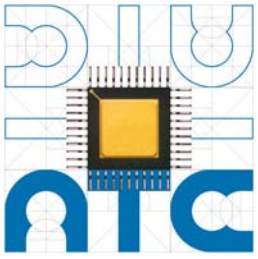
Departamento de Arquitectura y Tecnología de Computadores

E.T.S. Ingenieria Informática y de Telecomunicación

Universidad de Granada



TESIS DOCTORAL

“Alternativas de externalización para la interfaz de red. Análisis y optimización mediante simulación de sistema completo”

Autor

Andrés Ortiz García

Directores

Dr. Julio Ortega Lopera
Dr. Alberto Prieto Espinosa

Editor: Editorial de la Universidad de Granada
Autor: Andrés Ortiz García
D.L.: GR. 2218-2008
ISBN: 978-84-691-6773-1

Agradecimientos

Esta memoria, resultado de varios años de trabajo duro e intensa dedicación, no habría sido posible sin la ayuda y los consejos de diferentes personas a las que me gustaría darles las gracias.

En primer lugar, a mi esposa María José y a mis hijos Andrea y Pablo, por entender desde el principio la dedicación y el tiempo que ha requerido este trabajo y que no he podido dedicar a ellos.

A mis padres y hermano, por su esfuerzo y el ánimo que desde siempre me han dado.

A mis directores de tesis, Julio Ortega y Alberto Prieto, por su inestimable ayuda, colaboración, esfuerzo y constante ánimo durante la realización de esta tesis así como por sus valiosos consejos y por todo lo que he aprendido de ellos, no sólo en lo relacionado con este trabajo.

A mis compañeros de departamento, José María, Jorge, Alberto, Lorenzo, Isabel y Ana María, por hacer del trabajo una tarea agradable.

A mis compañeros de Telefónica, en especial a Ricardo, Jesús, José Ramón y Fran, por haber compartido sus conocimientos conmigo y con los que tantas horas de trabajo he compartido.

A Marcelo Cintra, por darme la oportunidad de trabajar con él y facilitarme los medios necesarios durante mi estancia en la Universidad de Edimburgo.

A mis compañeros de la Universidad de Edimburgo, en especial a Pedro Díaz y Martin Schindewolf, por las interminables conversaciones sobre arquitectura y sus consejos sobre Simics.

A Ignacio Rojas, al Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada y a los proyectos de investigación TIN2004-01419 y TIN2007-60587 por ofrecer los medios necesarios para concluir este trabajo.

A todos, muchas gracias.

A María José, Andrea y Pablo

Indice

Thesis Summary 9

Presentación 37

Capítulo 1: Diseño de la interfaz de red 45

1.1 Redes de comunicación en sistemas de altas prestaciones 46
1.2 El camino de comunicación 48
1.3 La pila de protocolos TCP/IP en Linux 53
1.3.1 Integración en el sistema operativo 58
1.3.2 Colas en TCP/IP 58
1.3.3 Interacción con el sistema operativo 59
1.4 Optimización del subsistema de comunicaciones TCP/IP 61
1.5 Protocolos ligeros 65
1.5.1 El protocolo CLIC (Communication on Linux Clusters) 65
1.5.2 El protocolo GAMMA (Genoa Active Message Machine) 66
1.6 Interfaces de red a nivel de usuario. Estándar VIA 67
1.7 Externalización de protocolos de comunicación 68
1.7.1 Estimación de prestaciones. El modelo LAWS 72
1.7.2 La simulación como técnica de análisis y evaluación de arquitecturas de comunicación 79
1.8 Resumen 80

Capítulo 2: Metodología experimental basada en la simulación de sistema completo 83

2.1 La Simulación de Sistema Completo 85

2.2 El simulador Simics 91
2.3 Simulación de la comunicación con Simics 94
2.4 Modelos funcionales 97
2.5 Modelos de Simulación desarrollados para arquitecturas de comunicación. 101
2.6 Modelado del tiempo en Simics 105
2.6.1 Modelos de temporización 105
2.6 Resumen 109

Capítulo 3: Propuestas para la externalización de la interfaz de red 111

3.1 Introducción 112
3.2 Externalización de protocolos 116
3.2.1 Ubicación de la interfaz de red en el sistema 118
3.2.2 Externalización TCP mediante offloading: hardware para la externalización del protocolo TCP 123
3.2.3 Externalización TCP mediante *onloading* 127
3.2.4 Offloading vs. onloading 129
3.4 Modelo de simulación para el sistema sin externalización 131
3.5 Modelo de simulación para la externalización offloading 133
3.5.1 Modelo Hardware para la simulación de *offloading* 135
3.5.2 Distribución del software en el sistema 139
3.7 Modelo de simulación para la externalización onloading 148
3.7.1 Modelo hardware para la simulación de *onloading* 148
3.7.2 Distribución del software en el sistema 152
3.9 Validación de los modelos de simulación 153
3.10 Resumen 155

Capítulo 4: Estudio experimental de las alternativas de externalización 157

4.1 Introducción 158
4.2 Benchmarks para la medida de prestaciones de red 159
4.2.1 Netperf 161
4.2.2 Netpipe 162
4.2.3 Hpcbench 164

4.2.5 Oprofile 165
4.3 Análisis de las prestaciones de la externalización mediante offloading 165
4.3.1 Carga de la CPU del sistema 167
4.3.2 Latencia de acceso a memoria 169
4.3.3 Efecto de la latencia de memoria en el ancho de banda efectivo de la red. 171
4.3.4 Efecto de la tecnología de la NIC 175
4.3.5 Efecto de la latencia de memoria en la latencia de la red. 176
4.4 Análisis de las prestaciones de la externalización mediante Onloading 181
4.4.1 Carga de la CPU del sistema 181
4.4.2 Ancho de banda y latencia 182
4.5 Comparación de prestaciones offloading/onloading 187
4.5.1 Utilización de CPU 187
4.5.2 Comparación de anchos de banda y latencia 189
4.6 Estudio de la aplicabilidad del modelo de prestaciones LAWS 196
4.6.1 Simulación con Simics 197
4.6.2 Evaluación experimental del modelo LAWS 200
4.6.3 Propuesta de un modelo LAWS modificado 202
4.6.4 Efecto del parámetro σ (*wire ratio*) 208
4.7 Efecto de la memoria cache en la externalización de protocolos 212
4.7.1 Efecto del tamaño de la memoria *cache* 221
4.7.2 Análisis con el modelo LAWS 227
4.9 Comparación de las alternativas de externalización para un servidor web 228
4.10 Tiempos de simulación 231
4.11 Resumen 233

Capítulo 5: Esquema híbrido de externalización 235

5.1 Alternativas para la reducción de la sobrecarga de comunicación 236
5.2 Modelos de externalización híbridos 239
5.2.1 Modelo híbrido de externalización *offloading/onloading* 239
5.3 Evaluación del modelo híbrido de externalización 244
5.3.1 Ancho de banda y latencia de red 244
5.3.2 Uso de la CPU principal 246
5.10 Prestaciones de la interfaz híbrida de externalización en un servidor web 247

5.11 Resumen 249

Capítulo 6: Conslusiones y líneas futuras 251

6.1 Contribuciones y resultados 254
6.2 Líneas futuras 259
6.4 Principales publicaciones 261

Conclusions and summary of contributions 265

Apéndice I: Características y configuración de Simics 271

A.1 Características de Simics 271
A.1.1 Arquitecturas soportadas por Simics 271
A.1.2 Configuración de Simics 274
A.1.3 Objetos de configuración 275
A.1.4 Componentes 275
A.1.5 Conectores 276
A.1.6 Instanciación de componentes 277
A.1.7 Utilidades de Simics 277
A.1.8 Checkpointing 278
A.1.9 HindSight *o simulación hacia atrás* 278
A.1.10 SimicsFS 278
A.1.11 Depuración 279
A.1.12 Seguimiento 279
A.1.13 Interfaz de línea de comandos de Simics (CLI) 279
A.1.14 API de Simics 280
A.1.15 API: funciones del núcleo del simulador 281
A.1.16 API: Funciones PCI 282
A.1.17 Modelos de temporización y ejecución de instrucciones 282
A.1.18 Conceptos de evento, paso y ciclo 283
A.1.19 Modos de Simulación o ejecución del simulador 285
A.1.20 Simulación en modo micro-arquitectura 285
A.1.21 Modo de ejecución stall 288
A.1.22 Step Rate 288

A.1.23 Sistema de memoria de Simics 289
A.1.24 Simulación de redes en Simics 291
A.1.25 Temporización del enlace. Latencia 292

Apéndice II: Modelado del tiempo en Simics 293

A.2 Modelo de tiempo 293
A.2.1 Modulo *cont_staller* 293
A.2.2 Configuración del modelo de tiempo 293
A.2.3 Instalación del modelo de tiempo 294

Apéndice III: Funciones de comunicación TCP/IP en Linux 297

Referencias 301

Thesis Summary

Abstract

In the last years, diverse network interface designs have been proposed to cope with the link bandwidth increase that is shifting the communication bottleneck towards the nodes in the network. The main point behind some of these network interfaces is to reach an efficient distribution of the communication overheads among the different processing units of the node, thus leaving more host CPU cycles for the applications and other operating system tasks. Among these proposals, protocol offloading searches for an efficient use of the processing elements in the network interface card (NIC) to free the host CPU from network processing. The lack of both, conclusive experimental results about the possible benefits, and a deep understanding of the behavior of these alternatives in their different parameter spaces, has caused some controversy about the usefulness of this technique.

On the other hand, the availability of multicore processors and programmable NICs, such as TOEs (TCP/IP Offloading Engines), provides new opportunities for designing efficient network interfaces to cope with the gap between the improvement rates of link bandwidths and microprocessor performance. This gap poses important challenges related to the high computational requirements associated to the traffic volumes and the wider functionality to be supported by the network interface has to support. This way, taking into account the rate of link bandwidth improvement and the ever changing and increasing application demands, efficient network interface architectures require scalability and flexibility. An opportunity to reach these goals comes from the exploitation of the parallelism in the communication path by distributing the protocol processing work across processors which are available in the computer, i.e. multicore microprocessors and programmable NICs.

Thus, after a brief review of the different solutions that have been previously proposed for speeding up network interfaces, this thesis analyzes the onloading and offloading alternatives. Both strategies try to release host CPU cycles by taking advantage of the communication workload execution in other processors present in the node. Nevertheless, whereas onloading uses another general-purpose processor, either included in a chip multiprocessor (CMP) or in a symmetric multiprocessor (SMP), offloading takes advantage of processors in programmable network interface cards (NICs). From our experiments, implemented by using a full-system simulator, we provide a fair and more complete comparison between onloading and offloading. Thus, it is shown that the relative improvement on peak throughput offered by offloading and onloading depends on the rate of application workload to communication overhead, the message sizes, and on the characteristics of the system architecture, more specifically the bandwidth of the buses and the way the NIC is connected to the system processor and memory. In our implementations, offloading provides lower latencies than onloading, although the CPU utilization and interrupts are lower for onloading.

With the background provided by the results which were obtained by using our offloading approaches, we propose a hybrid network interface that can take advantage both, of the offloading and onloading approaches.

We also explain the results obtained from the perspective of the previously described LAWS model and propose some changes in this model to get a more accurate approach to the experimental results. From these results, it is possible to conclude that offloading allows a relevant throughput and latency improvement in some circumstances that can be qualitatively predicted by the LAWS model. Thus, we have modified the original LAWS model, including three new parameters that enable the fitting of the experimental results more in a accurately way.

Finally, we use a real web server application for loading the server and making several experiments in order to get a general view of the behavior of our offloading approaches under a real and typical application.

Introduction

The availability of high bandwidth links (Gigabit Ethernet, Myrinet, QsNet, etc.) [51] and the scale up of network I/O bandwidths to multiple gigabits per second have shifted the communication bottleneck towards the network nodes. Therefore, an optimized design of the network interface (NI) is becoming decisive in the overall communication path performance.

The gap between the user communication requirements of in-order and reliable message delivery and deadlock safety and network features such as arbitrary delivery order, limited fault-handling, and finite buffering capabilities makes it necessary to have (layers of) protocols that provide the communication services which are not supported by the network hardware, but which are required by the applications [35]. Besides processing these protocols, the data to be sent or received needs to be transferred between the main memory and the buffers of the NIC (Network Interface Circuit) that access the network links through the media access control (MAC). Moreover, this data transference requires the cooperation between the device driver in the OS and the NIC, that have to keep coordinated by using information about the state of the frames that are being transmitted. In this way, the driver has to indicate to the NIC that there is data to be sent or buffer space for data receiving, and the NIC has to notify to the OS that data has been sent to or received from the network.

Thus, the sources of the network processing overhead are *protocol processing*; *operating system activities* such as kernel processes, device driver overheads, context switching, interrupt handling and buffer management; and *memory access* overheads due to packet and data copies and processor stalls. As higher link bandwidths are available, memory access and some operating system overheads are even more important due to the poor cache locality of the kernel network processing tasks [43]. Thus, the most part of the memory accesses are DRAM accesses and both the lower rate of improvement for DRAM access latency and the memory bus bandwidth shift the bottleneck to this overhead component. Moreover, as the OS and the NIC have to exchange a large volume of data and control information through the I/O buses, they are other important components in the network overhead [34].

Much research work has been carried out in order to improve the communication performance. This research can be classified into two complementary researching lines. One of these lines searches for decreasing the software overhead in the communication protocols either by optimizing the TCP/IP layers, or by using new and lighter protocols. Moreover, these new protocols usually fall into one of two types: the protocols that optimize the operating system communication support (GAMMA [27], CLIC [28], etc.), and the user level network interfaces [29], such as the VIA (*Virtual Interface Architecture*) standard [30]. These proposals include features that have been specifically

proposed to accelerate network processing in this context along with some improvements in the NICs. Some of these features are the following ones (Table 1 shows their relation with the above mentioned overhead sources):

- *Zero-copy* [36] tries to eliminate the copies between the user and kernel buffers by solving some difficulties related with the interaction of OS buffering schemes, virtual memory, and the API [38].
- *Interrupt optimization techniques* [29] either reduce the interrupt frequency by interrupting the CPU once multiple packets have arrived instead of issuing one interrupt per packet (*interrupt coalescing*), or use polling instead of interrupts (interrupts are only generated whenever the NIC has not been polled after a given amount of time) [37-39].
- *Jumbo frames*, proposed by Alteon [52], allow the use of frames of up to 9000 bytes, which are larger than the Ethernet maximum frame size of 1500 bytes, in order to reduce the per-frame processing overhead.
- *Checksum offloading* [40] allows the NIC to determine and insert the checksums into the packets to be sent, and to check the received packets in order to avoid the corresponding CPU overhead.
- *Header splitting* [7, 34] separates protocol headers and payload in different buffers to decrease the CPU cache pollution when the headers are processed and to aid in the zero-copy of the received payloads.
- *Large send offload (LSO)* [7, 34] builds in the NIC the (large) TCP segments to be sent.

Table 1. Relation among NI optimizing features and the main sources of communication overhead

	Protocol processing	Memory accesses	Operating System overhead
Zero-Copy		√	√
Interrupt coalescing			√
Jumbo frames	√		√
Cheksum offloading	√		
Header Splitting	√	√	√
LSO	√		

Another possibility to reduce communication overhead is the distribution of the network workload among other existing processors in the node. This way, the software is partitioned between the host CPU and another processor that executes the communication tasks. Thus, the host CPU does not have to process the network protocols and can devote more cycles to the user applications and other operating system tasks. Two main alternatives have been distinguished depending on the location of the processor where the communication tasks are executed.

One of these alternatives proposes the use of processors included in the network interface cards (NIC) for protocol processing. In this case, the NIC can directly interact with the network without the host CPU participation, thus allowing not only a decrease in the CPU overhead for interrupt processing, but also a protocol latency reduction for short control messages (such as ACKs) that, this way, do not have to access to the main memory through the I/O bus. There are many commercial designs that offload different parts of the TCP/IP protocol stack onto a NIC attached to the I/O bus [16-18]. These devices are called TCP/IP Offload Engines (TOE), and thus, this alternative is usually

called *protocol offloading*. Other techniques related with *protocol offloading* are *connection handoff* [15] and *network interface data caching* [41]. The *connection handoff* technique allows the operating system to control the number of TCP connections that are offloaded to the NIC to take advantage of its features without overloading it. *Network interface data caching* reduces traffic along the local buses by caching the frequently-requested data using on-board DRAM included in the programmable NIC.

Nevertheless, besides the works showing the advantages of protocol offloading, some papers have presented results arguing that this technique does not benefit the user applications. Particularly, TCP/IP offloading has been highly controversial because, as some studies have demonstrated, TCP/IP processing costs are small (particularly after that task were optimized in the late 1980s [7, 8]) compared to data transference overheads and the costs of interfacing the protocol stack to the NIC and the operating system. As the experimental results provided by the papers advocating each alternative are not conclusive because they depend on the specific technologies, systems and applications used in the experiments, interest in offloading performance analysis still remains.

Another alternative to release host CPU cycles is the so called *protocol onloading* or, more specifically, *TCP onloading* [7]. This technique proposes the use of a general-purpose processor in a CMP or in an SMP for protocol processing. Although it has been proposed as opposed to NIC offloading, despite its name, it can also be considered as a full offload to another processor in the node, rather than to the NIC [5,15]. Nevertheless, in what follows, we will maintain both terms to distinguish between the use of a processor in the NIC for protocol processing (*offloading*) or other general-purpose processor (*onloading*).

There are some papers that compare both techniques [5,20]. Nevertheless, it is difficult to make experiments by using systems with similar characteristics and to explore the whole parameter space. Thus, simulation is necessary to obtain right conclusions about the performances of both techniques along a representative set of parameter values. In this thesis, we provide an approach to this by using a full-system simulator. We first provide a brief introduction to *protocol offloading* and *onloading* that outlines their characteristics, differences, and relative advantages and drawbacks (Section 2). Then, the previously proposed LAWS model [10] is described. It allows a first analysis of the possible benefits of moving the communication workload to other processors in the node in terms of the relative computation/communication workloads and the different technological capabilities of the processors (Section 3). In Section 4, we describe our offloading and onloading implementations and the experimental setup based on the full-system simulator SIMICS. Finally, Section 5 provides the experimental results and the conclusions are given in Section 6.

The LAWS model.

Some authors have proposed performance models ([10,11]) to understand the offloading fundamental principles under the experimental results and to drive the discussions over offloading technologies, allowing us the exploration of the corresponding design space. The paper [10] introduces the LAWS model to characterize the protocol offloading benefits in Internet services and streaming data applications. In [11], the EMO (Extensible Message-oriented Offload) model is proposed to analyze the performance of various offload strategies for message oriented protocols. Here, we will use the LAWS model to understand the behavior of the simulated systems because, although it was

proposed for offloading, it can in fact be also applied whenever the communication overhead is distributed between the host CPU and other processors in the node (either in the NIC, or in another CPU in a multiprocessor node). In the following summary of the LAWS model, we use the generic term *communication processor* (CP) to refer to the processor (in the NIC or in the multiprocessor node) that executes the networking tasks in case of offloading or onloading.

The LAWS model gives an estimation of the *peak throughput* of the pipelined communication path according to the throughput provided by the corresponding bottleneck in the system: the link, the CP, or the host CPU. The model only includes applications that are throughput limited (such as Internet servers), and thus fully pipelined, when the parameters used by the model (CPU occupancy for communication overhead and for application processing, occupancy scale factors for host and NIC processing, etc.) can be accurately known. The analyses provided in [10] consider that the performance is host CPU limited before applying the protocol offloading (this technique never yields any improvement otherwise).

Figure 1 illustrates the way the LAWS model views the system before and after offloading. The notation which is used is similar to that of [10]. Before offloading (Figure 1.a), the system is considered as a pipeline with two stages, the host and the network. In the host, to transfer m bits, the application processing causes a host CPU work equal to aXm and the communication processing produces a CPU work oXm. In these processing delays, a and o are the amount of CPU work per data unit, and X is a scaling parameter used to take into account variations in processing power with respect to a reference host. Moreover, the latency to provide these m bits by a network link with a bandwidth equal to B, is m/B.

Thus, as the peak throughput provided before offloading is determined by the bottleneck stage, we have $B_{before}=min(B,1/(aX+oX))$. After offloading, we have a pipeline with three stages (Figure 1.b), and a portion p of the communication overhead has been transferred to the CP. In this way, the latencies in the stages for transferring m bits are m/B for the network link, $aXm+(1-p)oXm$ for the CPU stage, and $poY\beta m$ for the CP. In the expression for the NIC latency, Y is a scaling parameter to take into account the difference in processing power with respect to a reference and β is a parameter that quantifies the improvement in the communication overhead that could be reached with offloading, i.e. βo is the normalized overhead that remains in the system after offloading, when $p=1$ (*full offloading*). A similar reasoning for onloading is also possible.

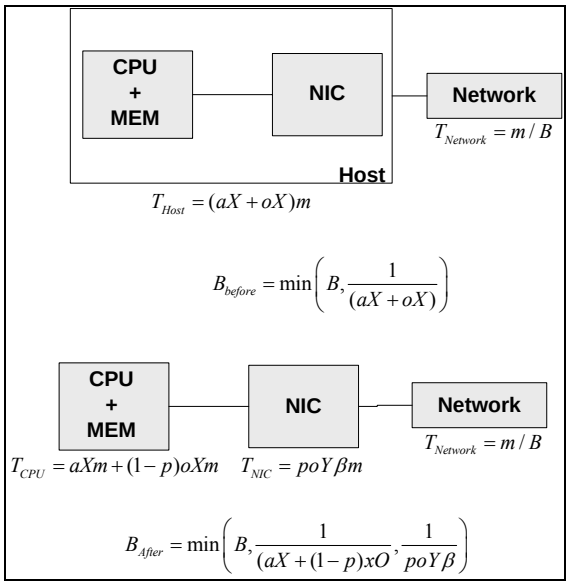


Figure 1. A view of the LAWS model before offloading (a) and after offloading (b)

In this way, after *offloading/onloading* the peak throughput is $B_{after}=min(B,1/(aX+(1-p)oX),1/ poY\beta)$ and the relative improvement in peak throughput is defined as $\delta b=B_{after}-B_{before})/B_{before}$. The LAWS acronym comes from the parameters used to characterize the offloading benefits. Besides the parameter β (**S**tructural ratio), we have the parameters $\alpha=Y/X$ (**L**ag ratio), that considers the ratio between the CPU speed to NIC computing speed; $\gamma=a/o$ (**A**plication ratio), that measures the compute/communication ratio of an application; and $\sigma=1/oXB$ (**W**ire ratio), that corresponds to the portion of the network bandwidth that the host can provide before offloading. In terms of the parameters α, β, γ, and σ, the relative peak throughput improvement can be expressed as:

$$\delta b = \frac{\min\left(\frac{1}{\sigma}, \frac{1}{\gamma + (1-p)}, \frac{1}{p\alpha\beta}\right) - \min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)}{\min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)}$$

From LAWS, some conclusions can be derived in terms of simple relationships among the four LAWS ratios [10]:

- Protocol offloading/onloading provides an improvement that grows linearly in applications with low computation/communication rate (low γ). This profile corresponds to streaming data processing applications, network storage servers with large number of disks, etc. In case of CPU intensive application, the throughput improvement reached by offloading is bounded by $1/\gamma$ and goes to zero as the computation cost increases (i.e. γ grows). The best improvement is obtained for $\gamma=\max(\alpha\beta,\sigma)$. Moreover, as the slope of the improvement function $(\gamma+1)/c-1$ is $1/c$, where $c=\max(\alpha\beta,\sigma)$, the throughput improvement grows faster as $\alpha\beta$ or σ decrease.
- Protocol offloading may reduce the communication throughput (negative improvement) if the function $(\gamma+1)/c-1$ is able to take negative values. This means that $\gamma<(c-1)$ and as $\gamma>0$ and $\sigma<1$, it should verify that $c=\alpha\beta$ and $\alpha\beta>1$. Thus, if the NIC speed is lower than the host CPU speed ($\alpha>1$), offloading may reduce

performance if the NIC get saturated before the network link, i.e. $\alpha\beta>\sigma$, as the improvement is bounded by $1/\alpha$ (whenever β=1). Nevertheless, if an efficient offload implementation (for example, by using direct data placement techniques) allows structural improvements (thus, a reduction in β), it is possible to maintain the offloading usefulness for values of $\alpha>1$.

- There is no improvement in slow networks ($\sigma>>1$) where the host is able to assume the communication overhead without aid. The offload usefulness can be high whenever the host is not able to communicate at link speed ($\sigma<<1$), but in these circumstances, γ has to be low, as it has been previously said. As there is a trend to faster networks (σ decreases), *offloading/onloading* can be seen as very useful techniques. When σ is near to one, the best improvement corresponds to cases with a balance between computation and communication before offloading/offloading ($\gamma=\sigma=1$).

The communication path is not fully pipelined as it is supposed by LAWS. Thus, this performance model can only provide an upper bound for performance. Nevertheless, LAWS can be useful, as it gives an idea about the trends in the improvement that could be achieved according to the changes implemented in some relevant parameters of real communication systems. The simulation results provided in this thesis also give certain experimental validation of the conclusions provided by the LAWS model.
However, in order to fit the experimental results more accurately that original LAWS model does, we proposed a modified LAWS model that includes three new parameters. With this new parameters we can model deviations in the application or communication load as well as in the residual workload in CPU0 after offloading. In this way the new LAWS model is as:

$$\delta b = \frac{\min\left(B, \frac{1}{(a(1+\delta_a)X + (1-p)o(1+\delta_o)X)(1+\tau)}, \frac{1}{\beta o(1+\delta_o)poY}\right)}{\min\left(B, \frac{1}{aX+oX}\right)} - 1$$

Where δ_a represent a deviation in the application workload, δ_o is a deviation in the communication workload and τ is a deviation in the residual workload on CPU0 after offloading.

Offloading and onloading approaches to improve communication

As it has been said, many works have proposed improvements in the communication architecture of high performance platforms that use commodity networks and generic protocols such as TCP/IP. The use of other processors in the computer to reduce the communication overhead in the host CPUs that run the applications has been proposed in many works since years. Moreover, the parallelization of network protocols and the exploitation of parallelism in programmable network interfaces have been also proposed and analyzed [44,45,46].

Thus, an alternative, usually called protocol offloading, proposes the use of processors included in the NIC, whereas the onloading [7] strategy tries to take advantage of the existence of multiple cores in a CMP or processors in an SMP. These two strategies have been commercially released through implementations such as TOEs,

TCP Offloading Engines [16-18]; and the Intel I/OAT, *I/O Acceleration Technology* [22,23,42], that includes an optimized *onloaded* protocol stack as one of its features along with header splitting, interrupt coalescing, and enhanced DMA transfers through *asynchronous I/O copy* [34].

Onloading and offloading have common advantages and drawbacks and also have differences that determine their specific performance. Among their common advantages, we have the following ones:

- The availability of CPU cycles for the applications increases as the host CPU does not have to process communication protocols. The overlap between communication and computation also increases.
- The host CPU receives less interrupts to attend the received messages.
- The use of specific programmable processors with resources that exploit different levels of parallelism could improve the efficiency in the processing of the communication protocols and enable a dynamic protocol management in order to use the more adequate protocol (according to the data to communicate and the destination) to build the message.

In addition to these advantages, offloading offers some others:

- As the NIC implements the communication protocols, it can directly interact with the network without the CPU involvement. Thus, the protocol latency can be reduced, as short messages, such as the ACKs, do not need to travel through the E/S bus; and the CPU does not have to process the corresponding interrupts for context changing to attend these messages.
- As protocol offloading can contribute to avoid traffic on the I/O bus (commands between the CPU and the NIC, and some DMA transferences between the main memory and the NIC), the bus contention could be reduced. It is also possible to improve the efficiency of the DMA transferences from the NIC if the short messages are assembled to generate less DMA transferences.

With respect to the drawbacks of offloading, some works [6-9] provide experimental results to argue that protocol offloading, in particular TCP offloading, does not clearly benefit the communication performance of the applications. Among the reasons to support these conclusions, we have the following ones:

- The host CPU speed is usually higher than the speed of the processors in the NIC and, moreover, the increment in the CPU speeds according to Moore's law tends to maintain this ratio. Thus, the part of the protocol that is offloaded would require more execution time in the NIC than in the CPU, and the NIC could become the communication bottleneck. Moreover, the limitations in the resources (particularly memory) available in the NIC could imply restrictions in the system scalability (for example, limitations in the size of the IP routing table).
- The communication between the NIC (executing the offloaded protocol), and the CPU (executing the API) could be as complex as the protocol to be offloaded [6,9]. Protocol offloading requires the coordination between the NIC and the OS for a correct management of resources such as the buffers, the port numbers, etc. In case of protocols such as TCP, the control of the buffers is complicated and could hamper the offloading benefits (for example, the TCP buffers must be held until acknowledged or pending reassembly) [6].

The main specific advantage of onloading is precisely related with the first drawback of offloading: as in onloading, the processor that executes the communication software has the same speed and characteristics as the host CPU. In particular, it can access the main memory with the same rights as the host CPU. This alternative exploits the current trend towards multi-core architectures or SMPs. However, in relation with this point, some objection could be raised, as the use of simpler and more power and area efficient cores added to the general purpose host CPU (a microarchitecture usually found in network processors) could reach similar network processing acceleration with lower cost in area, power, and complexity [21].

In some recent proposals, onloading is applied along with other strategies to define technologies for accelerating network processing [22]. They are sometimes considered as alternatives opposed to the use of TOEs [20]. Nevertheless, some of the strategies comprised in these technologies can also be implemented although protocol processing is carried out in the NIC. For example, the improvement in the number of interrupts, the DMA transference optimizations, and the use of mechanisms to avoid bottlenecks in the receiver-side such as split headers, asynchronous copy by using DMA, and multiple receive queues [23, 25] could be also implemented in TOE-based approaches.

In any case, it is clear that network interface optimization requires a system approach that takes into account not only the processors present in the computer, but also the chipset, the buses, the memory accesses, the operating system, the computation/communication profile of the applications, and the corresponding interactions among these elements. Thus, it is not easy to predict when offloading is better than onloading or vice versa.

In this thesis, we also explore the possibilities that a full-system simulator provides for that purpose. Thus, we have used the SIMICS full-system simulator with models developed by us to evaluate the onloading and offloading performance. These models allow to overcome some limitations of SIMICS, which does not provide either accurate timing models or TOE models by itself.

The simulation environment and our proposed network interfaces

Simulation can be considered the most common technique to evaluate computer architecture proposals. It allows to explore the design space of the different architectural alternatives independently of specific (commercial) implementations available at a given moment. Nevertheless, the research in computer system design issues dealing with (high-bandwidth) networking requires an adequate simulation tool that allows running commercial OS kernels (as the most part of the network code runs at the system level), and other features for network-oriented simulation, such as a timing model of the network DMA activity and a coherent and accurate model of the system memory [2]. Some examples of simulators with these characteristics are M5 [3], SimOS [14], and some other simulators based on SIMICS [1,4], such as GEMS [12] and TFsim [13].

SIMICS [1,4] is a commercial full-system simulator that allows the simulation of application code, operating system, device drivers and protocol stacks running on the modeled hardware. Although SIMICS presents some limitations for I/O simulation, in [19], we propose a way to overcome them in order to get accurate experimental evaluation of protocol offloading. The proposed simulation models, shown in Figure 3, include customized machines and a standard Ethernet network connecting them in the same way as we could have in the real world. The timing behavior is provided by developing the timing models shown in Figure 3. Nevertheless, in the simulation

models of [19], the cache behavior is simulated just taking into account the mean access time to memory, and not by simulating a detailed memory hierarchy. In this thesis, we use SIMICS to simulate not only offloading but also onloading implementations, and we have also developed more complete SIMICS models that include a detailed cache model in the nodes.

We have also developed three different system implementations for carrying out our simulations: a base system and offloaded and onloaded implementations. Once the machines are defined, SIMICS allows an operating system to be installed on them (Debian Linux with a 2.6 kernel in our case). To enable us to run Linux in this simulation model (neither requiring any kernel change nor the design of a new driver), we have taken advantage of some SIMICS features. By default, all the buses in SIMICS are simply considered as *connectors*. Nevertheless, although there is not any functional difference between a PCI bus and the system memory bus, it is possible to define different access latencies to memory and to the PCI devices. Moreover, it is also possible to define different delays for interrupts coming from the PCI device, other processor, etc, through the timing model that we have developed. This timing model is shown in Figure 3.

The first simulated platform corresponds to a base system, in which we have used a superscalar processor and NIC models provided by SIMICS for PCI based gigabit Ethernet cards. With this model (Figure 2), we have determined the maximum performance which we can achieve using a machine with one processor and no offloading/onloading effects. Thus, the host CPU of the system (CPU0) executes the application and processes the communication protocols.

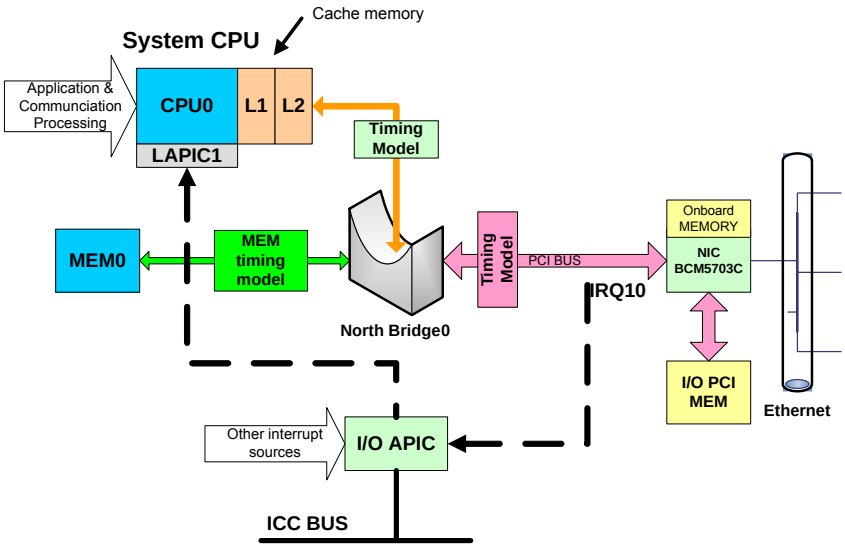


Figure 2. Simulation model for the base system

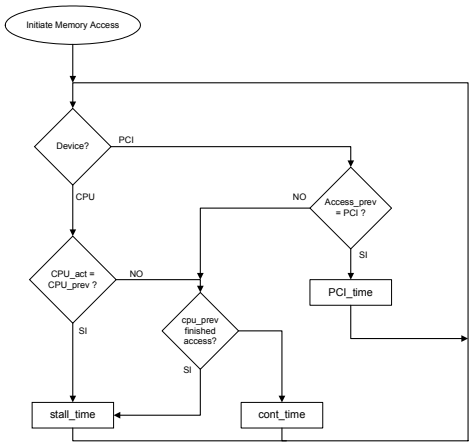
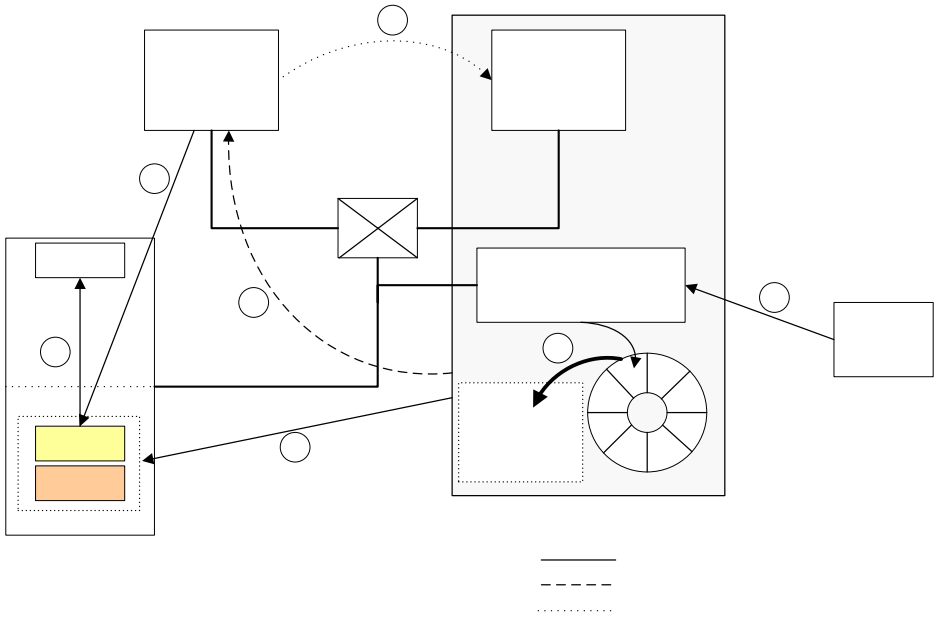


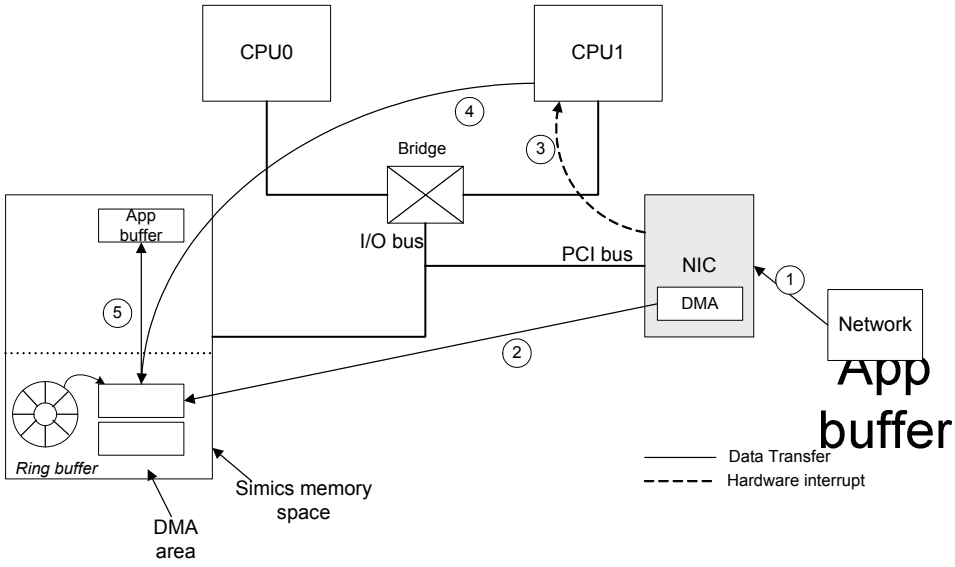
Figure 3. Developed timing model

The operation of the network interface with offloading is summarized in Figure 4.a. After a network transfer (1), the network interface card at the receiver starts to fill the *ring buffer* with the received packets. When this buffer is full, data is transferred to the NIC memory (2). When the memory transfer is completed, an interrupt is sent to the host CPU (CPU0) (3), to start the NIC's driver and a forced *softirq* [50] on the CPU at the programmable NIC (CPU1) (4). This *softirq* will process the TCP/IP stack and then will copy the data to the TCP socket buffer (5). Finally, the CPU0 copies (6) the data from the TCP socket buffer to the application buffer (7) by using socket-related receive system calls [50]. So, since an interrupt is only generated every time the receive ring is full, we are not using the approach "one interrupt per received packet" but the approach "one interrupt per event", whenever the received packet fits into the NIC's receive ring in the same way a TOE does.

Figure 4.b provides the network interface operation with onloading. In this case, the operating system has been configured to execute the NIC driver in the CPU1 instead of CPU0. Thus, once a packet is received (1) and stored in the receive ring (2), the NIC generates an interrupt that arrives at the I/O APIC (3), and the operating system launches the NIC driver execution in the CPU1. Thus, the corresponding TCP/IP thread is executed in the same CPU1. Finally, after the software interruption, the operating system copies the data into the user memory, (4) and (5).



(a)



(b)

Figure 4. (a) Offloading model operation. (b) Onloading model operation

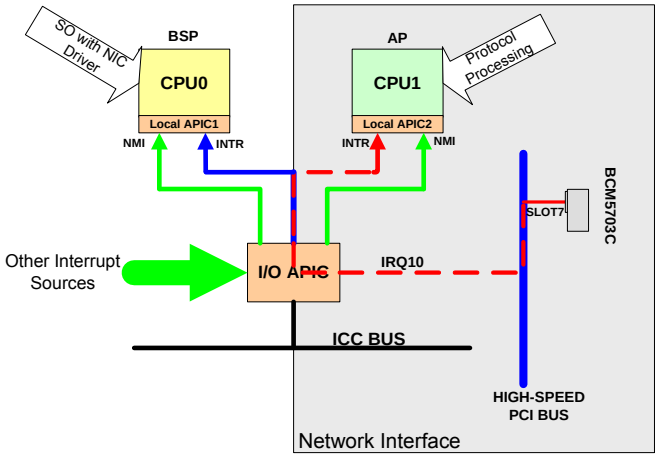
Figure 4 shows the functional models used to simulate offloading and onloading in SIMICS. Offloading has been implemented by using a PCI bus directly connected to the north bridge along with two processors (CPU0 and CPU1) also connected to the north bridge. Nevertheless, the connection between the PCI bus and CPU1 is no more than a *connector*. In the same way, CPU1, which executes the protocols, performs a fast access to the onboard memory on the NIC. Thus, from a functional point of view, this is equivalent to having together these two devices. As it has been said, the connectors do not model any contention at simulation time. The way to simulate the contention and

timing effects is by connecting a *timing model interface* [4] to each input of the bridge where access contention is possible. Thus, an accurate simulation of the contention behavior is provided, as it can be seen from the experimental results (Section 5).

