This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

A Memory Unit for Priority Management in IPSec Accelerators

Luigi Dadda
ALaRI, University of Lugano
Lugano, Switzerland
DEI, Politecnico di Milano
Milano, Italy
Email: dadda@alari.ch

Alberto Ferrante
ALaRI
University of Lugano
Lugano, Switzerland
Email: ferrante@alari.ch

Marco Macchetti
C.E.C.
Altran Group
Milano, Italy
mmacchetti@ceconsulting.it

Abstract— **This paper introduces a hardware architecture for high speed network processors, focusing on support for Quality of Service in IPSec-dedicated systems. The effort is aimed at defining a secure system on chip environment, where the speed and security requirements are of utmost importance. In particular, a method is devised to introduce and support Quality of Service through priorities at this level. An architecture of a memory system that provides automatic priority management is proposed.**

I. Introduction

The constant growth of the Internet and the undeniable importance of on-line services in every-day life and in the business world, generate new engineering problems. Bandwidth obtainable with current interconnection techniques (40Gbit/s for a single fiber-optic link and well beyond by using wave multiplexing) largely exceeds the bandwidth supported by network apparatus; therefore, several technical issues must be solved to adapt the communication devices (routers, gateways, servers) to operate at this speed. In fact the bandwidth request is growing even faster than the interconnection bandwidth.

In this paper we address the problem of providing Quality of Service (QoS) for fast secure communications. Here we concentrate on a hardware architecture for QoS management in an IPSec [1] environment. We present a memory architecture that enables QoS management in IPSec flow-through accelerators. Flow-through accelerators are placed on the data path; therefore, all the network-related data flows through them.

Nowadays specialized memory architectures are largely utilized; for example, content addressable memories (CAMs) are highly utilized in network apparatus. To our knowledge, no specialized memory structure exists for the purpose of supporting QoS. The memory architecture that we propose in this paper provides the ability to support QoS without any performance loss. The other great advantage of this architecture is the total lack of memory fragmentation. This problem may easily occur in any network related system due to packets having very different sizes. Supporting QoS through priority queues also introduces the problem of correctly dimensioning the memory allotted to each priority level. Fix dimension queues may not be the best choice due to possible variability in traffic related to each level. Our memory system natively supports dynamic queue resizing. The hardware support structure is mainly composed of some registers and of an additional small memory for the free memory block addresses.

As explained in the next section, a certain number of complex protocols and algorithms need to be used together to create a Virtual Private Network (VPN). Due to space constraints we do not discuss all of them here. In this paper we also do not deal with jumbo packets. These packets - that can be up to 4Gbyte long in IPv6 - might need to be processed in a different way than the one presented in this paper.

In Section II we describe the main technologies involved in VPN creation; in the same section we also give a brief overview of network Quality of Service (QoS). In Section III we describe the architecture of the memory system that allows for QoS management. In Section IV we show the model we have used for simulations and we discuss the results we have obtained.

II. Virtual Private Networks

A. Virtual Private Network Technologies and Protocols

Virtual Private Networks (VPNs) provide the ability to support communications on a shared mean (e.g., the Internet) in a secure way: secure protocols are used to build secure communication channels; machines connected to these channels can act as they were connected to a private network. Therefore, a private network is virtually built over a public one [1], [2].

A typical application for the VPN technology is to connect two private networks. This configuration is often used when two or more seats of the same company need to communicate and to share information. In each VPN there is one fundamental network component, the *secure gateway*. This machine manages the secure communications and it usually implements a firewall, a gateway, and a VPN server.

VPNs are created by associating two different protocols, one for data security and one for emulating a point to point connection. The latter is for routing privately addressed packets through a publicy addressed infrastructure. This task is usually performed by layer 2 tunnelling protocols. The most used point to point protocols are L2TP [3], [4], PPTP [5], and L2F [6].

To date the most widely used secure communication protocol for VPNs is IPSec. This protocol supports the creation of secure tunnels in a native way and it was thought for being used on secure gateways. 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, 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 [7], [8], [9]. 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 [10]. To secure typical bi-directional communications 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 may be performed through a protocol named Internet Key Exchange (IKE) [9], [11]. 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. The inbound and outbound security policies as well as the inbound and outbound SAs must be kept in separate databases.

