High-level Architecture of an IPSec-dedicated System on Chip

Alberto Ferrante
ALaRI, Faculty of Informatics
University of Lugano
Lugano, Switzerland
ferrante@alari.ch

Vincenzo Piuri
Department of Information Technologies
University of Milano
Milano, Italy
piuri@dti.unimi.it

***Abstract*—IPSec is a suite of protocols which adds security to communications at the IP level. Protocols within the IPSec suite make extensive use of cryptographic algorithms. Since these algorithms are computationally very intensive, some hardware acceleration is needed to support high throughput.**

In this paper we propose a high level architecture of a System on Chip (SoC) which implements IPSec. This SoC has been thought to be placed on the main data path of the host machine (*flow-through* architecture), thus allowing for transparent processing of IPSec traffic. The functionalities of the different blocks and their interactions, along with an estimation of the internal memory size, are also shown.

I. Introduction

IPSec is mainly composed of two protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP). The former allows authentication of each IP datagram's headers or – depending on the operational mode that has been selected – of the entire IP datagram. The latter allows encryption – and optionally authentication – of the entire IP datagram or of the IP payload, depending on the operational mode that has been selected, namely the transport and the tunnel modes. The former was designed for being used in host machines, while the latter is for secure gateways. In tunnel mode the entire original IP datagram is processed; the result becoming the data payload of a new IP datagram with a new IP header. In transport mode only parts of the original IP datagram are processed (e.g. the data payload for the ESP protocol) and the original IP header is kept with some small modifications. Through encryption, authentication, and other security mechanisms included in IPSec (e.g. anti-reply), data confidentiality, data authentication, and peer's identity authentication can be provided [1], [2], [3]. In each of the protocols within the IPSec suite, many choices for cryptographic algorithms are available (for example AES, DES, Triple-DES, and many others can be used within the ESP protocol). The concept of Security Association (SA) is fundamental to IPSec. A Security Association is a simplex "connection" that afford security services to the traffic carried by it [4]. To secure typical bi-directional communication between two peers, two SAs (one in each direction) are required. Security services are afforded to a SA by the use of AH, or ESP, but not both. Security association establishment can be performed through a protocol named Internet Key Exchange (IKE) [3]. IKE is a two-phase protocol: in the first phase a bidirectional SA – named IKE SA – is established; in the second phase this SA is used to negotiate the parameters (protocol, protocol settings, keys, ...) for the IPSec SAs to be created. The use of IKE is not mandatory for IPSec and SAs can be established either manually or through other suitable protocols. Two databases are involved in processing IP traffic relative to security associations. These two databases are the Security Policy Database (SPD) and the Security Association Database (SAD). The former specifies the policies that determine the disposition of all IP traffic. The latter contains parameters that are associated with each SA. For each packet traversing the IP communication layer, the SPD needs to be queried. If, in conformance with the SPD, an IP datagram needs to be processed by IPSec, the SAD needs also to be queried to discover the parameters of the considered SA. Information about whether a SA has already been created or not are contained in the SPD. If a suitable SA for the IP datagram to be processed does not exist, it needs to be established, for example through IKE.

IPSec has proved to be computationally very intensive [5], [6], [7]. Thus, some hardware acceleration is needed to support large network bandwidths, as may be required even in small secure gateways. Most of the presently used accelerators are placed outside the main data path (*off-line*). An *on-line* implementation has been proposed in [8]: the accelerator is placed on the main data path. Thus, all the network-related data flows through it; for this reason it is also called *flow-through* architecture. When such an architecture is adopted, all the other host machine processing units may be completely unaware of IPSec: all the IPSec processing is performed inside the accelerator and normal IP packets need only to be processed by all the other parts of the system. A family of commercial products which are based on the flow-through architecture have been developed by Hifn [9]. These accelerators combine policy lookup, SA management, packet processing, and multi-algorithm encryption/authentication in a single chip. An on-board IKE implementation is also optionally available. These chips allow for in-band and, optionally, out-of-band control lines; they allow for 1Gbit/s full duplex IPSec traffic processing and they are capable of processing up to one million of packets per second. These accelerator provide

2006938380/07/$25.00 © 2007 IEEE

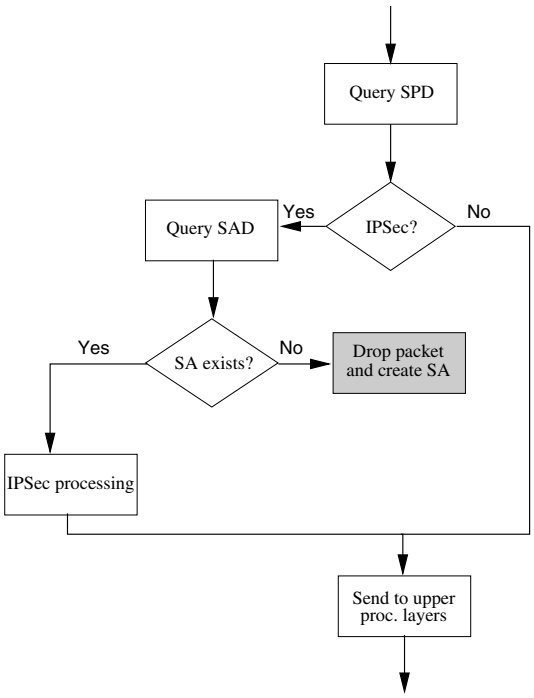

Fig. 1. Main packet processing steps for outbound IP datagrams.

the capability of storing 512 policy entries and 200 SAs on chip. Up to 256,000 SAs are supported through off-chip RAM.