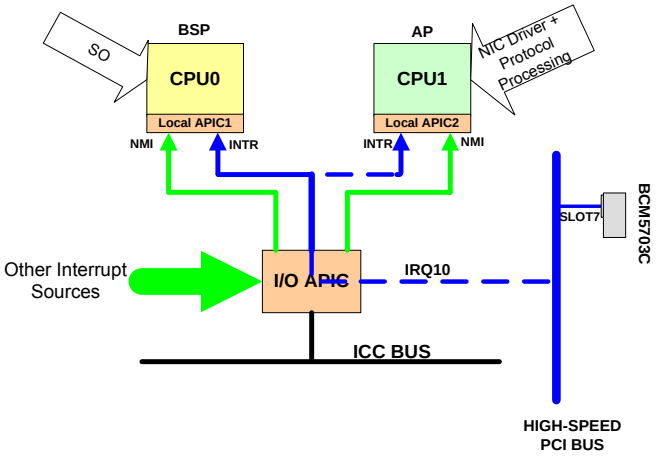
Our offloading model uses DMA for transferring data from the NIC to the main memory, and the CPU1 can handle the data on the DMA area to process the protocol stack with a fast access, as mentioned above, in order to simulate a fast onboard memory on the NIC.

In the offloading simulations, the interrupts generated by the NIC directly arrive at the CPU1 without any significant delay as, in real systems, CPU1 is the processor included in the NIC that executes the network software. This ensures that no interrupt is arriving CPU0 under any condition (i.e. whenever DMA is not used). Although SIMICS does not support the simulation of nodes with more than one I/O APIC, it is possible to configure them in such a way that it would be possible to redirect the interrupts coming from the PCI bus towards CPU1.

For onloading (Figure 4.b), we have used two CPUs connected to the north bridge. The interrupts have to go through an APIC bus and through the I/O APIC to reach these CPUs. This produces a delay in the interrupt propagation due to the simulated interrupt controller. Moreover, this controller has to decide about the interrupts that finally reach the CPU.



(a)



(b)

Figure 5. Functional models for offloading (a) and onloading (b)

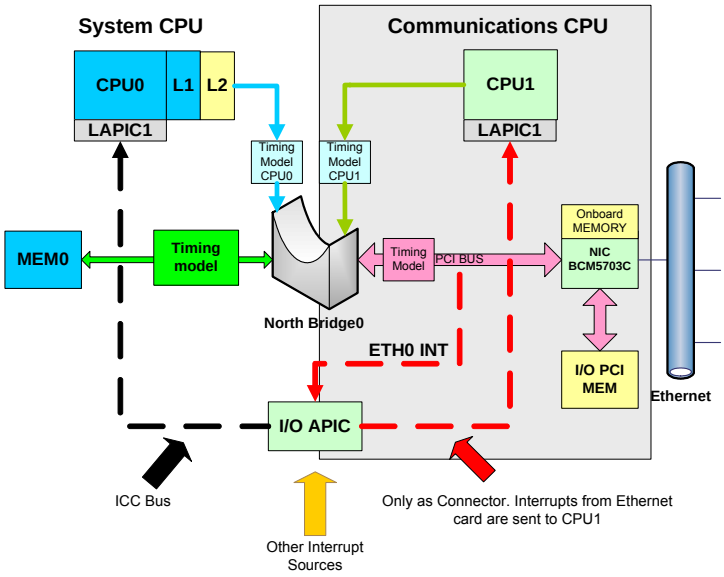
In SIMICS, the PCI I/O and memory spaces are mapped in the main memory (MEM0 in Figure 4). So, at hardware level, transferences between these memory spaces would not necessarily require a bridge, because SIMICS allows us the definition of hardware for full-custom architectures. We add a north bridge in our architecture in order to simulate a machine where can install a standard operating system (i.e.: Linux). This way, the main differences between the models for onloading and offloading are the following ones:

a) Whenever a packet is received, the interrupt directly arrives at the CPU1 without having to go through any other element. In the onloading model, the interrupts have to go through the interrupt controller, the APIC bus, etc. These elements have been simulated as in an SMP.

b) In the offloading model, the NIC driver is executed in the CPU0 that also runs the operating system.

c) In both cases, offloading and onloading, the TCP/IP threads are executed in the corresponding CPU1.

Figure 6 shows all the elements that have been included in the SIMICS simulation model for offloading (6.a) and onloading (6.b). The computers of Figure 5 include two CPUs, DRAM, instructions and data L1 cache, a unified L2 cache, an APIC bus, a PCI bus with an attached PCI-based gigabit Ethernet card, and a text serial console. As our simulation models include a memory hierarchy with two cache levels, it will be also possible to get accurate conclusions about the real behavior of the network interface and the influence of the different memory levels in the communication overheads. It is important to notice that the gap between processor and memory performance (the *memory wall* problem) is even more important in packet processing than in other usual applications [48]. Nevertheless, there are not many studies about the cache behavior of network interfaces.



(a)

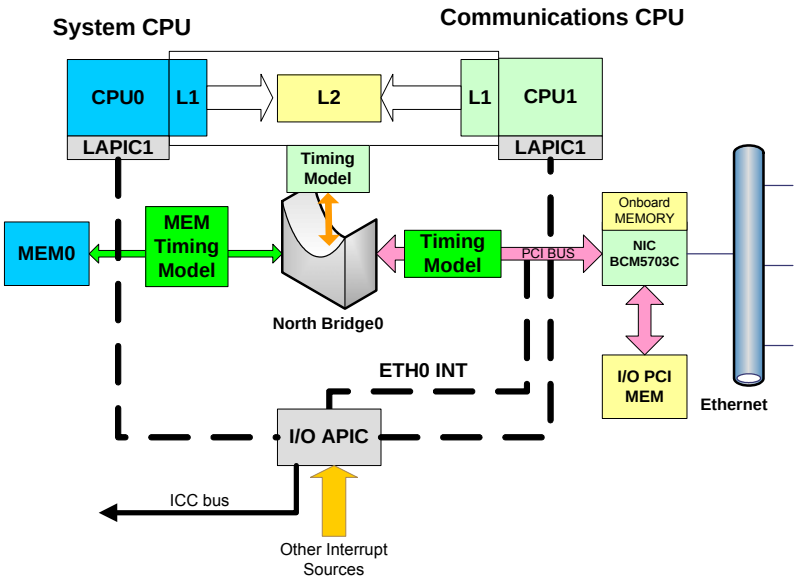


Figure 6. Simulation models for the proposed offloading (a) and onloading (b) implementations

In [43], a performance study of memory behavior in a core TCP/IP suite is presented. In this thesis, the protocols are implemented in the user space, as the authors cannot access the source code of the operating system used in the SGI platform where the study is done. It is concluded that, in most scenarios, instruction cache behavior has a significant impact on performance (even higher than data cache). This situation should continue to hold for small average packet sizes and zero-copy interfaces. In [43], it is also concluded that larger caches and higher associativity improve communication performance (mainly for TCP communication) as many cache misses in packet processing are conflict misses. Moreover, it is also indicated that network protocols should scale with clock speed except for *cold* caches (cases where caches do not store correct information of data and instructions at the beginning of packet processing) where cache performance shows an important decrease.

The papers by Mudigonda and cols. [47,48] analyze the memory wall problem in the context of network processors. Paper [47] concludes that data caches can reduce the packet processing time significantly. Nevertheless, in such communication applications where there is low computation per memory reference, each miss can lead to significant stall time. Thus, in these applications, data caches should be complemented with other techniques to manage memory latencies and achieve acceptable processor utilization. In this same context, in [48] the way the memory accesses limit the network interface performance is also analyzed. This thesis considers the mechanisms used by network processors to overcome the memory bottlenecks and concludes that data caches and multithreading strategies must cooperate to achieve the goals of high packet throughputs and network interface programmability.

Figure 7 shows the hybrid model operation. As mentioned above, the proposed hybrid model takes advantage of offloading and onloading techniques. In this model, CPU2 is the processor included in the NIC and executes the communication protocols, CPU1 executes the driver in the same way the onloading model does, but this CPU1 is also able to execute other tasks such as system calls for copying the data from the TCP sockets to the application buffers. The interrupts are received by CPU1, which also executes the driver, as in the onloading alternative. So, the hybrid model does not disturb the CPU0 while receiving data. Therefore, as the CPU0 executes the application

and it is only focused on being the data sink, the reached throughput is higher than the throughput for offloading or onloading cases.

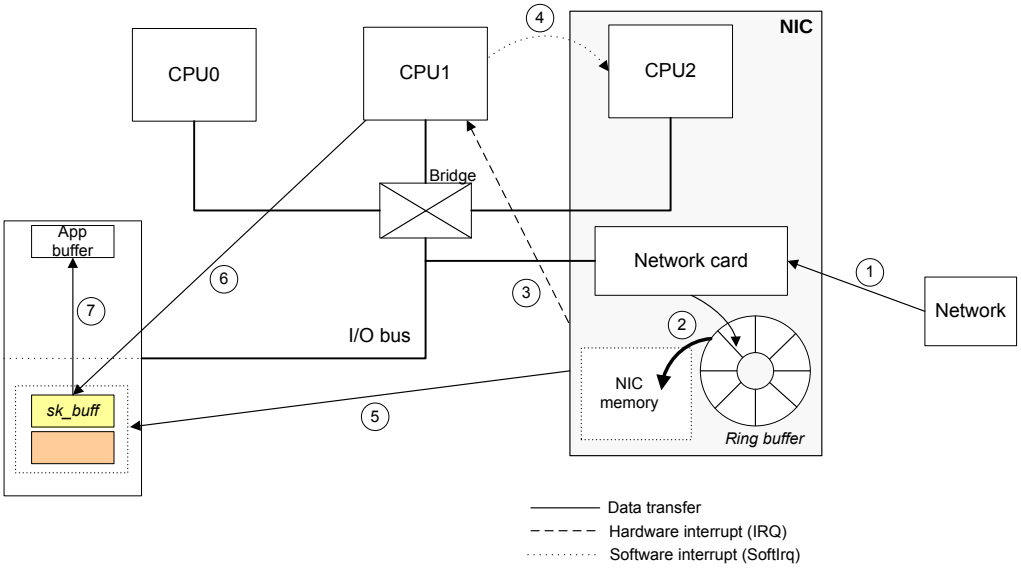


Figure 7. Hybrid model operation

In Figure 7, after receiving a packet (1), the NIC stores it in the *ring buffer*. Whenever this buffer is full or the packet transfer is finished, the packets are moved from the ring buffer to the NIC memory (2). Then, the NIC requests a hardware interrupt to CPU1 (3). This interrupt causes the execution of the function *do_irq* [50] to call the corresponding interrupt routine defined in the device driver to copy the data into the *sk_buff* structures [50]. Then, the driver starts a *softirq* [50] routine in CPU2 (4) to process the packets according to the parameters of the *sk_buff* structures. These parameters correspond, for example, to the protocols used in the higher layers (for example IP and TCP). Once the protocol stack has been processed, the data is moved to the TCP socket (5) where they can be accessed by the application, (6) and (7).

Figure 8 shows the simulation model corresponding to the hybrid network interface. Moreover, in this hybrid alternative, the NIC is directly connected to the north bridge (or directly to the multi-core chip) to avoid a bottleneck on the PCI bus and also includes one level of cache memory. This feature is implemented in the simulation model by fitting the time models of cache and PCI bus. As in the offloading model, we are using a NIC that contains an onboard memory that serves as a receive ring. So, although the interrupts are not arriving at the main host CPU (CPU0), the "per event interrupt" approach aids to reduce the interrupts generated to the CPU1. As a result, there are more idle cycles on CPU1 and it is possible to use them to execute some application threads, although through prioritizing the communication tasks (i.e. the driver execution).

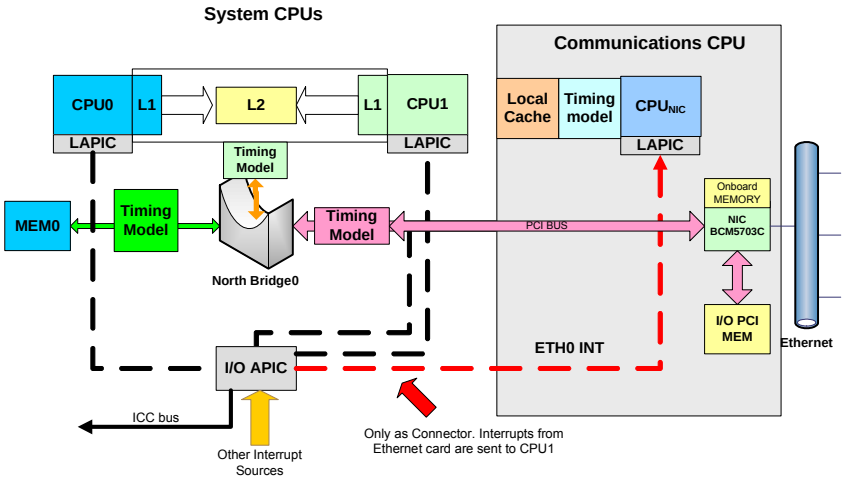


Figure 8. Simulation models for the proposed hybrid network interface

Experimental results

In our SIMICS models, we have used Pentium 4 processors running at 400 MHz as CPU0 (this frequency is enough to have up to 1 Gbps at link level and not to slow down the simulation speed). We have also included in our system a 128 MB DRAM, an APIC bus, a 64 bit PCI bus with a Tigon-3 (BCM5703C) gigabit Ethernet card attached, and a text serial console. The characteristics of the cache memories used in the models are provided in Table 2.

Table 2. Characteristics of the cache memories on the simulated machines

	Instruction Cache L1	Data Cache L1	Cache L2
Write policy	-	*write-through*	*write-back*
Number of lines	256	256	8096
Line size (bytes)	64	64	128
Associativity (lines)	2	4	8
Write back (lines)	-	-	1
Write allocate (lines)	-	-	1
Replacement policy	LRU	LRU	LRU
Read latency (cycles)	2	3	5
Write latency (cycles)	1	3	5

Figure 9 and 10 provide our first throughput comparisons between onloading and offloading. They have been obtained by using TCP as transport protocol and *netpipe* [24] as benchmark. This test measures the network performance in terms of the available throughput between two hosts and consists of two parts, a protocol independent driver and a protocol specific communication section that implements the

connection and transfer functions. For each measurement, *netpipe* automatically increases the block size (see [24] for details).

The simulations corresponding to the results of Figure 7 and 8 have been carried out by using low application workloads. In these experiments, there is almost no workload apart from the communication tasks. This initial situation allows us to validate the simulation models and the timing effects.

Figure 9 shows that both onloading and offloading cause improvements in the peak throughput, although these improvements depend on the size of the messages. Figure 9 also provides the throughput curves for offloading simulations that consider different speeds in the processor at the NIC (CPU1) with respect to the host processor (CPU0). Thus, the parameter α (lag ratio) in the LAWS model, which considers the ratio between the cycle times of the CPU and the NIC, is equal to 1 whenever CPU0 and CPU1 have the same speed. As the CPU1 becomes slower than the host CPU, the parameter α decreases (i.e., α=0.5 means that the host CPU is twice faster than the CPU at the NIC), and the improvement obtained by offloading also decreases, as Figure 9 shows. Moreover, in the case of slower CPUs at the NIC, the throughput could be even worse than in a non offloaded system. This circumstance can be explained from the LAWS model (see the δb expression on Section 3), and it can also be observed in Figure 9, that shows how the peak throughput decreases as α is reduced.

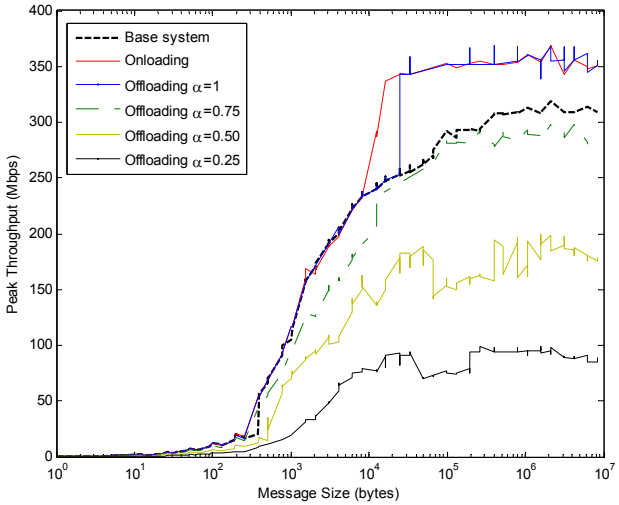
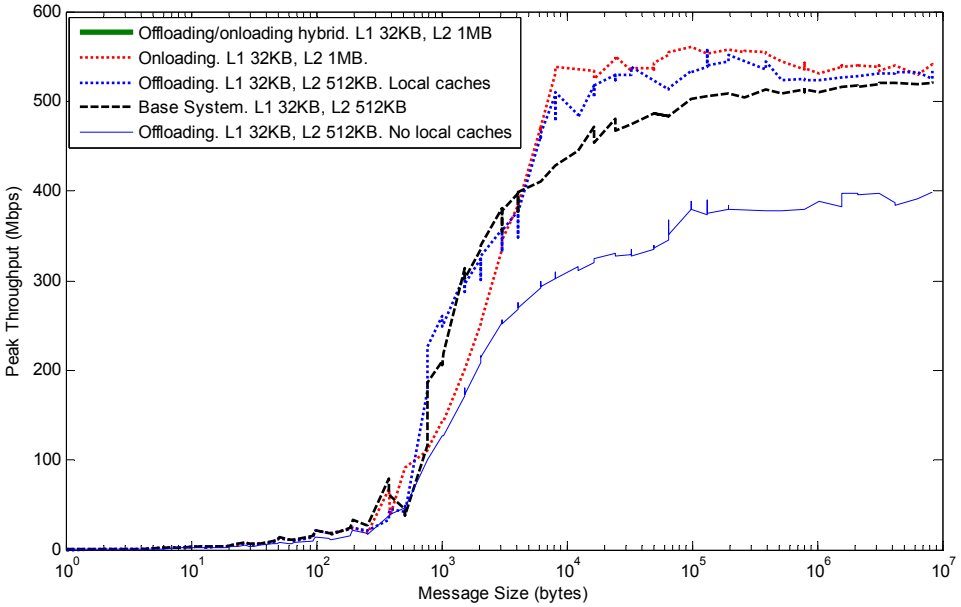
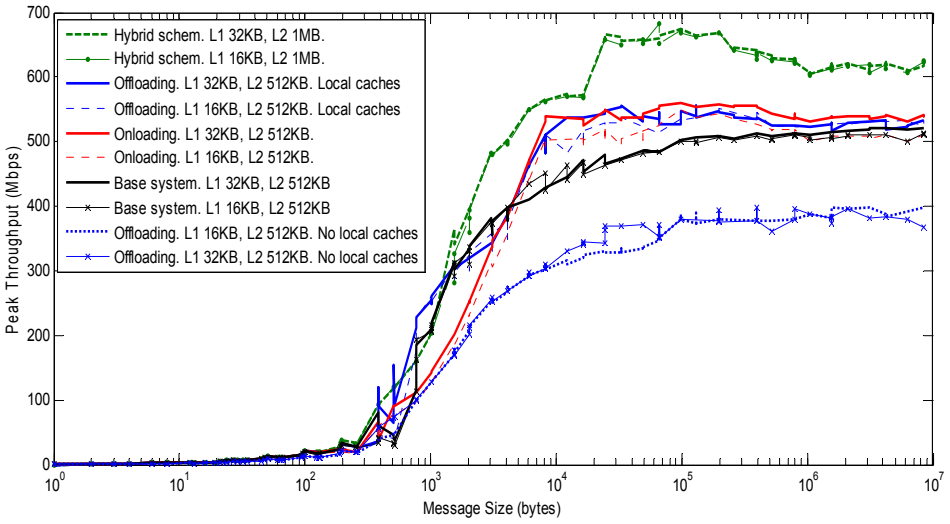


Figure 9. Peak throughput comparison (including the effect of the LAWS parameter α for offloading)

In the experiments corresponding to the results of Figure 6, we have only simulated the cache through the mean access time. Figures 8.a and 8.b show the effect of having a detailed two-level cache model caches in the throughput. From Figure 8.a, it is clear that caches improve the throughput provided by each network implementation alternative. Nevertheless, the amount of this improvement is different for each case. Thus, when a detailed cache model is included, the interface in the base system and the onloaded interface show higher improvements than the offloaded network interface. This can be explained by taking into account that, as the interface is mainly processed in the NIC in this case, the presence of a cache in the node almost does not affect its performance because the CPU in the NIC uses its local memory. However, if we simulate the effect of having a local cache in the NIC, the throughput is increased as shown in Figure 10.a. Figure 10.b shows that the increase in the cache size (L1 data cache size) only produces a slight throughput improvement in case of our onloading implementation.



(a)



(b)

Figure 10. Effect of the cache hierarchy in the throughput improvement: (a) comparison of throughputs with and without caches; (b) effect of the increase in the L1 data cache size.

The benefits of onloading and offloading should affect not only throughput but also message latency. They are provided in Table 3. As can be seen from this table, the cache hierarchy contributes to reduce the latencies more in the base and in the onloaded interfaces than in the offloaded interface. This situation is similar to that observed in the throughputs. Thus, the onloading strategy provides better latencies whenever a cache hierarchy is present in the node, as it is usual.

Table 3. Network latency for different interfaces and cache hierarchy configurations

Interface	Without detailed cache model	With detailed 2-level cache model (32KB L1 cache/512KB cache)
Base system	68µs	38 µs
Offloading	60 µs	60 µs
Offloading. Local caches	36 µs	36 µs
Onloading	64 µs	36 µs

Figure 11 provides the behavior of the latencies with respect to the message sizes and different cache hierarchy configurations (Figure 11.a and 11.b). It can be seen that, if the caches are disabled, the latencies are lower for offloading (Figure 8.b). As the protocol is offloaded to the network interface, it can interact with the network through less I/O buses transferences. Nevertheless, the presence of caches changes this situation and a higher reduction in the latencies for the base and the onloaded interfaces than in the offloaded one is shown. These results demonstrate the known fact that the characteristics of the interaction between the network interface and the memory hierarchy of the node have an important role in the communication performances.

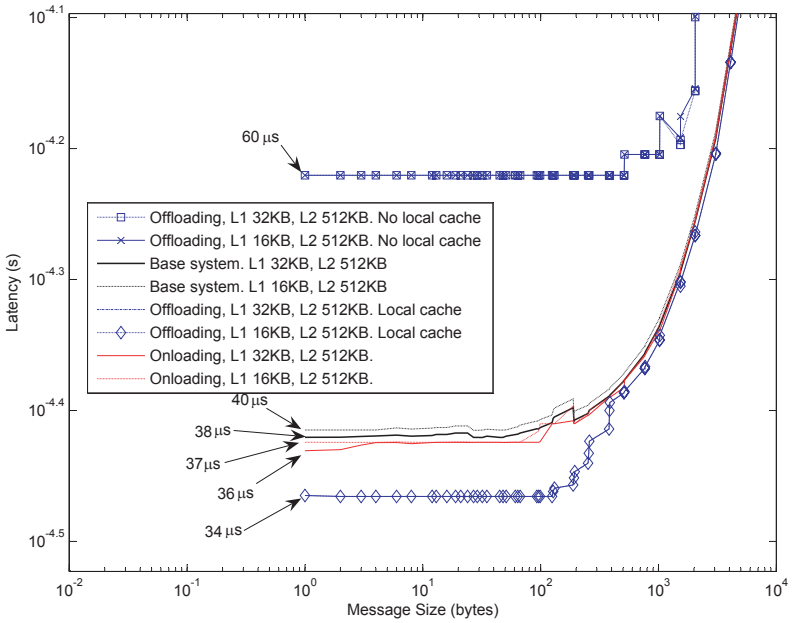


Figure 11. Latency. Effect of increasing L1 size.

In the experimental results which we have presented up to now, we have mainly compared the behavior of the onloading and offloading strategies against changes in the size of the messages. We are also interested on the performance as the application workload changes. The LAWS model has allowed us to organize our exploration of the space of alternatives for onloading and offloading.

In Figure 12 we show the experimental results and the model fitting using both, the original and the modified LAWS model. In this Figure, we present two different fits using different values for the new parameters that have been added to the original LAWS model (δ_a,δ_o,τ)

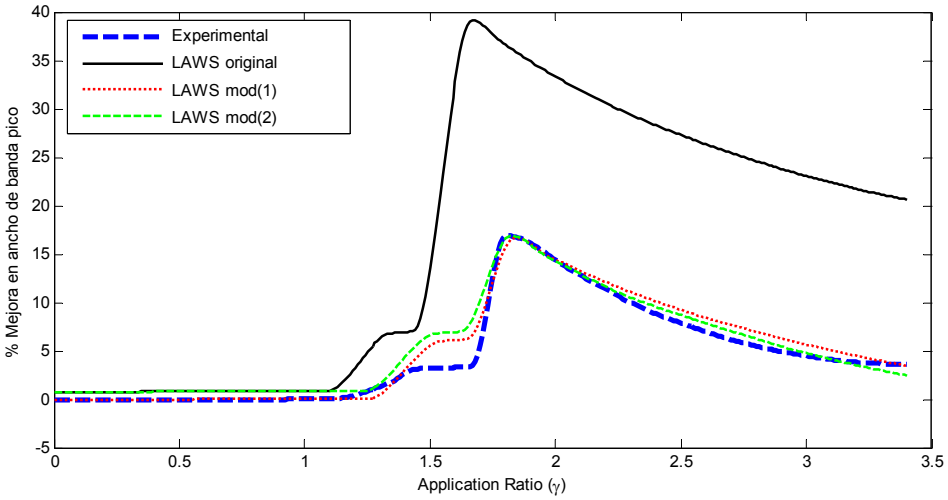


Figure 12. LAWS model and modified LAWS model

As it has been said, the LAWS model provides a way to understand the performance behavior under different application workloads and technological characteristics of the processors in the system. The characteristics of the application with respect to communication requirements is considered by using the rate of application workload to communication overhead, γ, and the technological impact is taken into account through the lag ratio, α. In LAWS, the message size can be taken into account through the communication overhead in the application ratio, γ, and through the wire ratio, σ (as the message size also affects the throughput provided by the host). Nevertheless, in our experiments, we use a given message size while we change the application workload with respect to the communication overhead to get the different values for the parameter γ.

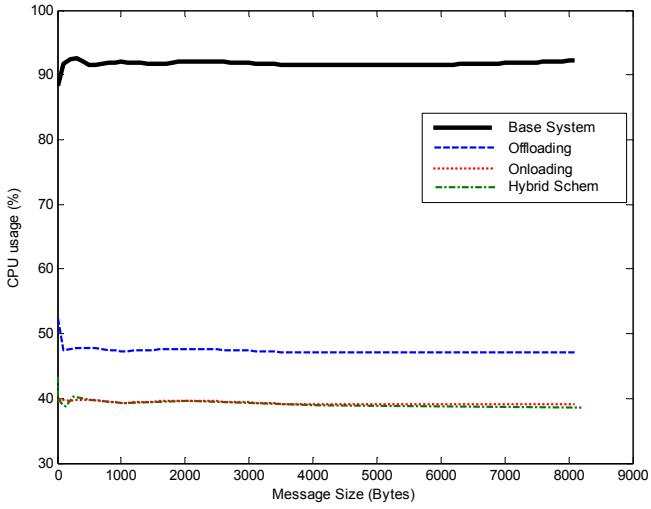


Figure 13. Comparison of CPU utilization (with hpcbench)

Figure 13 shows the host CPU utilization for offloading, onloading, the hybrid network interface and the base system at different message sizes. The results have been obtained using *Sysmon hpcbench* [26] as benchmark. In the simulations, we have used a very intensive communication workload. As we can see, since onloading allows the

NIC driver to be executed in CPU1, the CPU0 load is even lower than in the offloading case. In this case, although all the protocol processing is offloaded to the NIC, the driver is executed in the host CPU (CPU0) and this causes a higher load.

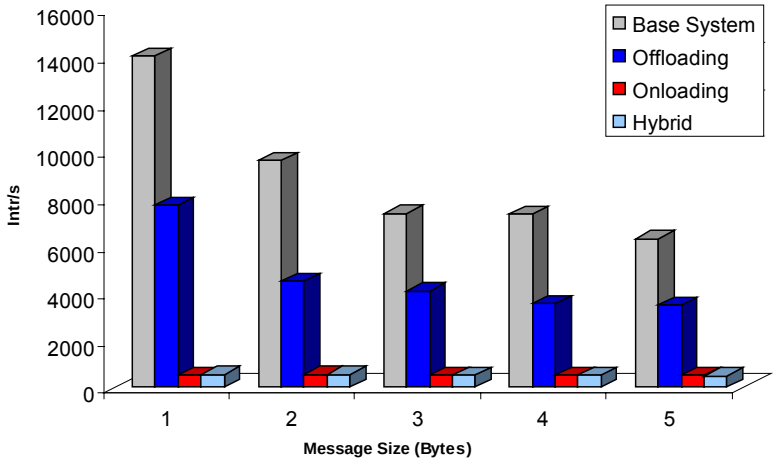


Figure 14. Comparison of interrupts vs. message size

Figure 14 shows the interrupts attended by CPU0 considering different message sizes. The number of interrupts has been obtained by counting them along a given amount of time (one second). In all our SIMICS simulations (base system, offloading, and onloading), we have included the mechanisms to reduce interrupts that are currently common in the NICs, such as *Jumbo frames* and *interrupt coalescing*. This is the reason for which the number of interrupts for offloading and non-offloading is very similar. In [19], the provided decrease in the interrupts per second obtained by offloading with respect to non-offloading was about 60% for TCP, and about 50% for UDP. In these simulations, the modeled NICs do not include either Jumbo frames or interrupt coalescing. Figure 14 also demonstrates the high reduction in the number of interrupts received by CPU0 with onloading. In this case, CPU1 executes all the communication software, including the NIC driver. In the case of offloading and the base system, the number of interrupts decreases with the size of the messages as the number of received messages decreases.

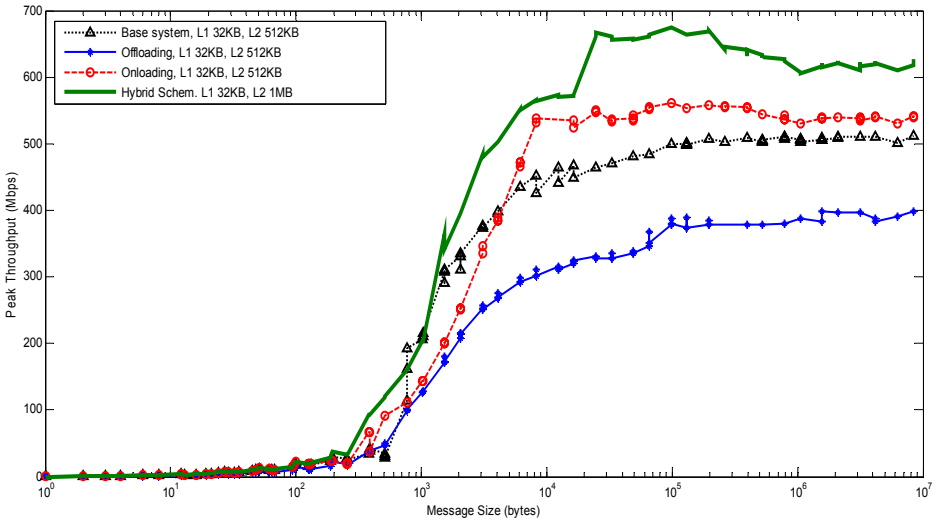


Figure 15. Comparison of throughputs for the hybrid model

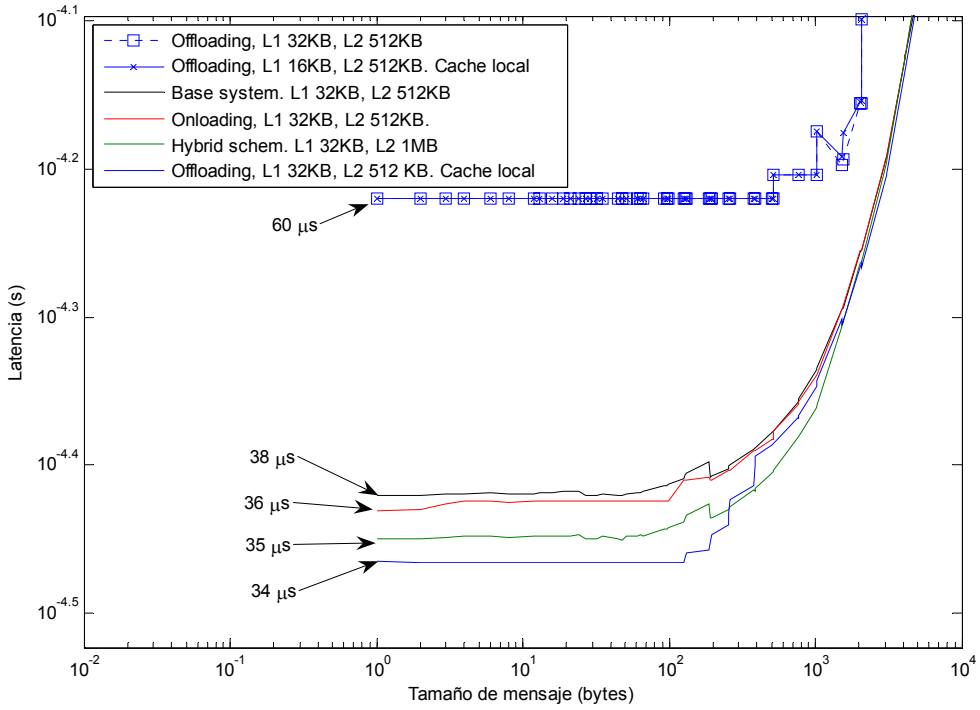
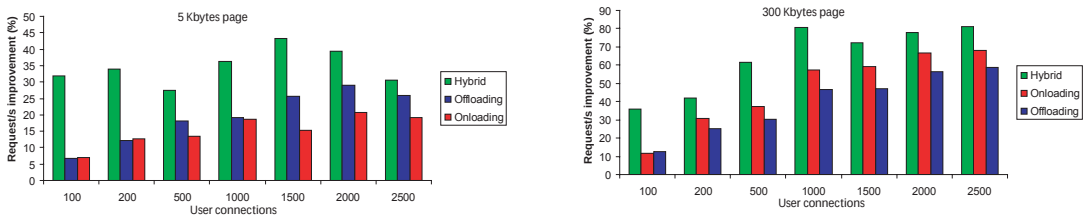


Figure 16. Comparison of latencies for the hybrid model

From the shown experimental results, it is clear that onloading and offloading offer different advantages that can be jointly exploited and drawbacks that can be avoided if we are able to take advantage of processors in the NIC and other additional processors with the same privileges for memory accesses as the host processor (CPU0). Thus, we have implemented a network interface that hybridizes some onloading and offloading strategies. Figure 16 shows that our hybrid interface provides better peak throughput improvement than either onloading or offloading. Moreover, from Figure 16, the fact that the latency achieved by this hybrid scheme is almost the same that the lowest latency, obtained by onloading, can be concluded.

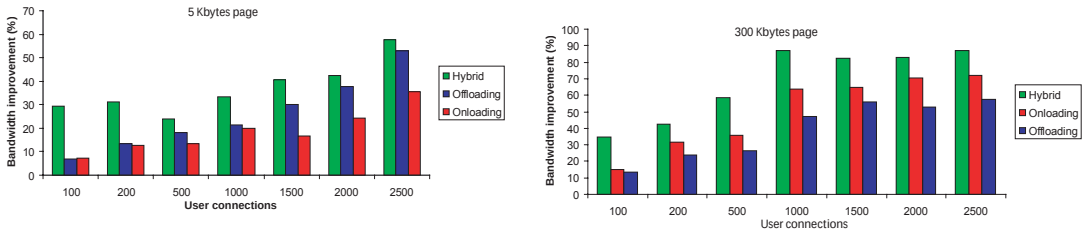


(a) *(b)*

Figure 17. (a) Request per second improvement for (a) 5 Kbytes page (b) 300 Kbytes page

In order to analyze the performance of our proposed interfaces, it is also interesting to use not only benchmarks but also a real application such as a web server. In this way, we have loaded our simulation platform with an Apache 2.0 web server. The experiments performed to obtain Figures 17 and 18 consist of measuring the requests per second provided by each proposed interface and varying the number of concurrent user connections from 100 up to 2500.

As can be seen in Figure 17a, while the offloading interface provides a higher number of requests served per second than onloading for small pages (5 Kbytes), the highest improvement is provided by the hybrid network interface and the change of this improvement when the user connection increase is less marked. As shown earlier on this section, the offloading interface provides better results for small messages than onloading does, and the hybrid interface provides higher bandwidth and lower latency values than offloading or onloading.
In Figure 17b, we show the results of the experiments for a 300 Kbytes page. In this experiments, the improvement in requests per second is also higher for the hybrid interface, but onloading provides better results than offloading in this case. This experiments also show that the improvement increases as the server load increases (user connections), according to LAWS model prediction.



(a) *(b)*

Figure 18. Bandwidth improvement for (a) 5 Kbytes page (b) 300 Kbytes page

In Figure 18, we shown the bandwidth provided by the network interfaces. The improvement in bandwidth is similar to the improvement in request per second because more requests served per second mean a higher bandwidth if the network interface is able to deliver the needed bandwidth.
Although some of the results obtained with the web server could be predicted with the knowledge of the analysis provided earlier on this section, it is clear that the improvement depends on the specific application profile. In this way, a web server application doesn't fit to a streaming or ping-pong profile but it is a mixture of different communication profiles and depends for instance, on the size of the client request and server response.

References

[1] Magnusson, P. S.; et al.:"Simics: A Full System Simulation Platform". IEEE Computer, pp.50-58. February 2002.

[2] Binkert, N.L.; Hallnor, E.G.; Reinhardt, S.K.:"Network-oriented full-system simulation using M5". Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CECW). February, 2003.

[3] M5 simulator system Source Forge page: http://sourceforge.net/projects/m5sim

[4] Virtutech web page: http://www.virtutech.com/

[5] Westrelin, R.; et al.:"Studying network protocol offload with emulation: approach and preliminary results", Proc. 12th Annual Symp. IEEE on High Performance Interconnects, pp.84-90, 2004.

[6] Mogul, J.C.:"TCP offload is a dumb idea whose time has come". 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.

[7] Regnier, G. et al.:"TCP onloading for data center servers". IEEE Computer, pp.48-58. November2004.

[8] Clark, D.D. et al.:"An analysis of TCP processing overhead". IEEE Communications Magazine, Vol. 7, No. 6, pp.23-29. June, 1989.

[9] O'Dell, M. :"Re: how bad an idea is this?". Message on TSV mailing list. Noviembre, 2002.

[10] Shivam, P.: Chase, J.S.:"On the elusive benefits of protocol offload". SIGCOMM'03 *Workshop on Network-I/O convergence: Experience, Lesons, Implications (NICELI)*. August, 2003.

[11] Gilfeather, P.; Maccabe, A.B.:"Modeling protocol offload for message-oriented communication". Proc. of the 2005 IEEE International Conference on Cluster Computing (Cluster 2005) , 2005.

[12] Martin, M.M.; et al.: "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset ". Computer Architecture News (CAN), 2005.

[13] Mauer, C.J.; et al.:"Full-System Timing-First Simulation". ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, June, 2002.

[14] Rosenblum, M.; et al.:"Using the SimOS machine simulator to study complex computer systems". ACM Tranns. On Modeling and Computer Simulation, Vol.7, No.1, pp.78-103. January, 1997.

[15] Kim, H.-Y.; Rixner, S.:"TCP offload through connection handoff". ACM Eurosys'06, pp.279-290, 2006.

[16] http://www.broadcom.com/, 2007.

[17] http://www.chelsio.com/, 2007.

[18] http://www.neterion.com/, 2007.

[19] Ortíz, A.; Ortega, J.; Díaz, A. F.; Prieto, A.:"Protocol offload evaluation using Simics". IEEE Cluster Computing, Barcelona. September, 2006.

[20] Competitive Comparison. Intel I/O Acceleration Technology vs. TCP Offload Engine: http://www.intel.com/technology/ioacceleration/316126.pdf

[21] Wun, B.; Crowley, P.:"Network I/O Acceleration in Heterogeneous Multicore Processors". In *Proceedings of the 14th Annual Symposium on High Performance Interconnects (Hot Interconnects)*. August, 2006.

[22] Intel I/O Acceleration Technology: http://www.intel.com/technology/ioacceleration/index.htm

[23] Vaidyanathan, K.; Panda, D.K.:"Benefits of I/O Acceleration Technology (I/OAT) in Clusters" Technical Report Ohio State Univ. (OSU_CISRC-2/07-TR13)

[24] Snell, Q.O., Mikler, A; Gustafson, J.L.:"NetPIPE: A Network Protocol Independent Performance Evaluator," *IASTED International Conference on Intelligent Information Management and Systems,* June 1996.
[25] Andrew, G.; Leech, C.; "Accelerating network receiver processing". http://linux.inet.hr/files/ols2005/grover-reprint.pdf
[26] Huang, B.; Bauer, M.; Katchabaw, M.: "Hpcbench – a Linux-based network benchmark for high performance networks". 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05). 2005
[27] Ciaccio, G.: "Messaging on Gigabit Ethernet: Some experiments with GAMMA and other systems". Workshop on Communication Architecture for Clusters, IPDPS, 2001.
[28] Díaz, A. F.; Ortega, J.; Cañas, A.; Fernández, F.J.; Anguita, M.; Prieto, A.: "A Light Weight Protocol for Gigabit Ethernet". Workshop on Communication Architecture for Clusters (CAC'03), IPDPS'03. April, 2003.
[29] Bhoedjang, R.A.F.; Rühl, T.; Bal, H.E.: "User-level Network Interface Protocols". IEEE Computer, pp.53-60. November, 1998.
[30] http://www.vidf.org/ (Virtual Interface Developer Forum, VIDF, 2001).
[31] Chakraborty, S. et al.:"Performance evaluation of network processor architectures: combining simulation with analytical estimation". Computer Networks, 41, pp.641-645, 2003.
[32] Thiele, L. et al.:"Design space exploration of network processor architectures". Proc. 1st Workshop on Network Processors (en el 8th Int. Symp. on High Performance Computer Architecture). February, 2002.
[33] Papaefstathiou, I.; et al.: "Network Processors for Future High-End Systems and Applications". IEEE Micro. September-October, 2004.
[34] GadelRab, S.:"10-Gigabit Ethernet Connectivity for Computer Servers". IEEE Micro, pp.94-105. May-June, 2007.
[35] Karamcheti, V.; Chien, A.A.:"Software overhead in messaging layers: where does time go?". Proc. of ASPLOS-VI, San Jose (California), October 5-7, 1994.
[36] Chase, J.S.; Gallatin, A.J.; Yocum, K.G.:"End system optimizations for high-speed TCP". IEEE Communications, pp.68-74. April, 2001.
[37] Maquelin, O.; et al.:"Polling watchdog: combining polling and interrupts for efficient message handling". Proc. Int`l. Symp. Computer Architecture, IEEE CS Press, pp.179-188, 1996.
[38] Welsh, M.; et al.:"Memory management for user-level network interfaces" IEEE Micro, pp.77-82. March-April, 1998.
[39] Tezuka, H,; et al.:"Pin-down Cache: a virtual memory management technique for zero-copy communication". Proc. Int'l Parallel Processing Symp., IEEE CS Press, pp.308-314, 1998.
[40] Foong, A.; et al.:"TCP performance revisited". Proc. IEEE Int`l Symp. Performance Analysis of Software and Systems, IEEE CS Press, pp.70-79, 2003.
[41] Kim, H.; Rixner, S.; Pai, V.S.:"Network Interface Data Caching". IEEE Trans. On Computers, Vol. 54, No.11, pp.1394-1408. November, 2005.
[42] Lauritzen, K.; Sawicki, T.; Stachura, T.; Wilson, C.E.:"Intel I/O Acceleration Technology improves Network Performance, Reliability and Efficiency". Technology@Intel Magazine, pp.3-11. March, 2005.
[43] Nahum, E.M.; Yates, D.; Kurose, J.F.; Towsley, D.:"Cache behaviour of Network Protocols". SIGMETRICS'97, 1997.

