A Packet Scheduling Algorithm for IPSec Multi-Accelerator Based Systems

Fabien Castanier
Advanced System Technology
ST Microelectronics
Email: fabien.castanier@st.com

Alberto Ferrante and Vincenzo Piuri
Department of Information Technologies
University of Milan
Email: {ferrante, piuri}@dti.unimi.it

Abstract

IPSec is a suite of protocols that 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 discuss a scheduling algorithm for distributing IPSec packet processing over the CPU with a software implementation of the cryptographic algorithms considered and multiple cryptographic accelerators. High-level simulations and the related results are provided to show the properties of the algorithm. Some architectural improvements suitable to better exploit this scheduling algorithm are also presented.

1: Introduction

IPSec is a suite of protocols that adds security to communications at the IP level. This suite of protocols is becoming more and more important as it is included as mandatory security mechanism in IPv6. IPSec is mainly composed of two protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP). The former allows authentication of each IP datagram's selected header fields 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, 4]. IPComp, a protocol for data payload compression, is also included in the IPSec suite of protocols [5].

IPSec is often used to create Virtual Private Networks (VPNs). A VPN is an extension of a private network on a public network (e.g., the Internet) [6, 7]. The extended part of the network logically behaves like a private one. Typical usage scenarios for VPNs are: remote user access to a private LAN over the Internet and connection of two private networks. In these cases a virtual secure channel needs to be created, respectively, from the user's PC to the LAN public access point or from one LAN to the other. Private network public access points are called *secure gateways*. A secure gateway is a router or a router/firewall also running a VPN-enabled software (e.g., an IPSec implementation). All the traffic inside the LAN is usually not protected, while the traffic going out or coming in the LAN through the secure gateway is protected by some security mechanisms.

IPSec has proved to be computionally very intensive [8, 9]. Thus some hardware acceleration is needed to support large network bandwidths, as may be required even in small secure gateways. Usage of mixed hardware-software solutions for this, especially for low-end systems, has become a common practice in the last years [10, 11], since it provides flexibility and performance.

While using multiple accelerators for the same task is not a very common practice especially for low end systems, it may allow for expandability at reduced cost. The scheduling algorithm presented in this paper allows to schedule packets to be processed either by the CPU with a software implementation of the considered cryptographic algorithms or by the accelerators. This also enhances scalability: one may use a accelerator tailored for the bandwidth normally required for VPNs and use the CPU to have a further processing capability when a higher bandwidth is required (the secure gateway may also provide support for non-encrypted traffic, thus the total network bandwidth that is available may be larger than the VPNs one). This may be convenient so that the short and not too high peaks over the estimated VPNs bandwidth can be supported at no additional hardware cost. While this may not fit for high-end systems, it may do so for network edge ones, like, for example, secure gateways run by small companies.

In [12] an overview of multiprocessor scheduling algorithms is given. None of these algorithms fully exploits the following two characteristics which are typical in IPSec packet processing (see Section 3): processing time a priori known and no data dependency exists between different packets. In [13] an algorithm for scheduling a set of n independent tasks in a multiprocessor is given. This algorithm is not suitable for our problem, since it requires prespecified processor allocation. [14] discusses the problem of load balancing (both static and dynamic) on multiple processors. While this could be interesting for many problems, our goal was different: to minimize latency and, possibly, maximize throughput.

In this paper we present a new scheduling algorithm that is suitable to tackle the envisioned application and fully exploits the two characteristics mentioned above. Section 2 describes the architecture of the considered system. Section 3 describes the scheduling algorithm, section 4 presents the model for the simulations and finally the obtained results.

2: System Architecture

The system considered here is composed by a host computer and N cryptographic accelerators connected to the normal system bus; a PCI bus (32bit, 66MHz) [15] is adopted here as an example. We consider heterogeneous accelerators, i.e., accelerators implementing different cryptographic algorithms and allowing different processing speeds. CPU-memory communication is performed on a faster bus, as in most modern personal computers. The network card is also connected to the faster CPU bus. Only cryptography-related operations are offloaded to the accelerator(s). This means that all the IPSec header processing is done by the CPU. Pieces of data to be processed are stored in main memory and each processor loads them in its local memory by using DMA.

