Scheduling Small Packets in IPSec-based Systems

Antonio Vincenzo Taddeo*, Alberto Ferrante†, and Vincenzo Piuri†
* ALaRI Institute, University of Lugano
Lugano, Switzerland
Email: taddeo@alari.ch
† Department of Information Technologies,
University of Milan
Milano, Italy
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. IPSec accelerator performance may heavily depend on the dimension of the packets to be processed. When packets are small, the time needed to transfer data and to set up the accelerator may exceed the one to process the packets (e.g. to encrypt) by software. In this paper, we propose a solution for this problem. High-level simulations and the related results are provided to show the properties of the algorithm.**

I. 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 a 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]. 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. 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) [4].

IPSec is gaining importance as it 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) [5], [6].

IPSec has been proven to be very computionally intensive [7], [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 in low-end systems, has become a common practice in the last years [10], [11], since it provides flexibility and performance.

It is also known that the encryption process might be inefficient for small IP packets. Some performance measurements for small packets processing are shown in [7]. This problem cannot be neglected; as matter of fact, according to an analysis of the Cooperatives Association for Internet Data Analysis (CAIDA), the most probable dimension of Internet packets is around 40 bytes [12].

This work is based on [13], [14]. The former presents a scheduling algorithm allowing to distribute packet processing over multiple crypto-accelerators and the CPU (running a software implementation of the same cryptographic algorithms). The latter extends the scheduler to support quality of service (QoS). In these papers, no specific solutions for the small packets problem is presented.

In this work, we investigate the effects of small packets on system performance and we propose a solution to improve overall system efficiency. Our scheduler bundles small packets belonging to the same SAs to form bigger packets. This allow to avoid performance loss due to the transfer of small packets between memory and processors.

Section II describes the architecture of the considered system. Section III describes the scheduling algorithm. Finally, section IV presents the model for the simulations and the obtained results.

II. System Architecture

The system considered here is composed of a host computer and a cryptographic accelerator connected to the normal system bus; a PCI bus (32bit, 66MHz) [15] is adopted here as an example. CPU-memory communication is performed on a faster bus, as in most modern personal computers. The

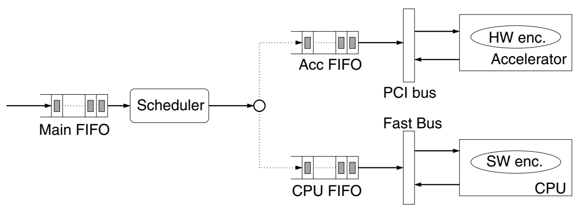

Fig. 1. Model used in simulations.

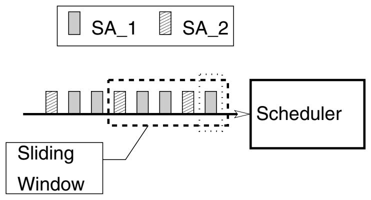

Fig. 2. Sliding Window schema

network card is also connected to the faster CPU bus. Only cryptography-related operations are offloaded to the accelerator. This means that all 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.

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.

III. The Sliding Window Scheduling Algorithm

In the envisioned architecture, each processor (CPU or hardware 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. The scheduler runs on the CPU.

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.

A. 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 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]. How to obtain data independency among packets for AES is shown in [16].

B. Description of the Algorithm

Our algorithm is based on two ideas: to allocate the packets to be processed on the processor (the CPU or the accelerator) which can provide the shortest processing time and to bundle together packets belonging to the same SAs. These bundles are sent to the selected processors; thus avoiding multiple DMA setups and algorithm initialization phases.

One key concept of this algorithm is the use of a *sliding window* over the incoming data. As shown in Figure 2, the sliding window provides a way to observe a specific-size part of the input data stream. This allows to group packets into *multi-packets* thus providing the possibility to send them to the processors in a single transfer. The sliding window allows to analyze the packets by comparing their SA identifiers:

1) the SA identifier of the first packet in the window is taken as a reference;
2) the SA identifiers of the incoming packets are compared with the reference one:
   a) the packets having the same identifier as the reference one are grouped together to form a *multi-packet*;
   b) if no packets with the same identifier as the reference one are found, the multi-packet is composed of the the first packet only;
3) once the window is full or a certain timeout expires, the multi-packet is sent for processing;
4) when multi-packets are formed and sent, the related single packets are removed from the queue; packets remaining in the queue are then rearranged (arrival order is preserved) and re-processed through the mechanism described in the previous points.