[44] Nahum, E.M.; Yates, D.J. ; Kurose, J.F.; Towsley, D.:"Performance issues in parallelized network protocols". *Proc. Of the Operating Systems Design and Implementation*, pp. 125-137, 1994.
[45] Kim, H.; Pai, V.S.; Rixner, S.: "Exploiting task-level concurrency in a programmable network interface". Proc. of the ACM PPoPP'03, 2003.
[46] Brogioli, M.; Willman, P.; Rixner, S.:"Parallelization strategies for network interface firmware". Proc. of 4th Workshop on Optimization for DSP and Embedded Systems, ODES-4, 2006.
[47] Mudigonda, J.; Vin, H.M.; Yavatkar, R.:"Managing Memory Access Latency in Packet Processing". SIGMETRICS'05, pp.396-397, 2006.
[48] Mudigonda, J.; Vin, H.M.; Yavatkar, R.:"Overcoming the memory wall in packet processing: hammers or ladders?". Proc. of the ACM ANCS'05, 2005.
[49] Turner, D.; Oline, A.; Chen, X.; Benjegerdes, T.:"Integrating new capabilities into NetPIPE". 10th European PVM/MPI User's Group Meeting, 2003.
[50] Benvenuti, C.: "Understanding Linux Network Internals". O'Reilly Media Inc. December, 2005.
[51] Balaji, P.; Feng, W.; Panda, D.K.:"Bridging the Ethernet-Ethernot performance gap". IEEE Micro, pp.24-40. May-June, 2006.
[52] Alteon Websystems: "Extended Frame Sized for Next Generation Ethernets" http://staff.psc.edu/mathis/MTU/AlteonExtendedFrames_W0601.pdf

Presentación

Es previsible que las necesidades sociales, las exigencias del mercado, y las ideas innovadores que buscan satisfacerlas, sigan generando aplicaciones que demandan prestaciones cada vez más elevadas en cuanto a tiempo de procesamiento, capacidad de almacenamiento y velocidad de comunicación en servidores de cálculo, servidores de ficheros, servidores web, etc. Así, han ido surgiendo microprocesadores que han aprovechado el paralelismo entre instrucciones, el procesamiento multihebra y, más recientemente, la inclusión de varios núcleos de procesamiento (microprocesadores multinúcleo o multiprocesadores monochip, *Chip Multiprocessors*, CMP) para aumentar sus prestaciones incorporando las mejoras tecnológicas y las innovaciones arquitectónicas. Por otra parte, también se han extendido las plataformas de cómputo de altas prestaciones basadas en la interconexión de procesadores a través de las arquitecturas de memoria compartida (como los multiprocesadores simétricos, *Symmetric Multiprocessors*, SMP) y las de memoria distribuida con tecnologías de interconexión específicas o estándares como es el caso de los *clusters*. En cualquier caso, las necesidades de comunicación son cada vez mayores, tanto dentro de las plataformas multiprocesador, como en las conexiones a los servidores.

Con la tecnología actual, las prestaciones disponibles en los microprocesadores hacen que se puedan construir plataformas de altas prestaciones mediante los llamados *clusters de computadores,* los cuales proporcionan una buena relación prestaciones/coste, pudiéndose utilizar para altas prestaciones o para alta disponibilidad. Más aún, con el desarrollo actual de Internet, los clusters de computadores pueden estar distribuidos, de modo que se pueden constituir grandes servidores de altas prestaciones a partir de PCs de usuarios conectados a Internet, como los proyectos de computación

distribuida SETI [SET07] o BOINC [BOI07] y servidores con tecnología GRID [ORA08].

Actualmente, cada vez es más patente la limitación de los procesadores actuales para aprovechar el ancho de banda que proporcionan las redes. Además este problema puede ser cada vez más acuciante si se atiende a la ley de Gilder, que predice un crecimiento del doble del ancho de banda de los enlaces cada nueve meses. Suponiendo que el aumento de las prestaciones de los microprocesadores se mantiene según lo que establece la ley de Moore (el doble cada 18 ó 24 meses), el desfase entre ancho de banda disponible y prestaciones de los procesadores aumentará a gran velocidad. Lo mismo que el desfase entre la velocidad de los procesadores y la memoria ha influido de manera decisiva en la forma de organizar el computador generalizando el uso de jerarquías de memoria (junto con las correspondientes estrategias para resolver las consecuencias de incluir dichas jerarquías), el aprovechamiento de los cada vez mayores anchos de banda tendrá consecuencias importantes en el diseño de las interfaces de red de los computadores.

En este sentido, existen diferentes alternativas en cuanto a la elección de una red de interconexión de altas prestaciones. Entre ellas, Myrinet [MYR07], SCI [DOL07], Infiniband [INF07], o QSNet [QSN07], concebidas para su uso en clusters de computadores formados por workstations, PCs, servidores, o computadores monoplaca, así como otras redes tales como ATM, Gigabit Ethernet [EST99] o 10 Gigabit Ethernet [EST06], 40 Gigabit Ethernet [ETS06] incluso 100 Gigabit Ethernet que actualmente está en desarrollo, las cuales, aunque se pueden utilizar también en clusters de computadores, están orientados a la interconexión de máquinas que requieran un gran ancho de banda. Concretamente, las redes Ethernet, debido a su bajo coste y facilidad de implementación y mantenimiento, han sido las más extendidas en los últimos tiempos, de forma que incluso algunas empresas están sustituyendo sus redes *backbone AT*M por Gigabit Ethernet [EST99, PET05, SKO97]. Esto es debido, en parte, a la posibilidad de utilizar par trenzado. Cabe destacar que el estándar 10-Gigabit Ethernet no gozó de gran popularidad en sus inicios, dado que, a pesar de que se está desarrollando un estándar 10-Gigabit Ethernet sobre par trenzado de categoría 6A, el coste de la implantación de una red 10-Gigabit Ethernet es muy superior al de Gigabit Ethernet. Debido a esto, se ha desarrollado el estándar 40-Gigabit Ethernet, y actualmente se encuentra en desarrollo 100-Gigabit Ethernet, para competir con otras redes de similares prestaciones. En cualquier caso, las redes Gigabit Ethernet, debido al coste comparado con otras

alternativas antes mencionadas y a la facilidad de implantación y mantenimiento, se utilizan comúnmente en aplicaciones que necesiten de gran ancho de banda y baja latencia [GRA02, HUG05].

Este avance de la tecnología de las redes de interconexión basadas en Ethernet, ha proporcionado un salto de un orden de magnitud en el ancho de banda. Esto ha hecho que el ancho de banda proporcionado por estas redes sea comparable al proporcionado por redes como Myrinet [MYR07] o SCI [DOL07], las cuales están especializadas para el uso en plataformas paralelas de altas prestaciones. Por otro lado, las redes especializadas no Ethernet, también llamadas *Ethernot* [BAL06], tienden a permitir compatibilidad con redes Ethernet, propiciando la convergencia *Ethernet-Ethernot* [BAL06].

Sin embargo, el aprovechamiento del ancho de banda que proporcionan los enlaces de red no siempre es posible, debido a la sobrecarga generada por los procesos de comunicaciones, que pueden llegar incluso a colapsar el nodo.

Para mejorar las prestaciones de las comunicaciones, y poder aprovechar el ancho de banda de los enlaces de red de altas prestaciones, evitando que el nodo se convierta en un cuello de botella que limite las prestaciones globales del sistema, se ha desarrollado un activo trabajo de investigación en dos vertientes: la primera se centra en la mejora de los protocolos de red utilizados, proponiendo alternativas que reducen la sobrecarga generada debida a los procesos de comunicaciones, como GAMMA [CIA01] o CLIC [DIA03], o protocolos de red a nivel de usuario [BHO98, VID01]. Esta línea ha desembocado en un estándar como VIA [VID01], incorporado en Infiniband. La segunda vertiente, trata de mejorar las prestaciones del hardware de la interfaz de red, de modo que este pueda realizar todo o parte del proceso de los protocolos de red, liberando así a la CPU del nodo de este trabajo [PAP04].

El uso de protocolos ligeros constituye una de las soluciones al problema de la liberación de carga de la CPU, aunque existe una tendencia a utilizar el protocolo estándar TCP/IP, ya que permite a estos servidores operar directamente sobre Internet, más aún cuando se trata de servidores que proporcionan servicios en Internet, como es el caso de servidores web. El protocolo TCP es el protocolo básico de comunicación tanto en Internet como en diversas redes privadas o de área local en todo el mundo. Además, existe una tendencia a reemplazar redes como ATM, SCI, Fiberchannel por Ethernet [SKO97], utilizando TCP/IP. Otro ejemplo de utilización de TCP/IP, se encuentra en los servidores de almacenamiento, en los cuales se utiliza en muchas

ocasiones SCSI sobre Ethernet (con el protocolo iSCSI que consiste en la transferencia de comandos SCSI sobre TCP/IP). Actualmente, incluso se está utilizando TCP de forma comercial para realizar transferencias de *streaming* de video [WAN04].

Sin embargo, el procesamiento de la pila de protocolos TCP/IP y la implementación sobre redes de altas prestaciones genera una gran sobrecarga en el host [CHU96], mayor que la que generan los protocolos ligeros mencionados anteriormente, por lo que se hace necesario utilizar mecanismos que reduzcan dicha sobrecarga: para poder aprovechar el ancho de banda de los enlaces, las prestaciones ofrecidas por las redes basadas en TCP deben crecer al mismo tiempo, mejorando para ello, las prestaciones de la pila de protocolos TCP y de la interfaz de red.

Con este objetivo, existen trabajos que proponen mejoras del protocolo TCP para utilizarlo en redes de alta velocidad [SAL03], u otros que proponen llevar el procesado del protocolo a un procesador externo a la CPU principal del nodo [RAN02, DEL06, REG04A, IOA05]. Esta técnica se conoce como *externalización de protocolos de comunicación.* Alternativas comerciales como [IOA05] incluyen en su implementación de la externalización ambas técnicas: una pila TCP optimizada en cuanto al consumo de recursos, y la externalización del procesado del protocolo en un procesador diferente a la CPU principal del nodo.

Sin embargo, frente a trabajos como [WES04, REG04A, KIM06, DEL06], que ponen de manifiesto las ventajas de la externalización de protocolos, existen otros como [MOG03] que ponen en duda estas ventajas, especialmente para la externalización del protocolo TCP.

Para externalizar los protocolos de comunicación se han propuesto diferentes técnicas: aquellas que pasan el procesado de los protocolos a un interfaz de red externo, conocidas como *offloading*, las que utilizan uno de los procesadores de un CMP o un SMP para procesar los protocolos, llamadas *onloading* (y las que utilizan protocolos TCP/IP modificados que se ejecutan en modo usuario, llamados *uploading)*. Mientras que existen implementaciones comerciales de offloading y de onloading, la llamada técnica de *uploading* se suele utilizar para la investigación, existiendo diferentes propuestas experimentales para su implementación, como son [RAD04] y [DIN02].

En esta memoria, se estudian alternativas para la implementación tanto de *offloading* como de *onloading*, debido a la gran controversia que actualmente existe en cuanto a las prestaciones proporcionadas por cada una de estas técnicas [MOG03] [INT04], con el fin de aportar claridad en cuanto a las condiciones en las que puede ser

mas beneficioso utilizar una técnica u otra. En este análisis, se han tenido en cuenta el espacio de parámetros de diseño del sistema en el que se externalizan los protocolos de comunicación, utilizando para ello el modelo teórico LAWS [SHI03]

Para evaluar las prestaciones de una arquitectura de computadores o de un sistema de comunicaciones, habitualmente se utiliza la técnica de simulación [TRA94, ENG03]. Sin embargo, el simulador utilizado debe ser capaz de modelar el sistema con un grado de detalle suficiente para permitir la ejecución de aplicaciones reales, con el fin de reproducir lo más fielmente posible, las condiciones de trabajo de una máquina real. En nuestro caso, necesitamos configurar máquinas a medida, en las que sea posible modificar parámetros del hardware, así como poder simular una red de interconexión y analizar el comportamiento de todo el sistema desde la capa de aplicación: es decir, será necesario además, instalar un sistema operativo y una aplicación.

Existen diferentes simuladores que pueden servir para nuestro propósito, pero finalmente para se ha elegido el simulador Simics [MAG02, VIR08a], el cual, aunque presenta ciertas limitaciones que hay que resolver, se ajusta a nuestros requisitos: permite simular arquitecturas a medida, y simular el sistema de comunicaciones, al mismo tiempo que permite ejecutar en dicha arquitectura aplicaciones reales. Por tanto, en este trabajo se utilizará Simics para construir los modelos de simulación necesarios que permitan evaluar el rendimiento del sistema de comunicaciones las mejoras introducidas por la externalización, así como comparar las dos técnicas mencionadas sobre las que existe una gran controversia: *onloading* y *offloading*.

Para poder evaluar ambas técnicas de externalización y determinar cual de ellas puede resultar más beneficiosa y bajo que condiciones, se ha utilizado el simulador Simics. Ya se ha comentado que Simics es un simulador de sistema completo, y puede utilizarse para el estudio de E/S y de arquitecturas de comunicación, pero al ser por defecto un simulador funcional, es necesario incluir en los modelos de simulación, interfaces de modelado del tiempo.

Así, en la presente memoria, se analiza y aplica un método para la simulación y evaluación de los sistemas de comunicaciones bajo diferentes condiciones tando de la red como de carga de trabajo del servidor, y se muestran las ventajas de la técnica utilizada para la externalización de los protocolos de comunicación así como un modelo teórico que permita evaluar y predecir las mejoras que proporcionará cada técnica en unas determinadas condiciones.

Con el fin de evaluar la mejora introducida por la externalización de protocolos, en base a diferentes parámetros, el modelo LAWS proporciona un punto de vista que permite organizar la exploración del espacio de parámetros de diseño de la externalización. No obstante, los resultados obtenidos muestran que debe mejorarse dicho modelo para ofrecer una estimación más precisa de las prestaciones, y por tanto de la mejora introducida por un sistema en el que se han externalizado las comunicaciones, ya sea mediante *offloading* o mediante *onloading*. Para ello, en esta memoria también se han introducido modificaciones en el modelo LAWS original, dando lugar a un nuevo modelo teórico que proporciona un mejor ajuste a los resultados experimentales.

Organización de la memoria

Esta memoria se ha organizado en los capítulos que se resumen a continuación:

Capítulo 1: Introducción. En este capítulo se realiza una introducción la problema de las prestaciones en redes de computadores, que se abordará en los capítulos siguientes. Se describen las posibles alternativas existentes para evitar los cuellos de botella que pueden aparecer en el camino de comunicación soluciones y concretamente, se analiza la mejora de las prestaciones mediante la externalización de los protocolos de comunicación, centrándose en la externalización del protocolo TCP. Para poder evaluar la mejora introducida por la externalización, presenta la técnica de simulación de sistema completo, con sus ventajas e inconvenientes, así como el simulador Simics, que se utilizará con tal propósito. Finalmente, se menciona el modelo teórico LAWS, que se tratará en esta memoria y se utilizará para predecir la mejora en las prestaciones introducida por la externalización.

Capítulo 2: Simulación de sistema completo de las arquitecturas de comunicación. Se realiza una introducción a los diferentes simuladores disponibles actualmente para nuestro propósito y se justifica la utilización de Simics. A continuación se describe brevemente el funcionamiento de Simics, sus ventajas y limitaciones y la forma en que se ha utilizado así como las modificaciones que han sido necesarias para la realización del trabajo de simulación. Además, se describe de forma somera como se ha

implementado el modelo de simulación propuesto, que ha sido utilizado para la evaluación de la externalización y que se detalla en el capítulo 3.

Capítulo 3: Modelo de simulación de la externalización de protocolos de comunicación. En este capítulo se describe el uso de la externalización de protocolos como alternativa para mejorar las prestaciones del sistema de comunicaciones. Se describen diferentes alternativas a la hora de realizar la externalización en cuanto a la arquitectura del sistema de comunicaciones y a la localización de la interfaz de red dentro del sistema. Además, se muestra como se ha utilizado Simics para la evaluación de la externalización de protocolos así como los modelos de simulación propuestos para este fin, tanto en la capa hardware como en la distribución del software en el sistema y el proceso de transmisión y recepción de un paquete. Finalmente se realizan propuestas para la mejora de las prestaciones del sistema de comunicaciones .

Capítulo 4: Estudio experimental de prestaciones. Una vez presentado el simulador que se va a utilizar (Capítulo 2) así como los modelos de simulación, se presentan en este capítulo los resultados de los diferentes experimentos realizados para evaluar las prestaciones ofrecidas por las alternativas de externalización propuestas. Además, se realiza un estudio comparativo de las prestaciones ofrecidas por la externalización mediante *offloading* y *onloading*, donde se muestra como cada técnica puede resultar beneficiosa en ciertos casos. Finalmente, se realiza un estudio de ambas técnicas utilizando el modelo LAWS, que nos permite estudiar el comportamiento de la externalización frente a diferentes niveles de sobrecarga en el host, debida a la aplicación y a los procesos de comunicaciones.

Capítulo 5 : Modelo híbrido de externalización. En el Capítulo 4 se ha visto como la mejora proporcionada por cada alternativa de externalización depende de la situación concreta. En este capítulo se propone una interfaz de red que incorpora las ventajas de la externalización mediante *offloading* y mediante *onloading* y se realiza una evaluación de las prestaciones que proporciona.

Capítulo 6: Conclusiones y resumen de principales aportaciones. En este capítulo, se describen las conclusiones que se derivan de nuestra investigación, así como las principales aportaciones realizadas para la evaluación de sistemas de comunicaciones

externalizados reales. Se plantea además el trabajo futuro que puede surgir a partir de los resultados descritos en la memoria.

La memoria concluye con las referencias y dos apéndices dedicados respectivamente a la configuración de Simics y al modelado del tiempo en el mismo.

Capítulo 1

Diseño de la interfaz de red

Por un lado, la necesidad de disponer de prestaciones de comunicación elevadas entre los nodos de los computadores paralelos, particularmente en los clusters, con interfaces de red y redes estándar, como Ethernet, que no han sido pensadas para supercomputación, ha promovido la investigación en interfaces eficientes. Junto a esa circunstancia, la disparidad entre el ritmo de crecimiento del ancho de banda de los enlaces y las prestaciones de los microprocesadores también ha contribuido al interés en la mejora de la arquitectura de comunicación de los servidores. En este capítulo se introducen los problemas que deben resolver las arquitecturas de comunicación y se analiza el trabajo de investigación realizado hasta el momento para mejorar las prestaciones de la interfaz de red. Así, una vez descrito el camino de comunicación y tras introducir los aspectos que limitan las prestaciones de la interfaz de red se analizan las características y la implementación de la familia de protocolos TCP/IP, que son los protocolos más utilizados tanto en los clusters de computadores como para la comunicación a través de Internet. Después se describen los trabajos de investigación más relevantes realizados para mejorar la eficiencia de los protocolos y se analizan las líneas de trabajo en protocolos ligeros e interfaces de red de usuario. Otra de las alternativas a tener en cuenta en el trabajo de optimización de la interfaz de red pasa por el aprovechamiento del aumento del número de procesadores disponibles en cada nodo, tanto por la incorporación de procesadores en las trajetas de red como por la utilización de procesadores multinúcleo o el uso de servidores SMP. Así, se analizan las alternativas asociadas a la externalización de la interfaz de red y se describe un modelo para ayudar al análisis de las mejoras que cabe esperar de estas propuestas. Finalmente, el capítulo termina con las consideraciones relativas a la necesidad de disponer de

herramientas de simulación adecuadas para hacer posible la investigación en el área del diseño de arquitecturas de comunicación eficientes.

1.1 Redes de comunicación en sistemas de altas prestaciones

La evolución de los computadores está determinada por un mercado que demanda aplicaciones que cada vez necesitan de más recursos para ser ejecutadas eficientemente [VAJ01]. Por otro lado, las mejoras tecnológicas hacen que sea posible disponer de procesadores de altas prestaciones a un precio asequible poniendo al alcance de muchas instituciones los servidores de altas prestaciones.

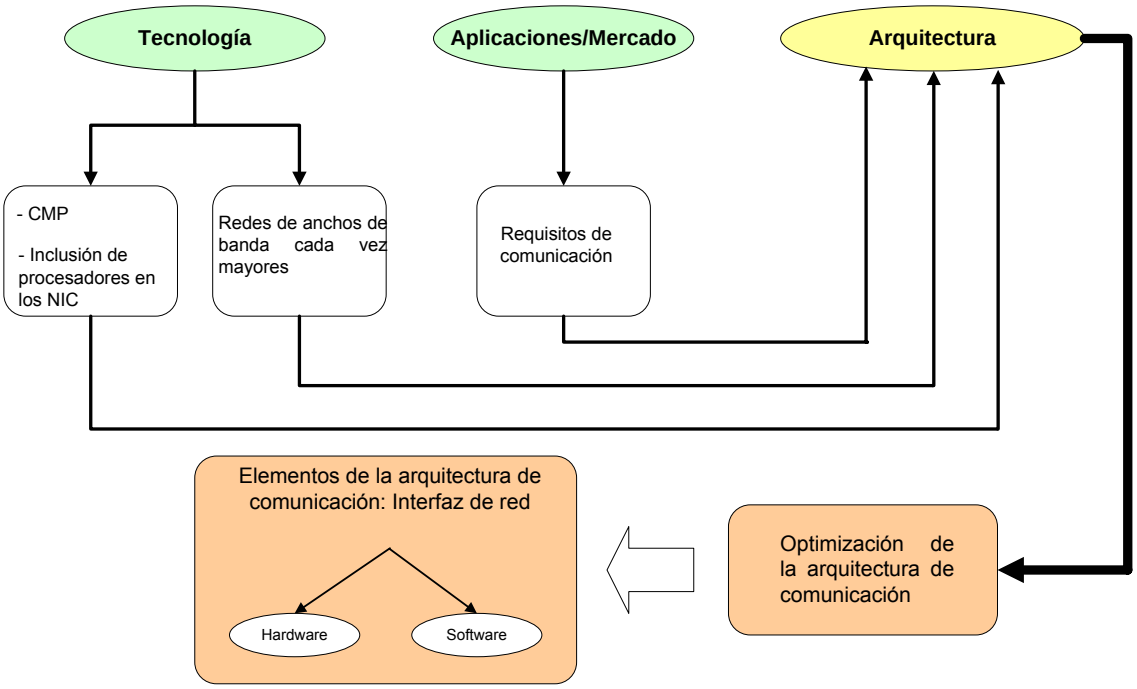


Figura 1.1. Interacción Tecnología/Aplicaciones-Mercado/Arquitectura de comunicación

En dichos servidores de altas prestaciones, la red de interconexión entre los nodos es un elemento determinante en las prestaciones del sistema y por otro lado, el mercado que demanda procesadores de altas prestaciones para la ejecución de ciertas aplicaciones, a la vez demanda una comunicación entre nodos cada vez más rápida y por tanto, redes con anchos de banda cada vez mayores.

La mejora de la tecnología de redes hace posible disponer de enlaces de red con anchos de banda elevados (Gigabit Ethernet [EST99], 10-Gigabit Ethernet [EST06], 40-Gigabit [EST06]), con lo que cabe esperar un salto de un orden de magnitud en las prestaciones de comunicación y mejoras en las plataformas altas prestaciones, como son

los clusters de computadores, u otras plataformas que requieran de una red con un ancho de banda efectivo suficientemente elevado. Por otro lado, el continuo incremento en los anchos de banda de las redes y el interés en los sistemas de almacenamiento basados en IP, así como la tendencia a reemplazar las redes específicas de almacenamiento [PET05] (SCSI, FiberChannel, etc.) por redes como Gigabit Ethernet o multigigabit Ethernet han alimentado el debate sobre las prestaciones ofrecidas por el protocolo TCP/IP y las técnicas para mejorarlas.

La implementación de protocolos tales como TCP/IP en redes de altas prestaciones plantea un problema considerable de comunicación, debido a que el aprovechamiento del ancho de banda de estas redes requiere una gran cantidad de recursos de CPU, y el cuello de botella de las comunicaciones pasa de la red a los nodos. Por tanto, las prestaciones de la interfaz de red son determinantes en el rendimiento de los servidores. Dicho interfaz de red, proporciona la conectividad del sistema con otros sistemas (computadores) y puede estar integrado en la arquitectura de tres formas diferentes:

a) Como una tarjeta de expansión: conectada a un bus que esté implementado en el sistema, tal como ISA, PCI ó PCI-express, o recientemente, a buses de alta velocidad como *HyperTransport*.
b) Como parte del *chipset*: pudiendo estar internamente conectada o no a un bus PCI del sistema o directamente conectada al puente norte.
c) Como parte del procesador y del *chipset*: en las arquitecturas más recientes, en las que se incluyen procesadores con más de un núcleo, uno de estos núcleos puede implementar un procesador de red que se encargue del procesamiento de los protocolos de comunicación y de la gestión de las comunicaciones. En este caso, la capa física (Ethernet) estará implementada por otros elementos del *chipset*.

En el caso de redes no Ethernet, la interfaz de red suele ser una tarjeta conectada a un zócalo del bus PCI o PCI-express.

Como puede verse en la Figura 1.1, la tecnología y las aplicaciones que demanda el mercado, determinan la arquitectura de los computadores actuales. La necesidad de anchos de banda efectivos elevados, hace necesaria la optimización de la arquitectura de comunicación y por tanto de los elementos que la componen, como la interfaz de red.

En las secciones 1.2 y 1.3, se presentan las diferentes opciones que permiten mejorar las prestaciones del sistema de comunicaciones, las cuales se basan en la optimización de la parte software (protocolos, soporte del sistema operativo a las comunicaciones, gestión de las conexiones, etc.) o de la parte hardware (interfaz de red) y como pueden ser implementadas.

1.2 El camino de comunicación

Tradicionalmente, el procesamiento de los protocolos de comunicación así como la gestión de las comunicaciones ha recaído en el procesador principal del sistema, el mismo procesador encargado de la ejecución de las aplicaciones y del sistema operativo. Esta implementación, que puede resultar casi inocua para el sistema cuando se utilizan redes con anchos de banda del orden de Mbps, genera una sobrecarga considerable en el procesador principal cuando se utilizan redes de altas prestaciones [JEF01, DIA03, GIU01, PET94, MAR02, BIN05, BHO98], sobrecarga que puede llegar a no ser asumible por el mismo, provocando incluso el colapso del sistema limitando por tanto el rendimiento global.

En el camino de comunicación existen diferentes elementos que pueden suponer un cuello de botella a las prestaciones del sistema. En el emisor, los datos son estructurados convenientemente de acuerdo al protocolo de comunicación utilizado, y dichas estructuras son transferidas al interfaz de red (NIC) desde la memoria de usuario, normalmente mediante transferencias DMA. En el receptor se realiza el proceso contrario: una vez que los datos llegan al interfaz de red (NIC) desde la red, este realiza una transferencia DMA de los datos recibidos hasta la memoria principal. Una vez que dicha transferencia DMA se ha completado, la NIC genera una interrupción indicando al procesador principal la disponibilidad de los datos recibidos en memoria, para que éste pueda procesarlos. Por otro lado, las interfaces de red suelen conectarse a un bus de E/S estándar (como PCI, PCI-X o *Hypertransport*), con el fin que estos sean independientes de la arquitectura del sistema. Las transferencias entre la memoria principal y la NIC se realizan a través de un bus de E/S que dispondrá de cierto ancho de banda y las interrupciones generadas por la interfaz de red requieren de cierto tiempo de CPU para ser atendidas, puede suponer una limitación de las prestaciones en el caso de redes con anchos banda del orden de gigabits/s.

Si se analiza el camino de comunicación de forma analítica, el tiempo de comunicación para un paquete de n bits puede expresarse como:

$$T_{comunicación} = O_s + O_r + L + \frac{n}{BW_{red}} \tag{1.1}$$

donde O_s representa la sobrecarga en el emisor (tiempo que tarda el emisor en preparar el paquete para ser enviado y en transferir los datos desde la memoria principal hasta la red) y O_r la sobrecarga en el receptor (tiempo invertido en el receptor para transferir los datos desde la red hasta la memoria principal donde sean accesibles por la aplicación). El tercer término de la expresión (1.1), L, hace referencia al tiempo de vuelo en la red (latencia) y cuarto término, n/BW_{red}, al tiempo asociado a la transferencia de n bits en una red con ancho de banda BW_{red}. Por tanto, la suma de estos dos términos $L+ n/BW_{red}$, es el tiempo que tarda la red en transmitir los n bits que conforman el paquete.

Teniendo en cuenta que tanto en el emisor como en el receptor, las transferencias entre la NIC y la memoria principal se realizan a través de buses de E/S que tienen un ancho de banda determinado, la suma O_s+O_r puede expresarse como el tiempo de transferencia entre la memoria y la NIC más el tiempo asociado a la gestión de dichas transferencias (que no se solape con ellas). De esta forma, la suma $O_s+ O_r$ puede expresarse como:

$$O_s + O_r = M \frac{n}{BW_{memoria}} + N \frac{n}{BW_{E/S}} + \Delta t \tag{1.2}$$

donde M y N representan el número de veces que se atraviesan los buses de memoria y de E/S respectivamente y Δt el tiempo empleado en el procesamiento de los protocolos de comunicaciones, en los cambios de contexto, gestión de interrupciones, etc. Y que no se solapa con los tiempos de transferencia de información. Si en (1.1) se desprecia L, teniendo en cuenta la expresión (1.2), el ancho de banda efectivo (ancho de banda de la red que consiguen aprovechar las aplicaciones) puede expresarse en términos del ancho de banda de la red como:

$$BW_{efectivo} = \frac{BW_{red}}{1 + M \frac{BW_{red}}{BW_{memoria}} + N \frac{BW_{red}}{BW_{E/S}} + \frac{\Delta t \times BW_{red}}{n}} \tag{1.3}$$

De la expresión (1.3) se pueden extraer las siguientes conclusiones:

a) El ancho de banda de los buses de E/S y de memoria así como el tiempo asociado al procesamiento de los protocolos de comunicación y a la gestión de las transferencias de datos hacen que el ancho de banda aprovechado por la aplicaciones sea menor del ancho de banda de la red.

b) Cuanto menor sea el tamaño del paquete *n*, mayor es la influencia del tiempo asociado al procesamiento de protocolos y a la gestión de las transferencias, Δt. De hecho, éste término será determinante en la latencia de mensajes cortos.

c) Si los anchos de banda de la memoria $BW_{memoria}$ del bus de E/S $BW_{E/S}$ son muy elevados frente a $MxBW_{red}$ y $NxBW_{red}$ respectivamente, el término dominante en el denominador de la expresión (1.3) y por tanto en la reducción del ancho de banda efectivo es $(BW_{red} \; x \; \Delta t)/n$. Por tanto, interesa reducir o solapar el tiempo asociado al procesamiento de protocolos, cambios de contexto, gestión de interrupciones, etc.

d) A medida que el ancho de banda de la red BW_{red} es mayor, es más difícil alcanzar un ancho de banda efectivo $BW_{efectivo}$ próximo a BW_{red}, ya que se necesitarían buses de memoria y de E/S con anchos de banda suficientemente elevados, a la vez que valores suficientemente pequeños de *M* y *N*, y se debe reducir el tiempo Δt o utilizar valores de n muy elevados (paquetes muy grandes).

Como puede verse, en el proceso de transmisión / recepción de datos, interviene una combinación de elementos hardware y software, que serán los que limiten las prestaciones finales de la comunicación. De la expresión 1.3, puede deducirse, que al incrementar el ancho de banda del enlace de red (BW_{red}), como ocurre al pasar de redes Gigabit Ethernet [EST99] a 10-Gigabit Ethernet [EST06], si se mantiene constante el ancho de banda de los buses de memoria y E/S, disminuye el ancho de banda efectivo. Por otro lado, la sobrecarga generada en el nodo debida principalmente a la gestión de las interrupciones, la API de sockets, las múltiples copias de datos (y por tanto, las transferencias en el nodo a través de los buses de E/S), los cambios de contexto y el procesamiento de la pila de protocolos TCP [GAD07, STE94a, FOO03, BIN05a, MAR02] hace que el ancho de banda efectivo sea menor que el de ancho de banda del enlace de red. Por tanto, para poder aprovechar el ancho de banda que proporciona un enlace de red multigigabit, es necesario reducir el tiempo de acceso a memoria, a la vez que optimizar las transferencias de datos en el nodo (buses de E/S) y disminuir la sobrecarga asociada a los procesos de comunicaciones para disponer de un ancho de banda efectivo ($BW_{efectivo}$) del orden del ancho de banda de la red. Debido a las

transferencias de datos necesarias, la ubicación de la NIC dentro del sistema, el número de copias de datos necesarias o el solapamiento entre las transferencias a lo largo del camino de comunicación, afectan a las prestaciones del sistema de comunicaciones.

En la Figura 1.2 se muestran tres alternativas para la ubicación de la NIC y los caminos de comunicación asociados en cada caso. En la Figura 1.2, mientras que las alternativas (c) y (b) se corresponden, respectivamente, con una conexión de la interfaza través de un bus PCI (c) o PCI de alta velocidad (b), la alternativa (a) se corresponde con una implementación más actual, dada la tendencia a incluir más de un núcleo de procesamiento en un mismo chip y al uso de buses de altas prestaciones como *Hypertransport*.

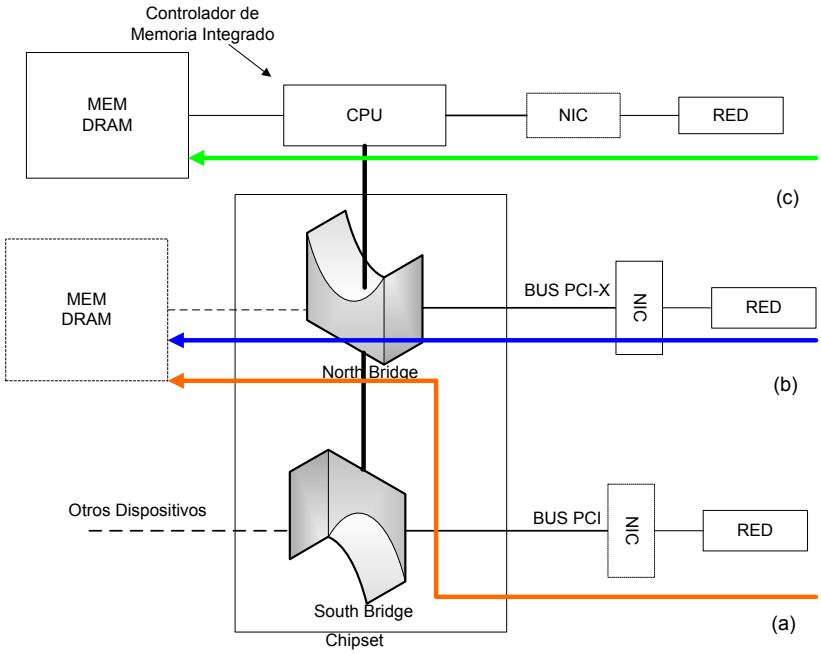


Figura 1.2. Posibles ubicaciones de la interfaz de red y caminos de comunicación

En la década de los 80, la mejora de la tecnología de redes, propició trabajos de investigación con el objetivo de conseguir un mejor aprovechamiento del ancho de banda que ofrecían los enlaces de red y mejorar así las prestaciones de las comunicaciones entre nodos. Dicho trabajo de investigación se centró inicialmente en la optimización de la pila de protocolos TCP, ya que para reducir la sobrecarga generada por el procesamiento de los protocolos y de esa forma incrementar las prestaciones de la red [CLA89]. De hecho, dichas optimizaciones aún se conservan en las pilas TCP/IP de los sistemas operativos actuales. Sin embargo, trabajos como [CLA89] analizan la importancia de otros factores como la sobrecarga generada por las múltiples copias de

datos o por la interacción con el sistema operativo en las prestaciones del sistema de comunicaciones, y concluyen que el procesamiento de la pila TCP no es realmente la fuente principal de sobrecarga.

Este trabajo de investigación iniciado en los años 80, ha continuado hasta la actualidad, motivado por el avance en la tecnología de redes, que ha conseguido pasar de los enlaces del orden de varios Mbps de que se disponía en los años 80 a los enlaces multigigabit de que se disponen en la actualidad. Con el avance en la tecnología de redes, los problemas asociados al aprovechamiento de las prestaciones ofrecidas por los enlaces de red han ido cambiando así como la medida en que afectan a las prestaciones las fuentes de sobrecarga citadas en [CLA89]. Trabajos posteriores como [STE94a, CHA01, MAR02] muestran como la principal fuente de sobrecarga se encuentra en los buses de E/S, en el sistema de memoria y en la interacción con el sistema operativo, así como en la gestión de interrupciones.

Por esta razón, se han propuesto diferentes alternativas para reducir la sobrecarga de comunicación, que van desde la optimización de las pilas TCP/IP [CLA89, JAC92], la utilización de protocolos ligeros como CLIC [DIA03], GAMMA [CIA01] o protocolos a nivel de usuario como el estándar VIA [VID01], a la introducción de mejoras hardware, siempre con el fin de reducir la sobrecarga generada por los procesos de comunicaciones.

Otra propuesta para reducir la *sobrecarga* de comunicación, consiste en traspasar el procesamiento de los protocolos a la tarjeta de red (NIC) o en general, evitar que la CPU que ejecuta la aplicación se encargue también de los procesos de comunicación, es decir, se propone la *externalización de los protocolos de comunicación* (*offloading*). De esta forma, se libera a la CPU del sistema de este trabajo, dejando más ciclos disponibles para la aplicación. Además, de esta forma, la NIC puede interactuar con la red sin la intervención de la CPU que ejecuta la aplicación, disminuyendo el número de interrupciones, y la latencia del protocolo para mensajes cortos (como ACKs) ya que éstos no tendrán que ir hasta la memoria principal del sistema, atravesando el bus de E/S.

Sin embargo, frente a los trabajos como [WES04, REG04A, KIM06, DEL06] que muestran las ventajas aportadas por la externalización de protocolos, existen otros estudios como [MOG03] que proporcionan resultados que ponen en duda dichas ventajas. Particularmente, en el caso de TCP/IP, los supuestos beneficios de externalización han sido muy discutidos, argumentándose que el coste de CPU para

procesar la pila TCP/IP es pequeño comparado con el coste de las transferencias de datos y de la interfaz entre la pila TCP/IP y el sistema operativo, incluyendo las interrupciones.

Modelos como LAWS [SHI03] y EMO [GIL05] se han propuesto como marco de interpretación de los resultados experimentales y para orientar la exploración del espacio de diseño de las tecnologías de externalización. No obstante, se trata de modelos aproximados que no pueden incluir detalles característicos de aplicaciones específicas y con patrones de comunicación no regulares que generan complejas interacciones entre los distintos elementos hardware. Así, dado el gran número de parámetros y la compleja casuística a tener en cuenta a la hora de evaluar las posibles mejoras aportadas por la externalización, es imprescindible disponer de las herramientas y técnicas adecuadas con las que analizar las diferentes arquitecturas de comunicación propuestas. Además, como se ha indicado, la externalización afecta de manera diferente a diferentes aplicaciones, dependiendo del uso que estas hagan de los recursos del sistema.

En las secciones siguientes, se introduce la pila de protocolos TCP/IP, concretamente la utilizada en los sistemas operativos Linux, así como la interacción con el sistema operativo y la aplicación, las diferentes capas del mismo. Se muestran sus principales caracteristicas asi como la integración de la pila de protocolos en el sistema operativo, para poner de manifiesto la sobrecarga que supone en un nodo el utilizar TCP/IP como protocolo de comunicaciones y las posibles optimizaciones que puedan reducir dicha sobrecarga e incrementar las prestaciones del sistema de comunicaciones.