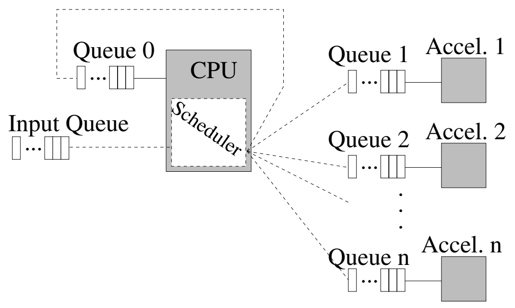

Figure 1. Reference scheme for the scheduling algorithm.

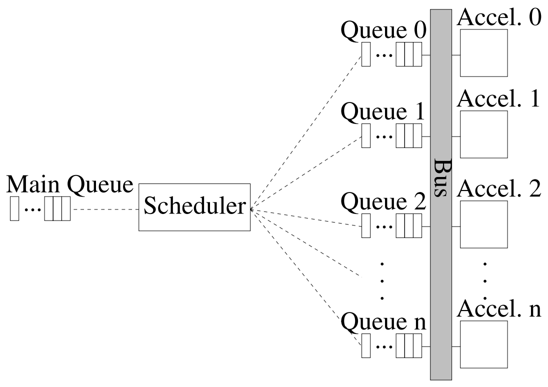

Figure 2. Simulation scheme.

This is only a sample architecture we have used to test the properties of our scheduling algorithm. Higher throughput systems should use different system architectures.

3: The Scheduling Algorithm

In the envisioned architecture each accelerator can support different sets of algorithms and different processing speeds. A common interface (i.e., an API) is therefore needed to allow for uniformly accessing to all the cryptographic devices. This common interface should also allow for accessing the software implementations of the cryptographic algorithms.

In this section we present the assumptions on which our scheduling algorithm is based and their motivations, the scheduling algorithm, and some limitations of our approach. In this paper we use the word *job* to identify a set of cryptographic operations that have to be performed on a packet.

3.1: Assumptions

Our algorithm is based on two fundamental characteristics: the first one is that the processing time for packets is known (at least approximately) in advance. This is true for symmetric-key cryptographic algorithms which are normally used within the IPSec context: their processing time only depends on the number of data blocks to be processed. The only exception is for the software implementations of these algorithms: in this case the computation time may vary depending on the current CPU load. The second characteristic is that each packet can be processed independently from the others (i.e., there are no data dependencies between different packets). This comes from IPSec specifications: each packet must carry any data required for its processing [3]. In [16] it is explained how to obtain data independency among packets for AES. Our approach fully exploits these characteristics to achieve high performance.

3.2: Description of the Algorithm

The goal of our algorithm is to minimize the average waiting time of the packets, thus minimizing their average processing time. Since scheduling jobs on multiple processors is a NP-complete problem, we adopted a heuristic approach. This is the first attempt published in the literature to tackle this problem within a comprehensive design framework. Exhaustive evaluation of heuristic scheduling algorithms to identify the optimal one is behind the scope of this paper.

The main idea underlying our algorithm is to allocate the packets to be processed on the processor (i.e., either one of the accelerators or the CPU) which can provide the shortest processing time. Our scheduling algorithm processes each packet as follows:

- a set of processors that are able to perform the cryptographic algorithm(s) required by the considered packet is selected,
- the *finishing time* for each of these processors is computed, being the *finishing time* defined as the sum of the *waiting time* and of the *processing time* of the packet scheduled on the considered processor,
- the packet is scheduled to the processor with the lowest finishing time. When all of the queues are empty new tasks are scheduled to the first fastest accelerator. Each processor processes the assigned packets in FIFO order.

In Figure 1 the reference scheme is of the algorithm is presented. The scheduler runs on the CPU. Considering N accelerators and the CPU, the computational complexity of the scheduling algorithm is $O(N)$.

The *waiting time*, $w_i(t)$, for the i-th processor at time t is defined as the sum of the processing time of all the packets contained in the related queue and of the *residual processing time* of the packet being processed at that time by the processor. In this paper we identify the CPU as the processor 0.