In this paper we propose an architecture of a System on Chip (SoC) which is based on the flow-through idea. This chip has been designed to support a high throughput for secure gateways. The input/output interface has been designed to be especially efficient, thus allowing for fast processing also of the non-IPSec traffic. This interface, along with the design of the internal architecture, should exceed the processing capabilities of 10Gbit/s of traffic. Our chip has been initially designed with the original version of the IPSec protocols (AH v.1 [1] and ESP v.1 [2]) in mind; the support for the second version of these protocols (AH v.2 [10] and ESP v.2 [11]) can be easily added as the basic processing flow of the packets is very similar in the two versions. No comparisons can be performed with the internal architecture of the Hifn products, as just some high-level schemas are available for them.

In Section II we present an analysis of the main steps which are required to process IPSec packets. In Section III we show the internal architecture of our SoC, we identify the different functional blocks, their functionalities, and their way of interacting. In the same section we also discuss the size of the internal memory of the SoC.

II. Analysis of the Data Processing

Figure 1 shows the different operations that must be performed for each IP packet: a database query is performed and, if a proper SA exists, the packet is processed accordingly with the results of the query. The SA negotiation phase might take a long time compared to normal IPSec packet processing, as it involves communication between the two parties. Therefore, related network traffic can be discarded until the SA is created. This does not normally introduce any problem as packets may be later retransmitted by the other peer. All of these operations translate into the following processing steps:

1) the SPD is queried;
2) if the packet requires some IPSec processing:
   a) the SAD is also queried, thus discovering the operations to be performed on the packet;
   b) these operations are translated into commands for the different processing blocks;
   c) the previously generated commands are applied to the packet;
   d) the *I/O Unit* read each packet from the memory and outputs it on the proper I/O channel.

For nested SAs, step 2.c must be repeated as many times as the nested SAs are.

One important fact is that at least one SPD query must be performed for each outbound IP datagram. Considering an overall traffic of 1Gbit/s, and the worst possible case (i.e. the packets are received at the maximum possible rate and their size is the smallest possible one, that is 40 bytes), the SPD needs to be queried $3,355,443$ times per second. In a normal system operating at the same speed queries are usually less then one million per second. This is due to the fact that interarrival time between packets is often greater than zero. Packet size is also often greater than just 40 bytes [12]. When short bursts of packets exceeding the database query supported speed occur, packets may be queued and processed later. Longer bursts may cause system saturation. In that case packet discarding may occur, depending on the policy implemented on the system.

III. Description of the Architecture

In a flow-through device all the IPSec protocols need to be implemented. This meaning that all the cryptographic algorithms as well as the database query functionalities and IPSec header processing must be implemented on chip. Depending on the processing speed which needs to be supported, different hardware/software partitions for the functional units can be studied. The tradeoff between hardware and software should also include considerations about flexibility. Usually, hardware implementations provide better performance, but no flexibility. Software implementations, on the opposite, provide great flexibility, but non optimized performance. In our system some flexibility might be required in the implementation of the IPSec protocols; we have anyway to consider that protocol updates are not very frequent as they usually require a couple of years to be studied and accepted. Some flexibility might also be required for the cryptographic algorithms: hardware implementations of the commonly used algorithms can be provided; less common or newer algorithms can be supported by means of software implementations, as also discussed in [13]. Another solution, that may also be used for the implementation of the protocols, is to use reconfigurable hardware. This solution can provide good performance (even if not as good as with

standard hardware), and great flexibility. For example, if the cryptographic algorithms are implemented by means of reconfigurable hardware, this hardware can be reconfigured, either statically or dynamically, when new algorithms are required. Dynamic reconfiguration can help supporting many different algorithms with limited reconfigurable hardware resources. The main problem of this solution being the reconfiguration delays. In fact, each reconfiguration may require a long time and should not be performed too often, as discussed in [14].

The I/O interfaces and the internal architecture of the SoC need to be carefully designed to allow reaching the desired performances.

A. External Interface

An efficient interface with the communication channels is fundamental for reaching high performances. Loading data from outside the chip may take a long time if compared with the on-chip timings. As shown, for example, in [13], a good practice is to decouple the I/O of the processing units and of the chip from the data processing. Input and output buffers can be used effectively for this purpose. The input buffers can be easily managed as input packets must be processed in FIFO order.