1.3 La pila de protocolos TCP/IP en Linux

Como ya se ha comentado, el incremento del ancho de banda de los enlaces de red junto con las necesidades del mercado que demandan aplicaciones y servicios que utilizan anchos de banda cada vez mayores, y que frecuentemente requieren una garantía de un cierto nivel de *calidad de servicio* (*QoS*) [GOP98, BAS05, CHU00], han generado una gran actividad investigadora en el campo de los protocolos de red. Además, muchos de estos servicios se proporcionan a través de Internet, red en la que el protocolo dominante en la actualidad es TCP [CLA01]. Por esta razón, resulta muy importante tratar de mejorar las prestaciones que se pueden ofrecer utilizando dicho protocolo. De hecho, existen numerosos trabajos que estudian las prestaciones y la

sobrecarga generada por el protocolo TCP/IP [CLA89, DAV89, STE94a, BIN05, WU06] y otros trabajos que tratan de mejorar las prestaciones que ofrece el protocolo TCP tomando como base la implementación tradicional del mismo en *Linux*. En este sentido, en [BHO98] se propone el uso de protocolos a nivel de usuario y en otros como [GOP98, PRA04] se proponen implementaciones experimentales de pilas de protocolos TCP/IP a nivel de usuario para Linux, de forma que la aplicación pueda interactuar directamente con el protocolo sin intervención del sistema operativo. Por otro lado, en [GIL02a, GIL02b] en lugar de puentear el sistema operativo, como pretenden los protocolos a nivel de usuario, se propone hacer un uso eficiente del mismo por parte de los procesos de comunicaciones. En esta misma línea, en [PRY98] se propone el protocolo BIP, diseñado para susituir a TCP/IP en redes de altas prestaciones. Otros trabajos como, [STE98] proponen el uso de técnicas de cero copias para mejorar las prestaciones, y otros como [GIL02a], donde se analizan los efectos de diferentes técnicas para optimizar la comunicación en redes Gigabit, como cero copias, externalización del cálculo de la suma de comprobación, tramas jumbo y coalescencia de interrupciones.

Por otro lado, el avance de la tecnología de los procesadores de propósito general presentes en los actuales servidores, y la diferencia de velocidad entre estos y la memoria, hace que el tiempo de procesamiento de cada capa del protocolo TCP/IP esté cambiando a la vez que desplaza el cuello de botella a otras partes del sistema como los buses de E/S o la memoria [BIN05b, CHA01, MAR02, STE94a]. De hecho, mas allá de la propia implementación del protocolo, habrá que tener en cuenta aspectos determinantes del rendimiento del sistema de comunicaciones como la gestión de los *buffers*, la inicialización de la pila de protocolos, los procedimientos utilizados para el envío y recepción de los datos y el camino de comunicación (sección 1.2) o la integración de la pila de protocolos en el sistema operativo.

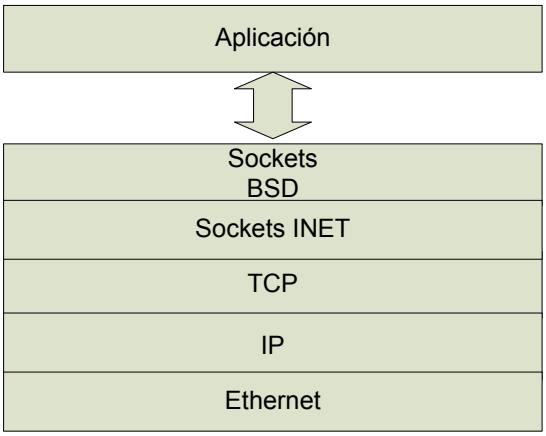


Figura 1.3. Estructura de la pila de protocolos en Linux

En la Figura 1.3 puede verse la estructura de la pila de protocolos TCP/IP en Linux. En la capa más alta, los *sockets* actúan de interfaz entre los protocolos de red, el sistema operativo y las aplicaciones. Los *sockets* BSD pueden proporcionar (y de hecho así ocurre en Linux) la interfaz necesaria con otros protocolos como X.25. Esta capa proporciona por tanto un interfaz con la aplicación, transparente al protocolo de red utilizado. Los *sockets* BSD de Linux soportan diferentes tipos de comunicación como *secuencia*, *datagramas* o *paquetes*. Por debajo de los *sockets* BSD se encuentran los *sockets* INET, que proporcionan la interfaz con la familia de protocolos de Internet y por tanto con la pila de protocolos TCP/IP.

Aquellas aplicaciones que utilizan *sockets* para las comunicaciones, utilizan un modelo cliente-servidor, es decir, un servidor que proporciona ciertos servicios como puede ser una base de datos o un servicio *web*, y un cliente que hace uso de estos servicios. La capa de *sockets* es responsable de identificar el tipo de protocolo y redirigir el control a la función apropiada del protocolo específico, así como de llamar a las rutinas de la capa de transporte apropiadas.

Por debajo de los *sockets* se encuentran las capas TCP e IP. El protocolo TCP está orientado a conexión y proporciona una comunicación fiable entre dos extremos. Por otro lado, IP es un protocolo de la capa de transporte, utilizado también por otros protocolos además de TCP, como es el caso de UDP (UDP/IP).

TCP proporciona la transmisión fiable de los datos entre dos aplicaciones, garantizando que no se pierdan datos ni se dupliquen en el extremo receptor. Otros protocolos como UDP ofrecen servicio de datagramas y no disponen de mecanismos para garantizar una transmisión segura (no se tiene seguridad de que los datos hayan llegado al destino) ni que los datos no se dupliquen en el destino.

En el transmisor, cuando la rutina de envío TCP se ha invocado desde la capa de *sockets*, dicha rutina debe esperar a que se establezca una conexión con el otro extremo, ya que como se ha comentado, TCP está orientado a conexión y por tanto no puede realizar una transferencia de datos sin que exista una conexión activa. Una vez que la conexión se ha establecido, se copian los datos a transmitir desde el espacio de usuario a una estructura *sk_buff* en el espacio del núcleo del sistema operativo para ser enviados. Posteriormente, la capa TCP añade la cabecera y envía el paquete a la capa IP, la cual transporta los paquetes TCP, siendo la responsable en cada extremo de la transmisión y recepción de paquetes IP.

Como ya se ha comentado, TCP/IP es un protocolo basado en capas, es decir, cada capa usa los servicios de la capa que tiene por debajo, añadiendo los datos propios de cada capa (como son las cabeceras) de forma que los datos se van encapsulando en cada capa, e incluyendo un identificador del protocolo de la capa inferior, en el caso de que esta no sea única. Este es el caso de la capa IP, que puede tener por debajo a TCP o UDP, por lo que en esta capa se incluye al paquete un identificador del protocolo que se ha encapsulado. Este identificador es un byte de la cabecera IP, como se muestra en la Figura 1.4.

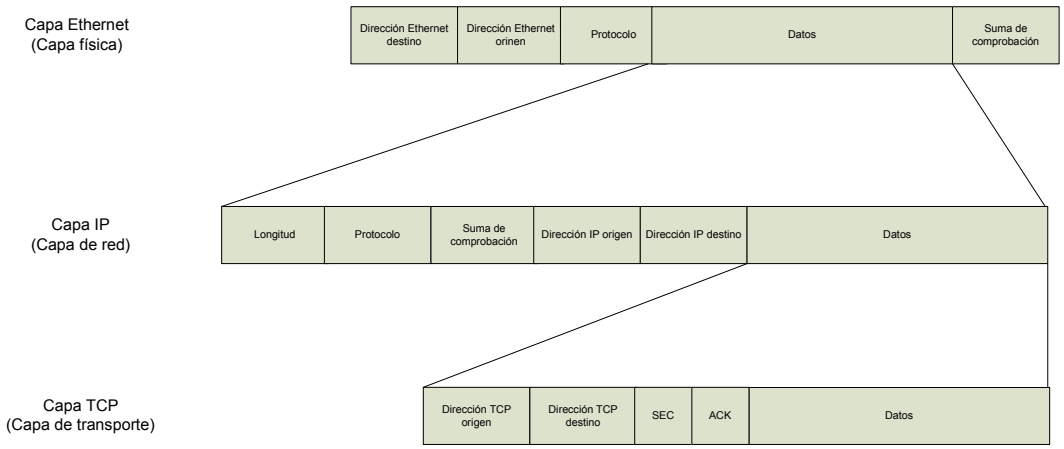


Figura 1.4. Capas TCP/IP y Ethernet

La capa IP recibe el paquete de la capa TCP, y le añade la correspondiente cabecera. Las rutinas de la capa IP incluyen los mecanismos de enrutamiento y envío, utilizando el FIB (*Forwarding Information Base*). Además, en esta capa, el paquete se fragmenta si es necesario.

Por debajo de la capa IP, se encuentra la capa de enlace de datos. Esta capa es la responsable de diferentes funciones además de la gestión de los paquetes sobre la capa física. Es esta capa la responsable de la política de colas, así como de las funciones de conformado de tráfico y de copiar los datos a los *buffers* de la interfaz de red para ser enviados. Si la transmisión se ha realizado con éxito, se libera el espacio de memoria asignado a la estructura *sk_buff* y si no, el paquete se encola de nuevo para volver a ser transmitido.

En la Figura 1.4 se ha considerado que Ethernet constituye la capa física, aunque el protocolo IP puede utilizar otros medios físicos (como PPP o SLIP), como se muestra en la Figura 1.5. En cualquier caso, la capa de enlace de datos añadirá también su propia cabecera. En el caso de Ethernet, se añade una dirección única formada por 6 bytes. Por tanto, esta será la única dirección visible desde la capa física y por tanto, la única dirección admisible por la interfaz de red. Sin embargo, normalmente cuando se transmiten datos a un receptor, la aplicación utiliza la dirección IP de este y no la dirección Ethernet.

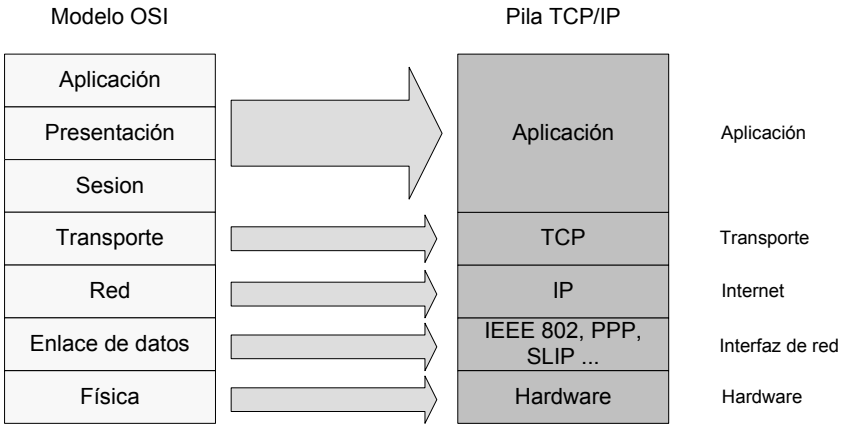


Figura 1.5. Correspondencia de capas OSI - TCP/IP

Es necesario, por tanto, un mecanismo de traducción entre la IP a la que se quieren enviar los datos y la dirección Ethernet del receptor. Este mecanismo está implementado en Linux mediante el protocolo ARP (*Address Resolution Protocol*), que permite la traducción de direcciones IP en direcciones físicas, que no tienen por que ser solo Ethernet, sino que el protocolo ARP puede también resolver otras direcciones físicas. En el caso de los enrutadores, que necesitan realizar la función inversa, es decir, traducir direcciones físicas en IP, utilizan el protocolo RARP (*Reverse ARP*).

En las secciones siguientes, se presentan determinados aspectos de la pila de protocolos TCP/IP que constituyen una fuente de sobrecarga en el sistema, tales como

la integración en el sistema operativo y la interacción con el mismo, así como la necesidad del uso de colas en TCP/IP y por tanto, de las correspondientes políticas de gestión de las mismas para conseguir un uso eficiente y optimizar las prestaciones. Estos aspectos han de ser tenidos en cuenta para optimizar las prestaciones ofrecidas por un sistema de comunicaciones que utilice TCP/IP.

1.3.1 Integración en el sistema operativo

En las versiones actuales de Linux, la pila de protocolos TCP/IP es parte del núcleo del sistema operativo, en el que se ha realizado una implementación de acuerdo a la estructura de cinco capas que se muestra en la Figura 1.3.

En la manipulación de los datos que se van a recibir o transmitir, se utilizan dos estructuras críticas. La primera de estas estructuras es sk_b*uff*, la cual, almacena un paquete recibido o a transmitir, conteniendo información tanto de gestión que es utilizada por todas las capas de red para, como de gestión como son las cabeceras. Otra estructura crítica es la llamada *net_device*, que contiene información acerca del hardware (interfaz de red) y su configuración.

La estructura *sk_buff* es probablemente la más importante en el código de red de Linux, y utiliza un espacio de memoria contiguo par almacenar un paquete completo. Linux por tanto, calcula el espacio necesario a reservar para la estructura *sk_buff*, incluyendo no solo la información de usuario sino también datos de gestión tales como las cabeceras. De esta forma, se tendrán diferentes estructuras *sk_buff* que contendrán los paquetes recibidos o a transmitir. En las siguientes secciones se comenta la importancia de procesos como la gestión de colas o la planificación de los procesos que implementan los protocolos de red en las prestaciones del sistema de comunicaciones.

1.3.2 Colas en TCP/IP

Como ya se ha comentado en la Sección 1.3, TCP/IP utiliza una estructura de 5 capas. Además, TCP/IP es un protocolo de tipo *store-and-forward*, por lo que será necesario disponer de colas en determinados niveles de la pila de protocolos. De hecho, los mecanismos de gestión de colas (políticas de gestión de colas) afectarán en gran medida a las prestaciones del sistema de comunicaciones y a la sobrecarga generada en

el sistema [CHU00]. En Linux, existen dos colas en el lado transmisor, una de ellas en la capa TCP y otra para la interfaz de red (controlador de la interfaz de red). En el receptor, existe una cola en la capa TCP y otra en la capa IP. El tamaño de las colas en la capa TCP se basa en el número de bytes y en la capa IP en el número de paquetes. De esta forma, en el lado receptor, las colas TCP proporcionan un almacenamiento temporal de los datos y un control de flujo, así como en el lado transmisor, un mecanismo para almacenar los paquetes generados por la aplicación, hasta que estos puedan ser enviados a la red.

La política de gestión de colas utilizada, determinará el flujo de recepción y envío de paquetes, afectando al rendimiento del sistema de comunicaciones.

1.3.3 Interacción con el sistema operativo

La interacción de los protocolos de red con el sistema operativo es otro factor determinante de las prestaciones finales del sistema de comunicaciones. Esto se debe a que el sistema operativo proporciona las funciones de gestión de memoria, gestión de interrupciones y planificación. Concretamente, la planificación de los protocolos de red es determinante en la eficiencia del protocolo y en la capacidad de respuesta en tiempo real del mismo.

Una vez que se recibe un paquete en la interfaz de red, este genera una interrupción en el procesador principal del sistema. La rutina de servicio de esta interrupción copia el paquete desde los *buffers* de la interfaz de red a la memoria principal (normalmente, utilizando transferencias DMA) y procesa el paquete según los protocolos que se estén utilizando. Sin embargo, la copia de los datos desde los *buffers* de la interfaz de red a la memoria principal y el procesamiento del paquete se llevan a cabo en dos fases. Por un lado, la copia de los datos desde la interfaz de red se lleva a cabo en el código del controlador de la interfaz de red (*driver*). La planificación de la ejecución del código del controlador de la interfaz de red se realiza mediante los llamados *bottom-half-handlers*, gestores de los *bottom-halves*. [BEN05, BOV05]. El origen de los *bottom-halves* está en la reducción de los bloqueos de CPU asociados al procesamiento de las interrupciones ya que, durante la interrupción, la CPU no puede ejecutar otras tareas. Los *bottom-halves* permiten colocar la ejecución de los controladores de dispositivos u otras tareas del núcleo en una cola para ser ejecutadas con posterioridad. Por otro lado, Linux utiliza las llamadas *softirq* [BEN05, BOV05] para realizar el procesamiento de los protocolos de red. Las *softirq* tienen el mismo

objetivo de los *bottom-halves*, es decir, detener el procesador el mínimo tiempo posible debido a la ejecución de rutinas de servicio a interrupciones. Las *softirq* implementan en Linux la gestión de tareas que se consideran prorrogables o no críticas. Una característica interesante de las *softirq* es la posibilidad de que se ejecuten varias concurrentemente en varios procesadores. Sin embargo, no se permite la ejecución de varias *softirq* de forma concurrente en un mismo procesador.

En este contexto, se plantea el problema del posible bloqueo en el procesamiento de los paquetes entrantes. En el caso de recibir paquetes a una velocidad suficientemente alta, el núcleo del sistema operativo, debido a los mecanismos de planificación de tareas comentados anteriormente, nunca llegaría a procesar ningún paquete, debido a que se están generando interrupciones de forma demasiado rápida. Este fenómeno se conoce con el nombre de *interrupt livelock* [BEN05]. Es decir, el procesador dedica el 100% de los ciclos al procesamiento de las interrupciones hardware. Para solucionar este problema surge el llamado NAPI (New API) [SAL01, BEN05, BOV05], incluido en la mayoría de los controladores de red en las versiones de los núcleos de Linux a partir de la 2.4.20. Básicamente, NAPI se basa en el uso de una estrategia mixta de interrupciones y sondeo (*polling*), de forma que los paquetes se van recogiendo por sondeo, y se genera una interrupción cada cierto número de paquetes recibidos. Por esto, esta técnica es también conocida como coalescencia de interrupciones adaptativa. NAPI incluye además técnicas para prevenir problemas causados por una inundación de paquetes, descartándolos en el caso de que el anillo DMA esté lleno.

Como se ha podido ver en esta y en las secciones anteriores, no sólo el protocolo es responsable de las prestaciones del sistema de comunicaciones sino que la integración e interacción con el sistema operativo y con el hardware de red afectan a la sobrecarga de comunicación de forma decisiva. En cualquier caso, y como se comentó en la Sección 1.2, el uso de los llamados protocolos ligeros, son una de las alternativas para reducir la sobrecarga generada por los procesos de comunicaciones en redes de altas prestaciones. En la siguiente sección se tratan las posibles optimizaciones de un sistema de comunicaciones TCP/IP y en la sección 1.5, el uso de protocolos ligeros, como GAMMA [CIA01] y CLIC [DIA03]. Con el mismo objetivo que los protocolos ligeros surgen las interfaces de comunicación a nivel de usuario, para las cuales incluso se ha propuesto el estándar VIA que se introduce en la sección 1.6.

1.4 Optimización del subsistema de comunicaciones TCP/IP

A lo largo de los años, los avances en la tecnología de redes han generado un amplio trabajo de investigación, destinado a optimizar el procesamiento de comunicaciones en el nodo. La investigación en procedimientos y técnicas para mejorar las prestaciones de comunicación tiene dos vertientes complementarias. La primera, estudia la forma de reducir la *sobrecarga* asociada al procesamiento de los protocolos de comunicación, bien optimizando la pila TCP/IP, bien proponiendo nuevos protocolos que optimicen el soporte del sistema operativo a las comunicaciones como es el caso de los protocolos GAMMA [CIA01] ó CLIC [DIA03], bien proporcionando interfaces de red a nivel de usuario como el estándar VIA (*Virtual Interface Architecture*) [VID01]. La otra vertiente trata de mejorar el hardware de la interfaz de red con el fin de mejorar las prestaciones del sistema de comunicaciones.

En las secciones anteriores, se ha puesto de manifiesto la influencia en las prestaciones del operativo, la gestión de los *buffers* y la planificación de procesos, así como la interfaz de red y su controlador. En las diferentes etapas del procesamiento de un paquete, se genera un tráfico en los buses de E/S debido a las copias de datos entre los *buffers* de la interfaz de red y la memoria principal y una sobrecarga en la CPU debida a procesos como la asignación de espacio de memoria para alojar los *buffers* y su posterior liberación, el cálculo del códigos de error el procesamiento de los distintos protocolos (por ejemplo la pila TCP/IP). Sin embargo, aunque el uso de procesadores rápidos puede mejorar en cierta medida las prestaciones, no es una solución cuando se tienen enlaces de red con anchos de banda del orden de gigabits/s [MAR02], debido a que el procesador puede estar la mayor parte del tiempo esperando a que se completen transferencias de memoria. Disponer de interfaces de red que puedan realizar ciertos cálculos sencillos (por ejemplo, el cálculo de la suma de comprobación de un paquete para rechazarlo desde la interfaz de red si contiene errores) así como de buses de E/S con anchos de banda elevados, puede mejorar considerablemente las prestaciones del sistema de comunicaciones [STE94a, STE98, CHA01, GAD07]. Por otro lado, se ha comentado la importancia de la integración de la interfaz de red en el sistema operativo, o lo que es lo mismo, la cooperación entre el sistema operativo y la interfaz de red. Para que este aspecto no suponga un cuello de botella en el camino de comunicación, la interfaz de red debería estar integrado en el sistema operativo de forma que la sobrecarga asociada a los accesos al mismo sea lo más pequeña posible. Por tanto, en

términos generales, la optimización del sistema de comunicaciones y especialmente en un sistema basado en TCP/IP, debe abarcar desde aspectos relativos a la optimización del hardware y la arquitectura del sistema para reducir la sobrecarga asociada a los procesos de comunicaciones, hasta la optimización de la integración con el sistema operativo, incluyendo la mejora de la gestión de los recursos asignados estática o dinámicamente al sistema de comunicación, la optimización de la pila TCP/IP, de parámetros relativos a la pila de protocolos, etc.

Normalmente, a la hora de evaluar la sobrecarga asociada a los procesos de comunicaciones, se distingue entre sobrecarga por paquete y sobrecarga por byte [CHA01]. La sobrecarga por paquete se atribuye al coste de procesamiento de la pila de protocolos, a la gestión de buffers y a los cambios de contexto. La sobrecarga por byte se atribuye a las copias de datos dentro del nodo y al cálculo de la suma de comprobación. Para reducir la sobrecarga por paquete se pueden adoptar dos soluciones:

a) Utilización de paquetes de mayor tamaño: con lo que se reduce el número de paquetes necesarios para transferir una determinada cantidad de información, lo que se consigue utilizando *tramas jumbo* (paquetes con un tamaño superior a 1500 bytes). Sin embargo, al incrementar el tamaño de los paquetes, se incrementa la sobrecarga por paquete debida a las copias de datos en el sistema. Por tanto, la mejora de prestaciones que puede suponer el incremento del tamaño máximo del paquete dependerá de las prestaciones de los buses de E/S y del subsistema de memoria [GAL99, CHA01, GAD07]

b) Reducción del número de interrupciones asociadas a la recepción de paquetes [CHA01, GAD07]. Las dos soluciones a) y b) están relacionadas, ya que al utilizar paquetes de mayor tamaño, se reduce el número de paquetes y por tanto el número de interrupciones generadas a la recepción de una determinada cantidad de información. Sin embargo, conjuntamente con esta técnica puede utilizarse técnicas que permiten reducir el número de interrupciones por segundo generadas, como la llamada coalescencia de interrupciones, que permite retrasar la generación de la interrupción un determinado número de paquetes (MTUs) recibidos, o técnicas de sondeo (*polling*), mediante las cuales se comprueba periódicamente la llegada de paquetes al interfaz de red. La mayoría de los controladores actuales de interfaces de red gigabit o multigigabit de Linux, utilizan ambas técnicas, implementadas mediante NAPI [SAL01, BEN05, BOV05], como se ha comentado en la sección 1.3.1.

Como se ha comentado, la sobrecarga por byte se debe a las copias de datos y por tanto a la latencia asociada a las transacciones a través de los buses de E/S. Así pues, reducción de dicha sobrecarga requiere de la mejora del sistema de E/S, la memoria y los mecanismos de copia de datos. Para evitar la intervención de la CPU en la primera copia de datos tras la recepción de un paquete, las interfaces de red actuales utilizan técnicas de DMA para transferir los paquetes directamente a la memoria principal sin la intervención de la CPU principal del nodo, u otras técnicas que permiten reducir la sobrecarga asociada a las copias de datos como la técnica low-cost copy de Intel [LAU05] utilizada en la Intel I/O Acceleration Technology [IOA05] la cual, basándose en que una de las principales fuentes de sobrecarga en el procesamiento de los protocolos está en el movimiento de datos entre buffers de memoria, y en que la memoria siempre será más lenta que la CPU principal, propone la introducción de lógica entre la CPU y la memoria que planifique las copias de memoria. Otra técnica para reducir la sobrecarga asociada a las copias de datos es el RDMA o DMA remoto. El uso de RDMA que permite la transferencia de datos entre la memoria de dos nodos, supone una ventaja frente al uso de sockets ya que reduce la intervención de la CPU en las copias de memoria y disminuye el tráfico de datos en el bus de memoria [BAL04]. Además, en las arquitecturas actuales, dotadas de buses de alta velocidad como son el bus *FSB* (*Front Side Bus)* de Intel o el bus *Hypertransport* de AMD, podrían utilizarse interfaces de red conectados a estos buses de gran ancho de banda y baja latencia. En este caso, el bus *Hypertransport* de AMD parece más apropiado, ya que este bus está diseñado para conectar dispositivos de una forma directa con el procesador, el cual suele disponer de un controlador de memoria integrado. El bus *FSB* sin embargo, no es realmente un estándar de interconexión de interfaces externos (como es el caso de la interfaz de red) sino un bus de E/S interno para la interconexión de los diferentes dispositivos integrados en la misma placa base.

En resumen, a medida que se dispone de enlaces de red de mayor ancho de banda, se tiende a utilizar paquetes de mayor tamaño (mayor MTU) para disminuir la sobrecarga por paquete. Sin embargo, el subsistema de E/S y de memoria se convierte en el cuello de botella, al incrementarse el número de transacciones para decodificar cada paquete y hacer un uso intensivo de estos recursos.

Para mejorar las prestaciones y mitigar los problemas anteriores, las tarjetas de red actuales (NICs) para enlaces de red Gigabit o multigigabit, incorporan mejoras para reducir la sobrecarga de comunicaciones, mejorando así las prestaciones del sistema de

comunicaciones. Estas mejoras, son transparentes para el usuario y no requieren cambios en las aplicaciones. Solamente requieren de pequeños cambios en el sistema operativo, ya que estas mejoras traspasan parte de la sobrecarga de comunicación a la propia tarjeta de red. Las mejoras que comúnmente incorporan las interfaces de red Gigabit o multigigabit son el cálculo de la suma de comprobación y la validación de los campos FCS (*frame check sequence*). En [FOO03] se presentan resultados que muestran una reducción de la carga de la CPU principal del 10% al utilizar esta técnica. Otra técnica utilizada es la agrupación de interrupciones, ya comentada en esta misma sección como estrategia para la reducción de la sobrecarga por paquete, debido a que la sobrecarga debida a la atención de las interrupciones generadas en la transmisión o recepción de paquetes supone hasta el 10% de la sobrecarga total de comunicaciones si se utiliza la pila de protocolos TCP/IP. Se ha previsto el uso de esta técnica en las pilas de protocolos TCP/IP actuales incrementando el tamaño de la ventana de congestión TCP, no en una cantidad de bytes constante, sino basándose en el número de bytes cubiertos por el último ACK recibido, de forma que el retraso en la recepción de los ACKs no suponga un problema [ALL03]. La técnica LSO (Large Segment offload) consiste en el traspaso del proceso de creación y encapsulación de paquetes TCP sucesivos para ser enviados. Esta técnica tiene un impacto considerable en las prestaciones. De hecho, trabajos como [REG04] muestran mejoras en las prestaciones de hasta el 50% utilizando esta técnica. Otra mejora de las interfaces de red gigabit y multigigabit consiste en traspasar el proceso de separación de las cabeceras (*header splitting en inglés*) a la interfaz de red, mejorando la eficiencia de la memoria *cache* al no tener que procesar la CPU cada cabecera [NAH97].

Por otro lado, la técnica presentada en [KIM05] propone disponer una memoria *cache* en la tarjeta de red (NIC) de forma que aquellos datos que se envíen por la red frecuentemente, estén disponibles en dicha memoria *cache* y no haya que transferirlos cada vez desde la memoria principal. Esta técnica tiene como objetivo la reducción del tráfico de E/S en el nodo que como ya se ha comentado es uno de los cuellos de botella más importantes en el camino de comunicación.

Las mejoras que se han visto, afectan tanto al sistema operativo, como a la interfaz de red (NI). Las mejoras que se han comentado, incluidas en las tarjetas de red actuales contribuyen a mejorar las prestaciones de un sistema de comunicaciones que utilice los protocolos TCP/IP. Sin embargo, hay otras alternativas para mejorar las prestaciones del sistema de comunicaciones. Dichas alternativas se presentan en las secciones siguientes,

y consisten, bien en la utilización de protocolos ligeros (Sección 1.5), interfaces a nivel de usuario (Sección 1.6) o la externalización de los protocolos de comunicación (Sección 1.7).

1.5 Protocolos ligeros

Como ya se ha comentado en la presentación y en la sección 1.1, la sobrecarga generada en un sistema de comunicaciones que utilice enlaces de red de anchos de banda del orden de gigabits/s, hace ineficiente el uso de la pila de protocolos TCP/IP en los sistemas estándar. Esta ineficiencia es mayor cuanto mayor sea el ancho de banda del enlace, como ocurre en redes como Myrinet o redes multigigabit, en las que el tiempo de transmisión es prácticamente despreciable frente al tiempo necesario para procesar los protocolos de red [CIA01]. Una alternativa a la mejora de las prestaciones del sistema de comunicaciones, consiste en la utilización de pilas TCP/IP optimizadas, nuevos protocolos que requieran un tiempo de procesamiento menor que el requerido por la pila TCP/IP y que mejoren el soporte del sistema operativo a las comunicaciones, o incluso en pilas de protocolos a nivel de usuario [PAK97, PRY98]. Mediante los protocolos llamados *ligeros*, se pretende evitar los cuellos de botella que normalmente suponen los buses de E/S y la memoria en el camino de comunicación. Dentro de esta línea, además de los protocolos a nivel de usuario [DUN98, VID01] y protocolos que son mejoras de TCP/IP [CLA89, JAC92], están otros protocolos ligeros, como [GEO97], [JIN01], y otros más recientes como GAMMA [CIA01] y CLIC [DIA01, DIA03], los cuales, sustituyen las capas TCP/IP y mejoran el soporte que sistema operativo ofrece a las comunicaciones.

1.5.1 El protocolo CLIC (Communication on Linux Clusters)

El protocolo CLIC está embebido en el núcleo de Linux, y proporciona un interfaz a las aplicaciones de usuario. Como se muestra en la Figura 1.6, CLIC reduce el número de capas del protocolo, respecto a las utilizadas por la pila TCP/IP, lo que disminuye la sobrecarga software y el número de copias necesarias para el procesamiento de la pila de protocolos. Por ejemplo, CLIC elimina los protocolos IP con enrutamiento, ya que está pensado para su uso en clusters de computadores y por tanto abarca sólo redes de área local. Por tanto, CLIC implementa las funciones de interfaz con la aplicación de

usuario, transporte (TCP) y red (IP), constituyendo un interfaz único entre la aplicación de usuario y el controlador de la interfaz de red. Esta arquitectura reduce el tiempo empleado en una llamada al sistema para envío o recepción de datos [DIA03]. En el caso de transmisión de un paquete, la llamada al sistema generada ejecutará directamente el módulo CLIC del núcleo de Linux. En el caso de recepción de un paquete, el *bottom-half-handler* ejecutará el módulo CLIC.

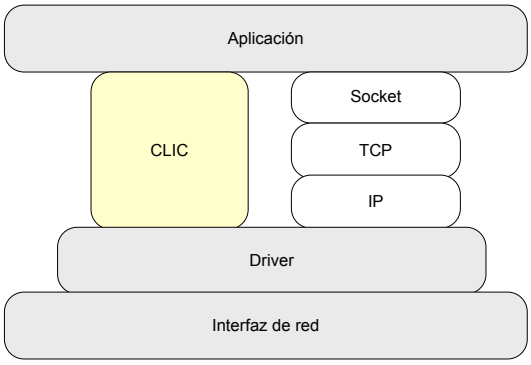


Figura 1.6. CLIC vs. TCP/IP [DIA03]

Aunque CLIC proporciona un ancho de banda ligeramente inferior al que proporciona GAMMA y una latencia ligeramente superior, mejora las prestaciones ofrecidas por TCP/IP y tiene algunas ventajas frente al protocolo GAMMA [CIA01], tal y como se verá a continuación.

1.5.2 El protocolo GAMMA (Genoa Active Message Machine)

Igual que CLIC, GAMMA es un protocolo ligero. Al igual que CLIC, el protocolo GAMMA está embebido en el núcleo de Linux [CIA01] y puede coexistir con servicios basados en IP en la misma red de área local. Sin embargo, en este caso, el protocolo se basa el uso de puertos activos para de reducir la sobrecarga generada por la pila de protocolos TCP/IP. Como en el caso de CLIC, GAMMA surge para mejorar las prestaciones del sistema de comunicaciones en redes Fast Ethernet para después extenderse a redes de tipo Gigabit Ethernet. Además, El protocolo GAMMA busca reducir el uso de CPU y de los buses de E/S. Para ello, utiliza mecanismos de cero copias. Se consigue que en recepción, el uso de CPU sea proporcional al tamaño del paquete. Al mismo tiempo GAMMA proporciona control de flujo como para evitar la

congestión y no garantiza unas prestaciones constanres sino que dichas prestaciones se ofrecen en modo de "mejores prestaciones" (*best effort*), por lo que no se garantiza una tasa de transferencia determinada. Por otro lado, GAMMA no dispone de mecanismos de retransmisión de paquetes, basándose en la baja probabilidad de error.

Un inconveniente del protocolo GAMMA es que depende de la interfaz de red, siendo necesario adaptar el conjunto de funciones de bajo nivel que se implementan en la interfaz BIND [CIA01] a cada interfaz. Por tanto, la portabilidad de GAMMA conlleva la adaptación de este interfaz.

1.6 Interfaces de red a nivel de usuario. Estándar VIA

Como se comentó en las secciones 1.3.3 y 1.4, una alternativa para mejorar las prestaciones del sistema de comunicaciones consiste en utilizar pilas de protocolos que generen una sobrecarga de comunicación menor que la pila de protocolos TCP/IP, no sólo por el propio protocolo sino por la integración con el sistema operativo del mismo. Estos son los protocolos que se han comentado en la sección 1.5 y que se denominan protocolos ligeros. Otra forma de disminuir la sobrecarga de comunicaciones consiste en el uso de interfaces de red a nivel de usuario, para lo que se ha desarrollado el estándar VIA [VID01]. El objetivo del estándar VIA es reducir la sobrecarga de comunicación entre la CPU, la memoria y la interfaz de red para mejorar las prestaciones en redes de altas prestaciones [DUN98, VID01]. Con dicha reducción en la sobrecarga del sistema de memoria y de la CPU, VIA consigue mejorar la latencia y el ancho de banda entre dos nodos dentro de un cluster de computadores.

El estándar VIA proporciona una comunicación orientada a conexión, y da a las aplicaciones un acceso directo al interfaz de red, evitando copias intermedias de datos así como la intervención del sistema operativo. De esta forma, se evitan interrupciones y cambios de contexto, disminuyendo la sobrecarga en la CPU principal del nodo. Además, VIA proporciona un mecanismo mejorado de comunicación entre procesos en un cluster de computadores que permite disminuir la latencia.

En la siguiente sección, se presentará otra técnica que permite reducir la sobrecarga de comunicación, basada en el traspaso de la carga de trabajo asociada al procesamiento de las comunicaciones a otro procesador diferente del procesador principal del nodo.

1.7 Externalización de protocolos de comunicación

El rendimiento de un servidor de altas prestaciones depende de las características arquitectónicas del servidor, es decir, del sistema de entrada/salida (buses de entrada salida), del sistema de memoria y del procesador, así como los mecanismos de interconexión entre dichos elementos, como puede verse en la Figura 1.3.

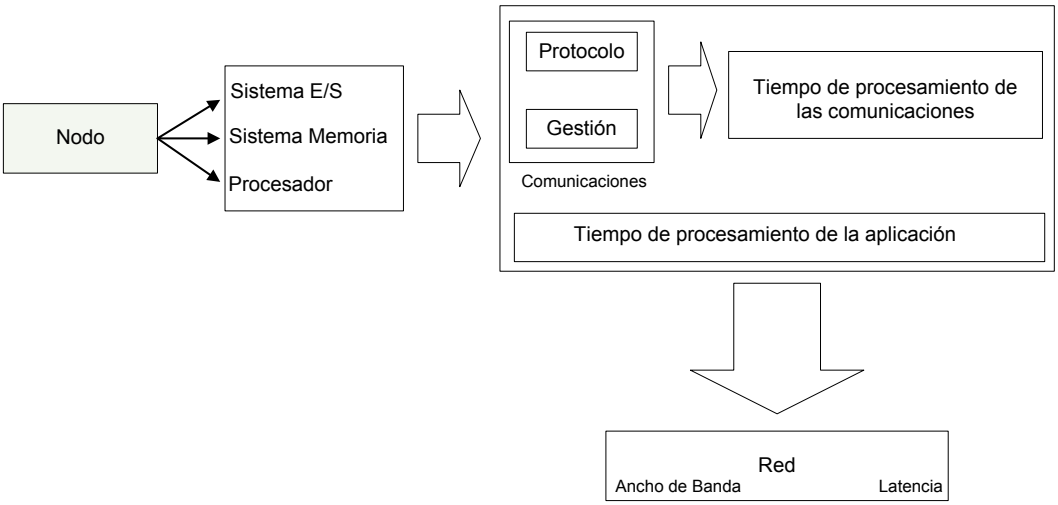


Figura 1.3 Elementos que determinan las prestaciones del sistema de comunicaciones

Estas características, afectan al sistema de comunicaciones, en cuanto al tiempo de procesamiento del protocolo, necesario para que los datos extraidos queden a disposición de la aplicación como al tiempo empleado en tareas de gestión de conexiones, gestión de buffers, etc. normalmente realizadas por el sistema operativo. A este tiempo, hay que sumarle el tiempo que la aplicación emplea en procesar la información, para obtener el tiempo neto de transferencia de una cierta cantidada de datos entre dos aplicaciones de dos nodos de una red (tiempo de procesamiento de las comunicaciones). Como se muestra en la Figura 1.3, y como se dijo en la Sección 1.2 donde se analiza el camino de comunicación, el ancho de banda efectivo, dependerá de la suma del tiempo de procesamiento asociado a los procesos de comunicaciones y el tiempo de procesamiento de la aplicación. Por tanto, se puede liberar al procesador principal en mayor o menor medida de la sobrecarga asociada a los procesos de comunicaciones, con el objetivo de dejar más ciclos libres para la aplicación es, bien optimizando el procesamiento de los protocolos como se ha mostrado en la Sección 1.4,

bien utilizando protocolos que hagan un uso más eficiente de los recursos disponibles, en cuanto a la arquitectura y al sistema operativo, como se ha mostrado en la Sección 1.5 y 1.6. En definitiva, se trata de dejar más ciclos libres disponibles para la aplicación.

Además de los intentos de mejorar la implementación del software de la interfaz de red con el desarrollo de nuevos protocolos (que en algunos casos implican algunas modificaciones en el hardware), existe existe otra vertiente que puede considerarse ortogonal a la anterior que busca incrementar los recursos hardware disponibles en la interfaz de red, incluyendo de procesadores de red (NPs) [CHA03, THI02, PAP04] en la NIC. De esta forma, la CPU del sistema se liberaría de todo o parte de la sobrecarga relacionada con los procesos de comunicación y se podría realizar una implementación más eficiente de los sistemas de comunicación. Esta técnica, consistente en dedicar un procesador diferente al principal para el procesamiento de las tareas de comunicaciones se conoce con el nombre de externalización (*en inglés offloading)*.

Los beneficios que puede aportar la externalización de protocolos dependen del tipo de implementación que se haga de la misma. En cualquier caso, puede decirse que mediante la externalización de protocolos podrían obtenerse los siguientes beneficios:

1. El incremento en el número de ciclos de CPU disponibles para la aplicación.
2. La reducción de la latencia del protocolo ya que los mensajes cortos como los ACKs no tienen que llegar a memoria principal ni atravesar el bus de E/S ni se generan interrupciones.
3. El incremento de la eficiencia del DMA si se agrupan mensajes cortos
4. La reducción del número de colisiones en el bus E/S
5. La posibilidad de gestionar dinámicamente el protocolo utilizado en función de la naturaleza de los mensajes a enviar/recibir.

Sin embargo, hay trabajos [MOG03, REG04A, CLA89, ODE02] que defienden que la externalización, especialmente en el caso del protocolo TCP, no produce mejoras claras en el rendimiento de las aplicaciones. En general, las razones para este escepticismo se derivan de dificultades de implementación, mantenimiento y *test* de los protocolos externalizados [MOG03]. Además, la propia comunicación entre la NIC (con el protocolo externalizado), la CPU y la API podría ser tan complicada como el protocolo a externalizar, al necesitar un conjunto de funciones suficientemente extenso para comunicarse con la NIC que podría llegar a ser tan complicada como el propio

protocolo [ODE02]. En el caso de TCP, la gestión de los *buffers* es complicada y podría dificultar la mejora producida por la externalización. Además, no sería adecuado externalizar los protocolos en una CPU más lenta que la CPU principal del nodo [REG04a]. Probablemente, estas razones no son definitivas para decidir sobre la utilidad de la externalización de protocolos pero hay que sopesarlas frente a los posibles beneficios.

Como se ha comentado en la sección 1.4, inicialmente se supuso que la principal fuente de sobrecarga estaba en el procesamiento del protocolo TCP/IP. Sin embargo, análisis más detallados como el realizado en [CLA89] pusieron de manifiesto la principal fuente de *sobrecarga* se debe al sistema operativo, la gestión de las interrupciones, las múltiples copias de datos, la gestión de los *buffers*, etc. Ya se ha visto que existen diferentes técnicas para reducir la sobrecarga asociada a un sistema de comunicaciones que utilice TCP/IP, tales como la coalescencia de interrupciones, cuyo objetivo está en reducir el tiempo de CPU dedicado a la atención de las mismas, la externalización de ciertas funciones simples en la interfaz de red, como el cálculo de la suma de comprobación de los paquetes o la segmentación TCP (*LSO, Large segment offload*) [REG04], ya que el procesamiento de la capa TCP y de la cabecera IP es menos costosa desde el punto de vista computacional si los paquetes son más pequeños (se han segmentado los paquetes). Además, se han desarrollado otras técnicas como el uso de técnicas híbridas interrupciones/sondeo o de sondeo, como NAPI [SAL01], actualmente implementado en la mayoría de los *drivers* para interfaces de red gigabit y multigigabit, con el fin de reducir la *sobrecarga* asociada a la gestión de las interrupciones [MOG97].