Let's be $\widetilde{w}_i(t)$ the waiting time, $w_i(t)$, without considering the residual processing time of the packet being processed at the time t by the i-th processor. $\widetilde{w}_i(t)$ can be computed as:

$$\widetilde{w}_i(t) = \sum_{j=1}^{J_i} c_{ij}, \quad 0 \leq i \leq N \tag{1}$$

where: j is the position of the job into the FIFO queue of the accelerator, nearest to the head of the queue. J_i is the length of the queue. c_{ij} is the computation time of the job j. The waiting time, $w_i(t)$, for each processor can be computed as:

$$w_i(t) = \widetilde{w}_i(t) + r_i, \quad 0 \leq i \leq N \tag{2}$$

where: r_i is the residual processing time of the tasks being currently executed on the i-th processor. r_i can be computed as:

$$r_i = c_i^p - (t - t_i^p), \;\; 0 \leq i \leq N \tag{3}$$

where: c_i^p represents the processing time needed for the packet that is being processed; t_i^p is the time at which the computation on that packet started (therefore $t - t_i^p$ represents the elapsed computation time).

From the previously given definitions, the finishing time, f_i can be computed as:

$$f_0 = \alpha_0 \left(w_0 + p_0 + \beta_0 \right), \qquad \alpha_0 \geq 1, \qquad \beta_0 \geq 0 \tag{4}$$

$$f_i = w_i + p_i, \quad 1 \leq i \leq N \tag{5}$$

where: p_i represents the processing time of the considered job on the i-th processor; α_0 and β_0 are two parameters that can be used to modify the finishing time computed for the CPU and control the CPU job allocation. The higher the value of α_0 and β_0, the higher the computed waiting time (that will be, in this case, different from the real one) for the CPU. Thus the lower will be the number of packets scheduled on it.

$\widetilde{w}_i(t)$ can be updated each time a datagram goes in or out of a queue. By assuming that the queue i is filled up to the J_i position, updating $\widetilde{w}_i(t)$ when a new datagram is added to the queue can be done in the following way:

$$J_i(t) = J_i(t-\tau) + 1 \tag{6}$$
$$\widetilde{w}_i(t) = \widetilde{w}_i(t-\tau) + c_{iJ_i}(t) \tag{7}$$

where $\tau > 0$ represents the time elapsed from the last update of the considered w_i (i.e., the last time a packet was added or was taken from the queue). In the same way, when a datagram is taken out from the queue it is:

$$J_i(t) = J_i(t-\tau) - 1 \tag{8}$$
$$\widetilde{w}_i(t) = \widetilde{w}_i(t-\tau) - c_{i0}(t-\tau) \tag{9}$$
$$c_i^p(t) = c_{i0}(t-\tau) \tag{10}$$
$$c_{ij}(t) = c_{ij+1}(t-\tau), \quad \forall j : 1 \leq j \leq J_i(t) \tag{11}$$

Once the cryptographic algorithm(s) to be applied to a packet is known, the *processing time* for this packet can be computed by using formulas corresponding to the considered cryptographic algorithm, while parameters depend on the characteristics of each processor. These characteristics can be provided by the manufacturer or determined through some simple speed tests. An example of a formula used to compute processing time is provided in Section 4.1.

3.2.1: Algorithm Limitations

As already said, this algorithm has been designed to process IPSec packets only with the assumptions given in Section 3.1. For a heterogeneous protocol environment our algorithm may have to be modified. This would be the case if, for example, the TLS protocol [17] also needed to be supported. In fact different TLS packets may exhibit data interdependency. Since our target system is a secure gateway, the situation in which IPSec and TLS are required to coexist should not be very common. TLS works above TCP and it would not make much sense having it on a gateway. Anyway, should this situation or other similar ones happen, two different solutions can be adopted: the first one consists of scheduling the packets that show interdependencies always on the same accelerator. In this way precedence problems are automatically resolved. The second, simpler, solution is to schedule a packet only when all data that are needed are available. If many packets exhibit data interdependencies, the scheduling algorithm may need to be radically changed.

While the algorithm presented above cannot guarantee the processing order for the packets, not too many of them are processed out of order in normal conditions. This is not a problem for IPSec per se.

4: Simulations

To validate our algorithm and evaluate its performance, we developed a queue-based model. Model simulations and results are here presented. The SystemC language [18] was selected to describe our model since it allows for specifying hardware-software systems. Delays associated with the performed operations can be easily modeled with this language.

4.1: Description of the Model

The SystemC model that can be used to describe our algorithm represents the queues of the cryptographic architecture and the flow of cryptographic requests of the IPSec packets. We are therefore not interested in specific operations performed on data, but only on their delays.