The packets that are not requiring any IPSec-related processing only require to be forwarded from one interface of the SoC to the other; these packets would need, in a conventional accelerator, to be transferred anyway inside the accelerator. Our SoC adopts the innovative idea to move the database query unit inside the I/O-dedicated part of the SoC. This way, packets are allowed into the accelerators only if some IPSec processing need to be applied on them.

B. Description of the Functional Blocks

Figure 2 shows all the different functional blocks of the SoC. The *Net I/O Manager* ad the *Host I/O Manager* process the incoming and outgoing traffic from the network or from the host, respectively. These two units store the incoming packets in the shared memory through dedicated communication channels; in the same way they output the outgoing packets that are read from the shared memory. A unit for managing fragmentation of the IP packets is also required. In fact, both for inbound and outbound packets, fragmentation might occur. Inbound packets may be fragmented by any of the nodes on the network. Therefore, packets must be reassembled before applying IPSec processing. Outbound packets might need to be fragmented when, after IPSec processing, their size becomes greater then the MTU of the system. As specified in [4], fragmentation may occur with tunnel mode only. In our SoC fragmentation is managed by the two I/O units.

The *In DB Manager*, *Out DB Manager*, and *IKE DB Manager* queue the different policy and SA databases and generate the commands to be executed by the different processing units. The memory for database storage is shown here as an off-chip unit but it can also be placed on-chip. An on-chip memory provides faster access times; an off-chip one provides easier expandability. Wether it is possible or not to place the memory on-chip mainly depends on its size. Smaller memories can fit on-chip, but might not provide enough room for all the database entries that might be required. Speed problems of an external memory can be masked by adopting a caching mechanism as shown in Figure 3. The two caches are implemented by using Content Addressable Memories (CAMs) [15], [16]. This kind of memory allows to index its cells through their contents. Therefore, it provides a good way to lookup the security associations and the security policy records. Different query techniques and different levels of multithreading can be used in the database query unit, depending on its target speed. A discussion of the detailed architecture of the database query units is outside the scope of this paper.

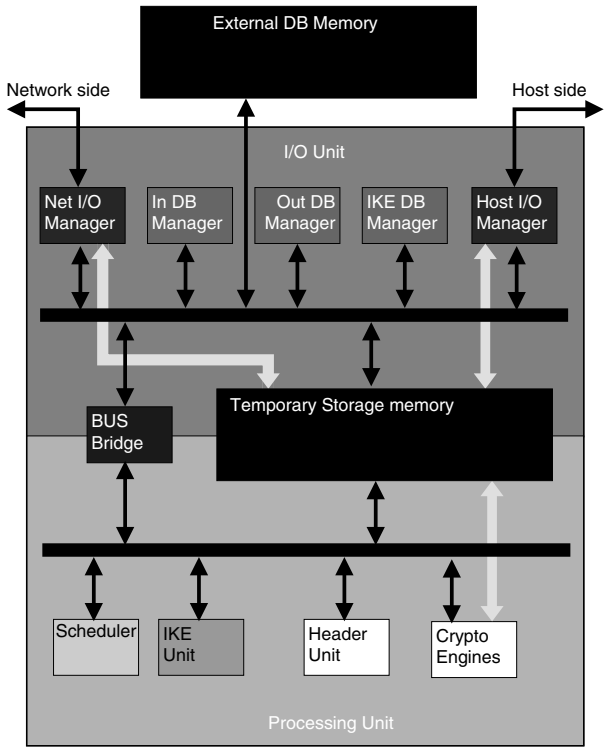

Fig. 2. High-level architecture of the SoC.

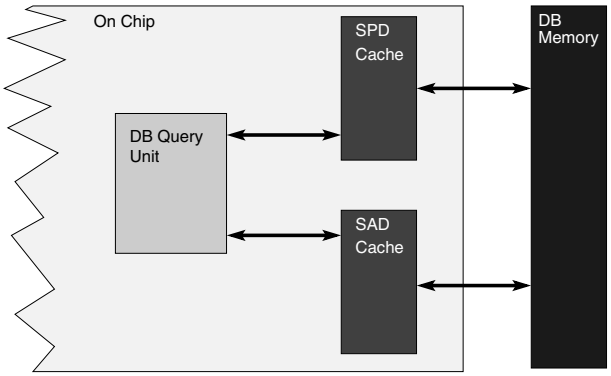

Fig. 3. A database query unit.

The *Scheduler* dispatches packets to be processed to the different processing units: the *Header Unit* and the *Crypto-engines*. The former is for processing the headers of the IPSec packets, while the latter is for applying cryptographic transformations to the data. The *Crypto-engines* unit contains different cryptographic cores (e.g., an AES, a DES, a SHA-1/2, and a MD5 engines); there can be multiple instances of each one of these cores. All of them might be enabled to run concurrently and the *Scheduler* should be able to handle them in a proper way. For example, a modified version of the algorithm proposed in [13] can be used for this purpose. In that paper a scheduling algorithm for an IPSec system with multiple cryptographic accelerators is presented. The scheduler runs on the CPU that can also be used as a data processor by means of software implementations of the cryptographic algorithms. In our case a CPU might not be present and if it is, the possibility to use it also for cryptography-related data processing should be carefully evaluated: this introduces the possibility to saturate the CPU, thus preventing the scheduler to assign the packets to be processed at the required speed.