Otra fuente de *sobrecarga* asociada a los procesos de comunicaciones en el servidor es la latencia de acceso a memoria. Este efecto, puesto de manifiesto en en trabajos como [NAH97, MIN95, IOA05, WUL95]. De hecho, trabajos como [REG04a] muestran como los recursos de un servidor basado en dos procesadores Intel Xeon se saturan para un ancho de banda de 1,5 Gbps. Además, la falta de localidad espacial y temporal de los datos en el procesamiento de la pila TCP/IP hacen que no se pueda aprovechar correctamente las ventajas que aporta la memoria *cache* a las aplicaciones convencionales. La pila de protocolos TCP/IP utiliza gran cantidad de información de control y por otro lado, los datos recibidos o a transmitir, son transferidos desde la memoria principal hasta la interfaz de red, lo que provocará fallos de *cache* cuando la CPU intente acceder a dichos datos (por ejemplo, información de control necesaria para la decodificación de un paquete) [NAH97].

Por otro lado, las múltiples copias de datos necesarias para transferir la información desde la red hasta la aplicación (memoria de usuario), supone otra importante limitación a las prestaciones del sistema de comunicaciones. Como se ha visto, normalmente son necesarias una o más copias de los datos para procesar un paquete y dejarlo disponible para la aplicación. Existen técnicas para reducir el número de copias de datos durante la transmisión de los mismos, llamadas técnica de copia cero (*zero-copy)*, mediante las cuales, la información se transfiere directamente desde la memoria de usuario al interfaz de red mediante DMA.

Hay otras razones que afectan los posibles beneficios proporcionados por la externalización. Uno de ellos es la relación entre la capacidad de procesamiento de la CPU del sistema y de la NIC. La CPU del sistema tendrá normalmente una capacidad de proceso mucho mayor que la de la NIC y bajo determinadas condiciones, la NIC puede incluso suponer un cuello de botella en la comunicación. El uso de procesadores de propósito general en la NIC suele ser un mal compromiso prestaciones/coste [CLA89]. Además, las limitaciones en los recursos disponibles en la NIC, pueden comprometer la escalabilidad del sistema (por ejemplo, si no hay mucha memoria disponible se podría limitar la tabla de encaminamientos IP).

Los problemas anteriores son más evidentes en aplicaciones que requieren un gran ancho de banda. En estos casos, la *sobrecarga* asociada a la gestión de las conexiones es más importante y mucho más difícil de evitar mediante la externalización. Según se indica en [MOG03], la externalización está más indicada para aplicaciones que requieren un gran ancho de banda, baja latencia y conexiones de larga duración.

Sin embargo, hay otros trabajos que ponen de manifiesto las ventajas de la externalización. En [WES04], se realiza un estudio experimental basado en la emulación de una NIC conectada al bus de E/S y controlada por uno de los procesadores de un SMP. En este estudio, se consiguen mejoras de entre un 600% y un 900% en la externalización emulada del protocolo TCP. También los modelos propuestos en [SHI03, GIL05] ponen de manifiesto la posibilidad de mejoras en la externalización.

De la misma forma, en [MOG03] se indica que las técnicas de externalización utilizadas para aumentar el rendimiento de las comunicaciones son aplicables en todos aquellos entornos en los que el tiempo proceso asociado a los procesos de comunicación supone un cuello de botella en el camino de comunicación. Este sería el caso, por ejemplo, de la utilización de redes Ethernet Gigabit ó 10-Gigabit para la realización de *backups* de sistemas de almacenamiento (*SAN)* en *robots de backup* remotos basados en

IP, en los que hay que hacer copia de cantidades masivas de datos en el menor tiempo posible, o siendo necesario un uso intensivo del sistema de comunicaciones con una red de gran ancho de banda. El interés por el incremento de las prestaciones de estas redes es mayor ya que incluso se están sustituyendo las redes *backbone* basadas en ATM en algunos casos por Gigabit Ethernet o 10-Gigabit Ethernet [PET05, SKO97].

En todo lo referente a las técnicas de externalización, es necesario tener en cuenta la aparición de los nuevos procesadores que incluyen dos o más núcleos (*multicore*) en un mismo chip. Estos núcleos pueden incluso ejecutar más de una *hebra* (*thread*) simultáneamente (*microengines*). Evidentemente, la aparición de estos procesadores ha supuesto un replanteamiento en las técnicas utilizadas para la externalización y sobre todo, de la arquitectura de los sistemas, en cuanto al lugar donde deben residir los protocolos. Dependiendo del entorno, puede resultar más beneficiosa una técnica sobre otra. También puede influir el coste final del sistema de comunicaciones. En esta línea surgen tanto los llamados TOE (*TCP Offload Engines*) [BRO07, CHE07, NET07a] como la propuesta de Intel I/O Acceleration Technology [IOA05].

Hay diferentes técnicas para conseguir la externalización en mayor o menor medida, que se irán tratando en esta memoria. Las dos más utilizadas se conocen como *offloading* y *onloading*. Aunque en los dos casos se trata, como ya se ha dicho, de liberar a la CPU que ejecuta la aplicación del proceso asociado a los procesos de comunicación, la diferenciación de ambas técnicas hace referencia al lugar dentro del sistema en el que se realizará el procesado de las comunicaciones.

1.7.1 Estimación de prestaciones. El modelo LAWS

En las secciones anteriores de este capítulo, se han presentado las principales dificultades a la hora de estudiar los beneficios aportados por las distintas técnicas propuestas para mejorar las prestaciones de la interfaz de red y conseguir aprovechar el ancho de banda del enlace, debido a la multitud de factores que intervienen en las prestaciones finales.

Para evaluar y predecir las prestaciones de un servidor bajo unas determinadas condiciones (tecnología del servidor, sobrecarga, ancho de banda de la red, técnica utilizada para disminuir la carga de trabajo asociada a procesos de comunicaciones, etc.), es de gran utilidad disponer de un modelo teórico que proporcione un valor para la mejora obtenida en el rendimiento del servidor, en función de estos parámetros.

Algunos trabajos recientes [SHI03] [GIL05], tratan de mejorar la compresión de los principios fundamentales de la externalización, más allá de los resultados experimentales obtenidos bajo condiciones y aplicaciones concretas. En [SHI03] se introduce el modelo LAWS para caracterizar los beneficios aportados por la externalización de protocolos en servicios de Internet y en aplicaciones de *streaming*.

En [GIL05], se propone el modelo EMO (*Extensible Message-oriented Offload*) para analizar las prestaciones proporcionadas por varias estrategias de externalización de protocolos de paso de mensajes. En la sección 4.6 de esta memoria, se realiza un análisis de los resultados experimentales a la luz del modelo LAWS. Por ello, pasamos a describir dicho modelo.

El modelo LAWS [SHI03] está basado en una estimación del ancho de banda pico proporcionado por un camino de comunicación segmentado a partir del ancho de banda determinado por el cuello de botella correspondiente en cada caso (el enlace, la NIC o la CPU del host). El análisis proporcionado en [SHI03] supone que las prestaciones están limitadas por la CPU antes de externalizar los protocolos de comunicaciones ya que en caso contrario, la externalización de protocolos no proporcionaría mejora alguna. Por tanto, el modelo incluye solo aplicaciones limitadas en ancho de banda (como es el caso de los servidores de Internet), cuando los parámetros utilizados por el modelo (como vemos son la ocupación de la CPU para procesos de comunicación y de las aplicaciones, factores de escala en la ocupación de la CPU del host y de la NIC, etc.) pueden conocerse de forma precisa.

Las Figuras 1.4a y 1.4b muestran nuestra visión acerca de cómo se ve el sistema desde el modelo LAWS, antes y después de externalizar respectivamente. La notación que se utiliza es similar a la de [SHI03]. Antes de externalizar (Figura 1.4a), el sistema se contempla como un cauce segmentado (*pipeline*) de dos etapas: el computador y la red. En el computador, para transferir m bits, los procesos de la aplicación generan una carga de CPU igual a aXm, y los procesos de comunicación generan una carga de CPU igual a oXm. En estos tiempos de proceso, a y o son la cantidad de trabajo de CPU por unidad de dato, y X es un parámetro de escala utilizado para tener en cuenta variaciones en la capacidad de procesamiento con respecto a un computador de referencia (que puede ser por ejemplo, el periodo de reloj al que funcione la CPU). Además, la latencia para proporcionar los m bits en un enlace de red con un ancho de banda igual a B, es m/B.

De esta forma, como el ancho de banda pico proporcionado antes de externalizar depende de la etapa que suponga el cuello de botella, vendrá determinado por la ecuación 1.4.

$$B_{before} = \min\left(B, \frac{1}{(aX + oX)}\right) \qquad (1.4)$$

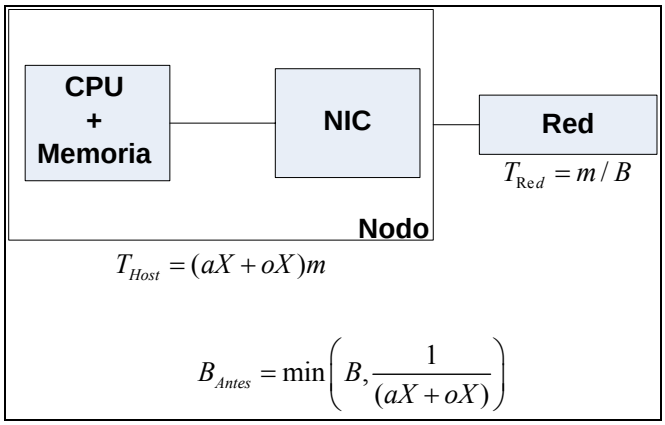
Después de externalizar, tenemos un cauce con tres etapas (como puede verse en la Figura 1.3b), y una porción *p* de la sobrecarga asociada a los procesos de comunicación se ha transferido al NIC (es la porción de sobrecarga asociada a los procesos de comunicaciones que se ha externalizado). De esta forma, las latencias de las distintas etapas para transferir *m* bits, son las que se pueden ver en las ecuaciones 1.5, 1.6 y 1.7.

$$T_{\mathrm{Red}} = m / B \qquad (1.5)$$

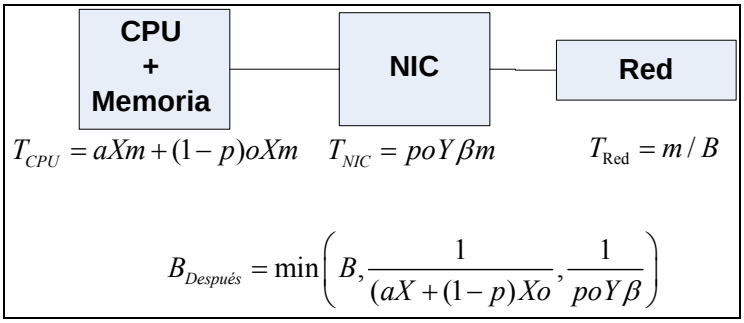
$$T_{CPU} = aXm + (1 - p)oXm \qquad (1.6)$$

$$T_{NIC} = poY\beta m \qquad (1.7)$$

En la expresión 1.5, que se corresponde con la latencia de la interfaz de red, *Y* es un factor de escala para tener en cuenta las diferencias de potencia de proceso con respecto al computador de referencia (el periodo de reloj del procesador de la NIC) y β es un parámetro que cuantifica la mejora en la sobrecarga asociada al sistema de comunicaciones que se puede llegar a conseguir sin *offloading* (es decir, β es la porción de sobrecarga normalizada remanente en el sistema después de externalizar [SHI03], dado que se pueden haber aplicado algunas técnicas que contribuyan a reducir el trabajo de procesamiento de la interfaz de red después de la transformación).



(a)



(b)

Figura 1.4 Modelo LAWS (a) antes de externalizar y (b) después de externalizar

De esta forma, después de externalizar, y tal y como puede verse en la Figura 1.4b, el ancho de banda viene dado por la expresión $B_{Después} = \min\left(B, \frac{1}{(aX + (1-p)oX}, \frac{1}{poY\beta}\right)$, y el incremento relativo en la mejora del ancho de banda pico se define como $\delta B = \left(\frac{B_{Después} - B_{Antes}}{B_{Antes}}\right)$.

$$\delta B = \frac{\min\left(B, \frac{1}{(aX + (1-p)oX}, \frac{1}{poY\beta}\right) - \min\left(B, \frac{1}{aX + oX}\right)}{\min\left(B, \frac{1}{aX + oX}\right)} \qquad (1.8)$$

El acrónimo LAWS viene de los parámetros utilizados para caracterizar las mejoras que aporta la externalización (***L**ag **A**pplication **W**ire **S**tructural)*. Además del parámetro β (*Structural* ratio) ya comentado, el modelo LAWS consta de otros parámetros:

- $\alpha = \frac{Y}{X}$ (***L**ag ratio*) : el cual considera la proporción entre la velocidad de la CPU y la velocidad de la NIC (Y es el periodo de reloj del procesador de la NIC y X es el periodo de la CPU)
- $\gamma = \frac{a}{o}$ (***A**pplication ratio*): mide la relación entre la carga de computación y comunicación de una determinada aplicación.
- $\sigma = \frac{1}{oXB}$ (***W**ire r**atio***): es porción del ancho de banda de la red que el host puede proporcionar antes de externalizar las comunicaciones.

Mediante estos parámetros, podemos estudiar el efecto de los diferentes factores que afectan a los beneficios aportados por la externalización y que se han comentado en la Sección 1.3. Utilizando el *Lag ratio* podemos analizar el efecto de la diferencia de prestaciones entre la CPU principal y la CPU de la interfaz de red. El *Application ratio*, permite describir la relación entre la sobrecarga generada por la aplicación y por los procesos de comunicaciones, el *Wire ratio* incorpora el efecto del ancho de banda del enlace de red, y mediante el *Structural ratio* se tiene en cuenta la parte de la carga del procesador principal que se reduce al externalizar (β<1).

En términos de los parámetros α, β, γ y σ, el incremento relativo en el ancho de banda pico viene dado por la ecuación 1.9.

$$\delta b = \frac{\min\left(\frac{1}{\sigma}, \frac{1}{\gamma + (1-p)}, \frac{1}{p\alpha\beta}\right) - \min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)}{\min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)} \qquad (1.9)$$

Del modelo LAWS se pueden derivar algunas conclusiones en términos de relaciones simples entre los parámetros descritos, como se muestra en la Figura 1.4 (obtenida a partir de la expresión de la ecuación 1.9:

- La externalización de protocolos proporciona una mejora que crece linealmente en aplicaciones con una baja relación computación/comunicación (bajo γ). Este perfil corresponde a aplicaciones de procesamiento de datos de streaming, servidores de almacenamiento, redes SAN (*Storage Area Networks*) con una gran cantidad de discos, etc. En el caso de aplicaciones que hacen un uso intensivo de la CPU, la mejora en el ancho de banda proporcionada por la externalización está limitada por 1/γ, y tiende a cero conforme el coste computacional crece (es decir, γ crece). El mejor valor de la mejora del ancho de banda que se puede alcanzar viene determinado por $\gamma = \max(\alpha\beta, \sigma)$. Además, como la pendiente de la función de mejora $\frac{(\gamma+1)}{c} - 1$ es 1/c, y al mismo tiempo, $c = \max(\alpha\beta, \sigma)$, la mejora en el ancho de banda crece más rápidamente conforme αβ o σ decrece.

- La externalización de protocolos puede reducir el ancho de banda (empeorar las prestaciones del sistema sin externalizar $\delta B < 0$) si la función de mejora $\frac{(\gamma+1)}{c} - 1$ puediese tomar valores negativos. Esto ocurriría cuando $\gamma < (c-1)$ y como γ>0 y σ<1, se debería verificar que c=αβ y que αβ>1. De esta forma, si la NIC es más lento que la CPU del sistema (α>1), la externalización provocaría una reducción en las prestaciones si la NIC se satura antes que el enlace de red, es decir, αβ>σ, ya que la mejora en el ancho de banda está limitada por 1/α (siempre que β=1). Sin embargo, si se realiza una implementación eficiente de la externalización (con el uso de técnicas que reduzcan el número de copias de los datos, por ejemplo) de forma que se introduzcan en el sistema mejoras estructurales (es decir, reduciendo β), sería posible sacar partido de la externalización para valores de α>1.

- En redes lentas (σ>>1) donde el host puede asumir toda la sobrecarga asociada a los procesos de comunicación sin ayuda, no se producirían mejoras con la externalización. La utilidad de la externalización es clara siempre que el host no sea capaz de comunicarse a la velocidad del enlace de red (σ <<1) (es decir, no sea capaz de aprovechar todo el ancho de banda del enlace), pero en ese caso, γ debe ser bajo, como se ha comentado anteriormente. Como actualmente existe una tendencia hacia redes con un ancho de banda cada vez mayor (σ decrece), la externalización se puede ver como una técnica muy útil para aprovechar el ancho de banda de redes rápidas. Cuando σ está cercano a 1, el mejor valor para la mejora que se puede obtener, corresponde con el caso donde hay un equilibrio entre computación y comunicación antes de externalizar (γ=σ=1).

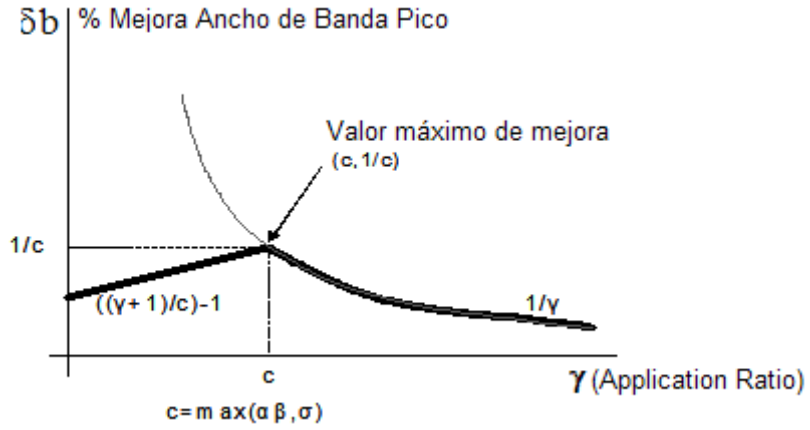


Figura 1.5. Comportamiento del ancho de banda pico de acuerdo al modelo LAWS

En la Figura 1.5, en la que se ha representado el *Application ratio (γ)* frente a la mejora en ancho de banda pico (δb), puede verse como para valores bajos del parámetro γ, la mejora en ancho de banda crece linealmente, según la función, $\frac{(\gamma+1)}{c}-1$, la cual es una recta de pendiente 1/c. El máximo en la mejora de ancho de banda pico se obtiene cuando Como se ha comentado, la externalización de protocolos podría empeorar las prestaciones del sistema sin externalizar en el caso de que la función de mejora tomase valores negativos. Por otro lado, al aumentar el valor de *γ,* la mejora tiende a cero, según la ley 1/γ.

Como se puede ver en la Figura 1.5, el valor pico para la mejora se corresponde con el valor para el *Application Ratio γ=c=max(αβ,σ)*, es decir, el máximo entre el ancho de banda que pueda proporcionar la NIC y el sistema de comunicaciones antes de

la externalización, lo que proporciona una mejora en el ancho de banda pico, de 1/c, según muestra la ecuación 1.10.

Vemos como el modelo LAWS presentado en esta sección permite disponer de una primera aproximación a los posibles beneficios aportados por la externalización, en diferentes condiciones.

1.7.2 La simulación como técnica de análisis y evaluación de arquitecturas de comunicación

La simulación es la técnica más frecuente para evaluar las propuestas en arquitecturas de computadores. El simulador utilizado para este fin, debe ser capaz de modelar el sistema con un grado de detalle suficiente, y permitir la ejecución de aplicaciones reales, tales como servidores *web*, bases de datos, etc., aparte de los correspondientes *benchmarks* estándar que pueden ser más o menos simples. Para esto necesitamos un simulador de sistema completo [ENG03] en el que se pueda reproducir el funcionamiento real del sistema lo más fielmente posible. Por tanto, los simuladores que sólo permiten la ejecución en modo usuario y una sola hebra no son adecuados para nuestro propósito [MAR05]. En el mercado existen simuladores que permiten, en principio, realizar las simulaciones que necesitamos, como son *M5* [M5S07], *Simics* [VIR08a] y las variantes basadas en *Simics* [MAG02, VIR04] o para plataformas Sparc o PA-RISC [M5S07, MAR05, MAU02, ROS97], pero habrá que seleccionar uno que proporcione un nivel de detalle suficiente, y que además, pueda ser configurado a medida, para que sea posible simular la arquitectura propuesta. Entre los simuladores que hemos evaluado, consideramos que Simics es el que cumple los requisitos necesarios, ya que permite realizar una configuración a medida de la máquina, modelar hardware a medida, e instalar un sistema operativo como Linux o aplicaciones comerciales que puede interactuar con el nuevo hardware. Además, Simics [VIR08a] es un simulador dirigido por eventos, lo que lo hace más rápido que otros simuladores con características funcionales similares, además de poseer herramientas de *generación de perfiles* y descripción del hardware. No obstante, si bien Simics es un simulador de sistema completo que se ajusta a nuestros requisitos, presenta algunos problemas y limitaciones que hay que resolver para poder llevar a cabo simulaciones realistas de las aplicaciones de comunicación, como se verá en el capítulo 2 de esta memoria.

1.8 Resumen

En este capítulo se ha presentado la influencia de las prestaciones del sistema de comunicaciones en las prestaciones finales de los servidores de altas prestaciones. Se ha analizado las fuentes de sobrecarga en el camino de comunicación, con el fin de identificar los cuellos de botella que limitan las prestaciones de la interfaz de red , y por tanto, que partes del sistema deben ser optimizadas para mejorar las prestaciones. Entre los elementos del sistema que se pueden mejorar, se propone en la sección 1.4 la optimización de la pila de protocolos TCP/IP, el uso de protocolos ligeros en la sección 1.5, o de interfaces de comunicación a nivel de usuario en la sección 1.6, los cuales, por un lado requieren una menor carga de trabajo para ser procesador y por otro lado optimizan la integración con el sistema operativo. Además, se presenta el efecto de los buses de E/S y de los accesos a memoria en las prestaciones del interfaz de red.

Finalmente, en la sección 1.7 se trata la técnica de externalización de protocolos (*en inglés offloading*) para mejorar las prestaciones de la interfaz de red, que consiste en traspasar una porción de la sobrecarga de comunicación a un procesador diferente del procesador principal del nodo. Esta técnica ha generado una gran controversia en cuanto a los posibles beneficios que puede aportar, por lo que resulta interesante el análisis de esta técnica bajo diferentes condiciones.

En definitiva, resulta muy conveniente disponer de una herramienta que permita simular unas condiciones de trabajo determinadas, y bajo estas, evaluar la mejora introducida por la externalización. Para ello, se propone un modelo de simulación utilizando el simulador de sistema completo Simics [VIR08a], mediante el cual, se pueden evaluar la mejora introducida al externalizar las comunicaciones, mediante offloading o mediante onloading, así como utilizar cargas de trabajo realistas. La ventaja de utilizar un simulador frente a sistemas reales es la posibilidad de modificar parámetros del hardware o de la arquitectura y realizar nuevos experimentos.

Por otro lado, resulta también de gran interés disponer de un modelo teórico que proporcione una predicción de la mejora obtenida al externalizar los protocolos de comunicaciones, en función de diferentes parámetros del sistema. En este sentido, el modelo LAWS proporciona un valor para la mejora obtenida utilizando diferentes técnicas de externalización y en función de las características de tecnológicas del

servidor y del ancho de banda de la red. De esta forma, puede utilizarse este modelo con el objetivo de evaluar las prestaciones y poder realizar predicciones, proporcionando la información necesaria para realizar cambios que mejoren el rendimiento global del servidor.

No obstante, este modelo ha de ser modificado con el fin de conseguir una predicción más precisa de los resultados experimentales. En el Capítulo 4, se propone una modificación del modelo LAWS, que incluye nuevos parámetros. Se utilizará el modelo LAWS modificado propuesto para conseguir un mejor ajuste a los resultados experimentales.

En el capítulo siguiente, se presenta la técnica de simulación como técnica principal a la hora de evaluar arquitecturas de comunicación, y se realizan propuestas de optimización con el fin de mejorar las prestaciones del sistema de comunicaciones, utilizando la técnica de simulación de sistema completo (*full-system simulation*), ya que como se ha visto en este capítulo, las prestaciones del sistema de comunicaciones no solo dependen de la interfaz de red, sino también del subsistema de E/S, la interacción con el sistema operativo y las aplicaciones. La simulación de sistema completo permite no solo simular una determinada arquitectura, sino también ejecutar un sistema operativo y aplicaciones comerciales o test de *benchmark*.

Capítulo 2

Metodología Experimental basada en la Simulación de Sistema Completo.

En el capítulo anterior, se presentó la problemática asociada al rápido incremento en el ancho de banda de los enlaces de red, debido a que los cuellos de botella en el camino de comunicación se desplazan hacia los nodos. Para identificar los diferentes factores que influyen en las prestaciones del sistema, y proponer mejoras en las arquitecturas de comunicación, resulta muy conveniente disponer de una herramienta que permita simular nuevas propuestas en las arquitecturas de comunicación y analizar su comportamiento, ya que evaluar las mismas en un sistema real puede ser complicado o incluso imposible, sobre todo cuando las propuestas incluyen modificaciones en el hardware. En primer lugar, es necesario disponer de sistemas con características suficientemente variadas para que se pueda abarcar un área significativa del espacio de diseño. En un sistema real no podemos modificar parámetros relacionados con la arquitectura del sistema o con las características de la tecnología hardware utilizada, como es la modificación del tiempo de acceso a memoria. En segundo lugar, en la mayoría de los casos, no es posible encontrar sistemas reales que implementen nuevas propuestas arquitectónicas. Así pues, la simulación puede considerarse la técnica más apropiada para la investigación en nuevas arquitecturas de computadores.

Existen simuladores tanto comerciales como de libre distribución, con los que ensayar y evaluar las propuestas para mejorar las prestaciones de comunicación. Entre ellas se encuentran simuladores de redes como [OPN07, WAN99], de protocolos de comunicación como NS2 [NS207], de interfaces de red programables como Spinach [WIL04] o de microprocesadores como WebMips [BRA04]. Con estos simuladores, es posible analizar las diferentes partes que intervienen en el camino de comunicación por separado. Sin embargo, dada la compleja interacción entre los diferentes elementos de la arquitectura de comunicación, sería muy conveniente poder analizar el comportamiento del sistema completo a través de una plataforma que no sólo simule el comportamiento de la red, el microprocesador o la tarjeta de red, sino también del sistema de entrada/salida y todo ello de forma conjunta, configurando un sistema completo en el que además se pueda ejecutar un sistema operativo junto con las correspondientes aplicaciones comerciales. De esta forma se podrían analizar de forma más realista las propuestas de mejora al poderse tener en cuenta las interacciones entre los diferentes elementos de la arquitectura, el sistema operativo y la aplicación. Los *simuladores de sistema completo (full-system Simulator)* surgen con ese objetivo [MAG02].

Actualmente, existen diferentes simuladores de sistema completo como Simics [VIR08a], M5 [M5S07], SimOS [ROS97] y los simuladores basados en Simics, GEMS [MAR05] y TFsim [MAU02], cuyas características más importantes se describen en la sección 2.1. Entre los simuladores de sistema completo actuales, para el trabajo de investigación que se presenta en esta memoria se ha elegido el simulador Simics [VIR08a], debido a una serie de características que lo hacen más apropiado que otros simuladores de sistema completo a los que nos hemos referido. Sin embargo, Simics [VIR08a] también presenta ciertas limitaciones que deben ser superadas para poder utilizarlo en la evaluación de las prestaciones de las arquitecturas de comunicación. Así, es preciso incluir modelos de temporización en los modelos de simulación de las arquitecturas propuestas con el fin de poder describir el comportamiento temporal del sistema y disponer además de la posibilidad de configurar ciertos parámetros como el tiempo de acceso a la memoria desde diferentes dispositivos. En este sentido se han elaborado dos modelos de simulación que incorporan los modelos de temporización antes mencionados, para simular el comportamiento de un sistema sin externalización y otro con externalización y, de esta manera, comparar los resultados obtenidos y evaluar las mejoras proporcionadas por la externalización. Utilizando dichos modelos, se

ejecutan pruebas de rendimiento (*benchmarks*) para proporcionar medidas de ancho de banda y latencia así como de tiempos de CPU en ambos casos (no-externalizado y externalizado).

Entre los programas de prueba o benchmarks para comunicación a través de TCP/IP, están Netpipe [SNE96], Netperf [NET07b], SpecWeb [SPE05], TTCP [TTC05], NetSpec [NET07d] o Hpcbench [HUA05], este último diseñado para realizar medidas en entornos de altas prestaciones. Dichos *benchmarks* se describen con más detenimiento en el capítulo 3. A la hora de elegir el conjunto de *benchmarks* a utilizar, hay que tener en cuenta que para comprobar la validez de las medidas y evaluar las prestaciones en las propuestas realizadas, resulta conveniente utilizar *benchmarks* que hayan sido ampliamente utilizados en otras propuestas.

En este capítulo se presenta la técnica de simulación de sistema completo para evalluar las arquitecturas de comunicación, describiendo los principales simuladores de sistema completo comerciales o de libre distribución actuales. Después, se describe con más detalle el simulador Simics, ya que como se ha comentado, presenta ciertas ventajas frente a otros simuladores que hacen que sea el elegido para obtener los resultados presentado en esta memoria. Dicha descripción de Simics abarca tanto sus características como las dificultades a la hora de simular con el mismo arquitecturas de comunicación. Posteriormente, se presentan los modelos de simulación que se han elaborado para evaluar las propuestas presentadas con el fin de mejorar las prestaciones de comunicación, utilizando los recursos de Simics y los modelos de temporiación o interfaces de modelado del tiempo desarrollados que permiten externder el comporamiento funcional de los modelos incorporando a los mismos un comportamiento temporal.

2.1 La Simulación de Sistema Completo

Generalmente, el método más utilizado para evaluar una arquitectura de comunicación es la simulación. Esta técnica permite la rápida modificación de parámetros software o incluso detalles del hardware del sistema con un coste mínimo [TRA94, WAN99, SKA03, WIL04]. De hecho, esta es la técnica comúnmente utilizada para la investigación en arquitecturas de computadores y de sistemas de comunicaciones. Mediante la simulación, deben poderse reproducir las características del un sistema real con la fidelidad necesaria. Así, normalmente se ha de llegar a un

compromiso entre precisión/detalle de la simulación y tiempo/coste en cuanto a recursos del sistema utilizados por la misma. Se trata de obtener resultados fiables en un tiempo lo suficientemente reducido como para que las simulaciones puedan ser útiles.

La simulación del comportamiento de arquitecturas de computadores es una técnica que se plantea en los años 50 [MAG02] y en la actualidad se utiliza de forma habitual para caracterizar el comportamiento de los diferentes componentes de un computador (microprocesador, comportamiento de *cache*s, buses de E/S, etc.). Si lo que se persigue es reproducir fielmente el comportamiento de todos los elementos que intergan un sistema real (no sólo el procesador y la memoria), se necesita un simulador de sistema completo (*Full-System Simulator)* [ENG03]. Es decir, un simulador que abarque tanto los elementos hardware de la arquitectura como el sistema operativo que se ejecuta y las aplicaciones que se utilizan como *benchmarks*.

En los últimos años se han popularizado los emuladores como VMWare [VMW07], QEMU [QEM07] o VirtualPC [VPC07], que se han utilizado incluso para poder disponer de varios sistemas operativos en un mismo computador. Se trata de lo que viene denominándose *virtualización*, o *máquinas virtuales*. Dichos emuladores suelen disponer de unas posibilidades limitadas para la configuración de las máquinas ya que, habitualmente, su objetivo es la evaluación o desarrollo de software sobre diferentes plataformas. En las plataformas en las que se emula otra máquina (virtual), que trata de reproducir la funcionalidad de una máquina real, se denomina máquina o sistema objetivo *(target)* al emulado y máquina o sistema huésped *(host)* al computador real donde se ejecuta la plataforma de emulación. Se trata de que, desde el punto de vista funcional, el sistema virtual se comporte con las mínimas diferencias con el sistema real. Esto permite la ejecución y evaluación de software comercial, sin la necesidad de disponer de la máquina real correspondiente. Sin embargo, las plataformas de máquinas virtuales diseñadas para este propósito, aunque son funcionalmente iguales al sistema real, están diseñadas para que la ejecución de las aplicaciones se realice de la forma más eficiente posible, sin reproducir el comportamiento temporal del sistema real. Además, no incluyen la posibilidad de modificar dicho comportamiento temporal, cambiar parámetros del hardware ni modelar nuevos elementos hardware que puedan ser conectados al sistema. Por tanto, este tipo de plataformas o proporcionan la precisión necesaria para predecir el rendimiento de una máquina real, por lo que, si bien, este tipo de plataformas pueden servir para la ejecución de software comercial, no

son válidas a la hora de investigar nuevas arquitecturas de computadores y estudiar su comportamiento y sus prestaciones.

Así, mientras que un *emulador* debe ejecutar una aplicación lo más rápidamente y sólo desde el punto de vista funcional, un simulador está diseñado para proporcionar información con el máximo nivel de detalle y precisión posible tanto de la ejecución de la aplicación como del funcionamiento del hardware simulado. Sin embargo, no siempre es fácil disponer de un simulador que nos permita reproducir fielmente la realidad. Para ello se necesitaría un simulador de sistema completo perfecto, que se comportara como un sistema real en cuanto a prestaciones y total precisión en cuanto a los detalles de la arquitectura y del hardware. Sin embargo, estas dos características son contradictorias: si se utiliza un simulador con mucha precisión, la ejecución de las aplicaciones será mucho más lenta. Habrá que llegar a un compromiso entre las prestaciones del simulador y su precisión. Además, si el simulador se pretende utilizar para investigar en arquitecturas de computador, hará falta disponer de ciertas características de configuración e inspección especiales. Es decir, debe ser posible configurar sistemas a medida, disponer de diferentes recursos hardware, inspeccionar en tiempo real el contenido de registros o posiciones de memoria, etc.

Concretamente, en nuestro caso, un simulador que reproduzca el sistema real debe tener las siguientes características [MAG02]:

1) *Configuración a medida de los sistemas simulados:* El simulador debe permitir la configuración de una máquina con elementos específicos como procesadores, memoria, interfaces de red y otros elementos hardware, así como la configuración de conexiones de red entre dispositivos con latencias prefijadas (que, por ejemplo, puede servir para simular la conexión entre dos máquinas que están a una distancia determinada).

2) *Simulación de sistemas multiprocesador:* Debe ser posible configurar un sistema con varios procesadores (por ejemplo, un SMP) así como modificar las características de los buses del sistema. Además, debe ser posible cambiar aquellos parámetros de los procesadores que afecten a la velocidad de ejecución de instrucciones tales como la frecuencia de reloj, o el número de ciclos por instrucción, etc.

3) *Simulación de redes de computadores:* debe poderse simular redes de computadores de área local ya que usualmente los clusters de computadores se configuran a partir de este ripo de redes. Además, debe ser posible simular diferentes interfaces de red comerciales y estándar, que se utilicen en sistemas reales, y cuyas características puedan configurarse o modificarse. En caso de simular comunicación entre más de dos máquinas interconectadas en la misma subred, o en subredes diferentes, sería necesario disponer, además, de un módulo que haga las funciones de encaminador *(router)*.

4) *Simulación de arquitecturas estándar:* Debe disponerse de modelos de simulación de todos los elementos hardware incluidos en arquitecturas estándar, es decir, arquitecturas compatibles con sistemas operativos comerciales sobre los que puedan ejecutarse aplicaciones comerciales. Esto implica, por ejemplo, disponer de modelos de simulación de buses estándar como *PCI* o *ISA,* de relojes de tiempo real, y controladores de interrupción como los elementos *APIC (Advanced Programmable Interrupt Controller)*. Por otra parte, los procesadores simulados, deben proporcionar soporte al núcleo de los sistemas operativos comerciales. Es decir, deben ser capaces de ejecutar las mismas instrucciones que el procesador real.

5) *Simulación eficiente:* Para que el simulador resulte útil en la práctica, la ejecución simulada de una aplicación no debería ocupar mucho más tiempo del que ocuparía en el sistema real. Lo ideal sería que las simulaciones fueran más rápidas que la ejecución real. Esta es una situación bastante difícil de alcanzar y depende del nivel de detalle que se necesite. Normalmente, para la simulación de los elementos hardware del sistema, se requiere gran tiempo de CPU en el *host* que ejecuta el simulador, y por supuesto, dicho tiempo aumentará cuanto mayor sea la complejidad de la arquitectura simulada.

6) *Definición de nuevos elementos hardware:* Además de los modelos de elementos hardware de que disponga el simulador, sería deseable poseer alguna herramienta para implementar modelos de simulación de dispositivos, a través de algún lenguaje de descripción hardware como podría ser VHDL [VHD93] o cualquier otro lenguaje de descripción estándar como Verilog [VER95] o

incluso lenguaje C/C++ [ISO03]. De esta forma, no se estará limitado a los modelos de elementos hardware con que cuente el simulador sino que se podrán construir modelos de dispositivos o interfaces comerciales, haciéndolos compatibles con los *drivers* del sistema operativo suministrados por el fabricante, o incluso diseñar dispositivos originales y programar sus controladores *(drivers)* para el sistema operativo. Esta característica es muy importante, ya que no es raro que no exista el hardware que queremos simular (debido a que lo estamos diseñando) o no se dispone del mismo en un sistema real.

7) *Herramientas de inspección y caracterización:* Cuando se está realizando la simulación o al finalizarla, es necesario disponer de herramientas que permitan inspeccionar el estado de la máquinatal y como se contempla desde el simulador. Es decir, se precisa acceder en *tiempo de ejecución* al estado de registros internos del microprocesador, conocer el contenido de posiciones de memoria concretas o incluso modificarlas, leer los registros de los dispositivos o conocer estadísticas de ejecución tales como el número de instrucciones ejecutadas, el número de ciclos totales o el tiempo transcurrido. En cuanto a la simulación de arquitecturas de comunicación, puede ser muy útil conocer el número de paquetes o bytes transmitidos/recibidos por la interfaz o que se han desechado, etc. Por tanto, es importante la capacidad para acceder a cualquier tipo de información que pueda ser útil para analizar el funcionamiento de la arquitectura simulada.

En definitiva, necesitamos un simulador de sistema completo con gran flexibilidad a la hora de definir la arquitectura a simular. Solo un simulador que cumpla de manera satisfactoria las siete características anteriores premite evaluar las prestaciones de una arquitectura sobre una aplicación concreta, de una forma suficientemente precisa, y sobre todo, proporcionar medidas que muestren claramente el comportamiento del sistema real.

En la actualidad, no existen muchos simuladores que cumplan los requisitos descritos más arriba. De hecho, por diferentes motivos (un modelo incompleto del sistema a simular, simulación demasiado lenta, etc.). Los simuladores utilizados en éste

ámbito suelen presentar limitaciones que se intentan soslayar de diferentes maneras [MAG02].

Algunos ejemplos de simuladores de sistema completo que pueden utilizarse para evaluar el sistema de entrada/salida y las arquitecturas de comunicación son:

- **M5 [M5S07].** Simulador de sistema completo de código abierto, desarrollado por la Universidad de Michigan, con la colaboración de empresas como IBM, Lucent Technologies o Intel. M5 proporciona dos modelos de CPU Alpha. Un primer modelo, más simple (SimpleCPU), que se utiliza para una ejecución rápida, aunque no permite realizar medidas ni recoger estadísticas, dado que se corresponde con un modelo de CPU con ejecución ordenada y sin segmentación de cauce. El segundo modelo de CPU (O3CPU) incluye segmentación de cauce y la posibilidad de ejecución desordenada de instrucciones máquina, así como la ejecución con multihebra simultánea (*SMT*). Sin embargo, la simulación utilizando este modelo es mucho más lenta. M5 también premite la simulación de redes de interconexión entre maquinas, aunque para ello, son necesarios diferentes módulos que deben realizarse a medida para cada simulación concreta.
- **SimOS [ROS97].** Es otro simulador de sistema completo que incluye modelos de los procesadores MIPS R4000, R10000 y Alpha. No dispone de versiones posteriores al año 1998, por lo que el código no está optimizado para las arquitecturas actuales. Esto hace que la simulación con cargas de trabajo reales sea muy lenta. Por otro lado, no dispone de modelos de interfaces de red detallados. Además, sólo permite la simulación de arquitecturas MIPS y Alpha como se ha comentado más arriba.
- **Simuladores basados en Simics [MAG03, VIR08a].** Entre estos simuladores están GEMS [MAR05] y TFsim [MAU02]. En realidad se trata de módulos de propósito específico creados para ser utilizados con Simics y evaluar arquitecturas multiprocesador y aplicaciones comerciales, respectivamente, superando algunas de las limitaciones de Simics a este respecto.

Entre los simuladores de sistema completo a que se he hecho referencia, Simics es el que mejor se adapta a nuestras necesidades y más se acerca a los siete requisitos que se han considerado antes. Otros simuladores como M5 [MAR05] o SimOS

[ROS97] que, o bien disponen ya de herramientas para la simulación de redes de forma detallada (el caso de M5 [MAR05]), o bien se pueden extender para ello (el caso de SimOS [ROS97]), conllevan de un gran esfuerzo si se quiere simular nuevas arquitecturas o modificar las existentes (al menos, un esfuerzo mucho mayor que el necesario para realizar lo mismo con Simics) al no disponer de la flexibilidad en la configuración o de ciertos bloques que proporciona Simics. Además, como más adelante se detallará, Simics es un simulador dirigido por eventos. Es decir, esto hace posible la simulación en tiempo real ó incluso más rápido que en tiempo real, según la arquitectura que se esté simulando y de la precisión que se requiera. Simics sólo consume tiempo de CPU cuando se producen eventos que hacen progresar la simulación (llegada de una interrupción, ejecución de una instrucción máquina, etc.).