Among all possible conflicts and delays due to communication between the architectural components, we only considered bus contention. RAM contention and other necessary communications between the CPU and the accelerators have been ignored. Providing a highly accurate performance estimation of the considered system is in fact beyond the scope of this work; our main goal is to prove that our algorithm works as desired. Ignoring RAM contention should not introduce a too coarse approximation anyway, since in the considered system the RAM access is much faster than the access to system bus.

A bus contention mechanism, simplified with respect to the PCI standard, has been modeled here. Access to the bus is given to each of the processors which have requested it, one at a time, in the same order as it has been requested. Bus transfer time (that is also the bus lock time) is computed as:

$$t_{bus} = t_{bus\_cycle}(c_{address} + \lceil l_{data}/b \rceil) \tag{12}$$

where: t_{bus_cycle} is the bus cycle period (i.e., $10^{-6}/66$); $c_{address}$ is the time needed to assert the address (1 bus cycle for PCI in DMA mode). l_{data} is the length of the piece of data to be transferred (measured in bytes); b is the number of bytes that are transferred in each cycle (here we consider $b = 4$).

In our model we decided to have AES encryption as the only available operation both in hardware and in software. Time for decryption, depending on the selected algorithm, may differ; for AES encryption and decryption time can be very similar, depending on the implementation. To compute the delays of the AES encryption operation we considered a formula that can be applied to all cipher-block algorithms working on 16-byte blocks (like, for example, AES):

$$t_{enc} = t_{init} + t_{block} \times \lceil b_{data}/16 \rceil \tag{13}$$

where: t_{init} is the algorithm initialization time; t_{block} is the time needed to encrypt a 16-byte data block.

The SystemC model is shown in Figure 2 and is mainly composed of 5 classes:

- the *accelerator* class describes the functionalities of an accelerator,
- the *scheduler* class implements the scheduling algorithm,
- the *fifo* class implements the FIFOs that are needed for interblock communications,
- the *fifo_accel* class implements the FIFOs used for accelerators,

Table 1. β_0 and corresponding number of bytes that can be processed in such time by one of the accelerators.

β_0 [ns]	Number of bytes
0	0
1.76×10^4	640
6.6×10^5	24,000
6.6×10^6	240,000

- the *bus* class models the PCI system bus.

To create the model for simulation, the following objects need to be instantiated:

- one *fifo* object to store the input data,
- one *scheduler* object,
- $N + 1$ *accelerator* objects (one represents the CPU),
- one *bus* object,
- $N + 1$ *fifo_accel* objects, used to connect the *scheduler* object to the $N + 1$ *accelerator* ones through the bus.

Inputs used for the simulations are taken from network trace files, e.g., the ones provided on the Internet Traffic Archive [19] website. These files contain long traces obtained by using the *tcpdump* tool [20] on various networks. We consider a trace taken from a 2Mbit/s gateway and containing about 3 million TCP packets. For our simulations we decided to use only 1 million of these packets to avoid having too long simulations. The only parameter we took from the considered tracefile was packet dimensions, while we ignored the timestamps. Being our target a gateway-like machine, in our simulation we processed the packets as if ESP in tunnel mode was being used. We incremented the packet size read from the tracefile by 40 bytes (i.e the size of a normal TCP/IPv4 packet header) since the sizes contained in ITA files are the ones of data payloads only.

In these simulations we considered the throughput for cryptography on the CPU to be of 110Mbit/s. This value is similar to the one that can be obtained on a Pentium III. Even if a more powerful CPU is used, it also has to execute, among other operations, IPSec header processing and TCP/IP packet processing, and to run the packet scheduler. Thus only a fraction of the overall computational capacity can be dedicated to running cryptographic algorithms. We should also remember that multi-GHz Pentium class CPUs are not able to process normal TCP/IP multi-Gbit/s traffic by themselves [21]. Therefore other form of hardware acceleration may be needed in large bandwidth systems.

When multiple accelerators were used, all of them were considered to have the same performance and to support the same cryptographic algorithm. We chose to have accelerators capable of processing a data traffic of about 230Mbit/s.

Values considered for the α_0 parameter are 1, 2.5, 5, 7.5, 10, 50, 100, and 500. Table 1 gives some values for the parameter β_0 (in nanoseconds) and the corresponding number of bytes that can be processed by one of the accelerators in such a time.