Different key exchange protocols can be utilized in conjunction with IPSec. In this work we have only considered the IKE protocol and the manual keying. IKE works at application level, therefore, it requires underlying network and transfer protocols: IP and UDP in this case. Our SoC works below the IP level, therefore the only way of providing an on-chip IKE implementation is to also implement IP and UDP on-chip. These must be used only during IKE negotiations and they can be implemented in software: SA negotiation does not require very fast packet exchanges. A different solution is to use a separate unit for IKE. This unit can use the host's network stack, but it requires to access the same SAD and SPD databases that are also accessed by the SoC. In any case the chip must provide access to these databases as manual keying or the use of different key exchange protocols may be required. As an example, Figure 2 shows an on chip IKE unit (*IKE*).

C. The Overall High-level Architecture

As shown in Figure 2, shared memory is adopted as data communication model; a bus hierarchy is used for control communications. The chip can be subdivided in two parts that need high internal connectivity and that are interconnected in a good way. These two parts are the *I/O* unit and the processing unit. The *I/O* unit needs dedicated interfaces for the incoming packets, but all the other communications between the functional blocks and the local memory can be performed by means of a lower speed local bus. The processing blocks need to be connected together and to the processing memory. Another local bus can be used for this purpose. A bus bridge allows for bus interconnection. Data are written into the shared memory by the I/O unit; commands are then sent to the processing unit through the bus bridge; results of the processing are written back into the shared memory by the processing unit and a "data ready" signal is sent to the I/O unit through the bus bridge. The I/O unit transfers the data on the output. In summary, shared memory allows to improve efficiency, by avoiding large transfers of data on internal buses.

Query requests are broadcasted by the *Net I/O Manager* and by the *Host I/O Manager* on the local bus of the I/O-dedicated part of the SoC. The first memory address of each packet is also broadcasted on the same bus, along with the corresponding query request. The database query units analyze these requests to understand which one of them they need to processes. As soon as the suitable DB unit completes the query, it generates the appropriate commands, thus avoiding to have a dedicated control unit in the SoC. The commands can be:

- forward the packet to the other interface: this command is sent to the appropriate I/O manager which then starts reading the packet from the memory and sending it on its output interface;
- apply IPSec processing (along with the details on the operations to be applied): this command is sent to the appropriate processing unit which can be the *Scheduler* or *IKE*;
- drop the packet: memory blocks related to the packet are freed and a message is sent through the appropriate output interface.

DB queries can start before full packets are received, as only the packet headers are required for these operations. This speeds-up the overall packet processing. Packet forwarding from one interface to the other, can start before they are completely received.

Dedicated buses are used for communications between the two I/O units and the memory and between the cryptographic units and the memory. If a single shared bus was used a high number of bus conflicts would have arisen, especially, but not only, when packets needed to be forwarded. The I/O unit bus is primarily a control bus, as most of the information that it is carrying are commands.

The communication channels of the two I/O units are used both for read and write operations. Both of them might need to be used at the same time for input and output operations. This requires a communication channel that is at least twice faster than the network speed we want to support. Incoming data are saved in a small buffer (its size is only of one bus transfer unit). The common channel is used for input operations during the even cycles and for the output operations during the odd ones. There is no overhead as the memory unit is able to store the context of the previous operation. By using time multiplexing transfers can be performed in the same way as in burst mode even if the channel is shared. The same thing happens on the interface of the *Crypto-engines*, but in this case the interface needs to be faster to support twice the load of the other two interfaces.

Four read/write ports are required on the memory unit, one for each of the buses. In fact, besides the three memory interfaces that are needed for the two I/O units and for the crypto-engines, an additional port is needed for interfacing it with the two buses. This interface will be used by the *Header* unit, the DB managers, and the *IKE* unit (if present). This port does not

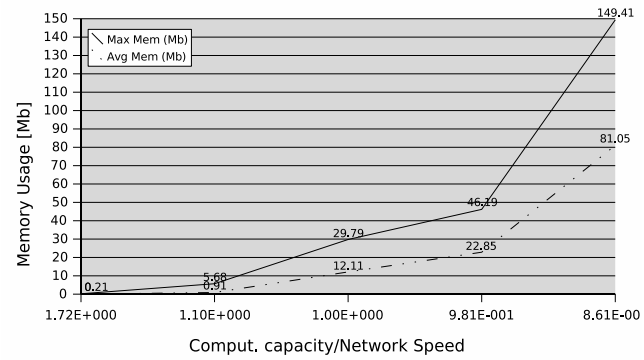

Fig. 4. Maximum and average memory usage comparison for different computational capacity over required network bandwith ratio.