En resumen, la utilización de simuladores, para desarrollar nuevas arquitecturas es la técnica normalmente utilizada, pero hay que tener en cuenta que el simulador que se utilice debe incluir o permitir la implementación de modelos del sistema a simular, con la precisión necesaria en cada caso. A continuación se describen las características más importantes de Simics, así como su utilización para la simulación de las arquitecturas de comunicación, cuyo estudio es el objetivo de este trabajo. Tal y como se comentará en la sección 2.2, aunque Simics proporciona la mayoría de las características necesarias para la evaluación de arquitecturas de sistemas de comunicaciones, tiene ciertas limitaciones que deben ser superadas.

2.2 El simulador Simics

Simics es un simulador de sistema completo desarrollado por la empresa Virtutech [VIR08a], surgida a partir de un *spin-off* del Swedish Institute of Computer Science (SICS) de Estocolmo. Simics es el producto principal que ofrece dicha empresa, y actualmente esta siendo utilizado por diferentes equipos de desarrolladores tanto de software como de hardware en diferentes paises del mundo [VIR08b]. Virtutech proporciona una versión académica gratuita para el uso de Simics en la enseñanza y en la investigación, que incluye todas las funciones del simulador, así como el soporte completo para las plataformas Pentium-4 de Intel entre otras [VIR08a].

Las utilidades de Simics para desarrollar, depurar y probar los sistemas, se han hecho muy populares en los últimos años, ocasionando que Simics sea una herramienta bastante común en empresas de sectores tales como los servidores para computación de

altas prestaciones, el diseño de hardware de red, los vehículos aeroespaciales y militares, y los automóviles [VIR08a], ya que permite a los programadores el desarrollo de *firmware* y software antes de que el hardware esté disponible. Entre las empresas que actualmente utilizan productos de Virtutech para la simulación están Cisco [CIS07], Ericsson [ERI07], Freescale [FRE07], Honeywell [HON07], IBM [IBM07] y General Electric Aviation [GEA07]. En 2005, Simics fue seleccionado por IBM para el desarrollo de la plataforma POWER6 [IBM07].

Simics [VIR08a] proporciona las herramientas necesarias para la simulación de sistema completo, incluida la simulación de elementos hardware y la interconexión de los mismos con bloques o módulos hechos a medida. Además, los sistemas simulados pueden estar basados en nueve arquitecturas diferentes (ver Apéndice 1), tales como arquitectura monoprocesador, y arquitectura multiprocesador simétrico o SMP. Además, proporciona modelos de hardware que hacen que el software, tanto en lo referente al sistema operativo como a las aplicaciones que se ejecutan sobre él, no note ninguna diferencia entre el sistema simulado y un sistema real. Además, Simics está diseñado para proporcionar el compromiso entre prestaciones y precisión a que se ha hecho referencia en la introducción de este capítulo, de forma que se disponga de las prestaciones y características funcionales suficientes para poder ejecutar aplicaciones reales, así como una precisión suficiente para poder simular modelos hardware detallados.

Mediante Simics, por tanto, además de simular el hardware, también se puede simular la ejecución de aplicaciones, sistemas operativo (como Solaris, Linux, Tru64 ó WindowsXP), y los controladores *(drivers)* y protocolos de comunicación de forma lo suficientemente rápida como para poder evaluar *benchmark*s o aplicaciones comerciales. De hecho, la capacidad de Simics [VIR08a] para poder utilizar cargas de trabajo comerciales es una de las características que lo diferencian de otros simuladores y que lo convierten en el simulador más apropiado en muchos entornos.

Sin embargo, Simics es en si un simulador funcional y su modelo de temporización puede resultar insuficiente para algunos propósitos. En [VIL05], se describen algunas de las posibilidades de Simics utilizando procesadores X86, y la simulación de computadores cc-NUMA con modelos precisos de caché. En esos casos se pone de manifiesto que la funcionalidad de Simics debería extenderse para poder evaluar aplicaciones reales. Además, como se ha dicho, Simics es un simulador dirigido

por eventos, lo que hace que, por defecto, el modelo de tiempo no sea lineal. De hecho, en [ALA03] se extiende Simics con modelos de temporización detallados.

Al añadir modelos de temporización a Simics, es posible modificar el comportamiento por defecto de Simics. En dicho comportamiento, la ejecución de un paso no supone tiempo de ejecución, y los pasos se ejecutan en el orden del programa. Este modelo se conoce como *modelo de Simics ordenado*, en el que las instrucciones son ejecutadas secuencialmente y de forma discreta. Se hace posible así una simulación funcional muy rápida, pero no se modela bien el comportamiento temporal que se tiene en los sistemas reales, donde hay una determinada latencia asociada a cada transacción. Este comportamiento por defecto, puede evitarse utilizando modelos de temporización, para que se pueda tener control sobre la temporización de las transacciones ya que los pasos dejan de ser operaciones atómicas que no consumen tiempo. Los modelos de temporización se implementan en Simics como interfaces (interfaces de modelado del tiempo) que pueden conectarse a los diferentes elementos del sistema. Cuando un objeto que tiene conectado un modelo de temporización realiza una transacción de memoria, ésta se lleva a cabo en un tiempo preestablecido que viene dado por un número de ciclos de reloj. Por otro lado, Simics permite ejecución desordenada para procesadores UltraSPARC y x86, aunque de forma experimental.

Otra de las características que diferencian Simcs de otros simuladores, es la posibilidad de definir hardware a medida mediante el lenguaje de descripción de hardware *DML* (Device Modeling Language) [DML07]. Así es posible desarrollar software y hardware conjuntamente.

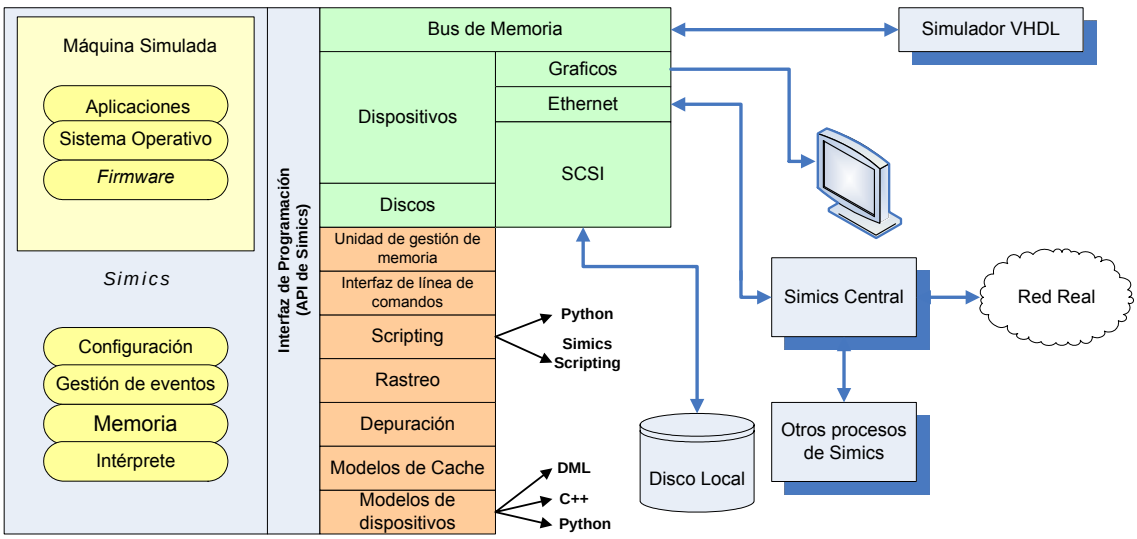


Figura 2.1. Esquema funcional de Simics [MAG02]

En la Figura 2.1 se muestra el esquema modular y funcional de Simics y la interacción entre los distintos módulos que lo componen. El nivel más alto se corresponde con la parte izquierda de la Figura 2.1, en la que está la máquina simulada que está ejecutando aplicaciones, el sistema operativo y el firmware que controla el hardware, como ocurre en un sistema real. Simics configura la máquina simulada y realiza la gestión de eventos de la memoria, además de proporcionar un intérprete de comandos. Por otro lado, los dispositivos conectados al bus de memoria utilizan la API (Application Programming Interface) de Simics para interaccionar con la máquina Simulada.La API es, además, la interfaz que utilizan las utilidades de rastreo o depuración, así como el interfaz de línea de comandos o los dispositivos que pueden modelarse y conectarse a la máquina simulada (por ejemplo, los modelos de memoria *cache*). Existe también la posibilidad de conectar un simulador VHDL a través del bus de memoria con el fin de modelar ciertos dispositivos sencillos mediante este lenguaje de descripción hardware.

2.3 Simulación de la comunicación con Simics

Como se ha indicado en la sección 2.2, Simics [VIR08a] es un simulador de sistema completo comercial, que proporciona modelos de hardware que hacen que el software no note ninguna diferencia entre el sistema simulado y un sistema real. Así, mediante Simics, podemos simular el hardware ejecutando aplicaciones, el sistema operativo, los *drivers* y los protocolos de comunicación. Como se ha dicho, Simics no proporciona un modelo de temporización preciso por defecto, aunque es un simulador funcional rápido que permite ser configurado a medida, y que entre otras herramientas dispone de la posibilidad de conexión de modelos de temporización específicas.

Para utilizar Simics en el estudio de las arquitecturas de comunicación y evaluar las prestaciones que proporcionan las mejoras que se pueden proponer con Simics, se necesita un modelo de interfaz de red en el que el procesamiento de los protocolos de comunicación puedan tener lugar tanto en la CPU del nodo como en otros procesadores disponibles, como por ejemplo en la tarjeta de red. De esta forma se podrá simular distintas opciones de externalización. Así, las principales limitaciones que presenta Simics de cara a simular la arquitectura de comunicación, y que habrá que tener en cuenta para nuestros propósitos son:

- Las redes se simulan a nivel de paquete y las transacciones se realizan como un evento (ya se ha indicado que Simics es un simulador dirigido por eventos). Por tanto, no se simulan los detalles en la transacción de paquetes, es decir, el envío detallado de los bytes individuales: la transacción completa se simula como una acción. Dado que la red y los dispositivos de E/S se simulan mediante transacciones, existe un importante inconveniente en las simulaciones de los procesos de comunicación, donde se necesitan modelos de temporización detallados de las transferencias de DMA asociadas a los paquetes que llegan y que deben enviarse.
- El ancho de banda del enlace simulado puede ser teóricamente infinito, pero en la práctica, la simulación de anchos de banda superiores a 1Gbps requieren un tiempo elevado al mismo tiempo que no se obtienen resultados realistas. Aunque el ancho de banda del enlace puede ser infinito, actualmente Simics no proporciona modelos de interfaces de red que soporten anchos de banda superiores a 1Gbps. No obstante, a partir de los resultados obtenidos con para 1Gbps, se pueden extraer conclusiones extensibles a anchos de banda mayores teniendo en cuenta las relaciones entre el ancho de banda del enlace, la frecuencia de reloj de los procesadores y los anchos de banda de los buses de E/S y memoria del nodo.
- Utilizando *slots* o intervalos de simulación pequeños para la conmutación entre procesadores en una simulación multiprocesador y multimáquina, el tiempo de simulación se incrementa. Dicho tiempo de conmutación entre procesadores es un parámetro configurable y deberá tomar valores bajos para obtener resultados de simulación lo más realistas posibles. De hecho, en las simulaciones que se han realizado en esta memoria, se ha utilizado el mínimo valor para el tiempo de conmutación de 1 (*cpu_switch_time=1*), para que el comportamiento del sistema simulado sea lo más parecido a la realidad posible, lo que significa que el simulador ejecutará solamente una instrucción seguida en cada procesador.
- Simics proporciona diferentes herramientas para construir modelos de simulación a nivel de hardware como se ha visto en la Sección 2.2. Sin embargo, presenta algunas limitaciones que pueden salvarse, por ejemplo,

mediante la simulación en modo *stall*, que permite modelar el sistema con determinadas latencias o tiempos de acceso, dotando al modelo de simulación de un comportamiento temporal determinado.

A pesar de estas limitaciones, consideramos que Simics sigue siendo preferible a otros simuladores como M5 o SimOS por la flexibilidad que tiene a la hora de cambiar los parámetros de simulación, crear modelos hardware, o incluir modelos de una gran variedad de procesadores, además del soporte técnico con el que se puede contar actualmente. De hecho, en Simics, mediante el lenguaje de descripción *DML* se puede modelar cualquier dispositivo hardware. Además, la depuración en Simics es mucho más fácil que con otros simuladores como M5 o SimOS ya que proporciona herramientas efectivas para la depuración y el desarrollo. Los simuladores GEMS [MAR05] y TFSim [MAU02] están basados en Simics y proporcionan modelos de temporización detallados, enfocados a la simulación de arquitecturas concretas. Por otro lado, Virtutech proporciona un efectivo soporte, que permite la resolución de fallos en el software *(bugs)* de forma rápida, cuestión muy importante ya que se trata de una herramienta en contínua evolución.

Teniendo en cuenta las cuestiones anteriores, hemos construido un modelo de simulación configurando dos máquinas *a medida* conectadas mediante un enlace Ethernet estándar. En la simulación se ha evitado el uso del módulo *Simics Central* [VIR08a]. Se trata de un módulo de Simics que actúa como encaminador para interconectar máquinas que se estén simulando en diferentes instancias de Simics o incluso para proporcionar conectividad con la red Ethernet real del host. Aunque en nuestro caso no es imprescindible la utilización del módulo Simics Central, este es necesario en el caso de que se quiera realizar una simulación distribuida, teniendo varias instancias de Simics interconectadas ejecutándose en *hosts* diferentes, que incluso pueden estar conectados a través de Internet. Además, en los modelos de simulación diseñados, se han incluido modelos de temporización que permiten simular las latencias que existen entre las CPUs, los dispositivos y la memoria que por defecto Simics no tiene en cuenta. Por otra parte, como se vera en el Capitulo 3, dedicado al modelado y evaluación de las técnicas de externalización de protocolos estudiadas en la presentación de esta memoria, se ha modificado el perfil de interrupciones, para reproducir el comportamiento del sistema de comunicación en sus distintas alternativas.

A continuación, se describen los modelos funcionales utilizados en las simulaciones y posteriormente, los modelos de simulación que hacen posible la utilización de dichos modelos funcionales para la evaluación del subsistema de comunicación.

2.4 Modelos funcionales

Para realizar simulaciones de las arquitecturas de comunicación se han configurado en Simics dos máquinas idénticas, interconectadas a través de una red Ethernet. Al configurar estas máquinas se intenta minimizar la carga de trabajo generada en el computador de forma que la simulación se realice lo más rápidamente posible. No se trata de algo irrelevante, sobre todo si se tiene en cuenta vamos a llevar al límite las posibilidades de Simics para simular enlaces de red con anchos de banda elevados para evaluar las prestaciones observadas tras cada modificación o propuesta en la arquitectura de comunicación.

En el esquema de la Figura 2.1, puede verse la arquitectura de cada una de las instancias de computador monoprocesador simuladas en Simics.

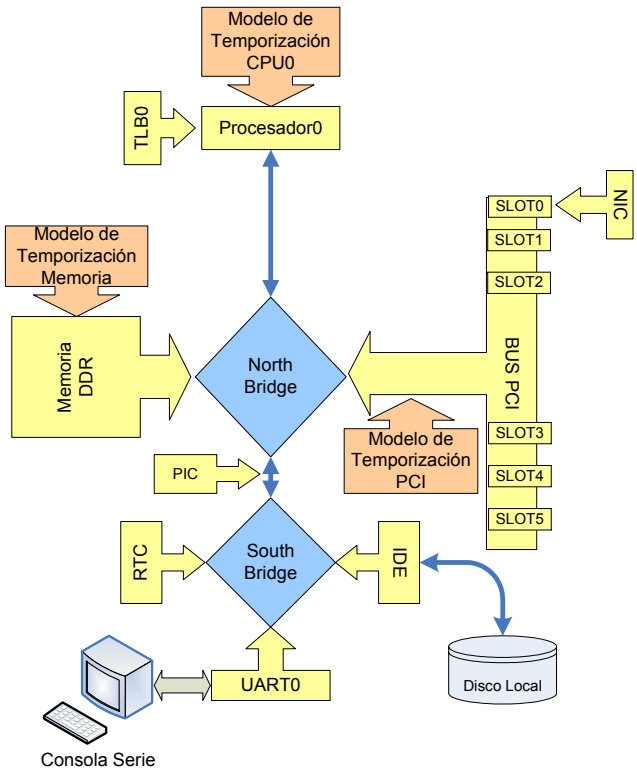


Figura 2.1. Modelo funcional de la arquitectura monoprocesador en Simics

Se trata de máquinas basadas en un procesador superescalar, concretamente en el Pentium-4 de Intel [INT07]. El chipset utilizado es el Intel X86-440BX [INT99] que permite la gestión de sistemas multiprocesador de hasta 16 CPUs. El chipset considerado está compuesto por un puente norte, al cual se conecta el bus de memoria DDR y el bus PCI, un puente sur al que se conecta la interfaz IDE, que se utilizará como interfaz con el disco, un reloj de tiempo real y una UART para proporcionar la conectividad serie necesaria para la consola de texto. Además, dispone de un controlador programable de interrupciones (*APIC*) [INT01]. La CPU además, se conecta un buffer TLB *(Translation Lookaside Buffer)* de 4MBytes. Como puede verse, se configura un sistema bastante parecido a uno real, donde será posible instalar un sistema operativo comercial (Linux en nuestro caso) y ejecutar aplicaciones reales que constituyen nuestro conjunto de *benchmarks* descrito en la sección 4.1.

En la Figura 2.1 se pone de manifiesto que hay modelos de temporización conectados a la CPU, la memoria y el bus PCI. A través de estos modelos de temporización se pretende conseguir:

1) Modelar las latencias de memoria
2) Paralizar un número determinado de ciclos de reloj las transferencias desde o hacia memoria para simular un determinado tiempo de acceso.

En el caso de disponer de más de una CPU como es el caso de los sistemas *SMP (Symmetric Multiprocessors)*, podremos conectar modelos de temporización diferentes a cada CPU para simular diferencias de latencia en los accesos a memoria según la CPU.

En Simics, todos los elementos actúan por defecto solo como conectores (Apéndice 1) proporcionando solo la infraestructura y las funciones necesarias para que no existan diferencias a nivel funcional entre la *máquina virtual* de Simics y una real. Sin embargo, nuestro objetivo no es solo el estudio funcional sino también el de prestaciones.

La optimización del tiempo de simulación es importante, ya que en las simulaciones que se han realizado para obtener los resultados mostrados en el Capítulo 4, se han simulando dos arquitectuas biprocesador Pentium-4, que componen dos máquinas *SMP* biprocesador junto con los demás componentes asociados. Así, es importante destacar que al conectar modelos de temporización a los elementos, las simulaciones generan mucha más arga de trabajo en el computador donde se ejecuta la

simuácion y todavía más si tenemos en cuenta que se están simulando varios procesadores Pentium-4 en un host de la misma generación. Esto puede suponer un incremento inaceptable del tiempo de simulación. Por tanto, como se ha comentado antes, es necesario configurar el simulador de forma que la carga generada en el host sea lo más pequeña posible, para que las simulaciones se realicen en el menor tiempo posible, pero manteniendo configuraciones que proporcionen resultados realistas. Para ello, se utiliza una única instancia de Simics. Para entender esto, hay que tener en cuenta que a la hora de simular mediante Simics varias máquinas conectadas en red, existen dos alternativas. La primera consiste en la ejecución de dos instancias de Simics y la configuración del módulo *Simics Central*. Ya se ha comentado que el módulo *Simics* Central actúa como encaminador, y puede utilizarse bien para conectar las máquinas simuladas con el mundo real, bien para proporcionar comunicación entre dos máquinas simuladas por instancias de Simics diferentes. Sin embargo, la ejecución del módulo *Simics Central* consume recursos del computador de simulación y por otro lado, la simulación de dos máquinas en dos instancias diferentes es más ineficiente. La otra alternativa consiste en la configuración de Simics para que las dos máquinas *virtuales* se ejecuten en una única instancia de Simcis. En este caso, la comunicación entre las dos máquinas no necesita del módulo encaminador, simulándose por tanto, una comunicación punto a punto. Nuestra configuración consiste en dos máquinas *virtuales* simuladas sobre una única instancia de Simics conectadas entre si mediante una red Ethernet, para lo cual no es necesario utilizar más de una instancia. Por otro lado, el módulo *Simics Central* introduce una latencia en el enlace de red que supone una limitación en la propia red Ethernet simulada. En esta memoria no se tratan posibles problemas que puedan aparecer en encaminadores, switches u otros elementos de la red, y la simulación en una única instancia permite simular una conexión punto a punto evitando problemas con el módulo encaminador.

En la Figura 2.2 se muestra la arquitectura simulada en el caso de una máquina con dos procesadores. Como puede verse en la Figura 2.1, se ha conectado a una ranura PCI una tarjeta de red Ethernet (NIC). Esa tarjeta proporciona la conectividad necesaria con la red Ethernet simulada. Como se muestra en la figura 2.3, el enlace entre las dos máquinas simuladas se realiza mediante el objeto *Ethernet-link* el cual simula un enlace de estándar Ethernet.

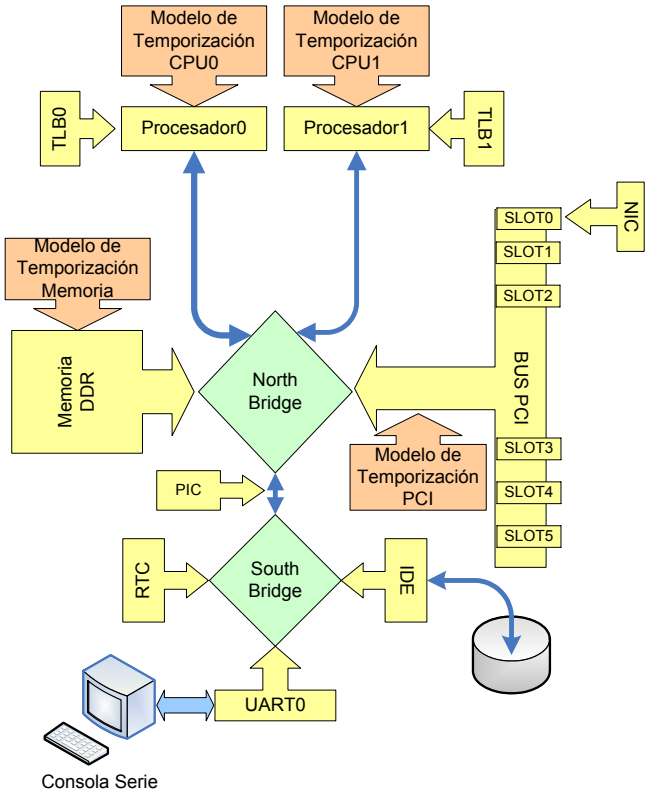


Figura 2.2. Modelo funcional de Arquitectura multiprocesador en Simics

Para la simulación eficiente mediante Simics de la externalización. En esta memoria nos basaremos fundamentalmente en el modelo de la Figura 2.2 y en los detalles que se van a considerar a continuación en la sección 2.5.

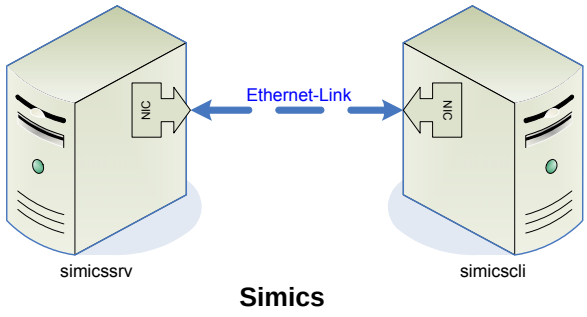


Figura 2.3. Red local

2.5 Modelos de Simulación desarrollados para arquitecturas de comunicación.

Los modelos de simulación se construyen a partir de los modelos funcionales descritos en el apartado 2.4 mas un modelo de red Ethernet estándar que conecta las máquinas simuladas entre si.

Uno de los requisitos de nuestro modelo de simulación es que sobre él se pueda ejecutar un sistema operativo real, en nuestro caso Linux versión 2.6. Se ha utilizado Linux 2.6, porque proporciona el soporte necesario para nuestra arquitectura a la vez que hace posible la implementación de los cambios que necesitamos para la simulación, y que permiten distribuir el software entre las CPUs del sistema, como se muestra en la sección 3.7.2. Para hacer que Linux se pueda ejecutar en nuestro modelo sin necesidad de modificar el *kernel* ni tener que programar un nuevo controlador (*driver*) a medida, se han aprovechado ciertas características de Simics que en un principio podrían considerarse desventajas o limitaciones del simulador. Como se ha comentado, por defecto, todos los buses en Simics se modelan simplemente como conectores. No obstante, como se ha dicho, es posible definir diferentes latencias en los accesos desde la CPU a dispositivos PCI y a la memoria mediante la inclusión de modelos de tiempo. Además, es posible incorporar diferentes retardos para interrupciones que provengan de diferentes dispositivos (otro dispositivo PCI, otro procesador, etc).

Para la evaluación de las distintas alternativas de mejora de la arquitectura de comunicació, necesitamos diferentes modelos que corresponden a configuraciones con diferentes alternativas de configuración en cuanto a la ubicación de la NIC y a la capacidad de cómputo, incluyendo la posibilidad de disponer de más de una CPU en el sistema. En la Sección 3.2 se describen los modelos alternativos para las distintas ubicaciones de la NIC en el sistema.

Para evaluar las prestaciones de una majora dada que se ha introducido en el sistema de comunicaciones, se han generado dos modelos de simulación. El primero corresponde a una arquitectura de simulación basada en un procesador Pentium-4 a 1 Ghz. En los experimentos realizados se ha visto que eliminando las limitaciones que suponen los buses de E/S del sistema, una frecuencia de reloj de 1 GHz es suficiente para tener 1 Gbps de ancho de banda y que la simulación no sea demasiado lenta. Además, se ha incluido una tarjeta de red Ethernet Gigabit BCM5703 PCI. En este

sistema base, la interfaz de red no dispone de procesador y la sobrecarga de comunicación recae sobre el procesador principal, que ejecuta la aplicación además de los protocolos de comunicación. Por tanto, mediante el modelo de la Figura 2.4, podemos determinar el máximo ancho de banda que se puede alcanzar con un procesador, constituyendo la configuración tomada como base o punto de partida sobre la que evaluar posibles mejoras.

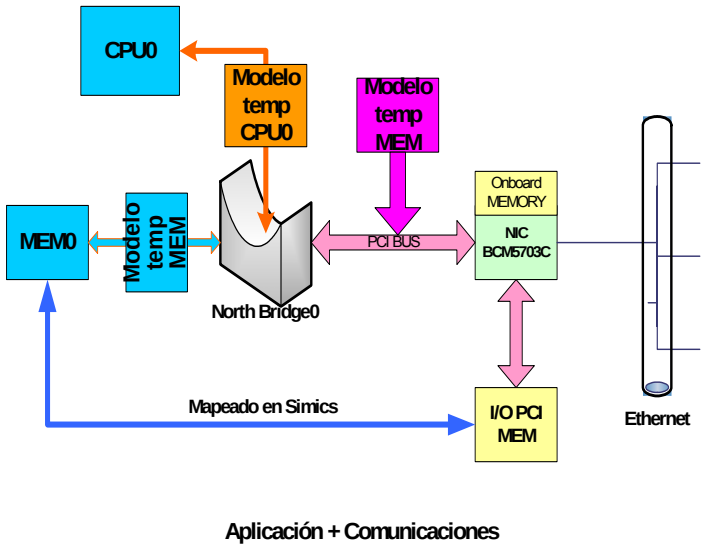


Figura 2.4. Modelo de simulación sin externalización

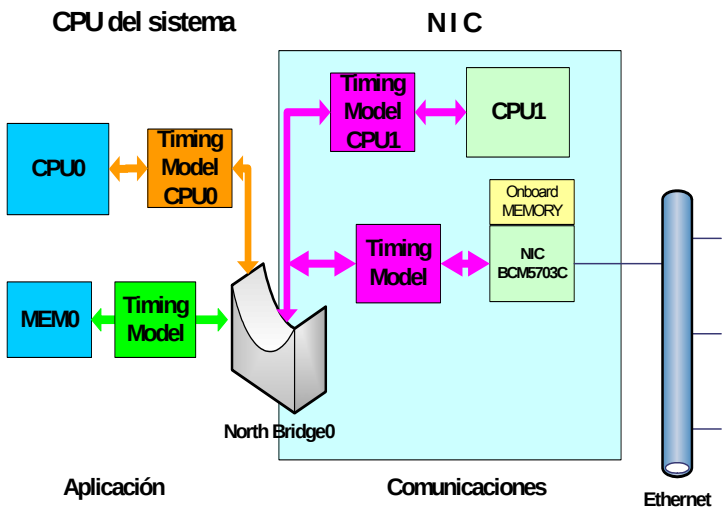


Figura 2.5. Modelo funcional para la simulación de la externalización

La Figura 2.5 muestra el modelo funcional utilizado para simular un nodo que dispone de una interfaz de red capaz de realizar parte del procesamiento de los protocolos de comunicación, absorbiendo así parte de la sobrecarga que en el sistema base de la Figura 2.4 recaía por completo en el procesador principal. Los detalles concretos relativos a las modificaciones en este modelo para simular las técnicas de externalización pueden verse en el Capitulo 3. En Figura 2.5 se muestran todos los elementos que se han incluido en el modelo de simulación para el caso de la externalización mediante *offloading* [ORT06a, ORT06b, ORT07a, ORT07b, ORT07c, ORT08a, ORT08b]. Así, El modelo de la Figura 2.5 incluye dos procesadores Pentium-4, un modulo de memoria DRAM de 128Mb, un bus APIC y un bus PCI con un interfaz de red Gigabit-Ethernet Tigon-3 (basado en el chip BCM5703C [BRO07]), así como una consola serie de texto.

Además, se han realizado modificaciones en el comportamiento de las interrupciones, como puede verse en la Figura 2.6. Para que el sistema operativo pueda ejecutarse con unas pequeñas modificaciones, como se describe en la sección 3.7, es necesario mantener todas las características funcionales del sistema y por tanto, no realizar modificaciones en el hardware que sean incompatibles con las versiones estándar del sistema operativo. Sin embargo, sí se pueden realizar cambios en el comportamiento de los componentes, como es el caso del controlador APIC que se comenta a continuación, sin modificar sus funciones.

Así, en cuanto al comportamiento de las interrupciones, se ha modificado el comportamiento temporal del APIC. Para simular la recepción directa en la CPU de las interrupciones de la interfaz de red, estas llegan a la CPU_{NIC} (CPU1) a través del *APIC*. Sin embargo, se éste se ha configurado de forma que no puedan llegar interrupciones a la CPU_{NIC}. Esto es posible mediante la configuración del APIC para que las interrupciones enmascarables correspondientes a la interfaz de red se redireccionen a la CPU_{NIC}. Además, las interrupciones que provienen de la interfaz de red alcanzarán la CPU_{NIC} con una latencia inferior a las interrupciones que puedan llegar a través del *APIC* a la CPU_0 desde otros dispositivos (IDE, RS232, etc.). Para ello, se configura el divisor de frecuencia entre la frecuencia de reloj utilizada en la CPU principal y la frecuencia de reloj utilizada por el bus APIC.

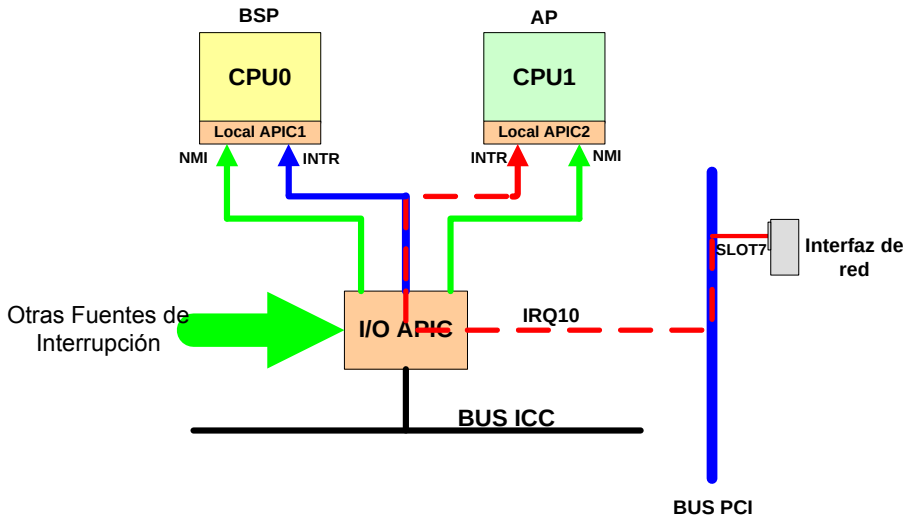


Figura 2.6. Modificación del perfil de interrupciones

Este modelo de puede utilizarse por tanto para :

a) Modelar el efecto de la latencia en los buses y de las colisiones en el puente norte

b) Modelar la interfaz de red con externalización de protocolos, al aislar la CPU que ejecuta los protocolos de comunicación del resto del sistema.

c) Simular la técnica la externalización de protocolos mediante Offloading.

De esta forma, podemos simular los sistemas mencionados, con la suficiente precisión. Existen otras alternativas que consisten en el uso de uno de los procesadores centrales en un SMP o CMP para realizar el procesamiento de los protocolos de comunicación. En este caso, también se han utilizado los modelos de temporización, pero no se ha modificado ni el perfil de interrupciones ni el comportamiento del controlador APIC no se ha modificado respecto al comportamiento en un sistema real.

Como se ha comentado, en el Capítulo 3 se describen detalladamente los modelos de simulación que se utilizan en el Capítulo 4 para obtener los resultados experimentales que nos permiten analizar distintas alternativas de diseño de las arquitecturas de comunicación. Además, en dicho capítulo se validarán los modelos comparando los resultados obtenidos con los que predice el modelo teórico LAWS [SHI03].

2.6 Modelado del tiempo en Simics

En la Sección 2.3 se ha mostrado que Simics presenta limitaciones en el modelado del comportamiento temporal del sistema simulado en cuanto al máximo ancho de banda de la red de interconexión simulada, y en relación con el tiempo de simulación, que hacen que no sea factible su uso directo en la evaluación de los subsistemas de E/S ni, por tanto, para el análisis de la externalización. Los modelos que se proponen en este apartado extienden la funcionalidad de Simics para evitar estos problemas. Así será posible analizar las prestaciones de cada una de las alternativas planteadas para analizar las distintas alternativas de implementación de la interfaz de red. Los resultados obtenidos se compararán con resultados correspondientes diferentes modelos que predicen el porcentaje de mejora que puede alcanzarse con la externalización del sistema de comunicación en base a diferentes parámetros.

2.6.1 Modelos de temporización

Los interfaces de modelado de tiempo implementan los modelos de temporización (*timing models*) que permiten especificar en Simics una determinada latencia para cada evento. De esta forma, es posible modificar el tiempo empleado por un objeto conectado a un *interfaz de modelado de tiempo* en completar transacción, indicando éste tiempo en términos del número de ciclos de reloj. Simics, por defecto relaciona el tiempo de ejecución con el número de instrucciones ejecutadas. Es decir, en el caso de ejecución ordenada de instrucciones, cada instrucción supondrá un número determinado de ciclos de reloj (cuya duración es configurable). En el caso de un sistema multiprocesador, aunque cada procesador tiene asignado un determinado tiempo durante el cual puede ejecutar instrucciones (también configurable), dado un intervalo de tiempo, todos los procesadores habrán ejecutado el mismo número de instrucciones. En el modo de ejecución desordenada de instrucciones no hay una correspondencia entre el número de ciclos transcurridos y el número de instrucciones ejecutadas.

En Simics, tanto los dispositivos como los procesadores pueden iniciar transacciones de memoria. Para saber a que objeto corresponde una determinada dirección física de memoria, Simics utiliza el concepto de espacio de memoria (*memory-space*). Un espacio de memoria asigna una dirección física a cualquier objeto

que pueda realizar una transacción. Esta dirección puede corresponder a una dirección de memoria RAM, memoria *flash* o de un dispositivo. De esta forma, cualquier acceso que se realice a un espacio de memoria se propaga automáticamente al destino correcto (dispositivo o memoria). Estos espacios de memoria pueden a su vez contener otros espacios de memoria creando una *jerarquía*.

Como se muestra en la Figura 3.6, la posibilidad de disponer de jerarquías de memoria, permite evitar la limitación inicial de Simics de no modelar latencias en los accesos, conectando modelos de temporización.

Internamente, Simics utiliza el ***S**imulator **T**ranslation **Cache*** o STC, con el fin de acelerar las simulaciones. El STC mantiene una tabla de traducción de direcciones con el fin de evitar pasar por toda la jerarquía de memoria. Sin embargo, el uso del STC hace que los modelos de temporización no se apliquen siempre. Estos solo se aplicarían en el caso de que la dirección de memoria dada (donde esté el dato a leer o donde haya que escribir) no esté almacenada en el STC. Por tanto, en nuestras simulaciones se desactiva el STC para conseguir un comportamiento temporal más real, aunque la simulación sea mucho más lenta. De esta forma, Simics se comporta internamente como muestra la Figura 3.6 en cuanto a la realización de transacciones.

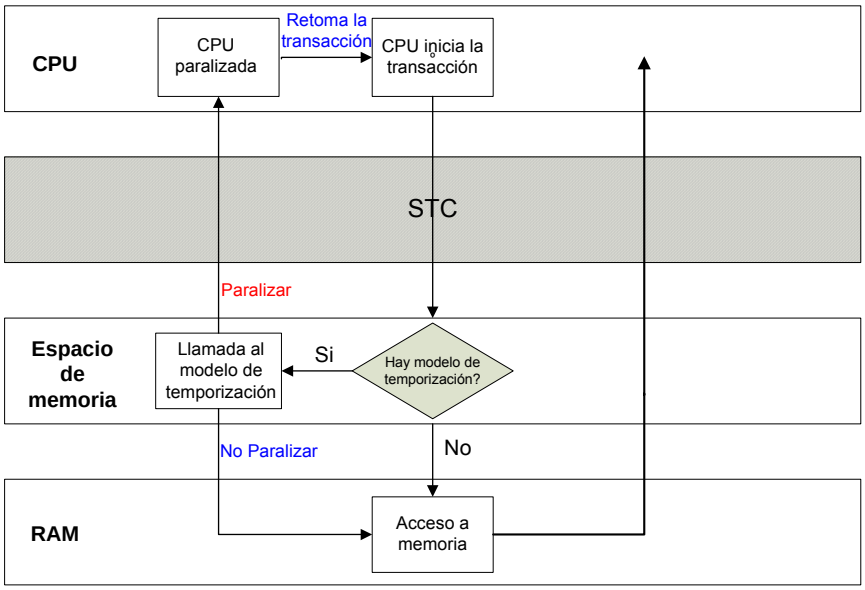


Figura 2.7. Transacciones en el sistema de memoria de Simics sin STC

Los modelos de temporización en Simics se programan a través de la función `timing_model_operate()` de la API de Simics, como puede verse en la Figura 2.8.

```
static cycles_t
timing_model_operate(conf_object_t *mem_hier,
                     conf_object_t *mem_space,
                     map_list_t *map_list,
                     generic_transaction_t *mem_op);
```

Figura 2.8. Función de propagación de los modelos de temporización

Los argumentos a pasar a la función `timing_model_operate()` son:

- *mem_hier* : modelo de temporización
- *mem_space*: espacio de memoria donde se va a conectar el modelo de temporización
- *map_list*: descriptor para el atributo que describe la operación de memoria
- *mem_op*: información sobre la operación de memoria actual

El uso de modelos de temporización evita la limitación inicial de Simics de ser solo un simulador funcional, permitiendo simular latencias y dotándolo de un modelado del comportamiento temporal. Así hemos desarrollado un modelo de temporización que se conectará a los dispositivos que realizan accesos a memoria: los procesadores del sistema y el bus PCI. De esta forma, aunque en principio no se tiene un modelo totalmente preciso de las colisiones, sí se consigue un comportamiento adecuado en las simulaciones, como muestran los resultados experimentales del Capítulo 4.

El esquema de funcionamiento del modelo de temporización que proponemos se muestra en la Figura 2.9.

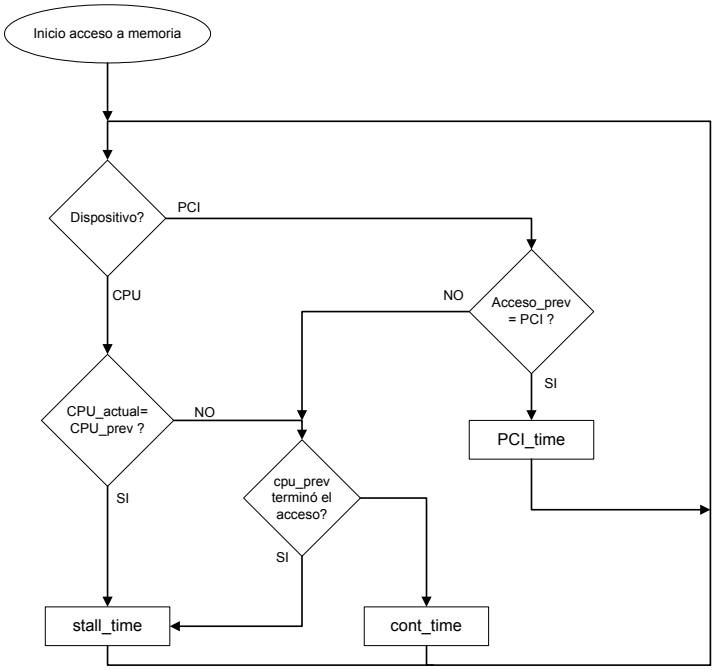


Figura 2.9. Diagrama de flujo del funcionamiento del modelo de temporización

Nuestro modelo, cuyos detalles de configuración se muestran en el Apéndice 2, se inicia en todos aquellos dispositivos a los que se haya conectado (como se ha dicho, procesador y la interfaz de red PCI) y, cuando estos intentan realizar un acceso a memoria, comprueba que no haya otro dispositivo utilizando el bus. En el caso de que haya otro dispositivo utilizando el bus, penaliza las transferencias con un número de ciclos determinado por el parámetro *cont_time*. En la Figura 2.10 se muestra un esquema del funcionamiento general del modelo de temporizacion frente a colisiones por accesos concurrentes a memoria. Mientras que un procesador utiliza el bus de memoria, los intentos de acceso a dicho bus por otro procesador o por un dispositivo provocan una colisión, conteniendose su acceso a memoria. Esta contención se traduce en una penalización de un número determinado de ciclos de reloj, configurable en el modelo de temporización desde la línea de comandos de Simics (Apéndice 2), hasta que los dispositivos puedan volver a intentar un nuevo acceso. Además, el modelo de temporización permite simular latencias en el bus PCI que, por defecto, en Simics es simplemente un conector. Esta latencia se configura a través del parámetro *pci_time*, expresándose en número de ciclos de reloj.