4.2: Simulation Results

Various figures of merit has been analyzed by simulation to evaluate the performance of our algorithm. The CPU usage (in percent) due to cryptography is shown in Figure 3. The

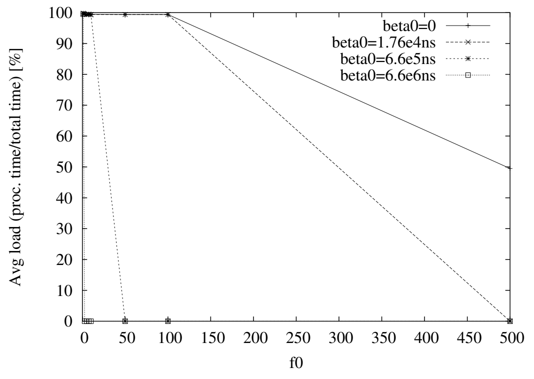

Figure 3. CPU load due to cryptography for a 2-accelerator system.

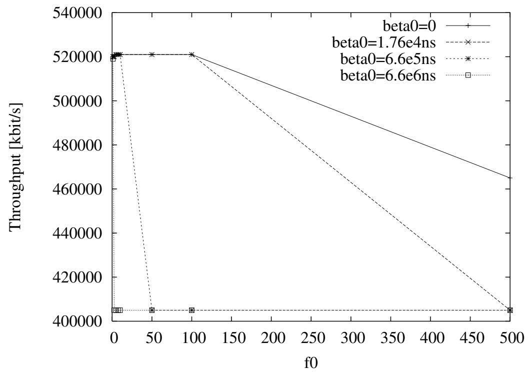

Figure 4. Throughput for a 2-accelerator system.

CPU usage can be controlled through the α_0 and β_0 parameters. The higher β_0, the lower the number of packets processed by the CPU, even for low values of α_0. The higher α_0, the lower the CPU load (and thus the number of packets processed by the CPU itself). These two parameters may even be used to dynamically control the main CPU load: when main CPU load grows, α_0 can be augmented (up to a certain upper limit) to try to avoid having more packets being scheduled on the CPU.

The throughput (in kbit/s) achieved by the whole system in performing cryptographic operations is shown in Figure 4. As can be seen from this figure, throughput changes accordingly with the CPU load. This is due to the fact that an overloaded system is considered (systems maximum throughput is slightly more than 500Mbit/s, while the required bandwidth would be of 1Gbit/s).

The average processing latency for cryptography is shown in Figure 5. This parameter represents the average time between when a packet is scheduled and when its processing is completed. Average processing latency has a minimum corresponding to the best trade off between having long waiting times and having too many or too big packets being processed by the CPU. The CPU is in fact considerably slower (in this case) than accelerators.

The average global latency for cryptography is shown in Figure 6. This parameter represents the time elapsing between the time when a packet is enqueued for being encrypted/decrypted and the time when its processing is completed. Since in our simulation we decided not to drop packets, but to keep them in a very long arrival queue, the global latency in a saturated network increases with the number of considered packets. Results concerning the average global latency are strictly related to the ones concerning the throughput of the system: the higher the throughput, the lower the global latency.

Results here shown were obtained considering a very high maximum length of the queues (1000 elements) on the accelerators. By only allowing shorter queues (50 elements) the results obtained concerning throughput, global latency, and CPU usage are very similar to the ones shown here. The only difference is that the effects obtained by changing the parameters α_0 and β_0 are amplified. Processing latency has a similar behavior both considering long or short queues, but its values are considerably lower in the second case. This is due to the fact that having shorter queues means having less time to wait for a packet to be processed.

By considering a non overloaded system (200Mbit/s of required bandwidth), results are different: the CPU is barely used and both latency and throughput do not change with α_0

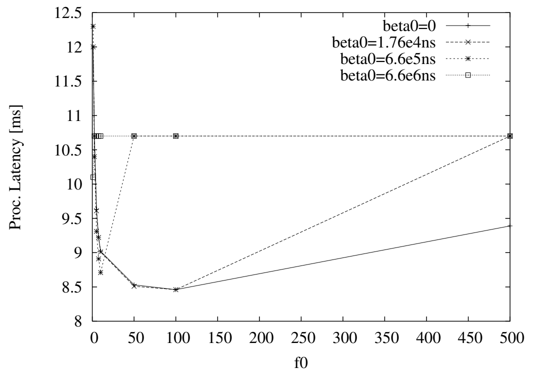

Figure 5. Average processing latency for a 2-accelerator system.