As previously said, VPN deployment involves the use of different protocols and algorithms. Different tasks to be performed spread from query of databases to cryptographic algorithms, the latter usually being the most resource consuming ones: they are considered to be at least two order of magnitude more complex than any other algorithm used in networking applications. Some performance measurements for IPSec are provided in [12], [13]. All of these evaluations mainly concentrate on traffic processing capabilities without considering other problems such as query of databases. All these documents anyway show that software implementations of IPSec are too slow for allowing support for multi-gigabit networking. Even when the necessary network speed can be supported, the CPU usage is very high. Therefore running other tasks such as the ones that are necessary for the layer 2 tunnelling protocols or routing functionalities becomes unfeasible. We also have to consider that query of the databases are quite frequent as the SPD must be queried for each IP packet. In [14] the final proof that just accelerating cryptographic functionalities is not enough is provided. In fact, in that paper a maximum throughput of 45Mbit/s is obtained when IPSec is implemented on a IXP 425 network processor.

The actual trend is to move toward flow-through solutions for IPSec. IPSec packets are processed by a dedicated IPSec processor and passed to upper network processors [15]. These network processors may be totally unaware of the IPSec processor since incoming and outgoing traffic is formed by non IPSec-protected IP traffic. The flow-through solution has many advantages, one of them is to allow improving the overall system efficiency and therefore performance.

B. *Quality of Service*

The network Quality of Service (QoS) is the ability to provide different levels of service to different fluxes of data [16], [17]. This is usually obtained by assigning different priority levels to these fluxes. The priority levels are determined at network level and, in IPv4, they may be associated to each IP datagram by using a 3-bit header field. In IPv6 the size of this field has been increased to 12 bits, thus allowing to support more priority levels [18]. Priorities can be managed in different ways. The simplest one is to use a First In First Out (FIFO) policy on the incoming packets, but other more effective ways exist: Priority Queuing (PQ), Custom Queuing (CQ), Flow-based Weighted Fair Queuing (WFQ), and Class-based Weighted Fair Queuing (CBFQ) are the most used ones [16] .

It is important to note that QoS is only useful in congestion management. When no congestion is experienced on the system, QoS does not introduce any benefice over the flux management and over the performance. The concept of QoS has to be supported at system and at network level.

QoS has been assuming an increasing importance in the VPN environment, as services requiring real time support such as voice over IP are emerging. So far QoS has been supported only for local networks or, at best, for intra-provider communications through Service Level Agreements (SLAs). The trend is now to develop inter-provider QoS, but this poses some challenges such as the ones described in [19], [20], [21]. VPNs, most of all, are driving the request for this extension of QoS in inter-provider communications.

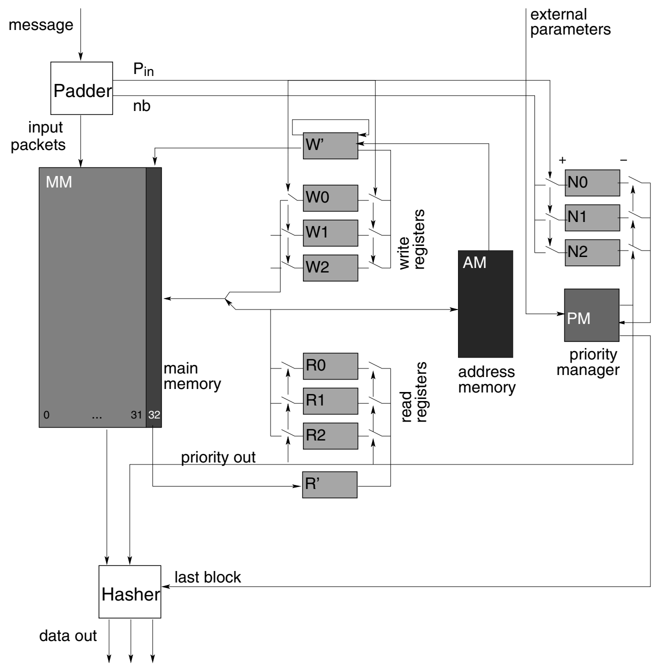

Fig. 1. The architecture of the memory for priority management (MQM). P_{in} is the priority of the packet being input.