From an implementation stand point, the sliding window is just a modification of the scheduler's input queue. A new system parameter needs to be considered: the sliding window size, W. This parameter needs to be tuned depending on the considered system and on network traffic characteristics.

The scheduler processes each multi-packet as follows:

- the *finishing time* for each of these processors is computed; the *finishing time* is 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 accelerator processes the assigned packets in FIFO order.

The equation to compute the *finishing time* for each processor i (where $i = 0$ is the CPU and $i = 1$ is the accelerator) is as follows:

$$f_i = w_i + p_i, \quad i = \{0, 1\} \tag{1}$$

where w_i is the waiting time and the p_i is the processing time of the examined packet. $p_i = p_i(b_{data})$ is a function of the packet dimension. The *waiting time* is defined as the sum of the processing time of all the packets contained in the related queue, the *time in queue*, q_i, and of the *residual processing* *time*, r_i, of the packet being processed at that time by the processor:

$$w_i = w_i(t) = q_i(t) + r_i(t) \quad (2)$$

In detail, we have the *time in queue* expressed by:

$$q_i = q_i(t) = \sum_{n=1}^{N} c_{in} \quad (3)$$

that is the sum of the computation time c_{in}, at time t, for the related queue i of dimension N. The q_i value can be efficiently updated each time a datagram goes in or out of a queue by a simple addition or subtraction, respectively.

The *residual processing time* r_i represents the residual computation time to complete the packet that is currently being processed. r_i can be computed as follows:

$$r_i = r_i^j(t) = c_i^j - \left(t - t_i^j\right) \quad (4)$$

Where c_i^j is the overall computation time required by the packet j, and t_i^j is the time at which the computation has started.

Once the cryptographic algorithms to be applied to a packet are 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 IV-A.

1) Algorithm Limitations: As was stated earlier, this algorithm has been designed to process IPSec packets only with the assumptions given in Section III-A. 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.

The sliding window mechanism proposed is strongly connected with network traffic characteristics and may increase processing latency of the packets. In applications where latency is important (e.g., real time network streams) the W parameter needs to be tuned in a proper way (typically small values need to be selected for these applications). In addition other mechanisms such as a timeout on the sliding window might be applied.

The algorithm presented above cannot guarantee the processing order of the packets. This is not a problem for IPSec per se.

IV. Simulations

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

A. Description of the Model

The SystemC model 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. A representation of the model is shown in Figure 1.

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 accelerator 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, since in the system considered, 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:

$$t_{bus} = t_{bus\_cycle}(c_{address} + \lceil l_{data}/b \rceil) \quad (5)$$

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) and 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:

$$t_{enc} = t_{init} + t_{block} \times \lceil b_{data}/16 \rceil \quad (6)$$

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

In our model, we considered a modified version of Equation 6 allowing for CPU processing rate variability. Here follows the modified version of the equation:

$$t_{enc} = T_{init-ratio} * t_{init} + T_{enc-ratio} * t_{block} \times \lceil b_{data}/16 \rceil \quad (7)$$

where: $T_{init-ratio}$ is the ratio between the encryption/decryption initialization time of the CPU and the one of the accelerator; $T_{enc-ratio}$ is the ratio between the encryption/decryption processing time of the CPU and the one of the accelerator. These two parameters allow to define the CPU processing rate relatively to the accelerator one. Both $T_{init-ratio}$ and $T_{enc-ratio}$ are 1 for the reference accelerator.

Mainly, the sliding window modifies the average packet dimension in the system datapath. The number of times it creates merged packets highly depends on network traffic characteristics. Information about packets inter departure/arrival

TABLE I
PATTERN PARAMETERS IN BYTES.