El modelo de temporización detecta por si mismo si el acceso a memoria proviene de un procesador o de un dispositivo PCI.

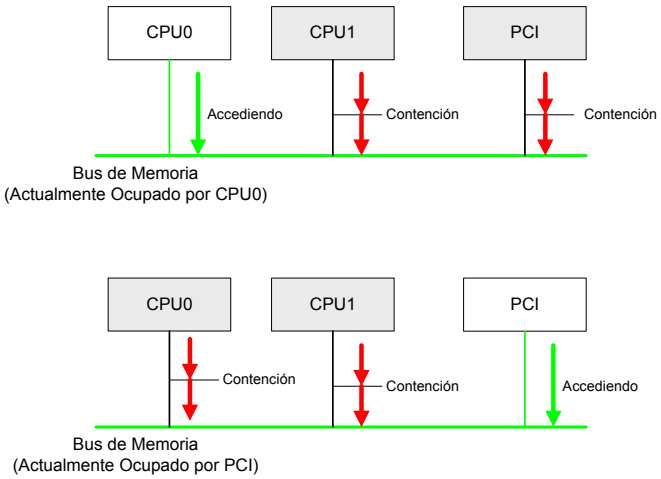


Figura 2.10. Funcionamiento del modelo de temporización

2.6 Resumen

En este capítulo se ha descrito las características que deben tener los simuladores para constituir alternativas útiles en el estudio y evaluación de las arquitecturas de comunicación. Concretamente, se ha puesto de manifiesto que la simulación de sistema completo nos permite, no solo simular el comportamiento del hardware del nodo computador, sino también incluir los efectos del sistema operativo como de las aplicaciones comerciales, y con ello obtener resultados realistas.

Se ha presentado Simics, el simulador de sistema completo utilizado para realizar nuestro trabajo experimental, poniendo de manifiesto sus ventajas y sus limitaciones para la simulación de las arquitecturas de comunicación. Precisamente, entre las aportaciones del trabajo que se presenta en esta memoria está la forma de evitar estas limitaciones y la construcción de modelos de simulación adecuados, que proporcionan un compromiso entre precisión y tiempo de simulación, permitiéndonos obtener resultados lo más parecidos a la realidad.

Capítulo 3

Propuestas para la externalización de la interfaz de red

En el Capítulo 1 (Sección 1.7) se ha definido la técnica de externalización como la implementación total o parcial de los protocolos de comunicación en un procesador distinto al procesador principal del nodo. En este capitulo se describe el uso de la externalización, así como dos de las principales alternativas que actualmente se pueden considerar para conseguir la externalización de los protocolos de comunicación y por tanto la liberación del procesador del sistema de la sobrecarga asociada al procesamiento de los mismos. Se trata de las técnicas de *offloading* y *onloading*. Se describe además el modelo utilizado para la simulación de la externalización en Simics, tanto en lo que se refiere a la configuración hardware simulada en Simics, como a la distribución del software en el sistema para proporcionar el soporte necesario a la alternativa simulada. Así, se describe el modelo de simulación desarrollado para la simulación de la externalización mediante *offloading* y *onloading*, respectivamente. Estos modelos se utilizarán en el Capítulo 4, para obtener los resultados experimentales y realizar una comparación de prestaciones de ambas técnicas de externalización.

En este capítulo se resume brevemente la problemática asociada al aprovechamiento del ancho de banda de los enlaces de red y concretamente los problemas asociados al uso del protocolo TCP/IP en cuanto a la sobrecarga generada en el nodo. Posteriormente se presenta la técnica de externalización de protocolos como

una de las alternativas para solucionar la problemática antes citada, y el efecto de las distintas ubicaciones de la interfaz de red en el sistema en las prestaciones del sistema de comunicaciones, así como en bus de interconexión utilizado entre la interfaz de red y el resto del sistema. A continuación, se presentan las técnicas de externalización mediante *offloading* y *onloading* así como los modelos de simulación utilizados para la evaluación de las alternativas que se presentan en esta memoria, mediante la simulación de sistema completo. Finalmente, se presentan otras propuestas para la mejora de las prestaciones del sistema de comunicaciones.

3.1 Introducción

Como se vió en la Sección 1.1, el incremento de prestaciones en las redes de área local ha hecho que el cuello de botella del camino de comunicación pase de la red a los nodos computadores. Esto no sólo es cierto en el caso de los clusters de computadores. Las redes de área local de altas prestaciones, con anchos de banda de hasta 40 Gbit/s, han aparecido recientemente y se están imponiendo incluso para sustituir algunas redes backbone ATM [PET05]. Además, las redes de altas prestaciones como [INF07, EST99, EST06] se utilizan desde hace tiempo para interconectar los sistemas de almacenamiento con robots de *backup* [PRES98]. Estos sistemas deben realizar copias de seguridad con tamaños del orden incluso de varios Terabytes, bien en tiempo real, a través de imágenes de disco, o bien consumiendo el menor tiempo posible. En estos casos, si se intenta utilizar todo el ancho de banda del enlace de una red de altas prestaciones, el nodo en cuestión puede quedar colapsado debido a la gran carga que supone el procesamiento de los datos para ser enviados a la red [GAD07]. Este problema se agrava aún más si se utiliza TCP como protocolo de transporte [MOG03, GAD07], debido a la sobrecarga que supone el componer los paquetes TCP o extraer los datos de dichos paquetes a la velocidad que requiere una red de altas prestaciones. Se ha comentado en la presentación de la memoria así como en la Sección 1.1, la importancia del protocolo TCP/IP. Dada la tendencia actual al uso de TCP/IP y la sobrecarga que supone la implementación de la pila de protocolos TCP en el nodo, resulta de gran interés la investigación de técnicas que permitan explotar el ancho de banda de los enlaces de red utilizando TCP/IP. En este contexto, surge la propuesta de externalizar el protocolo TCP, es decir, llevar el procesamiento del protocolo a un procesador diferente

al procesador principal del nodo, para liberarlo de la sobrecarga que supone el procesamiento de la pila TCP/IP. Sin embargo, hay una gran controversia sobre las ventajas de la externalización como método para mejorar las prestaciones del sistema de comunicaciones cuando se utiliza TCP en la capa de transporte. Además, existen trabajos como [CLA89, STE94a, CHA01, MAR02] que muestran como para conseguir un ancho de banda del 50% con una red de 10 Gbit/s, el uso del procesador será del 100%. En otros trabajos [FOO03] se muestra que la de frecuencia de reloj del procesador debe ser de 1 GHz por cada Gbit/s de ancho de banda de la red a que se conecta. Se necesitaría un procesador con un reloj de 10 GHz para aprovechar todo el ancho de banda de uan red de 10 Gbit/s. (aunque esta misma regla ha sido cuestionada en [REG04a] y de hecho, se puede comprobar experimentalmente que el ancho de banda depende en gran medida de otros factores como la latencia de acceso a memoria). Este efecto se ha reflejado, además, en trabajos como [NAH97, IOA05, MIN95]. Concretamente, en [MIN95] ya se proponía la integración de la interfaz de red en la jerarquía de memoria para evitar el cuello de botella que supone el acceso a memoria. En el Capítulo 4, este hecho se pone de manifiesto experimentalmente mediante la simulación de sistema completo que hemos realizado.

Por tanto, los avances en la tecnología de las redes de comunicación han motivado un trabajo de investigación orientado a la mejora de las prestaciones de comunicación de los sistemas del computador. El interés en esta línea también se justifica en gran medida, por el creciente uso de clusters de computadores que suponen una alternativa económica a los computadores de altas prestaciones. Como se ha visto en las Secciones 1.1 y 2.2, un cluster necesita un sistema de comunicaciones con unas prestaciones equilibradas con las prestaciones de los procesadores, cada vez más potentes, presentes en los nodos, de forma que el cuello de botella en el camino de comunicación no esté en dicho sistema de comunicaciones. Dicho de otra forma, los enlaces de red proporcionan anchos de banda cada vez mayores, con lo que el cuello de botella se ha desplazado del enlace de red al procesador.

En la Figura 3.1 se muestra el tiempo de procesador consumido por los procesos de usuario, sistema operativo y red en la ejecución de un servidor web Apache en un sistema basado en el procesador Pentium-II a 300 MHz, y utilizando una red de 100Mbps de ancho de banda [RAN02] y Linux 2.4 como sistema operativo. Este gráfico puede utilizarse para escalar el consumo de procesador a un procesador de prestaciones superiores.

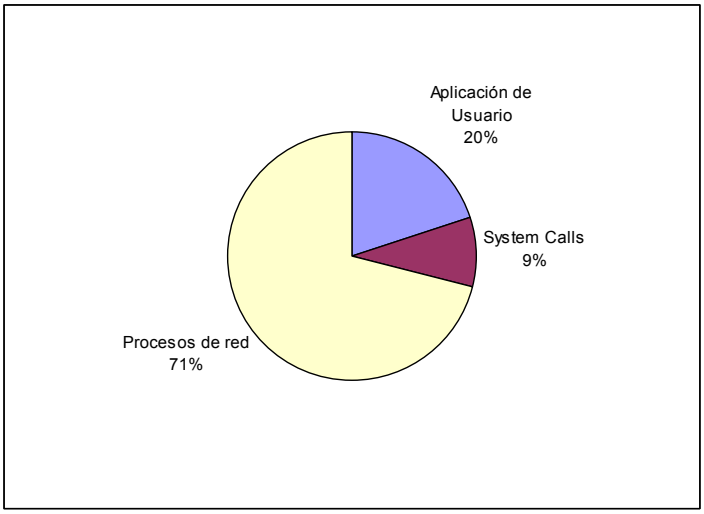


Figura 3.1. Distribución de tiempo de procesador en la ejecución de un servidor web Apache [RAN02]

Puede observarse como en la Figura 3.1, el mayor porcentaje del tiempo corresponde a los procesos de red, que se refieren a:

- La sobrecarga del sistema debida a la gestión de interrupciones, de *buffers* y los cambios de contexto del sistema operativo
- El procesamiento de la pila de protocolos TCP/IP
- Los accesos a memoria ocasionados por a las copias de datos necesarias, así como el efecto de los bloqueos producidos en el procesador por estos accesos.

En el caso de utilizar redes gigabit o multigigabit, la carga asociada a las comunicaciones aumenta al tener que procesarse más mensajes por unidad de tiempo, pudiendo llegar a ocupar hasta el 100% del tiempo de procesador.

De las alternativas posibles para evitar el cuello de botella en el sistema de comunicaciones, esta memoria se centra en la externalización de protocolos de comunicación.

Como se ha comentado en la sección 1.7, las alternativas basadas en la externalización mediante *offloading* [THI02, PAP04, DEL06, BIN06, CHE06] se implementan en las tarjetas de red (NIC) que incorporan ciertos elementos hardware entre los que pueden encontrarse procesadores con arquitcturas más o menos específicas. Entre estas tarjetas, se han hecho populares las llamadas TOE (*TCP*

Offloading Engine) [BRO07, CHE07, NET07a], que tratan de llevar el procesamiento de la pila TCP fuera del procesador principal del sistema, a un procesador separado que se incluye en la interfaz TOE.

Basada en presupuestos similares a lso de la externalización mediante *offloading,* existe otra alternativa que externaliza los protocolos en uno de los procesadores presentes en el sistema. Esta alternativa se suele denominar *onloading* [REG04a, REG04b], y para ella también se han propuesto implementaciones comerciales [IOA05].

En el modelo de simulación que hemos desarrollado, y que se describirá en la Sección 3.2, así como en todas las pruebas realizadas se utiliza TCP como protocolo de transporte. Como se ha comentado en la presentación de esta memoria, el protocolo TCP permite a los servidores operar directamente sobre Internet. Como se comentó en la Sección 3.1, dicho protocolo está ampliamente extendido e incluso se está utilizando en redes SCSI para servidores de almacenamiento, o actualmente para realizar transferencias de secuencias (*streaming*) de video [WAN04]. Si se utilizase un protocolo diferente a TCP, sería necesario usar puentes para la interconexión con redes TCP/IP. Así, la pila de protocolos TCP/IP es, de hecho, el estándar más utilizado en la actualidad para la comunicación a través de redes de área local (LAN), redes de área extensa (WAN) e Internet. Quizá por esto sea precisamente la externalización del protocolo TCP la que ha generado mayor controversia en cuanto a los beneficios que pueda aportar [PET05].

La arquitectura de capas de TCP/IP hace posible la división de las tareas de comunicación en diferentes procesos como el cálculo de la suma de comprobación (*checksum)* o la segmentación TCP. En TCP/IP, la capa de transporte TCP proporciona además de la capacidad de conexión [STE94b]:

- Fiabilidad en la entrega de los datos al destino: TCP proporciona al protocolo el chequeo de la suma de comprobación (*checksum*) para cada paquete, lo que garantiza la entrega sin errores, asegurando que los datos no se han corrompido en el camino de transmisión.
- Entrega ordenada de los paquetes, ya que IP transmite cada paquete como entidades separadas, y, en principio, los paquetes podrían llegar por diferentes rutas, y por tanto, desordenados.
- Mecanismo de control de flujo para no sobrecargar al extremo receptor.

- Multiplexación de conexiones, de modo que se pueden tener diferentes conexiones TCP abiertas al mismo tiempo, transmitiendo y/o recibiendo datos.

Existen numerosos trabajos [SHI03], [MOG03] que ponen de manifiesto los inconvenientes que tiene utilizar TCP en redes de altas prestaciones debido a la sobrecarga de comunicaciones generada en los sistemas que utilizan dicha pila de protocolos. Además, existe cierta controversia en cuanto a la fuente de dicha sobrecarga. Esta controversia ha dado lugar a trabajos de investigación que defienden el uso de sondeo (*polling*) en lugar de interrupciones [BRE06], junto a tros trabajos que proponen el uso de un método híbrido con interrupciones y sondeo para conseguir una mejor adaptación al flujo de datos y optimizar las prestaciones de la interfaz de red [DOV01]. Esta técnica no es incompatible con la externalización de protocolos, ya que puede icorporarse tanto en la interfaz del sistema operativo con la interfaz de red que externaliza los protocolos de red, como internamente, en el propio interfaz de red, para mejorar sus prestaciones. Actualmente existen modificaciones en fase experimental de los núcleos 2.6 de Linux para implementar la técnica de sondeo.

3.2 Externalización de protocolos

A continuación se presentan las diferentes técnicas de externalización de protocolos así como algunas implementaciones comerciales que se han propuesto para las mismas. La primera técnica que se verá es la llamada externalización mediante *offloading*, que consiste en implementar todo el procesamiento de los protocolos en una tarjeta de red. Esto implica que dicha tarjeta debe tener la suficiente capacidad de cómputo para procesar los protocolos aprovechando todo el ancho de banda del enlace de red. En la actualidad, existen diferentes interfaces de red que, si bien no externalizan por completo los protocolos (TCP), sí externalizan alguna parte de los mismos como puede ser el cálculo de la suma de comprobación (*checksum*) o la segmentación TCP. Por tanto, podemos tener distinto grado de externalización, como se ha comentado en la Sección 1.7.2 donde se presenta el modelo LAWS para el análisis teórico de los beneficios aportados por la externalización. En dicho modelo se utiliza el parámetro *p* para representar la parte de la sobrecarga de comunicación que se pasa a la tarjeta de red externa.

Los procesos de cálculo del *checksum* o de segmentación TCP (*TSO, TCP Segmentation Offload* [OPP05]), habitualmente se externalizan en las interfaces de red con anchos de banda a partir de 1 Gbps. De esta forma, parte de la sobrecarga de comunicaciones se pasa a la interfaz de red. A la hora de externalizar el protocolo TCP, la elección de las capas que se deben externalizar es complicada ya que puede llegarse a situaciones que dificulten la interacción con el sistema operativo. Realmente, en los TOE se implementan todas las capas de protocolos, incluyendo las capas inferiores a TCP. La pila de protocolos TCP/IP, así como la implementación usual en los TOE, puede verse en la Figura 3.2.

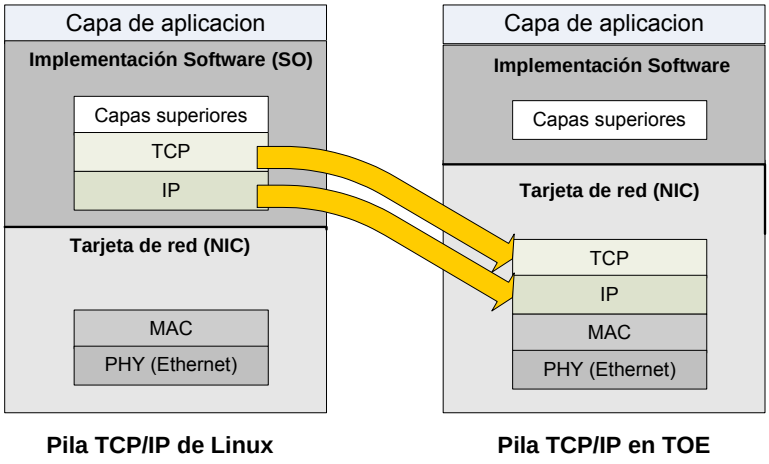


Figura 3.2. Implementación TCP/IP y externalización en un TOE

Además de la implementación de la externalización en la tarjeta de red (*offloading)* implementados comercialmente en [DEL06, BRO07, CHE07, NET07a], existen otras alternativas para externalizar las funciones de comunicación que se diferencian tanto por la forma como por el lugar dentro de la arquitectura del sistema al que se transfieran las funciones de comunicación. Estas alternativas reciben el nombre de *onloading*, cuando se hace uso de los núcleos del nodo en el caso de un CMP [WUN06], o de los procesadores de un SMP [IOA05].

Así, mientras que la externalización mediante *offloading* se basa en llevar las funciones de comunicación a una tarjeta de interfaz (NIC) externa, la técnica de *onloading* que se describe con detalle en la Sección 3.2.2, utiliza los recursos de alguno de los procesadores del nodo (en el caso de un *SMP* o *CMP*) para realizar el procesamiento de los protocolos de comunicación.

Recientemente, Intel ha propuesto una alternativa de externalización basada en una implementación de la técnica de *onloading* [IOA05]. Otros trabajos como [KIM06] proponen externalizar en la tarjeta de red solo algunas de las conexiones TCP, argumentando que una externalización total del protocolo TCP degradaría las prestaciones del sistema debido a las limitaciones, tanto de procesamiento, como de memoria de las interfaces de red.

3.2.1 Ubicación de la interfaz de red en el sistema

Existen trabajos [BIN05] que analizan el impacto que la ubicación de la interfaz de red dentro de la jerarquía de memoria tiene en el sistema de comunicaciones. Concretamente, en [BIN05] se proponen cinco alternativas para situar la tarjeta de red de red (NIC) en el sistema. Se muestran en la Figura 3.3:

1) NIC conectada a un bus de E/S (usalmente PCI), a través de un puente sur.

2), 3) NIC conectada al bus del sistema (como el FSB [REF] o HyperTransport [HYP06]) o integrado en el *chipset* (3).

4) NIC conectada a un bus de E/S (usualmente PCI de alta velocidad) de alta velocidad, directamente desde el puente norte.

5) NIC integrada en el mismo chip de la CPU.

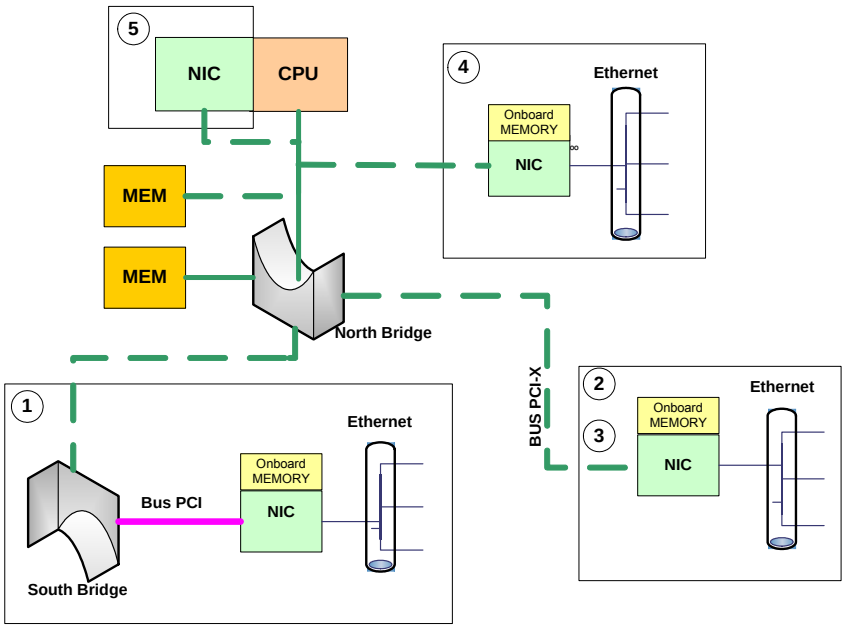


Figura 3.3. Diferentes alternativas de emplazamiento de la NIC

La implementación tradicional y más habitual en sistemas comerciales no orientados a proporcionar alto rendimiento, suele ser la número 1. Sin embargo, en esta implementación, cada acceso a memoria debe pasar forzosamente por los puentes norte y sur. El puente norte hace de interfaz entre el procesador y los dispositivos de alta velocidad y la memoria, a través del controlador de memoria integrado en el propio puente norte. El puente sur, es la interfaz con los dispositivos más lentos. Tener una NIC conectada al puente sur a través de un bus, incrementa notablemente la latencia en los accesos a memoria desde la tarjeta de red, debido a que a la latencia propia de la memoria (determinada por la tecnología correspondiente), hay que añadir la latencia introducida por la lógica de los puentes norte y sur y las colisiones en los mismos debidas a los accesos a memoria concurrentes desde distintas fuentes.

En [BIN05] se concluye que el mayor rendimiento se obtiene cuando la NIC está conectada (o integrada) en el propio procesador del sistema, como se muestra en las implementaciones número 3 y número 4 de la Figura 3.3.

En la Figura 3.4 se muestran las ubicaciones de la interfaz de red más frecuentes en las arquitecturas comerciales actuales de altas prestaciones. En la Figura 3.4a se muestra la implementación utilizando el bus de sistema FSB (*Front Side Bus*) en el que la memoria está conectada al puente norte mediante el correspondiente bus de memoria. En este caso, tendríamos tres posibles ubicaciones de la NIC: directamente conectada al FSB, conectada al puente norte mediante un bus PCI del alta velocidad (como PCI-X), o a un bus PCI estándar a través del puente sur.

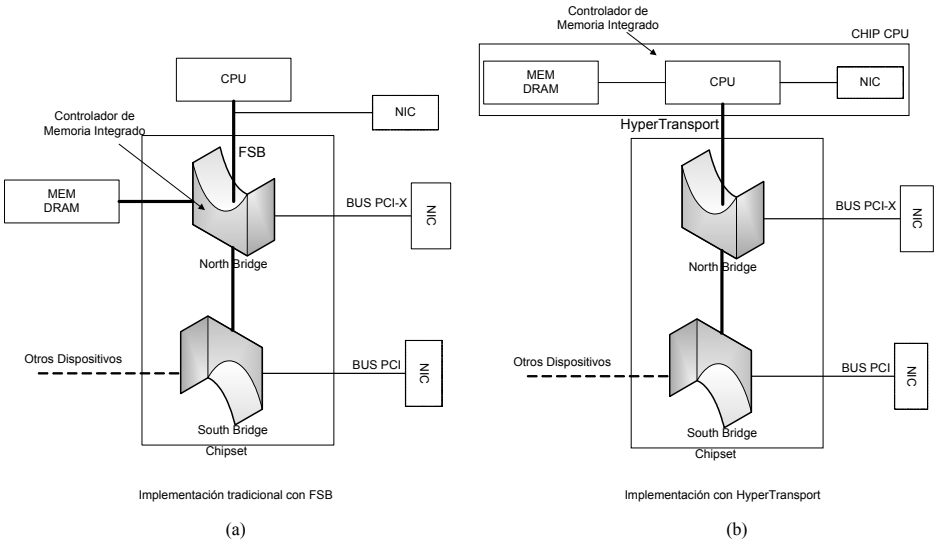


Figura 3.4. (a). Implementación con FSB. (b) Implementación con HyperTransport

La implementación que se muestra en la Figura 3.4b se corresponde a una arquitectura más reciente llamada *HyperTransport* [HYP06] de AMD, en la que el controlador de memoria se integra en el propio procesador, estando el procesador y la memoria conectados directamente, sin necesidad de un puente que haga de interfaz. La tecnología *HyperTransport* [HYP06] se basa en una conexión punto a punto de alta velocidad y baja latencia diseñada para incrementar la velocidad de comunicación entre componentes de la arquitectura (actualmente, los canales *HyperTransport* [HYP06] proporcionan un ancho de banda de 6,4 Gb/s). En sistemas en los que se usa esta tecnología, se reduce el número de buses en el sistema ya que la NIC está integrada en el microprocesador, junto con el procesador y el controlador de memoria.

Por tanto, las cuatro posibilidades de conexión e integración de la NIC mostradas en la Figura 3.4 pueden implementarse con las dos arquitecturas comerciales, FSB Figura 3.4a o *Hypertransport* [HYP06] Figura 3.4b, como se detalla a continuación:

1. NIC conectada al bus PCI estándar (ubicación 1 en la Figura 3.3). Es la ubicación tradicional de la interfaz de red. Sin embargo, no suele ser una ubicación apropiada para interfaces de red de ancho de banda elevado (como Gigabit Ethernet), debido a que el propio bus PCI supone un cuello de botella para las prestaciones de la interfaz.

2. NIC conectada al puente norte a través de un bus PCI de alta velocidad. Esta es actualmente la ubicación más frecuente para interfaces de red de altas prestaciones. Al estar conectado directamente al puente norte, la latencia se reduce considerablemente. Además, trabajos como el descrito en [HUG05] describen la posibilidad de acceder directamente a la memoria *cache* del procesador del sistema para incrementar las prestaciones. No obstante, hay que tener en cuenta la latencia introducida por el puente norte así como la contención en el mismo.

3. NIC integrada en el Chipset. Actualmente es común encontrar sistemas en los se han integrado ciertos de dispositivos en el chipset. De esta forma se reduce el numero de componentes y por tanto el coste del sistema. Del mismo modo, se puede integrar la NIC en el chipset, reduciendo así la sobrecarga asociada al paso de mensajes a través de los buses del sistema

y a la comunicación entre la NIC y el procesador. Es posible tener integrada la NIC en el chipset tanto en el caso de la implementación tradicional con FSB como en el caso de HyperTransport. En ambos casos, de forma similar a lo que ocurre cuando la NIC está integrada en el propio microprocesador, la NIC puede acceder a la memoria *cache* del procesador, reduciendo los accesos desde el driver a la memoria del sistema mediante DMA.

4. NIC conectada al FSB o a Hypertransport. Es equivalente a tener un sistema con más de un procesador. La diferencia, en este caso, consiste en que la NIC se integraría en el microprocesador. Esta implementación tiene ventajas evidentes, como la posibilidad de acceso desde la NIC a la memoria *cache* del procesador del sistema, así como el aprovechamiento del ancho de banda que proporcionan tanto el FSB como HyperTransport, y evitar el puente norte. Sin embargo, sería más adecuado conectar la NIC a HyperTransport debido a que es un sistema estandarizado, concebido para poder conectar dispositivos y no sólo como bus interno del sistema (como es el caso del FSB). Además, si se utiliza el FSB para transmitir paquetes de red a alta velocidad, se estaría reduciendo el ancho de banda del mismo para uso del procesador. La utilización de buses con gran ancho de banda para conectar la NIC, ya se propone de alguna forma en el trabajo de [MIN95], donde se estudian las posibles ventajas que podría tener una interfaz de red integrada en la jerarquía de memoria.

5. NIC integrada en el procesador. Con esta alternativa se pretende evitar que el bus del sistema se convierta en un cuello de botella del sistema de comunicaciones. Al integrar la NIC en el propio procesador, el ancho de banda para la comunicación entre ambos se incrementa notablemente y se reduce la latencia, a la vez que disminuye el número de transferencias DMA necesarias para la comunicación entre la NIC y el procesador. De esta forma, los accesos mediante PIO (Programmed I/O) para comunicar el procesador y la NIC se completan mucho más rápidamente. Además, el NIC podría acceder a algunos niveles de la memoria *cache* del procesador, evitando así muchas de las transferencias de DMA. Sin

embargo, implementar esta ubicación presenta numerosos problemas (es necesario dotar al procesador de la NIC de memoria suficiente, etc.). En [BIN06] se propone la utilización de esta técnica para mejorar las prestaciones de las redes TCP/IP con enlaces de gran ancho de banda.

Ya se ha comentado que los buses del sistema pueden constituir un cuello de botella importante para las prestaciones del sistema de comunicaciones. Por eso, las alternativas de ubicación número 4 y número 5 de la Figura 3.3 deberían proporcionar mayor rendimiento. En el caso de que se utilicen procesadores con más de un núcleo, uno de estos núcleos bien puede dedicarse a las funciones de red. Podría tratarse de un núcleo de propósito específico como un procesador de red. En la Sección 3.3 y siguientes, se verá con más detalle este tipo de implementación y en el Capítulo 4 se analizarán las prestaciones ofrecidas por cada una de ellas, así como las condiciones en las que pueden proporcionar mayores prestaciones.

Como se ha visto existen diferentes alternativas para aumentar las prestaciones de la interfaz de red, desde la mejora del propio interfaz hasta el cambio de su ubicación dentro del sistema para evitar cuellos de botella. Cabe distinguir por tanto, entre las alternativas de conexión e integración de la NIC mostradas en esta sección, y las alternativas de implementación de la interfaz de red (NI) a las que se hace referencia en las Secciones 1.5, 1.6 y 1.7, y que serán las que determinarán la sobrecarga de comunicación en la NIC o en el procesador.

En cuanto a la ubicación de la NIC, las prestaciones ofrecidas por cada una de las alternativas mostradas en esta sección dependen en gran mediada de la aplicación concreta y del tipo de sistema. Precisamente, identificar los cuellos de botella y poner de manifiesto la relación entre la ubicación y características de la interfaz con sus prestaciones es una de las motivaciones principales de este trabajo. Para ello, la metodología experimental que utilizamos se basa en la simulación de sistema completo con Simics.

Los modelos para la simulación de la externalización están inspirados en una arquitectura que, la interfaz de red está conectada al puente norte. Además, se han realizado modificaciones en la arquitectura simulada para que se pueda distinguir entre *offloading* y *onloading*, y comparar sus prestaciones. Estos cambios incluyen la utilización del puente norte solo como conector o modelando las colisiones que se

producen en el mismo debido a accesos concurrentes a memoria por diferentes dispositivos, y la modificación del perfil de interrupciones. Con *offloading*, las interrupciones generadas por recepción de paquetes llegarán exclusivamente al procesador de la tarjeta de red, mientras que el driver deberá ejecutarse en el procesador del nodo. Nuestro objetivo es aprovechar las posibilidades de modelado y configuración de Simics según interese en cada caso par que el modelo se comporte de una forma realista.

3.2.2 Externalización TCP mediante offloading: hardware para la externalización del protocolo TCP

Para poder explotar el ancho de banda de los enlaces multigigabit, cada vez más utilizados en redes de interconexión de larga distancia con sistemas de almacenamiento, en clusters de computadores, o en la interconexión de sistemas de almacenamiento con robots de backup [PRE98], es necesario, como ya se ha visto, evitar que el nodo computador se convierta en un cuello de botella. Se ha indicado en la Sección 3.2 la idea de los TOE (*TCP Offload Engine*) es llevar el proceso de la pila TCP/IP a un hardware especializado usualmente ubicado en la NIC y diferente del procesador que ejecuta las aplicaciones, el sistema operativo e incluso la gestión de las conexiones TCP/IP [KIM06]. De esta forma, se pretende conseguir un ancho de banda elevado sin disponer de un nodo con un procesador de altas prestaciones que soporte tanto la sobrecarga asociada a los procesos de comunicaciones como la de los procesos de las aplicaciones, liberando ciclos del procesador para dichas aplicaciones.

Las operaciones necesarias para procesar cada paquete (sobre todo en la recepción), desde que éste es recibido en la tarjeta de red hasta que los datos que contiene quedan disponibles para la aplicación, requieren de un gran número de ciclos de procesador. Los métodos normalmente utilizados por las interfaces de red convencionales (Sección 1.4) con enlaces de anchos de banda a partir del gigabit por segundo, no son suficientes para evitar que el nodo se convierta en el cuello de botella del camino de comunicación cuando se transfieren cantidades masivas de datos y se quiere aprovechar todo el ancho de banda del enlace.

Las interfaces de red usuales no externalizan toda la pila TCP/IP, aunque sí implementan algunas de las funciones en distintos elementos hardware de la interfaz de red [BRO07]. Un ejemplo típico es la externalización del cálculo de la suma de

comprobación de los paquetes (*checksum*) o la segmentación TCP (LSO o TSO en el caso de TCP) [DEL06].

La segmentación TCP es el proceso por el que los datos que envía la aplicación se dividen en unidades conocidas como MTU (*Maximum Trasnsfer Unit*), a las que se les añade las cabeceras TCP e IP para su transmisión. Al externalizar la segmentación TCP se puede reducir el uso de procesador incluso hasta la mitad en algunos casos, si se están transmitiendo paquetes grandes utilizando tramas jumbo [GUM03]. El uso de tramas jumbo, permite el envío de MTUs (*Maximum Transfer Unit*, o tamaño en bytes del datagrama más grande que puede transportar la capa física) mayores de 1500 bytes, el cual es el tamaño predeterminado en la mayoría de las implementaciones de TCP. La tarjeta de red informa al procesador de la llegada de un paquete mediante la generación de una interrupción (ya sea de la propia tarjeta de red o del controlador de DMA indicando que la transferencia se ha completado). Por tanto, incrementando el tamaño de la MTU se consigue retrasar la generación de la interrupción por recepción de paquetes o, lo que es lo mismo, enviar al procesador un menor número de interrupciones por segundo, con la consiguiente liberación de ciclos de procesador para la aplicación. Esta reducción del uso de procesador se producirá tanto en el extremo transmisor como en el receptor, en el que el uso de tramas jumbo reduce el número de interrupciones generadas. Sin embargo, si los paquetes transmitidos son pequeños o se requiere un ancho de banda muy elevado, esta técnica no reducirá lo suficiente la sobrecarga del procesador, además de aparecer otras limitaciones relacionadas con la latencia de acceso a memoria o los buses de E/S, con lo que el nodo seguirá siendo el cuello de botella [BIN05, IOA05, MIN95].

Por tanto, las mejoras ofrecidas por las interfaces convencionales, no son suficientes para evitar la congestión del procesador del nodo, siendo necesaria la externalización del sistema de comunicaciones (tanto en transmisión como en recepción). Así, la NIC consumiría ciclos del procesador del nodo para cada transferencia E/S hacia o desde la aplicación, evitando por ejemplo, los ciclos que se consumen por cambios de contexto en las rutinas de servicio de interrupción, que se producen con la recepción de cada paquete o las sucesivas copias de los datos a memoria. Estos mecanismos pueden ser implementados directamente en la NIC.

Sin embargo, el uso de interfaces de red externalizados mediante *offloading*, no siempre soluciona el problema de la sobrecarga de comunicación. Aunque el uso de TOEs puede reducir la carga de trabajo del procesador del nodo durante los procesos de

comunicación, la integración en el sistema operativo, genera latencias considerables. Ya se ha visto en la Sección 1.7 que la comunicación entre la NIC (TOE en este caso) y el sistema operativo podría ser tan complicada como el propio protocolo a externalizar [ODE02]. Por tanto, su uso solo mejorará el rendimiento del sistema con aplicaciones que transfieren paquetes muy grandes (*payload* de gran tamaño). El uso de TOEs con paquetes de tamaño pequeño provocará un gran número de interacciones con la CPU, generando una carga considerable, aunque dicha CPU no tenga que procesar la pila de protocolos, pudiendo llegar a colapsarse y disminuyendo el rendimiento del sistema de comunicaciones. Ya se ha comentado que una parte muy importante de la carga generada en el nodo viene de las rutinas de servicio de las interrupciones generadas a la llegada de un paquete, o cuando el TOE tiene datos listos para ser transferidos [IOA05, BIN05].

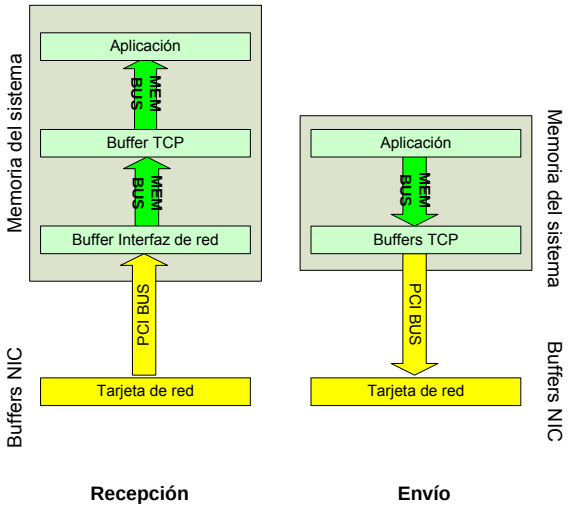


Figura 3.5. Caminos de envío y recepción

En definitiva, para aprovechar las ventajas que puede proporcionar la externalización del sistema de comunicaciones en la tarjeta de red, esta ha de realizar tres funciones básicas de manera eficiente:

- *El procesamiento de las interrupciones*. Con esto se trata de evitar que las interrupciones generadas por la llegada de paquetes (en realidad los cambios de contexto y la ejecución de las rutinas de servicio que las interrupciones ocasionan) colapsen el procesador del nodo. Gestionar cada una de estas interrupciones supone un

elevado número de cambios de contexto, de la aplicación (*user-space*) al núcleo (*núcleo-space*), y al contrario. Existen técnicas para reducir la carga asociada a las interrupciones como el llamado *interrupt coalescing* [CEB04] que hace que no se genere una interrupción hasta que no hayan llegado un número de paquetes prefijado. Además, cada transferencia de datos desde la interfaz de red produce, usualmente, una serie de copias de datos de la aplicación (desde los *buffers* de la aplicación a los *buffers* del sistema, y de aquí a los *buffers* de la tarjeta de red). Por tanto, es necesario optimizar la implementación de la gestión de las interrupciones para mejorar las prestaciones de la interfaz de red.

- *Las copias de datos en memoria*. Normalmente, para transferir un dato desde la aplicación a la red son necesarias tres transacciones: desde la memoria de usuario a los *buffers* del núcleo y desde aquí a la interfaz de red, como puede verse en la Figura 3.5. Esto genera un elevado tráfico en los buses de memoria y de E/S del sistema [PLA00]. Estas transferencias pueden reducirse a dos utilizando un interfaz que externalice el protocolo. Así, la interfaz de red puede copiar los paquetes recibidos en el *buffer* TCP y de aquí en los *buffers* de la aplicación. Además, si la interfaz de red utiliza RDMA (Remote Direct Memory Access) [BAL04], es posible utilizar algoritmos de *copia cero* ("*zero-copy*") [SKE01], de forma que la interfaz de red copia los datos directamente desde sus *buffers* a los *buffers* de la aplicación.

- *Procesamiento de la pila de protocolos*. Diferentes trabajos como [MOG03, REG04a, ODE02] muestran que el procesamiento de la pila de protocolos TCP no es la principal fuente de sobrecarga del sistema de comunicaciones. Sin embargo, la sobrecarga generada por dicho procesamiento, junto con las copias de datos y el procesamiento de las interrupciones, lleva a la conocida regla que indica que para conseguir una tasa de transferencia de 1 Mbps es necesaria un procesador de 1MHz o, lo que es lo mismo, que se consume un MHz de procesador por cada Mbps de ancho de banda [FOO03]. Por tanto, aunque ya se ha comentado que esta regla solo puede tomarse como una orientación, para enlaces de 10 Gbps no es difícil entender que prácticamente cualquier procesador actual quedaría colapsado por el procesamiento de la interfaz de red. Por tanto, sería interesante que el procesamiento de la pila TCP/IP resida en la NIC, o aprovechar el paralelismo asociado a la presencia de otros procesadores en el sistema. Sin embargo, esta estrategia plantea una serie de problemas

cuya solución no siempre es sencilla. Concretamente, como se ha dicho anteriormente, se ha de implementar un mecanismo de sincronización efectivo entre la interfaz y el sistema operativo, tanto para la gestión de las conexiones TCP como de los *buffers* de datos, ya que aunque el protocolo se procese en la interfaz de red, la gestión de las comunicaciones debe seguir residiendo en el sistema operativo [MOG03].

Por tanto, la incorporación de TOEs en nodos requiere que el hardware que incluyan esos TOE sea capaz de realizar las funciones que se han descrito, y que además, existan componentes software que cedan el procesamiento de la pila de protocolos a la NIC mientras que el funcionamiento del sistema de comunicaciones se hace transparente para las capas superiores permitiendo que sobre ellas puedan ejecutarse aplicaciones de usuario sin modificar.

En cuanto a la implementación de los TOE, estos pueden incluir procesadores de propósito general, aunque no es lo más recomendable [CLA89, MOG03]. Muchos de los TOE que actualmente existen en el mercado utilizan procesadores de propósito específico. Dichos procesadores pueden ser procesadores de red (*Network Processor o* NP) [BRO07, NET07a, CHE07], los cuales incluyen instrucciones específicas para el procesamiento de protocolos, consiguiendo una mayor eficiencia que en el caso de utilizar procesadores de propósito general [INT03a], o procesadores con varios núcleos que permiten disponer de un alto grado de paralelismo. Además, la mayoría de los procesadores de red actuales incluyen varios núcleos de procesamiento, además de utilizar tecnología multihebra [INT03a].