TABLE I
EXAMPLE OF A 16-ADDRESS *AM* WHEN ADDRESSES ARE MANAGED IN LIFO WAY.

	a			b			c			d			e			f			g	
O		I	O		I	O			O		I	O		I	O		I	O		I
	1111			1111			1111			1111			1111		●	1111		○	––––	●
	1110			1110			1110			1110			1110			––––	●		––––	
	1101			1101			1101			1101			1101			––––			––––	
	1100			1100			1100			1100			1100			––––			––––	
	1011			1011			1011			1011			1011			––––			––––	
	1010			1010			1010			1010			1010			––––			––––	
	1001			1001			1001			1001			1001			––––			––––	
	1000			1000			1000			1000			1000			––––			––––	
	0111			0111			0111			0111			0111			––––			––––	
	0110			0110			0110			0110			0110			––––			––––	
	0101			0101			0101			0101			0101			––––			––––	
	0100			0100			0100			0100			0100			––––			––––	
	0011			0011			0011			0011			0011			––––			––––	
	0010			0010		●	0010			0010			0010			––––			––––	
	0001		●	0001			––––	●	●	0000			0000			––––			––––	
●	0000	○		––––	●		––––			––––	●	●	0001	○		––––			––––	

● an active pointer	–––– do not care about cell contents
○ a non active pointer	O output address pointer
	I input address pointer

III. The Proposed Architectural Solution

We describe the architecture of a memory unit suitable for storing several queues of messages. Each queue is characterized by a given priority level. The memory is organized in blocks of 16 words; these blocks are linked according to the message to which they belongs and to their priority. Each memory block can be associated to different priority levels in different periods of time (blocks cannot belong to different priority queues at the same time). Once freed, memory blocks can be reused in any of the priority queues. This leads to an optimal use of the memory without the need of any memory defragmenting operation. Moreover there is no predetermined memory space allotted to the different queues and this gives full flexibility in matching the capacities required for them in different instants of time. This properties can be obtained by adding some hardware to a system without QoS management. The cost of this hardware appears to be well compensated by the advantages obtained.

The architecture of the priority management system called *Message Queues Memory* (MQM) is shown in Figure 1. We assume here that each IPSec packet has a priority level associated with it. A discussion about how to attribute priority levels to packets is outside the scope of this paper. Each message is subdivided in blocks of 16 words of 32 bits and written in the *Main Memory* (MM). The length of the packet in blocks, nb, is added to the content of the N_p register.

The principle on which the MQM operates can be described as follows. Assume that the first block of the first message of one of the queues is placed anywhere in the memory. The address of the first word of the block (the 1st word of the block) is given by a binary integer whose last four bits are "0". The remaining 15 words are addressed by changing those four bits to: 0001, 0010,$\ldots$, 1111.

We assume as a working example that the full-address of each word, i.e. the word address, is composed by 20 bits (i.e. the memory capacity is 2^{20} words). It can be partitioned in:

- the 4 least significant bits composing the in-block address;
- The 16 most significant bits composing the block address, common to all full addresses of a block words.

To each memory word we add a bit (the 33rd) that allows to store the address of the next block of the packet. This address is recorded during the write process. The full address of the next block will be available when the last word of the block is read, so that the next block can be addressed. Two of these 16 bits are used to identify the first and the last block of a packet. With the above assumptions, the number of addressable blocks in the memory will be 2^{14} $(16,384)$ that is equivalent to $1,048,576$ bytes. The addition of a 34th bit in each word could be used to increase the memory capacity, if necessary.