	Exponential	Uniform	Trace
range:	[20-7,065]	[0-1960]	[0-1,460]
average:	471	976	256
median:	326	–	162
std. dev.:	470	565	259

time was neglected as we considered the worst possible case, i.e., no delay between arrival of the packets. We therefore simulate a nominal network rate of 1 Gbit/s. Moreover, we focused only on the distribution of the dimension of the packets and their SA parameters.

In order to study the effects of the dimension of the packets on our model, we adopted seven different traffic patterns, each one composed of 500,000 samples. Two patterns are based on exponential and uniform statistic distributions; another four patterns are of data payload fixed dimension: 0, 24, 472, and 984 bytes. Inside these patterns random distribution of 100 SAs were considered. Besides these patterns, we also used a real network trace file provided on the Internet Traffic Archive [19] website. Files provided by ITA 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 500,000 of these packets to avoid having overly long simulations. The only parameter we took from the considered tracefile was packet dimension, while we ignored the timestamps. Table I shows the parameters of this trace and of the exponential and uniform patterns. In this case, SAs were associated to the destination IP address. This could be considered a kind of worst case estimation for gateway machines.

As our target is 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 tracefiles by 40 (40 bytes is the size of a normal TCP/IPv4 packet header) since the sizes contained both in ITA and in artificially generated files are the ones of data payloads only. We chose to have an accelerator capable of processing a data traffic of roughly 230Mbit/s. The processing capability of the CPU was varied from 32Mbit/s to 230Mbit/s, being the first value similar to the one that can be obtained on a Pentium III [9]. 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.

B. Simulation Results

In this section, we show the results we have obtained and we compare them with the ones obtained by considering a system in which the basic scheduling algorithm shown in [13] is adopted. Considered sliding window dimensions (W) are of 10 and 30.

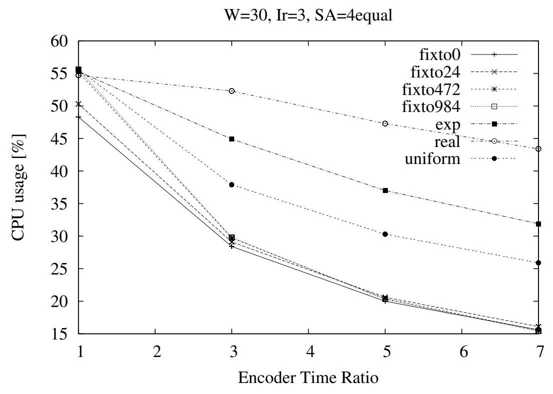

Fig. 3. Percentage of packets processed by the CPU; $W = 30$.

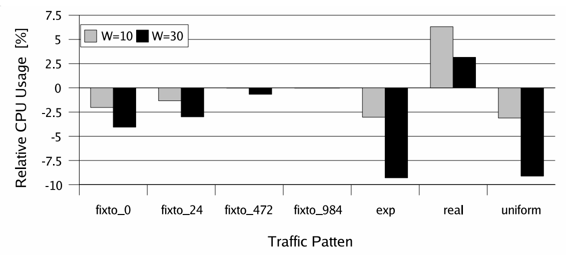

Fig. 4. Relative CPU usage. CPU parameters: $T_{init-ratio} = 3$; $T_{enc-ratio} = 3$.

The average number of multi-packets per window that have been obtained for $W = 30$ is 2.57 for the real trace and 1.33 for all other patterns. The latter has been obtained by considering 100 equiprobable SAs.

The relative CPU load due to cryptography for a system with a sliding window dimension W of 30 is shown in Figure 3. This figure shows the results obtained by considering the $T_{enc-ratio}$ parameter of the CPU to be 1, 3, 5, and 7; $T_{init-ratio}$ was instead fixed to 3. CPU usage decreases for increasing values of $T_{enc-ratio}$ and it does it faster for fix dimension patterns.