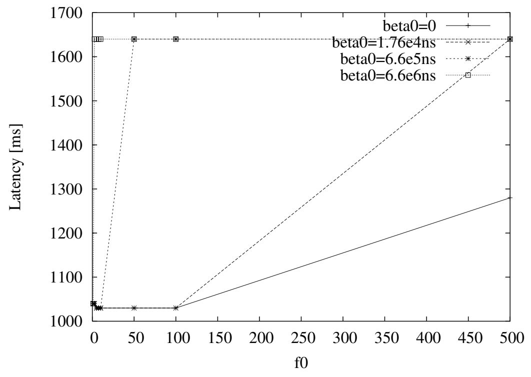

Figure 6. Average global latency for a 2-accelerator system.

and β_0. In fact, in this situation, all data traffic is always processed by the accelerators.

Results for a similar system with 4 accelerators were also obtained. In this case throughput was really limited by bus contention.

The 1-accelerator case is of special interest. Results similar to the ones shown above are obtained for throughput and latency. The only difference is that higher values of α_0 and β_0 are needed to decrease main CPU usage.

4.3: Enhanced System Architecture

The previous results showed that, especially when the throughput approaches 1Gbit/s, access to the system's bus really limits the performance. Coprocessors got idle in many cases waiting for the opportunity to write back the results in memory or to fetch data to be processed. We therefore propose two enhancements for the system architecture in order to avoid these problems and increase the overall system performance. The first one consists of introducing a write buffer in each accelerator. This allows for decoupling data output from packet processing; the accelerator can therefore continue to process incoming data while processed data are written back to the memory. The second enhancement consists of using data prefetch on each accelerator. Since packet processing order is determined by each processor's queue, preloaded data are always used, and used in that order. This makes prefetch very convenient in this case, since its cost is only due to some additional memory and control logic on the accelerators. Data prefetch allows for decoupling the data fetch phase from the data processing, thus allowing the accelerators to perform data processing, while data for the subsequent computation(s) are fetched.

To fully exploit the second architectural enhancement, a modification the the scheduler is also proposed. The scheduler believes that prefetched packets are in processing, and, consequently, computes a finishing time lower than the real one. Actually, prefetched packets are just waiting to be processed in the processors' queues. The scheduler can therefore be adapted for taking into account the processing time of the last k elements in the queue – where k is the prefetch buffer size – when finishing time is computed.

Comparisons of the results achieved with the enhanced architecture in terms of average processing latency and throughput are shown in Figure 7 and 8, respectively. A 1Gbit/s required bandwidth and a 1-accelerator system are considered. The β_0 parameter was set to $1.76 \times 10^4 ns$. Results for a 1-accelerator system in which our algorithm is not used (i.e.,

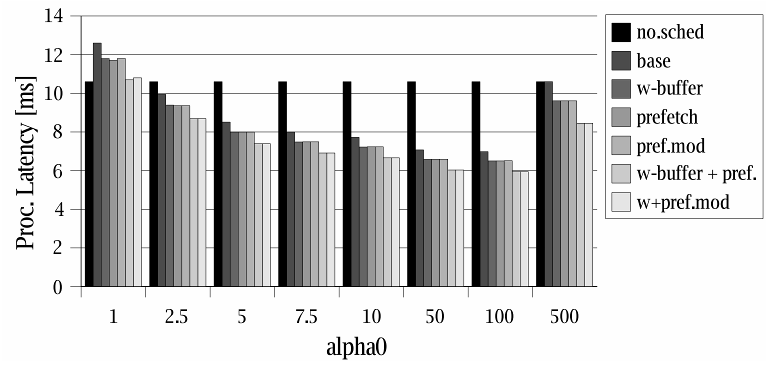

Figure 7. Average processing latency comparison for a 1-accelerator system.