Note that all blocks and all messages linked to the first address constitute a queue. Additional queues can be obtained by adding new "first blocks" that are not belonging to any of the other queues. For this a unit capable of managing the list of the unused addresses is required. This functionality is provided by the *Address Memory* (AM) of Figure 1. *AM* outputs addresses of free blocks; addresses of released blocks are written back to *AM*.

We will now show the operation of the system with some more details. Let us consider the *initial phase*, where registers W_0, W_1, and W_2 will be filled with addresses taken from *AM*. Registers R_0, R_1, and R_2 will take the values stored in W_0, W_1, and W_2, respectively. R_0, R_1, and R_2 contain the addresses of the first blocks of each queue and will be changed (automatically) during the reading phase (each message queue is managed by a First In First Out – FIFO – policy).

Let us now see how incoming blocks are stored in the different priority queues, and how those blocks can be read and sent to the cryptographic unit (i.e., the *operational phase*).

Assume that a block having priority 0 is input to *MM*. The priority index (P_{in}) will operate on the switches for selecting the appropriate registers, i.e., in this case, the registers W_0 and N_0. Simultaneously an address will be sent from *AM* to register W'. The successive 16 words of the input block will be stored in *MM*, the block address being given by W_0, the in-block addressing being generated within *MM*. During this process, the successive 16 bits of W' will be stored along the successive block words in the 33rd bit of the same words. The circulation around W' operates as a (left) shift to obtain at the output (at its left) the successive bits to be recorded. Note that registers W_j, at the beginning of the operational phase, store the addresses of the first block in the queues characterized by $p = j$. At the end of the store process, W' is transferred (in parallel) into W_j, W' being the address of the next block in the same queue.

Let us suppose that we want to read just one block in queue 0. Register R_0 will address the first word of the block and, by increasing its last four bits by 1 at each clock, all the remaining 15 words. On reading the successive words we will store in a register R' the bits constituting the address of the block following the one that is being read. R' will be then transferred in R_0 for reading a second block of same queue.

Note that while registers W_j are selected according to the priority assigned to the message to be written in *MM*, registers R_0, R_1, and R_2 and the right -1 inputs to N_0, N_1, and N_2 are controlled by the *Priority Manager* (PM).

Any address obtained from *AM* is first stored in W' then transferred to W_j and to the block itself; this address is then transferred in R', then in R_j and finally back to *AM*, (not necessarily in the same cell of origin).

The input to R' is derived serially from the 33rd bits of the block words; the end of block signal can be derived from the four stage counters producing the x_3, x_2, x_1, and x_0 generating the addresses of the 16 words block (precisely at the transition from 1111 to 0000 values). Suppose we now want to input another block which has the same priority. This will be done by following exactly the procedure described for the first block.

Note that the only blocks initially assigned to the different priorities are the ones whose address have been written in the initial phase. If a certain priority level is never used by the incoming messages, the respective first block will never be used too. This is a price to be paid for the flexibility obtained.

Addresses stored in *AM* need to be managed in a proper way. As explained before, addresses are taken out from *AM* and written back to *AM* when the corresponding memory blocks are freed. We use two pointers, one for the address to output, and another one for the address to be written back. Every time an address is output by *AM*, both pointers are moved by one position. Since the two pointers move synchronously up or down they can be implemented by a single up-down counter. By adopting this mechanism, addresses are output in a LIFO (Last In First Out) order. The output pointer references the head of the LIFO; the input pointer references the first free cell after the head of the LIFO. A decoding network for each possible position of the input-output pointers couple is required.

Table I shows the behavior of *AM*. The different columns show the contents of the memory and the position of the input and output pointers in different instants of time. In column a the input (I) and the output (O) pointers are at the bottommost position. I is non active as the memory is full and no further addresses can be written in *AM*. In column b, I and O have been moved one step up; address 0000 has been output; therefore, I has become active. Column c shows the same behavior for the address 0001. In column d the address 0000 has been released and therefore it has been written back in *AM*; consequently, I and O have been moved one step down. The same happens in column e with the address 0001; I has become inactive as the memory as become full. Column f shows *AM* when just one memory address is available. Column g shows *AM* when no memory addresses are available; therefore, O has been put in the inactive state.