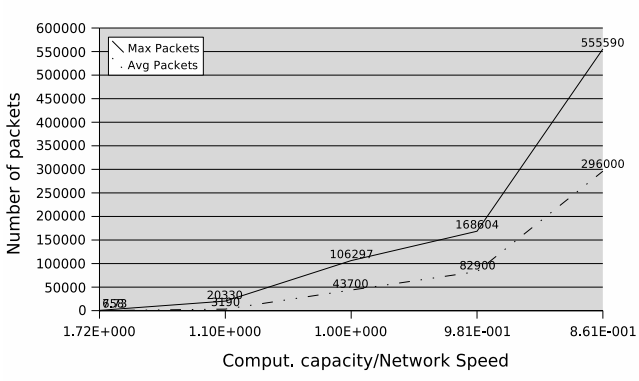

Fig. 5. Comparison between average an maximum number of packets that need to be stored depending on the ratio between computational capacity and required network bandwith.

require a large bandwidth. The memory is required to store the packets that are waiting to be processed. Theoretically, this memory should be just for buffering data and for storing intermediate results. In the reality, it will also be used for temporary data storage, as some packets will need to wait for some time before being processed. This is due to possible temporary burst of packets over the rate supported by the functional units. The size of the memory to be dedicated to this functionality can be roughly estimated through simulation. The SystemC [17] model we have developed for this is composed by a processing module and an input queue. This processing module receives information on the size of the packets and simulates the time which is needed to process them. Input data for the simulations have been taken from one of the Internet Traffic Archive traces [18]. Timestamps contained there have been multiplied by a suitable constant to obtain different average data rates. The considered trace contains data of about 3.8 million TCP packets. In these simulations we considered three different conditions: the required bandwidth is lower than one provided by the processing unit; the bandwidth is equal to the provided one; the bandwidth is higher than the provided one. In this work we are really interested only in the first two situations as the third one should happen only in poorly designed networking environments. The considered processing capability of the functional unit is of about 10Gbit/s. Average memory requirements are always considerably smaller than maximum ones as shown in Figure 4. Figure 5 shows that, even when the required bandwidth is supported by the processing units, a great number of packets need to be temporarily stored. We have anyway to take into account that in these experiments we were considering the average required bandwidth. This means that packets may arrive at a considerable higher rate than the supported one for short periods of time. When this happens, packets are stored to be processed later. In addiction we were considering the special condition in which no packet dropping happens. Therefore, in real life systems a quantity of memory which is slightly exceeding the average used one when the processing capabilities are slightly greater then the required bandwidth, can be adopted. This memory can be quantified, in our case, in 1Mb. Packets that cannot be stored will be dropped and may be later retransmitted by the other peer.

An efficient memory unit is crucial for system performances and an architecture for supporting efficiently four memory ports needs to be used. In this architecture data are managed by a separate interface for each connection to the unit; these interfaces contain read and write buffers: data are moved from or to the buffers by dedicated units. These units are inside the interface with the data channels. This way read/write operatios on the I/O ports become asynchronous with read/write operations into the shared memory. When read operations are performed, the whole requested packet (or parts of it) can be prefetched into the buffer. This is a convenient operation as complete packets, and not just parts of them, are usually processed or transferred to the output interfaces. It is convenient for the two interfaces which are dedicated to the I/O managers to share the addresses of the data blocks that are stored into the write buffers and, therefore, not yet written into memory. In this way, those two unitscan start outputting data while they are still being written in memory by the other interface. The number of addresses to check is in any case very small.

The central memory has high bandwidth requirements. Let us consider, for example, a chip designed for supporting a full duplex 10Gbit/s throughput: the total memory bandwidth that is required is of at least 80Gbit/s. Required physical memory performance can be reduced by exploiting the subdivision of the memory into different banks. Conflicts in accessing the banks can be reduced by using the modulo interleaving technique ([19, pp. 426–439]). By this technique memory addresses are distributed among different memory banks that can be accessed independently. Each memory bank need to be connected to each management unit; for this purpose, depending on the number of memory banks, different solutions, ranging from using direct separate connections to using an interconnection network, can be adopted. For example, the crossbar switch topology can be used to suitably interconnect the four managers to the memory banks [19, pp. 579–597].