Figure 4 shows the relative CPU usage with $W = 10$ and $W = 30$; considered reference case is the one without the sliding window mechanism. By comparing the average size of the packets processed by the CPU with the merged ones, we noticed that the scheduler always tries to assign merged packets to the accelerator. Only single small packets are usually sent to the CPU, especially when it is considerably slower than the accelerator. Our algorithm allows to save CPU usage by dispatching merged packets to the accelerator.

The sliding window system improves the overall system throughput as shown in Figure 5. The results obtained prove the efficiency of our algorithm to reduce the effects of small packet processing. The greater throughput increment is for small packets. We have to consider that with bigger patterns, processing capacity of the processors are saturated. This is due

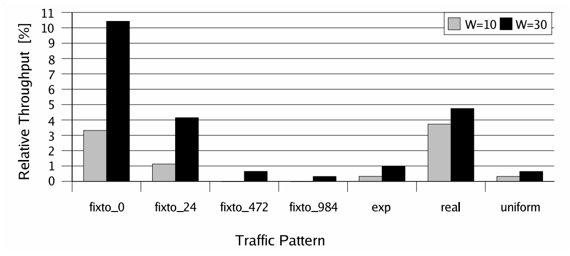

Fig. 5. Relative throughput. CPU parameters: $T_{init-ratio} = 3$; $T_{enc-ratio} = 3$.

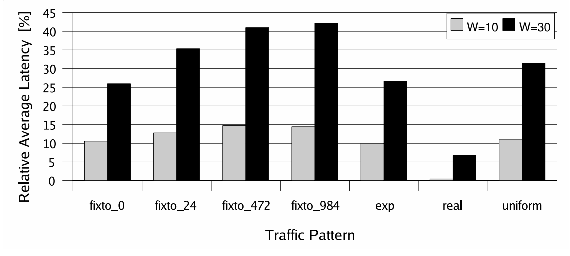

Fig. 6. Relative average latency. CPU parameters: $T_{init-ratio} = 3$; $T_{enc-ratio} = 3$.

to the fact that an overloaded system is considered (network rate is 1Gbit/s, while processing capability is of 500Mbit/s). In any case, throughput decreases for increasing values of the $T_{enc-ratio}$; this is due to the fact that processing capability of the CPU is decreasing accordingly.

Average processing latency represents the average time elapsing between when a packet is scheduled and when its processing is completed. In opposition with throughput, the latency does not grow with increasing values of $T_{enc-ratio}$. This is again because we have considered an overloaded system. Results obtained by considering the sliding window mechanism are shown in Figure 6. Average processing latency increases due to the time that is needed to fill the sliding window. The bad influence of the sliding window on this parameter is higher when traces with big average sizes of the packets are considered. The problem with latency increase can be solved by introducing an upper bound on time or on total number of bytes for the sliding window.

The not very large performance improvement is due to the fact that in average very few packets were identified to be related to the same SAs inside the sliding windows.

V. Conclusions and future work

We have provided a solution to the “small packets” problem. The sliding window mechanism was applied to bundle together packets belonging to the same SAs in order to avoid multiple DMA setups and cryptography algorithm initialization.

We also provided some high-level simulations to prove that our algorithm works as desired and that it can provide an overall performance enhancement, especially when the overall system is overloaded.

A real life implementation and test of our algorithm is ongoing.

Future research should address QoS support. A timeout mechanism and a limit on number of bytes contained in the sliding window will also be studied to improve performance of the algorithm in terms of latency. A dynamic parameter tuning mechanism will also be studied to adapt the algorithm to the incoming fluxes of data.

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

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

[7] 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.

[8] 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.

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

[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] S. McCreary. Packet Length Distributions at NASA Ames Internet Exchange (AIX). Internet Analisys of the Cooperative Association for Internet Data Analysis (CAIDA). [Online]. Available: http://www.caida.org

[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, Sept. 2004, pp. 387–397.

[14] Alberto Ferrante, Vincenzo Piuri, and Fabien Castanier, “A QoS-enabled Packet Scheduling Algorithm for IPSec Multi-accelerator Based Systems.” in *Computing Frontiers*. Ischia, Italy: ACM, May 2005, pp. 221–229.

[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, pp. 152–162.