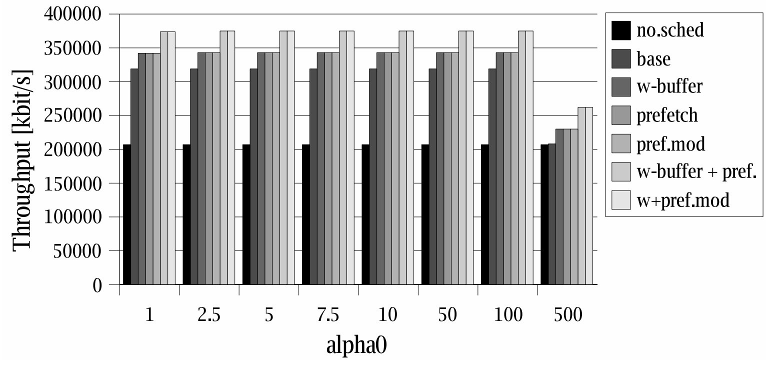

Figure 8. Throughput comparison for a 1-accelerator system.

all the packets are sent to the cryptographic accelerator) are also reported in these figures in the "no-sched" columns. These results do not depend on α_0 and β_0. The scheduling algorithm, especially for low values of the α_0 parameter, improves processing capability in terms of latency and throughput. The higher is the value of α_0, the closer is the performance of our system without the scheduling algorithm to the one with the scheduler. This is due to the fact that by increasing α_0, less packets are processed by the CPU, going closer to the case in which all packets are processed by the cryptographic accelerator.

The same performance improvement can be obtained by adopting either the write buffer or data prefetch on the coprocessors. By adopting both of them we can achieve some further benefits. Modifying the scheduling algorithm to consider data prefetch does not give any valuable benefit in term of throughput. In some cases this also worsens the average processing latency.

There are values of α_0 and β_0 for which packets scheduled on the CPU are the smallest ones. This is a good behavior since it avoids to transfer small packets to the accelerators. In fact for small packets the DMA overhead becomes relevant with respect to their processing time.

5: Conclusions and future work

We developed a scheduling algorithm which allows for distributing IPSec packet processing over the CPU and multiple accelerators. We also provided some high-level simulations to prove that the algorithm works as desired and that it can provide a performance enhancement especially when the overall system is overloaded.

Future research should address the case of packets having some form of correlation. The algorithm presented in this paper should be extended also to support Quality of Service (QoS). A study on the size of the packets that can be more conveniently processed in software is also in progress.

References

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

[2] ——, "IP Authentication Header – RFC2402," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

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

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

[5] A. Shacham, R. Monsour, R. Pereira, and M. Thomas, "IP Payload Compression Protocol (IPComp) – RFC2393," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

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

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

[8] 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.

[9] 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.

[10] F.T. Hady, T. Bock, M. Cabot, J. Chu, J. Meinecke, K. Oliver, and W. Talarek, "Platform Level Support For High Throughput Edge Applications: the Twin Cities Prototype," *IEEE Network*, vol. 17, no. 4, pp. 22–27, July 2003.

[11] John Freeman, "An Industry Analyst's Perspective on Network Processors," in *Network Processor Design*, P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, Eds. Morgan Kaufmann, 2003, vol. 1, ch. 9, pp. 191–218.

[12] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms," in *Hanbook of Scheduling: Algorithms, Models, and Performance Analysis*, Joseph Y. Leung, Ed. CRC Press, 2004, ch. 31.

[13] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, "Scheduling Independent Multiprocessor Tasks," in *European Symposium On Algorithms*, 1997, pp. 1–12.

[14] R. Rajaraman and S. Muthukrishnan, "An Adversarial Model for Distributed Dynamic Load Balancing," in *the 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1998, pp. 47–54.

[15] (2002) PCI comparison, 32 vs. 64-bit and 33MHz vs. 66MHz. [Online]. Available: http://www.buildorbuy.org/pdf/64bitpci.pdf

[16] S. Frankel, R. Glenn, and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec - RFC 3602," IETF RFC, Sept. 2003.

[17] T. Dierks and C. Allen, "The TLS Protocol Version 1.0 – RFC 2246," IETF RFC, Jan. 1999. [Online]. Available: http://www.ietf.org/rfc.html

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

[19] (2000) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[20] TCPDUMP Public Repository. [Online]. Available: http://www.tcpdump.org/

[21] Srihari Makineni and Ravi Iyer, "Architectural Characterization of TCP/IP Packet Processing on the Pentium M Microprocessor," in *Tenth International Symposium on High-Performance Computer Architecture*, Feb. 2004.