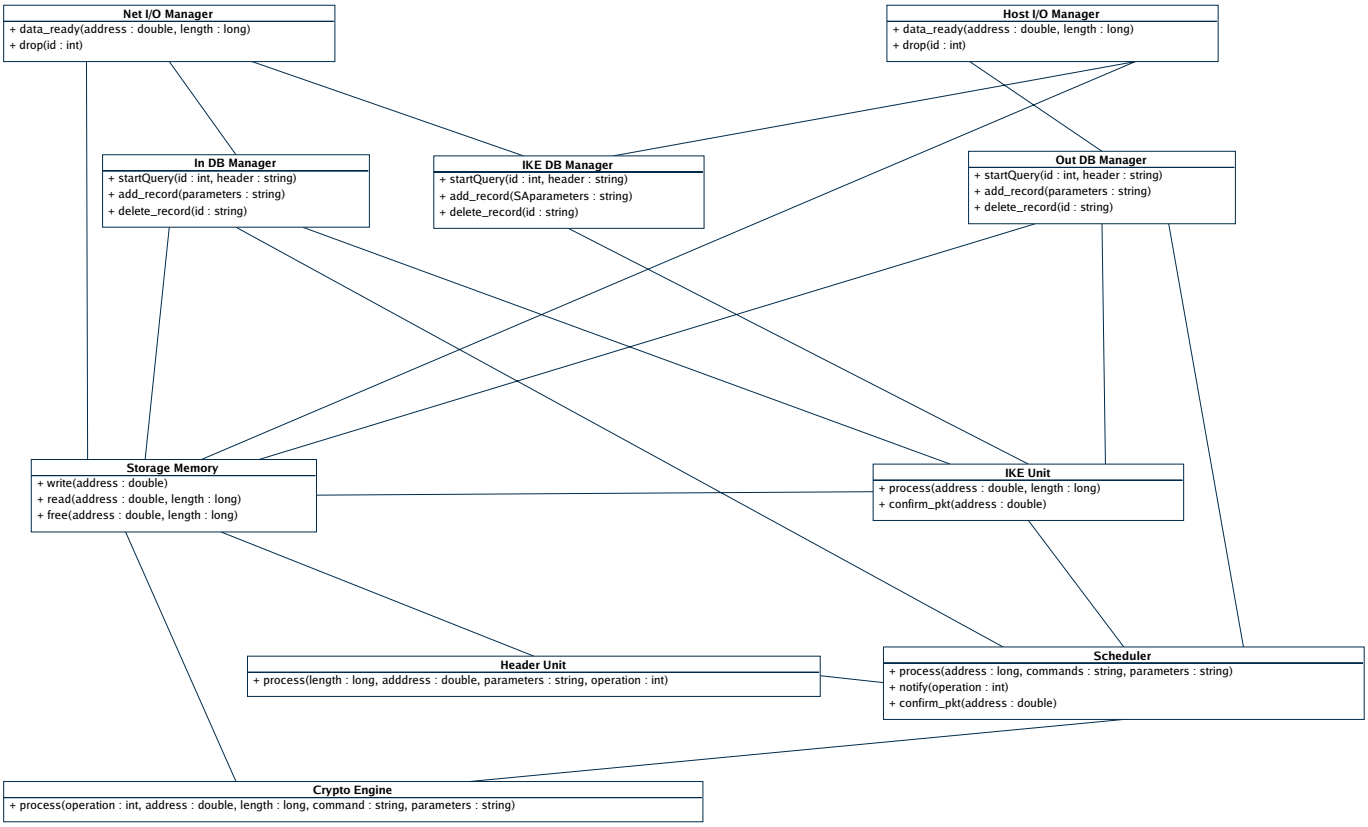

Fig. 6. UML class diagram describing the relations among the different parts of the SoC.

One interesting problem is related to the central memory management. Packets need to be saved into memory in an effective way to avoid fragmentation. An example on how to efficiently store packets is provided in [20]; this paper describes a memory architecture aimed at supporting packet priority management (for quality of service) in IPSec accelerators. A simplified version of this memory architecture can be used in non QoS-enabled devices. In fact, even if the quality of service mechanism proposed in that paper is not utilized, the memory address management one can be effectively used to avoid memory fragmentation.

D. Specifications of the Functionalities

Figure 6 shows a class diagram representing the different functional units of the SoC and their relationships. This diagram, along with a couple of sequence diagrams, specifies the behavior of the SoC. Figure 7 shows, as an example, the sequence diagram describing the behavior of the system when a non-IPSec inbound packet is received. Other sequence diagrams, which are not shown here due to space constraints, describe all the operations performed in the SoC. In all cases the packets need to be loaded into the memory by the I/O interface. In the meantime, simultaneous queries of the inbound and IKE databases start. As explained before, a query of the inbound databases happens only when the packet is identified not to be IKE-related. In the same way an IKE database query happens when a packet is identified to be IKE-related. These decisions are independently taken by the different DB units by looking at the headers. Other sequence diagrams describe the operations performed internally by each unit. For example, in Figure 8 the behavior of the *In DB Manager* unit is shown.

IV. Extending the Processing Capabilities of the SoC

There might be some cases in which our IPSec SoC is not fast enough to support the required throughput. Developing a faster chip might not always be possible. One possible solution to this problem is to use multiple accelerators in parallel; this is equivalent to using multiple network processors as shown in [21]. Unfortunately, this approach poses some problems related to the anti-reply IPSec mechanism. The anti-reply mechanism provides partial sequence integrity to detect duplicate IP packets within a certain window [4]. This service is implemented, both for AH and ESP, by means of a sequence counter [1], [2]. The verification and generation of this sequence number imposes some dependencies among different packets. Thus, the parallel processing of packets belonging to the same SA is not possible. Parallel processing of packets belonging to different SAs is still possible.