A FIFO policy can be also implemented instead of the LIFO one. Two mono-directional counters are required for this. After an address is output, the output pointer steps upward; after an address is written in the *AM* the input pointer steps upward pointing to an empty cell. When both pointers are pointing to the same full cell the output pointer prevails.

The priority management scheme presented above, requires some additional hardware to be implemented. *AM* – which is 1/32 of the data part of *MM* – needs to be introduced. In *MM* an additional bit for each 32-bit word of data is required. Therefore, another 1/32 of its size need to be added. Eleven 32-bit memory elements are also required for the N, R, and W registers. Roughly, by introducing this QoS management system, we enlarge the memory by the 7% of its normal size. Note that the influence of this additional hardware would be halved if a SHA2-512 algorithm was considered (the size of the blocks would be twice the one considered for SHA2-256).

IV. Simulation of the QoS Management Unit

The architecture that we described in the previous section has been simulated by means of a SystemC [22] model. Using this language, hardware-software systems can be easily modeled and simulated.

The first goal of this model is to provide a functional evaluation of the QoS management mechanism and of the memory. For this reason only the HMAC-SHA2 [23], [24] cryptographic functionality has been implemented. In real-life systems at least a symmetric cryptographic algorithm (e.g., AES) is also required. In any case, the HMAC-SHA2 algorithm is the one with longer computational times per data block among the ones that can be used in IPSec systems. See [25] for an example of a SHA2 implementation. The simulations also allowed to obtain some rough estimations of performance figures. The next subsections show the model and the results of the simulations.

A. The SystemC Model

The SystemC model describes the different parts of the architecture that has been presented before along with a

HMAC-SHA2 processing module. Each one of these units have been implemented by different SystemC classes: *MM*, *AM*, *control_block*, and *HMAC*. The control block has two different parts: the first one writes data blocks to memory, while the other one reads them. The model works as follows:

1) the *control block* receives input packets and write them to the memory (by means of the *write* function) in FIFO order;
2) the *HMAC* block requests blocks to be processed to the *Control Block*; this unit provides blocks of the suitable priority level;
3) data blocks are processed by the HMAC block and the result is output.

Three different packet discarding policies have been implemented in the *write* function. These policies are used when the system is saturated and incoming packets need to be discarded. The policies that have been considered are: unconditional packet discarding, proportional packet discarding, and uniform packet discarding. When the first policy is adopted packets are discarded regardless their priority level. When the second policy is adopted, packets are discarded when the memory occupied by other packets of the same priority level is higher than a certain limit called f_p. The fraction of memory available for the priority level p when P different levels are considered is

$$f_p = \frac{(p+1)}{\sum_{i=1}^{P} i}$$

When the third policy, the uniform discarding one, is adopted, packets are discarded when the quantity of memory occupied by other packets of the same priority level is higher than $f = 1/P$. Therefore, the fraction of the memory available to each priority level does not depend on the level itself, but just on the number of levels that are considered.

The priority level of the packets to be processed by the HMAC block is determined by the *read* function of the *control_block*. The priorities are changed following a proportional round robin scheme. During each round robin cycle, at each priority level is given a certain number of processing slots (i.e. data blocks). This number is proportional to the priority level. In our simulations, each priority level is allowed to process up to $10 \times (p + 1)$ blocks per each round robin cycle.

The inputs of the simulations are managed by the *testbench* class; this class reads packets from a binary data file and calls the write function of the control block. No interarrival time is considered for packets (worst case) and the network speed (i.e. the delays to be applied in the *testbench* class) is determined by the time required to receive each byte of the packets. The input data file has been obtained by considering artificially generated packets along with their HMACs and their keys. The lengths of these packets are read from network trace files. We have used two of them: an artificially generated one and a real life one. In the artificially generated one, packets of the same size are replicated on different IP addresses. The real life trace file has been chosen from the ones available on the Internet Traffic Archive website [26]. Two different ways for distributing packets among the different priority levels have been considered both for the artificial and for the real traces: packets distributed in a cyclic way (the 1st packet is assigned to priority 0, the 2nd to priority 1 and so on) and packets distributed depending on their source IP address (IP address modulo the number of levels).