For parallel accelerators there are two possible configurations to be adopted: the first one is to use one accelerator for inbound traffic and another one for outbound traffic; the second possible configuration corresponds to a more generic

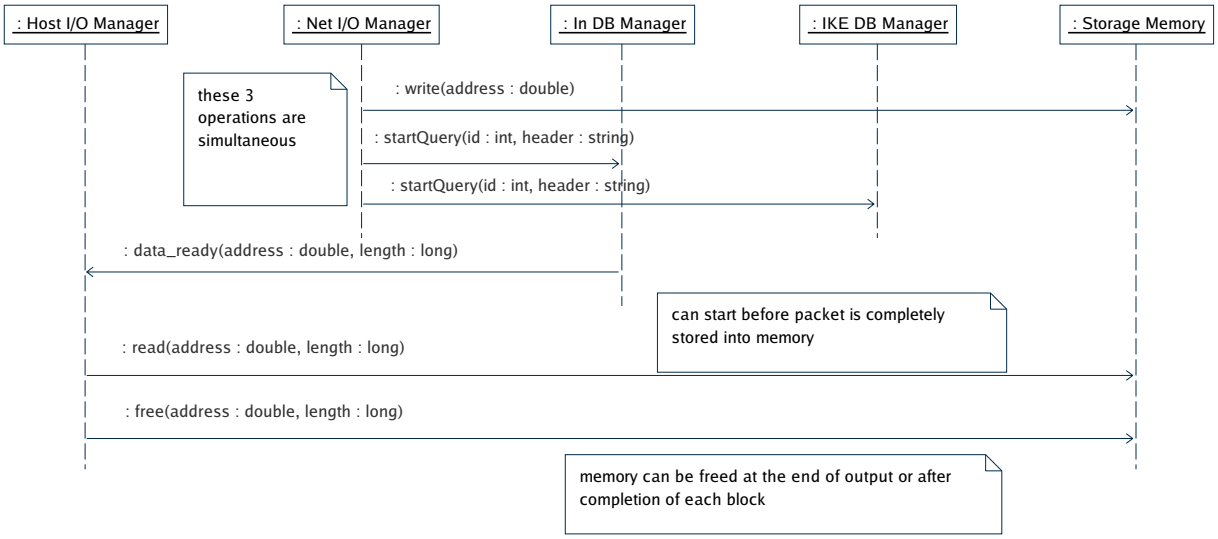

Fig. 7. Sequence diagram describing the behavior of the system when a non-IPSec inbound packet is received.

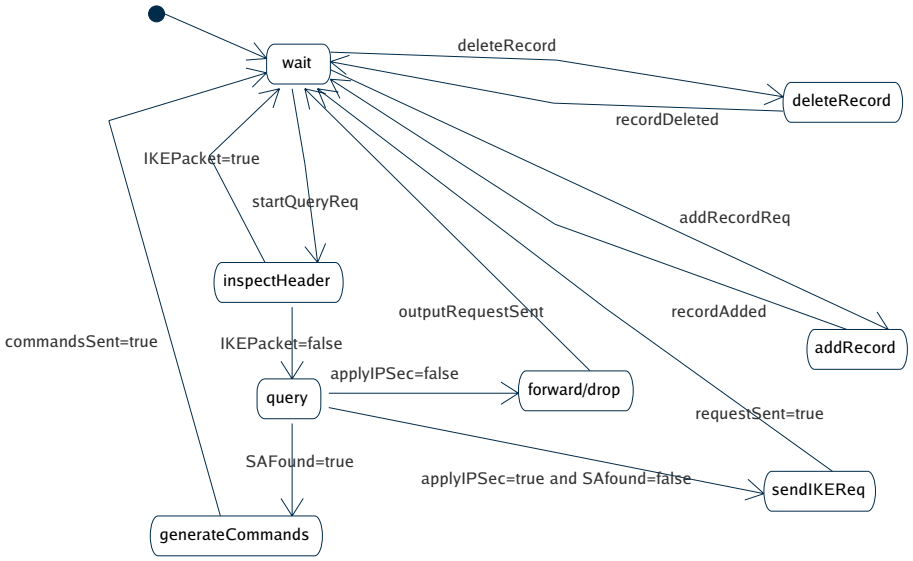

Fig. 8. Statecharts describing the behavior of the *In DB Manager* unit.

approach which provides the capability to use two or more accelerators without statically subdividing the flows related to the two directions (or subdividing them among more than just two accelerators). The former allows for a natural partition of the traffic, while the latter requires a load balancer to distribute it among the processors. As explained before, the load balancer must allocate just one packet per SA at a time for processing to avoid problems with the sequence numbers.

The management of the IPSec databases require either some kind of collaboration among the processors, or a static subdivision of the traffic. If the network traffic is statically distributed, each IPSec processor can manage its own databases independently from the others. The latter is the simplest solution, but it requires to perform an *a priori* estimation of the traffic and to statically allocate it to the processors. This may lead to have a non efficient use of the resources. Both by using dynamic load balancing and by using a static allocation of the inbound and outbound flows to different processors, require to share the databases or some of the information there contained among the processors. This can be performed in two ways: the first one is to share the database memory among the different processors; the second one is to keep local memories and to broadcast the modifications of the databases to all the other processors. The first technique should work well if coupled with local caches and a cache coherence protocol [19, pp. 654, 677]. The second technique is quite inefficient from the memory usage stand point as it requires to replicate the databases on the different processors. For each update, it also requires to broadcast the information about the modified records to all the other processors, thus requiring many interprocessor communications. An intermediate technique, which can be very effective in this case,

is to use a distributed shared-memory architecture [19, pp. 677, 693]. In this architecture every processor stores its own piece of the SAD and it shares it with the other processors. SPD can be centralized and completely shared. A directory-based coherence protocol can be used to preserve coherence of the SAD. This architecture works well coupled with a load balancer which tries to always allocate the processing related to already created SAs to the same processors; packets related to these SAs can be occasionally diverted to different processors when their usual processor is too busy. The idea beyond this load balancing technique has been taken from presently used load balancers for web servers.

V. Conclusions and Future Work

In this work we developed a high-level architecture of an IPSec-dedicated SoC. In this architecture data flows inside the chip are highly optimized. The speed of this architecture is mostly limited by the one of the memory. Actually this is a common problem for all network equipments [22, pp. 1, 8].

A study of the database query unit internal architecture is ongoing. Some modifications to the SoC for quality of service support are also being studied. Simulations of the architecture will be performed to verify its correctness, to tune its parameters (depending on the required processing speed), and to evaluate its performances. A study of the optimal IKE HW/SW partitioning is ongoing.

References

[1] S. Kent and R. Atkinson, "IP Authentication Header – RFC2402," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[2] ——, "IP Encapsulating Security Payload (ESP) – RFC2406," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[3] D. Harkins and D. Carrell, "The Internet Key Exchange (IKE) – RFC2409," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[4] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol – RFC2401," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[5] S. Miltchev, S. Ioannidis, and A. D. Keromytis, "A Study of the Relative Costs of Network Security Protocols." Monterey, CA: USENIX Annual Technical Program, June 2002.

[6] S. Ariga, K. Nagahashi, M. Minami, H. Esaki, and J. Murai, "Performance Evaluation of Data Transmission Using IPSec Over IPv6 Networks," in *INET*, Yokohama, Japan, July 2000.

[7] Alberto Ferrante, Vincenzo Piuri, and Jeff Owen, "IPSec Hardware Resource Requirements Evaluation," in *NGI 2005*. Rome, Italy: EuroNGI, 18 Apr. 2005.

[8] R. Friend, "Making the Gigabit IPSec VPN Architecture Secure," *IEEE Computer*, vol. 37, no. 6, pp. 54–60, 06 2004.

[9] Hifn Intelligent Packet Processing III (HIPP III). Hifn. [Online]. Available: http://www.hifn.com/technology/HIPP_III.html

[10] S. Kent, "IP Authentication Header – RFC4302," IETF RFC, Dec. 2005. [Online]. Available: http://www.ietf.org/rfc.html

[11] ——, "IP Encapsulating Security Payload (ESP) – RFC4303," IETF RFC, Dec. 2005. [Online]. Available: http://www.ietf.org/rfc.html

[12] (1997) WAN Packet Size Distribution. [Online]. Available: http://www.nlanr.net/NA/Learn/packetsizes.html

[13] Fabien Castanier, Alberto Ferrante, and Vincenzo Piuri, "A Packet Scheduling Algorithm for IPSec Multi-Accelerator Based Systems," in *ASAP 2004*. Galveston, TX, USA: IEEE Computer Society Press, Sep. 2004, pp. 387–397.

[14] Tyrone Tai-On Kwok and Yu-Kwong Kwok, "On the Design Of a Self-Reconfigurable SoPC Based Cryptographic Engine," in *ICDCSW*, vol. 07, no. 7. Computer Society, 2004, pp. 876–881.

[15] Kostas Pagiamtzis. CAM Primer. University of Toronto, Department of Electrical and Computer Engineering. [Online]. Available: http://www.eecg.toronto.edu/~pagiamt/cam/camintro.html

[16] K. Pagiamtzis and A. Sheikholeslami, "Pipelined Match-lines and Hierarchical Search-lines for Low-power Content-addressable Memories," in *IEEE Custom Integrated Circuits Conference*, 2003, pp. 383–386.

[17] "SystemC Official Website." [Online]. Available: http:/www.systemc.org/

[18] (2000) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[19] John Hennessy and Dave Patterson, *Computer Architecture: a Quantitative Approach*, 3rd ed. Morgan Kaufmann Pub, 15 May 2002.

[20] Luigi Dadda, Alberto Ferrante, and Marco Macchetti, "A Memory Unit for Priority Management in IPSec Accelerators," in *ICC07*. Glasgow, Scotland: IEEE Communications Society, 24 Jun. 2007, to appear in the proceedings of ICC 2007.

[21] John Marshall, "Cisco Systems – Toaster2," in *Network Processor Design*, P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, Eds. Morgan Kaufmann, 2003, vol. 1, ch. 11, pp. 235–248.

[22] P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, *Network Processor Design*. Morgan Kaufmann, 2003, vol. 1.