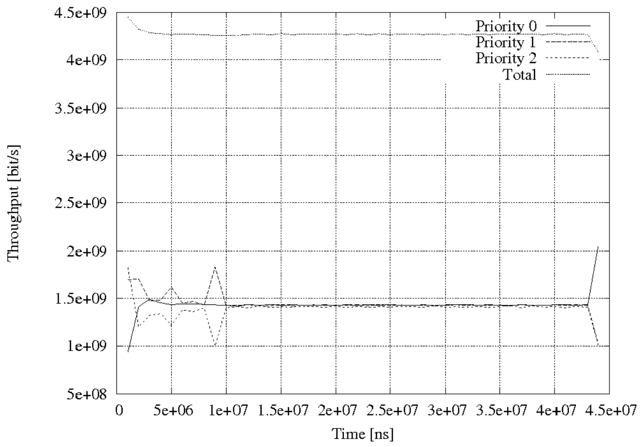

Fig. 2. Throughput obtained for the different priority levels when the artificial trace with cyclic distribution of priorities is considered. The discarding policy is the unconditional one.

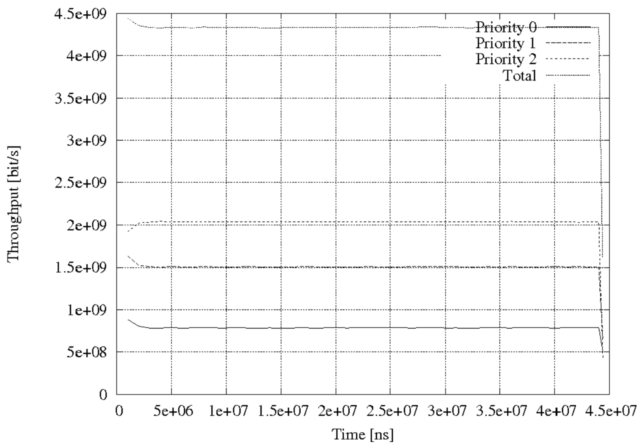

Fig. 3. Throughput obtained for the different priority levels when the artificial trace with cyclic distribution of priorities is considered. The discarding policy is the uniform one.

B. Simulation Results

As explained before, the main goal of these simulations was to perform some functional verification of the priority management architecture. Some performance figures can anyway be roughly estimated. Sixteen different simulations have been performed to consider all the different discarding policies that we have discussed above. All of them have been simulated both with the artificially generated traces and with the real ones. Both of the methods for distributing priorities among packets have been considered. The real network trace with an IP address based distribution of priorities gives closer to the reality results. The artificial trace with cyclic distribution of packets among priorities provides an easiest evaluation of simulation results.

Figure 2 shows the throughput obtained for the artificial trace with an uniform distribution of packets among the different priority levels. In this figure an unconditional discarding policy has been considered. In this case the throughput is practically the same for all the priority levels. This is due to the fact that the memory becomes very soon saturated by the blocks of the lower levels of priority. In fact, many of these packets need to be stored in memory, since higher priority level packets have more processing slots per cycle. By using a proportional or an uniform discarding policy, priorities are managed in a more effective way as shown in Figure 3. In fact, in this case the throughput obtained clearly depends on the considered priority levels. The lower levels are the ones obtaining lower throughput. The main difference in terms of performance among the proportional packet discarding, and the uniform packet discarding policies is in the number of discarded packets and in the processing latency for each level. The uniform discarding policy gives a better distribution of processing latencies among priorities. The processing latency obtained for each of the levels when this policy is adopted, is proportional to the level itself. Higher priority packets are processed with lower latencies. The proportional discarding policy allows to discard packets in a way that is more respectful of the priority levels, at least when the real network trace is considered. When an artificial trace with uniform distribution of priorities is utilized, both of these policies are equivalent from the packet discarding view point.

V. Conclusions and Future Work

In this paper we provided a hardware architecture for supporting quality of service in an IPSec accelerator. We also provided some simulations which prove that the priority management mechanism – which is mainly composed by a specifically designed memory – works as desired. The results obtained by simulations, show that packet discarding policies are fundamental to obtain good results with any QoS management system.

Lower level and, therefore, more precise simulations need to be developed to better estimate the device performance. At the moment only a HMAC algorithm has been simulated; future work includes simulations with encryption algorithms. The discarding policies, the algorithm parameters (e.g., the round robin cycle), and their interactions with priority management policies will be further studied and improved.

References

[1] R. Yuan and W. T. Strayer, *Virtual Private Networks*. Addison Wesley, 2001.

[2] J. Feghhi and J. Feghhi, *Secure Networking with Windows 2000 and Trust Services*. Addison Wesley, 2001.

[3] Layer 2 Tunnelling Protocol. Cisco. [Online]. Available: http:///www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t1/l2tpt.htm

[4] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter, “Layer Two Tunneling Protocol ”L2TP” - RFC 2661,” IETF RFC, Aug. 1999.

[5] K. Hamzeh, G. Pall, W. Verthein, J. Taardu, W.A. Little, and G. Zorn, “Point-to-Point Tunnelling Protocol (PPTP) - RFC 2637,” Tech. Rep., July 1999.

[6] A. Valencia, M. Littlewood, and T. Kolar, “Cisco Layer Two Forwarding (Protocol) L2TF - RFC 2341,” Tech. Rep., May 1998.

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

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

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

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

[11] C. Kaufman, “Internet Key Exchange (IKEv2) Protocol – RFC 4306,” IETF RFC, Dec. 2005. [Online]. Available: http://www.ietf.org/rfc.html

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

[13] 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.

[14] Yi-Neng Lin, Chiuan-Hung Lin, Ying-Dar Lin, and Yuan-Chen Lai, “VPN Gateways over Network Processors: Implementation and Evaluation,” in *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS 05)*, IEEE Computer Society Press, Ed. IEEE, Mar. 2005.

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

[16] Sean Convery, *Internetworking Technologies Handbook*. Cisco Press, 19 Apr. 2004, no. ISBN158705115X, ch. 49, pp. 49–1 – 49–32.

[17] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An Architecture for Differentiated Services – RFC2475,” IETF RFC, Dec. 1998. [Online]. Available: http://www.ietf.org/rfc.html

[18] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification – RFC2460,” IETF RFC, Dec. 1998. [Online]. Available: http://www.ietf.org/rfc.html

[19] Philip Jacobs and Bruce Davie, “Technical Challenges in the Delivery of Interprovider QoS,” *IEEE Communications Magazine*, vol. 43, no. 6, pp. 112–118, June 2005.

[20] Luyuan Fang, Nabil Bita, Jean-Luis Le Roux, and Jaime Miles, “Interprovider IP-MPLS Services: Requirements, Implementations, and Challenges,” *IEEE Communications Magazine*, vol. 43, no. 6, pp. 119–128, June 2005.

[21] Michael P. Howarth, Paris Flegkas, George Paviou, Ning Wang, Panos Trimintzios, David Griffing, Jonas Griem, Mohamed Boucadair, Pierrick Morand, Abolghasem (Hamid) Asgari, and Phanos Georgatsos, “Provisioning for Interdomain Quality of Servuce; the MESCAL Approach,” *IEEE Communications Magazine*, vol. 43, no. 6, pp. 129–137, June 2005.

[22] “SystemC Official Website.” [Online]. Available: http:/www.systemc.org/

[23] Descriptions of SHA-256, SHA-384, and SHA-51. [Online]. Available: http://csrc.nist.gov/cryptval/shs/sha256-384-512.pdf

[24] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication – RFC 2104,” Tech. Rep., Feb. 1997.

[25] L. Dadda, M.Macchetti, J. Owen, and S. Chakrabarti, “An ASIC Design for a High Speed Implementation of the Hash Function SHA-256 (384, 512),” in *GLSVLSI 2004*, Boston, Apr. 2004, pp. 421–425.

[26] (2000) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/