3.2.3 Externalización TCP mediante *onloading*

La externalización mediante *onloading* consiste en transferir el procesamiento de los protocolos a uno de los procesadores centrales del nodo, de forma que los paquetes TCP/IP sean procesados no por un procesador de propósito específico, sino por un procesador de propósito general distinto del procesador actual.

Por tanto, esta técnica propone por tanto, que el procesamiento de los protocolos tenga lugar en un núcleo de procesamiento de un CMP (chip multiprocesador o procesador monochip) o en uno de los procesadores de un SMP (Symmetric Multiprocessor). Aunque esta técnica se ha planteado como contrapuesta al *offloading*, en realidad también se trata de una externalización de los procesos de comunicación a

otro procesador disponible en el nodo, en lugar de en la NIC. En la Sección 3.2.1 se han comentado diferentes alternativas para la ubicación del interfaz de red en el sistema, entre las cuales se incluye la posibilidad de que dicho interfaz de red esté integrado en el propio microprocesador.

Existen trabajos que comparan las dos técnicas de externalización [INT04], aunque es difícil realizar experimentos utilizando sistemas de similares características. Por un lado, la implementación del *onloading* de Intel utiliza ciertas optimizaciones y modificaciones en el sistema operativo que no son públicas, así como una pila de protocolos optimizada para ser ejecutada en procesadores Intel Xeon, que tampoco es pública. Por otro lado, en el estudio comparativo de [INT04] se realizan medidas de prestaciones sólo para mensajes de hasta 64KB, cuando los trabajos que analizan la externalización con *offloading* coinciden en que dicha técnica de externalización resulta más adecuada para aplicaciones que requieren un ancho de banda elevado y para la transmisión de paquetes de gran tamaño. Por tanto, resulta complicado explorar todo el espacio de parámetros mediante experimentos con sistemas reales, más aún cuando no se conoce con precisión los detalles de implementación de las alternativas comerciales. Así, es muy conveniente disponer de una herramienta que permita modificar los diferentes parámetros que definen el espacio de diseño en un sistema simulado, lo más parecido posible a un sistema real, en el que se puedan modificar características software y hardware para explorar el espacio de parámetros y determinar el impacto de cada uno de estos parámetros en las prestaciones del sistema de comunicaciones.

Al igual que los TOE son implementaciones comerciales de la externalización mediante *offloading*, como se ha dicho, actualmente existe una implementación comercial del *onloading* diseñada por Intel: *Intel I/O Acceleration Technology* [IOA05]. A continuación se resumen las claves principales de la implementación del *onloading* de Intel:

- Utilización de un DMA mejorado, para reducir las latencias asociadas a las transferencias de datos. Ya se ha comentado en la Sección 1.7 que dichas transferencias suponen el cuello de botella más importante en el caso del procesamiento de la pila TCP/IP [MIN95, NAH97, IOA05].
- División de los paquetes recibidos para paralelizar el procesamiento de los paquetes.

- Pila de protocolos TCP/IP optimizada, para reducir la sobrecarga asociada al procesamiento de los protocolos TCP/IP.

Con respecto a la implementación mediante *offloading* en un TOE, se pueden citar las siguientes diferentias:

- Aunque requiere un sistema operativo que proporcione el soporte adecuado, dicho soporte es más ligero que el soporte necesario para un TOE (ya se ha comentado en la Sección 1.7 que la API necesario para la comunicación entre el TOE y el sistema operativo es complejo). En realidad, en una implementación de *onloading* se siguen ejecutando los procesos de comunicaciones en un procesador de un SMP o CMP. No es necesario una API que proporcione conectividad con una interfaz de red conectado a un bus (PCI ó PCI-X por ejemplo) y el control de las primitivas de comunicación TCP/IP.
- Se utilizan procesadores de propósito general, evitando el problema de la diferencia tecnológica existente entre los procesadores de red y los de propósito general (lo que en el modelo LAWS [SHI03], se modela a través del parámetro α , ***L****ag Ratio*).
- Es una alternativa de menor coste que la utilización de un TOE [IOA05] debido a que se están aprovechando los recursos hardware presentes en las arquitecturas de procesador que se están imponiendo hoy día, sin necesidad de adquirir ningún hardware adicional. Además, los problemas asociados al test y mantenimiento [MOG03] tienen una solución más fácil debido a que dicho test y mantenimiento afectará principalmente a la capa software ([IOA05] es fundamentalmente una implementación software).

3.2.4 Offloading vs. onloading

En la sección 3.2.1 se ha mostrado que el procesamiento de los protocolos TCP/IP genera una gran cantidad de transferencias de datos [PLA00, SKE01]. La alternativa de externalización mediante *onloading* trata de reducir la sobrecarga asociada al procesamiento de las comunicaciones, y concretamente al manejo de los datos, sin que ello suponga cambios en el hardware del sistema y de forma transparente para las aplicaciones.

Cuando la interfaz de red recibe un paquete, inicia una serie de operaciones que provocan la interacción de la interfaz de red con el procesador del sistema. Estas operaciones persiguen el procesamiento del paquete y proporcionar los datos a una aplicación determinada. Como se ha comentado en la Sección 1.3, todo este proceso se inicia con una interrupción generada por la NIC. Una vez que el procesador realiza ciertas comprobaciones con los datos del paquete recibido y una vez que ha verificado que este no contiene errores, los transfiere al espacio de memoria de usuario, desde donde la aplicación los recogerá.

El tiempo de cómputo del procesamiento del protocolo TCP/IP es pequeño, en comparación con el tiempo de la multitud de transferencias necesarias para procesar cada paquete [MOG03]. Estas transferencias provocan retardos, debido fundamentalmente a las latencias y las colisiones en los buses, la interacción con el sistema operativo y la latencia de acceso a memoria [IOA05, NAH97, MIN95], siendo la latencia de acceso a memoria la responsable de la mayor parte de la sobrecarga asociada al procesamiento de paquetes TCP/IP. Esto hace pensar que el uso de la externalización mediante offloading (TOE) debería proporcionar mayores beneficios en el caso de paquetes de gran tamaño, ya que el uso de paquetes pequeños generaría una gran latencia debido al gran número de accesos a memoria y a que la interfaz entre el sistema operativo y la NIC también introduce latencias [MOG03]. Por tanto, aunque el TOE fuese capaz de procesar los paquetes recibidos, no podría reducir la latencia asociada al interfaz NIC/sistema operativo y a las copias de datos en el espacio de usuario. Por consiguiente, en el caso de paquetes pequeños, la externalizació mediante offloading no proporcionaría una mejora sustancial respecto al sistema sin externalización.

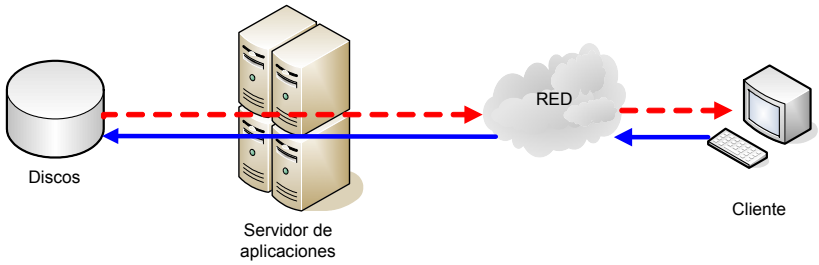


Figura 3.6. Camino de datos desde y hacia la aplicación

No obstante, es necesario realizar un análisis más detallado, en el que se tengan en cuenta los diferentes parámetros que intervienen en las prestaciones del sistema de

comunicaciones. Para ello, resulta útil el modelo LAWS descrito en la Sección 1.7.2, así como los modelos de simulación mediante Simics descritos en la Sección 3.4.

3.4 Modelo de simulación para el sistema sin externalización

Para analizar y evaluar las mejoras que supone la externalización y las propuestas que planteamos en esta memoria se han implementado dos modelos de simulación, que hacen uso de los modelos de temporización descritos en la Sección 2.6 (Capítulo 2). Uno para la simulación del sistema sin externalizar y otro para la simulación del sistema con externalización, de forma que podamos evaluar las prestaciones de nuestras propuestas de externalización. Gracias al *modelo de temporización* descrito en la Sección 2.6.1, y como se mostrará más adelante en esta sección, el modelo sin externalización representa un sistema en el que se tienen en cuenta las latencias existentes en los diferentes elementos (memoria, procesador, acceso a buses, red de interconexión, etc.), así como las colisiones en los buses.

En la Figura 3.7, se muestra un esquema del modelo en el cual se ha incluido un procesador, un puente norte, un bloque de memoria física, un bus PCI y una tarjeta de red. Simics proporciona un modelo hardware de interfaz con el chip BCM5703C, compatible con el driver estándar de Linux *tg3.o*. Además, se han conectado *modelos de temporización* descritos en la Sección 2.6.1 en distintas partes de la arquitectura, con el fin de poder modelar el comportamiento de los diferentes componentes (procesador, memoria y tarjeta de red). También se ha utilizado un enlace Ethernet estándar, configurado para soportar un ancho de banda determinado.

El procesador incluido en el modelo es de tipo Pentium4. En cuanto a la memoria, en la Figura 3.7 se muestra el espacio físico de memoria configurado en Simics. En este espacio se asigna al espacio de E/S y de configuración del bus PCI.

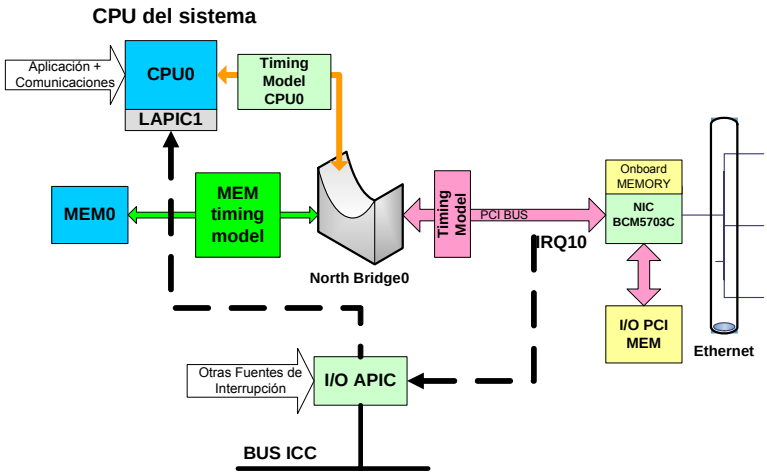


Figura 3.7. Modelo sin externalización

El esquema de la Figura 3.7 permite simular latencias en el acceso a memoria o al bus PCI, así como simular el comportamiento ante colisiones en el puente norte.

Así, mediante esta arquitectura que respeta el modelo de la Figura 3.7, se han realizado las simulaciones correspondientes a un sistema de comunicaciones no externalizado en el que el procesador ejecuta los procesos del sistema operativo y de la aplicación, y tiene que gestionar totalmente los procesos de comunicaciones. Como se ha indicado y vemos en nuestros experimentos, con redes de anchos de banda de enlace y cargas de trabajo reducidas, esta configuración es capaz de proporcionar todo el ancho de banda del enlace de red. Sin embargo, cuando aumenta el ancho de banda del enlace, o la carga de trabajo generada en el procesador debida a la aplicación suponga un cierto tiempo de procesador, el ancho de banda efectivo comenzará a ser inferior al del enlace. De hecho, hay trabajos donde se presentan modelos teóricos [SHI03] que muestran el efecto del incremento de la carga de procesador debida a la aplicación en el ancho de banda efectivo de la comunicación. De estos modelos teóricos se puede extraer una relación entre el ancho de banda efectivo y la relación entre la *sobrecarga de comunicación* y la *sobrecarga de aplicación*, poniéndose de manifiesto que no se podrá utilizar todo el ancho de banda del enlace cuando los procesos de comunicación y los demás procesos que se ejecutan en el nodo (es decir, los procesos del sistema operativo junto con los procesos propios de la aplicación) supongan el 100% del tiempo de procesador y no dejen ciclos libres.

En la Figura 3.8 se muestra la secuencia temporal de operación del modelo de simulación sin externalización. En dicha secuencia temporal aparecen las diferentes

operaciones que se llevan a cabo en el nodo desde que se recibe un paquete de la red hasta que la información que contiene dicho paquete está disponible para la aplicación.

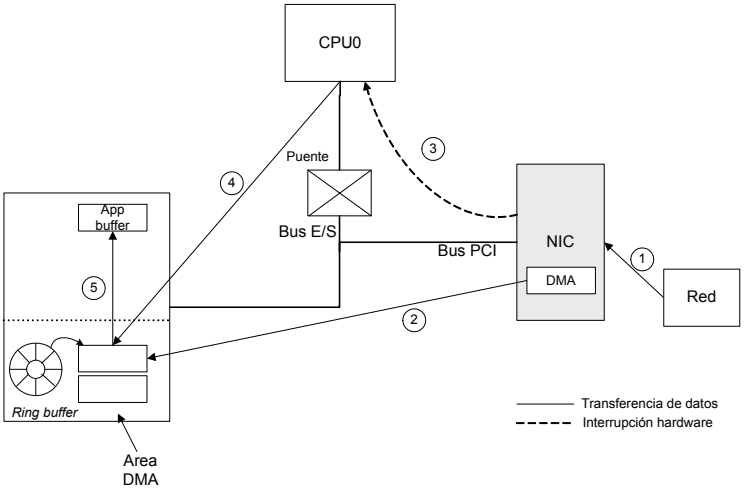


Figura 3.8. Operación del modelo sin externalización

En el esquema mostrado en la Figura 3.8, una vez que se recibe un paquete de la red ① en la tarjeta de red (NIC), éste se transfiere al *buffer* en anillo mediante DMA ②. Cuando dicha transferencia ha finalizado, la NIC genera una interrupción hardware (IRQ) ③ que alcanza al procesador del nodo (CPU_0). Dicha interrupción hardware hace que se ejecute el driver de la NIC y que se encole la interrupción software correspondiente, que copiará el paquete desde el *buffer* en anillo hasta el *socket* de recepción ④. Finalmente, los datos se copian desde el *socket* de recepción hasta los *buffers* de la aplicación ⑤, quedando disponibles para la misma.

En el Capítulo 4 se presentan los resultados de las simulaciones, realizadas para analizar las prestaciones de este sistema.

3.5 Modelo de simulación para la externalización offloading

El modelo de simulación utilizado para la externalización de protocolos mediante *offloading*, pretende aprovechar las características propias de Simics y las que hemos incorporado para reproducir un sistemarealista con externalización mediante un TOE a partir de un sistema con dos procesadores. Para ello, como se indica en [WES04], es necesario tener en cuenta dos restricciones:

1) El sistema de dos procesadores se ha de comportar como un sistema en el que hay un solo procesador central, con el segundo procesador en la NIC. Esto puede suponer un problema debido a que los procesadores de un SMP comparten recursos y y atienden los cross-calls. Esta restricción implica tener en cuenta determinados aspectos de la arquitectura y de la implementación que se hace de la externalización. Para conseguir que la máquina con dos procesadores (CPU_0 y CPU_{NIC}) se comporte como una máquina con un solo procesador (CPU_0) ya que el otro procesador (CPU_{NIC}) se utiliza para el procesamiento de los protocolos, es necesario asegurar que en la CPU_{NIC} no van a ejecutarse hebras del sistema operativo ni de las aplicaciones.

2) Una NIC real dispone de su propia memoria y buses internos. Por tanto la NIC simulada no debe compartir recursos del sistema que puedan ocasionar colisiones, es decir, buses de I/O, memoria principal o memoria *cache* para el procesamiento de los protocolos, si estamos simulando el comportamiento de una NIC externa, en la que el único recurso compartido con el resto del sistema es el bus de interconexión. Así, esta restricción supone un aislamiento desde el punto de vista hardware de la CPU_{NIC} del resto del sistema.

3) El número de ciclos de CPU utilizados para realizar el procesamiento de los protocolos debe ser similar al empleado por un TOE.

Este número depende en gran medida del código utilizado para implementar la pila TCP/IP. La mayoría de los interfaces, aunque habitualmente incluyendo ciertas optimizaciones para una arquitectura de procesador concreta, utilizan como base la pila de protocolos BSD. De hecho La pila TCP/IP BSD es la base de la implementación TCP/IP actual de Linux.

Para conseguir un comportamiento de acuerdo a las restricciones anteriores, y así simular la externalización de protocolos mediante *offloading,* se ha utilizado el modelo de simulación de la Figura 3.8, que corresponde a un sistema que hemos configurado de forma que el procesador que ejecuta los procesos de comunicación esté conectado a la interfaz de red de una forma más directa que en el caso de un sistema convencional. Esto se consigue conectando modelos de temporización diferentes a cada procesador. A la CPU_0 se le ha conectado un modelo de temporización que modela la latencia en los accesos a memoria. Aunque la latencia para los accesos a memoria desde la CPU_{NIC} es

la misma que para la CPU_0, el bus PCI, al que se ha conectado la interfaz de red, actúa solo a modo de conector hacia la CPU_{NIC}. De esta forma, las transacciones entre la interfaz de red y la CPU_{NIC} no se ven limitadas por la latencia impuesta por el bus PCI, simulando el estar juntas, como se haría en un TOE, en el que el procesador de la NIC se encuentra en la misma tarjeta de red y no conectadas mediante un bus PCI. La asignación de las latencias al bus PCI puede hacerse gracias al modelo de temporización que proponemos. Además, se ha modificado el perfil de interrupciones, de forma que las interrupciones que provengan de la interfaz de red sólo llegan a la CPU_{NIC}, como se verá en la Sección 3.5.1, y sin la latencia impuesta por el controlador de interrupciones incluido en el chipset (APIC [INT97]).

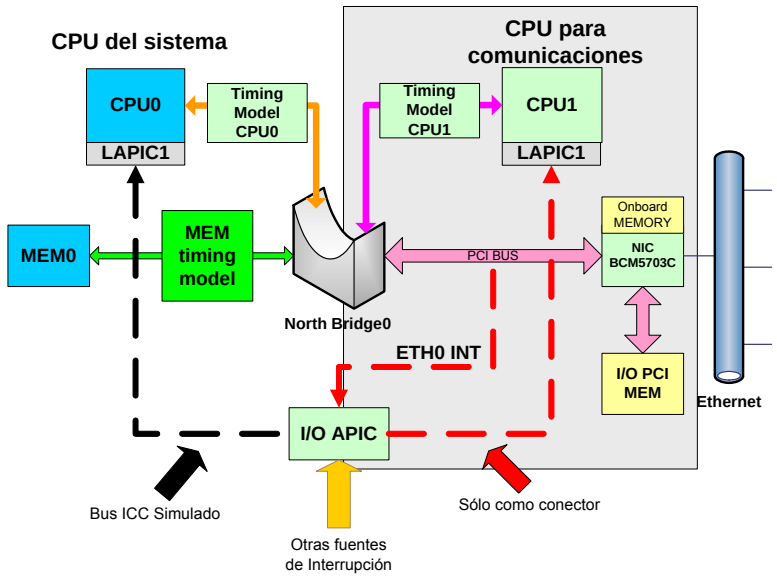


Figura 3.9. Modelo de simulación del sistema con externalizción offloading

A continuación se describirá el modelo en cuanto a la configuración hardware simulada en Simics y en cuanto a la distribución del software en el sistema.

3.5.1 Modelo Hardware para la simulación de *offloading*

En la Figura 3.9 se muestra el modelo utilizado para la simulación de la externalización mediante *offloading*.

Como se ha visto en la Sección 2.6, la forma de simular las latencias en los accesos a memoria en Simics consiste en conectar un modelo de temporización a cada

entrada del puente norte donde puede haber colisiones. De esta forma, en la Figura 3.8, se han conectado modelos de tiempo a las entradas del puente norte, de esta forma los buses no actúan solo como conectores, como lo harían por defecto en Simics.

Las transferencias de datos en Simics no requieren necesariamente un puente de interconexión ya que Simics permite definir un hardware totalmente a medida. Sin embargo, en nuestro modelo es necesario incluir el puente norte, así como otros elementos de la arquitectura (controlador de interrupciones, etc.) para simular un sistema real en el que se pueda instalar un sistema operativo estándar (como se ha dicho anteriormente, Linux). Al construir los modelos de simulación se parte de la premisa de poder ejecutar sistemas operativos comerciales y poder disponer de los resultados que proporcionen las mismas pruebas de rendimiento (*benchmark*) que se puedan ejecutar en una máquina real. Para que estos modelos funcionen correctamente con el sistema operativo y se modele correctamente el comportamiento con externalización, es necesario añadir una capa software que controle la gestión de los procesadores que hace el sistema operativo y que, a la vez, nos permita conseguir el aislamiento software que se describía en la Sección 3.6. Para esto se han utilizado las librerías CPUSET [BUL04], tal y como se describe más adelante, en la Sección 3.6.2. Además, se ha utilizado la capacidad *hotplug* [MWA04] incluida en los nuevos nucleos de Linux mediante la cual es posible pasar una determinada CPU del sistema (siempre que sea una CPU distinta a la CPU de arranque) a modo *offline*. Una CPU en modo *offline* no puede ser utilizada por el sistema operativo para ejecutar una hebra, aunque es posible forzar la ejecución de una determinada hebra. De esta forma, aseguramos que la CPU_{NIC} no va a ejecutar ninguna otra hebra que no sea la que realiza el procesamiento de los protocolos TCP/IP, y se minimiza la interacción de la CPU en modo *offline* con el sistema operativo y con el resto de procesadores del sistema (CPU_0).

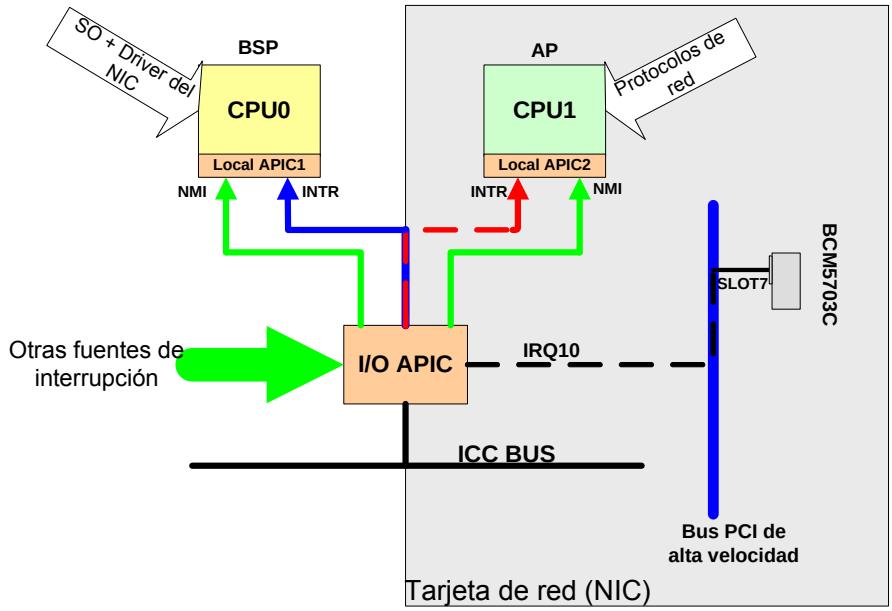


Figura 3.10. Interrupciones en el modelo de simulación para la externalización offloading

Por otro lado, la segunda restricción descrita en la Sección 3.6 implica minimizar o, si fuese posible, eliminar las interacciones o colisiones entre la CPU_0 y la CPU, que ejecuta los protocolos de comunicación (CPU_{NIC}), con los buses del sistema, la memoria y con la *cache*. En Simics es posible incluso eliminar totalmente dichas interacciones gracias a los modelos de tiempo, con los que es posible modelar (y por supuesto también eliminar) las colisiones en los buses o en los accesos a memoria. Para eliminar las colisiones en los buses del sistema se han modelado los buses de los que hace uso la CPU_{NIC} solo como conectores. De esta forma, los accesos concurrentes por parte de los dos procesadores al bus del sistema no generarán una colisión. Además, la CPU_{NIC} utiliza su propio espacio de memoria, como puede verse en la Figura 3.10, y se han configurado los modelos de tiempo para que sean posibles accesos concurrentes a memoria por parte de los dos procesadores.

Como se ha dicho el modelo presentado en las Figuras 3.9 y 3.10, tiene como objetivo cumplir con las restricciones anteriores, y así simular un procesador lo más próximo posible al interfaz de red, al mismo tiempo que lo más separada posible del resto del sistema. Además, para conseguir eso sin cambios significativos en el sistema operativo es necesario, además, distribuir el software en el sistema de forma apropiada, como se verá en la sección siguiente.

En el modelo de externalización se ha incluido un APIC local a cada procesador, un APIC de E/S y un bus ICC (Interrupt Controller Communication), a través del cual se comunican los APIC. El APIC de E/S recibe las interrupciones tanto de la interfaz de

red (desde el bus PCI) y del resto de dispositivos que pueden generar interrupciones como puede ser la interfaz de disco (Interfaz IDE o SCSI por ejemplo). De la misma forma que en un sistema real y siguiendo la especificación del fabricante (en nuestro caso Intel [INT97]).

Con el fin de simular la proximidad entre el procesador que procesará los protocolos y la NIC, se han utilizado diferentes interfaces de modelado de tiempo conectados a las entradas del puente norte, de forma que el bus PCI actúa en este caso sólo como conector y se simula solo a nivel funcional. Esto permite conectar directamente la tarjeta de red al procesador (CPU_{NIC}), desde el punto de vista de las transacciones realizadas entre ambos. Además, ya se ha comentado que se ha incluido un controlador de interrupciones APIC con el fin de mantener la funcionalidad del sistema. En este caso, se ha modificado el comportamiento del APIC para que las interrupciones generadas en la NIC lleguen directamente a la CPU_{NIC}, de forma, que mientras que las interrupciones que llegan a la CPU_0 lo hacen a través del APIC (que en este caso se está simulando a nivel funcional y de temporización) las interrupciones que llegan a la CPU_{NIC} desde la NIC, pasen a través del APIC (que en este caso solo se simula a nivel funcional). Así, a la CPU_{NIC} no le afectan las interrupciones que lleguen a través del APIC y que van dirigidas a la CPU_0.

En la Figura 3.11 se muestra la secuencia temporal de operación del modelo de externalización con *offloading* para la recepción de un paquete. Inicialmente ① se recibe en la tarjeta de red un número de paquetes en la interfaz de red, determinado por el uso de tramas jumbo (y su longitud). Una vez recibidos los paquetes de datos, el interfaz de los almacena en el buffer en anillo. Cuando el anillo está lleno, los paquetes se transfieren al espacio de memoria asignada a la NIC ②.

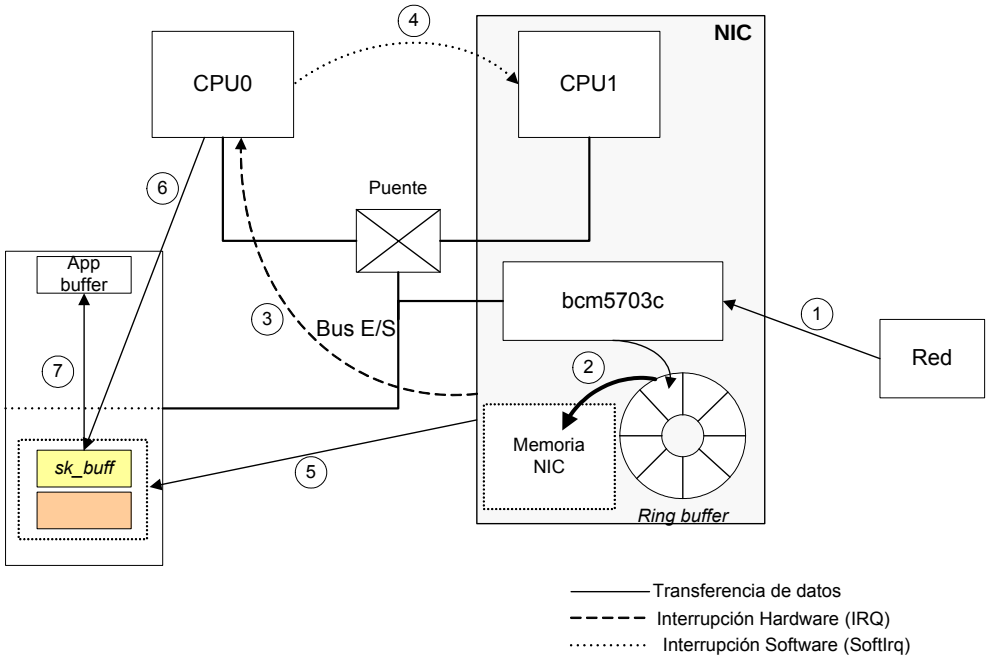


Figura 3.11. Operación del modelo de simulación offloading

Una vez que la transferencia anterior se completa, se envía una interrupción a la CPU principal del nodo (CPU_0) ③ donde se ejecuta el driver que, a su vez, ejecuta la interrupción software forzada (*softirq*) asociada al procesamiento de la pila TCP/IP ④ en la CPU de la NIC (CPU_{NIC}). Esta *softIrq* procesará la pila de protocolos TCP/IP y posteriormente copiará los datos al socket TCP. Finalmente, la CPU_0 copia ⑥ los datos desde el socket TCP al buffer de la aplicación ⑦ mediante las llamadas al sistema relativas a la recepción desde los sockets. Por tanto, como se genera únicamente una interrupción cada vez que el buffer en anillo está lleno, no se está utilizando la técnica "una interrupción por paquete" sino que se está utilizando la técnica "una interrupción por evento", siempre que el paquete recibido pueda ser alojado en el anillo de recepción de la NIC, de la misma forma que ocurre en un TOE real.

3.5.2 Distribución del software en el sistema

Como se ha adelantado, para distribuir el software en el sistema se han utilizado los objetos CPUSET. Se trata objetos ligeros que pueden integrarse en los *núcleos* de Linux a partir de la versión 2.6.9. La idea original de los objetos CPUSET se propone inicialmente en [BUL04] y consiste en hacer particiones de los sistemas con más de un

procesador, creando distintas áreas de ejecución. De esta manera, el sistema operativo dispone de una herramienta de administración de los recursos de procesador. El control de los recursos de procesador es útil en entornos de computación de altas prestaciones (HPC), en máquinas NUMA o en cualquier ámbito en que pueda ser necesaria la reasignación de recursos del sistema dependiendo de la carga del mismo. Mediante los objetos CPUSET es posible crear conjuntos de procesadores (de ahí la denominación *cpusets*) asignando una o varias CPU a cada conjunto. Además, la reasignación de procesadores entre conjuntos se puede realizar de forma dinámica.

Muchas de las aplicaciones HPC utilizan la política de *un proceso por cada procesador*: es lo que también se llama *cpu-bound applications* [FIG96]. Mediante los objetos CPUSET se puede conseguir que un procesador solo pueda ejecutar un determinado proceso. Se elimina así la necesidad de que el planificador del sistema operativo tenga que redistribuir los recursos, en cuanto a procesador o memoria se refiere, pudiendo ser estos asignados de forma manual por el administrador del sistema. En sistemas operativos como Solaris, existe la utilidad *pset* que permite la configuración del uso de los procesadores aunque de una forma menos flexible que CPUSET.

Un uso típico de CPUSET puede verse en máquinas destinadas a dar servicio web en las que, además del propio servidor web, debe ejecutarse una base de datos. En este caso CPUSET puede dividir un sistema en dos, de forma que un procesador y parte de la memoria ejecute solo el servidor web (Tomcat [TOM07], por ejemplo) y el otro procesador y memoria ejecute el servidor de base de datos (Oracle [ORA07], por ejemplo). Esto permite dimensionar la máquina de forma correcta, dedicando determinadas procesadores a la ejecución de los procesos de la base de datos y otros procesadores a los procesos del servidor web por ejemplo, distribuyendose los recursos de una manera más eficiente.

Otro uso típico de CPUSET es tiene en las aplicaciones críticas. Dado que se puede aislar uno o variose procesadores, se puede hacer que determinadas aplicaciones críticas utilicen un procesador y la memoria necesaria de forma exclusiva.

Dado que CPUSET se puede integrar en el *núcleo* de Linux, es posible dotar a las máquinas pequeñas de una capacidad de distribución de carga habitualmente solo disponible en sistemas mucho más grandes y costosos.

Los diferentes CPUSET definidos en el sistema se pueden configurar como abiertos o como exclusivos a través del *flags* de configuración *cpu_exclusive*. Aquellos CPUSET configurados como exclusivos (*cpu_exclusive=1)* solo permitirán en sus

procesadores la ejecución de procesos asociados a ese CPUSET. De la misma forma, el acceso a memoria puede ser también exclusivo para un CPUSET.

Aunque las librerías de CPUSET se puede utilizar en cualquier máquina en la que se ejecute un sistema operativo Linux con versión de *núcleo SMP* posterior a la 2.6, su mayor utilidad se tiene en sistemas con varios procesadores, en los que se puedan establecer particiones, reduciendo así la interacción entre procesos que de otra forma podrían estar ejecutándose en la mismo procesador. En el caso de que se haya activado el *flag* de exclusividad de un CPUSET, nos aseguramos que los procesos que se ejecuten en dicho conjunto, no estarán afectados de ninguna forma por otros procesos. En el caso de que un CPUSET contenga más de un procesador, el planificador del sistema operativo podrá decidir en que cpu ejecutar un proceso, pero siempre dentro del mismo CPUSET.

Con las librerías del conjunto CPUSET es posible incluir las hebras del *núcleo* del sistema operativo o las hebras asociadas a la gestión de interrupciones en un CPUSET. De esta forma, se puede conseguir que las hebras del núcleo o las que ejecutan las rutinas asociadas a interrupciones no afecten al resto del sistema, al no competir estas hebras con los demás procesos.

Por otro lado, los CPUSET pueden definirse como estáticos o dinámicos. El tipo de *cpuset* del que se ha hablado hasta ahora es estático, es decir, se asigna un grupo de procesador a un CPUSET y esta asignación no variará en tiempo de ejecución. En el caso de una asignación dinámica, es posible reasignar un procesador contenido en un CPUSET a otro, con el fin de redistribuir los recursos y no tener procesadores ociosos en el sistema. Para ello, existe además un gestor de carga de trabajo que en el caso de que esté activo y en caso de cargas de trabajo altas, puede crear un nuevo CPUSET y asignarle una tarea determinada, destruyéndolo a la finalización de la misma.

En la actualidad, la versión de CPUSET original que incluyen los núcleos a partir de la versión 2.6.11 implementa su gestión a través de un sistema de ficheros virtual. Por tanto, una vez instalado el correspondiente módulo del núcleo al montar el sistema de ficheros virtual */dev/cpuset*, se creará una estructura de directorios (como puede verse en la Figura 3.12) que permitirá la gestión.

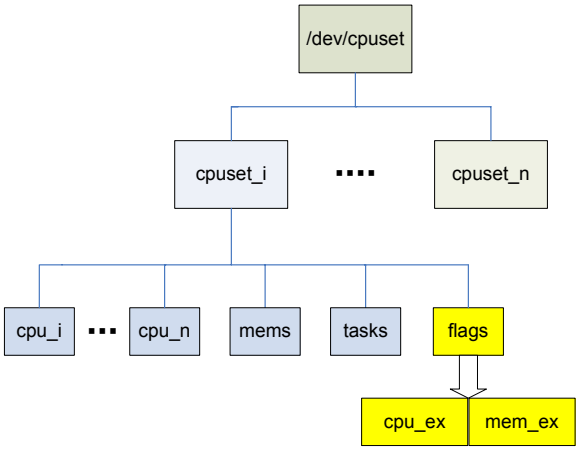


Figura 3.12. Estrutura de ficheros cpuset

Dentro de cada CPUSET pueden también crearse otros CPUSET hijos, dando lugar a una jerarquía de objetos, como puede verse en la Figura 3.13.

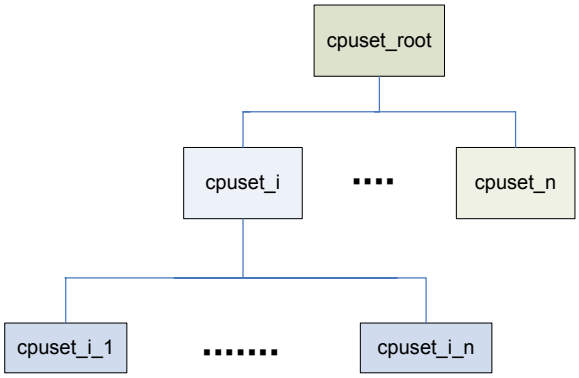


Figura 3.13. Jerarquía de objetos cpuset

En nuestro modelo de simulación la máquina se dividirá en dos partes. Una de ellas se encargará de ejecutar los procesos asociados a las comunicaciones. Esto puede hacerse simulando una máquina SMP en la que uno de sus procesadores actúa como *procesador para la externalización*, de forma similar a como se describe en [WES04]. Sin embargo, mientras que en [WES04] se realiza la emulación del *procesador para la externalización (Offload Engine)*, en nuestro trabajo se está simulando una máquina con un hardware configurado para que la separación entre la parte que ejecuta la aplicación y la parte que ejecuta los procesos de comunicaciones no este implementada sólo mediante software, sino que se han realizado las modificaciones en el comportamiento

del hardware (que se han descrito en la Sección 3.7.1) para modelar el comportamiento de un sistema con una tarjeta de red de tipo TOE. Como se verá, en el caso de la simulación del sistema con externalización mediante *onloading* se simula un SMP, y se utiliza CPUSET así como otras técnicas descritas más adelante en la Sección 3.7 para conseguir la externalización.

En las simulaciones realizadas se han utilizado máquinas con dos procesadores. Uno de ellos ejecuta la aplicación y el otro los procesos de comunicación. En el caso de que se hubiesen utilizado máquinas con más de dos procesadores, los procesadores que se encargaran de la aplicación se comportarían como un SMP, compartiendo los recursos, mientras que los procesadores encargados de las comunicaciones, aunque compartirían en determinados momentos los recursos del sistema (memoria, buses, etc.) lo harían solo a nivel funcional, y no en cuanto a la temporización, al quedar separados del resto del sistema como que se ha descrito en la Sección 3.6.1 para el caso de un procesador.

Para reproducir lo más fielmente posible las condiciones correspondientes al comportamiento de externalización, necesitamos que los procesos de comunicación estén asociados a un procesador en exclusiva, para la ejecución de estos procesos no se vea afectada por otros procesos que se están ejecutando en la máquina. Ya se ha visto en la Sección 3.6 que mediante el modelo de simulación, se consigue separar el procesador que se encargará de los procesos de comunicaciones del procesador que ejecutará los demás procesos del sistema operativo y la aplicación. Sin embargo es necesario que el funcionamiento del sistema operativo sea coherente con el modelo hardware. Precisamente para esto es para lo que se ha utilizado CPUSET. En un CPUSET *(cpuset_root*) permanecerán todos los procesos del sistema operativo y la aplicación, y en otro CPUSET creado tras el arranque se ejecutarán las hebras TCP/IP responsables de la comunicación, de manera exclusiva. Así, este procesador no atenderá peticiones del otro procesador del sistema que no se refieran a los procesos de comunicaciones a los que queda dedicad en exclusiva.

El funcionamiento de la máquina que hemos modelado, tal y como se muestra en las Figuras 3.8 y 3.9, se puede resumir de la siguiente forma:

1) *Arranque de Linux, como un sistema SMP*. La máquina sobre la que está ejecutándose este núcleo, tiene las características hardware descritas en la Sección

3.5 en cuanto a comportamiento de los procesadores, buses y puentes, y a su interacción con el resto del sistema.

2) *Configuración de CPUSET*. En el arranque, se inicializa CPUSET, configurando dos conjuntos de CPUs: el conjunto *raiz*, de donde ha arrancado el núcleo de Linux y otro conjunto, que denominamos *cpuset_1*, en el que se incluye la CPU_{NIC} y un nodo de memoria, ambos con acceso exclusivo. Todos los procesos del sistema operativo y los procesos que se arranquen a continuación se ejecutarán en el conjunto *raiz*.
3) *La* CPU_{NIC} *se pasa a modo offline*, permitiendo la ejecución solamente de las hebras *del software de comunicación* (hebras TCP/IP del núcleo).

Esta forma de operación es posible gracias al modelo multihebra del núcleo de Linux y a la forma en la que está implementada la pila de protocolos TCP/IP en las actuales versiones del núcleo de Linux. Linux arranca una hebra especial para ejecutar los gestores de paquetes (*packet handlers)* [WEN06]. Esta hebra se ejecuta de forma separada y concurrente a las que ejecutan el *driver* de la interfaz o las rutinas de servicio a las interrupciones. Por razones de optimización del tiempo de procesador invertido en las interrupciones (que afecta de forma directa y decisiva al rendimiento global del sistema), así como de la implementación por capas de TCP/IP en Linux por capas preferible que el procesamiento de la pila TCP/IP se realice en una hebra independiente al *driver*. Algunas implementaciones antiguas de TCP/IP en Linux procesaban los paquetes entrantes en el contexto de la propia rutina de servicio de la interrupción. Sin embargo, con los sistemas y redes actuales y el ancho de banda de los enlaces, esta implementación colapsaría todo el sistema [BEN05, MAT03]. Por esta razón, en las implementaciones actuales de Linux (las utilizadas en nuestro trabajo experimental) que proporcionan soporte para tarjetas de red gigabit y multigigabit, los paquetes entrantes no se procesan en el contexto de la propia rutina de servicio a la interrupción, sino en contextos diferentes. Hay que tener en cuenta que el núcleo actual de Linux versión 2.6 dispone de tres contextos de ejecución (contexto de procesos, contexto *bottom-half* y contexto de interrupción), utilizando además mecanismos para encolar la ejecución de rutinas de servicio a las interrupciones y otras tareas del núcelo, como son las *softirq*, los *tasklets* y los *bottom-halves* [MAT03]

En la Figura 3.14 se puede ver más detalladamente el proceso de recepción de un paquete desde la red, con la actual implementación TCP/IP de Linux.

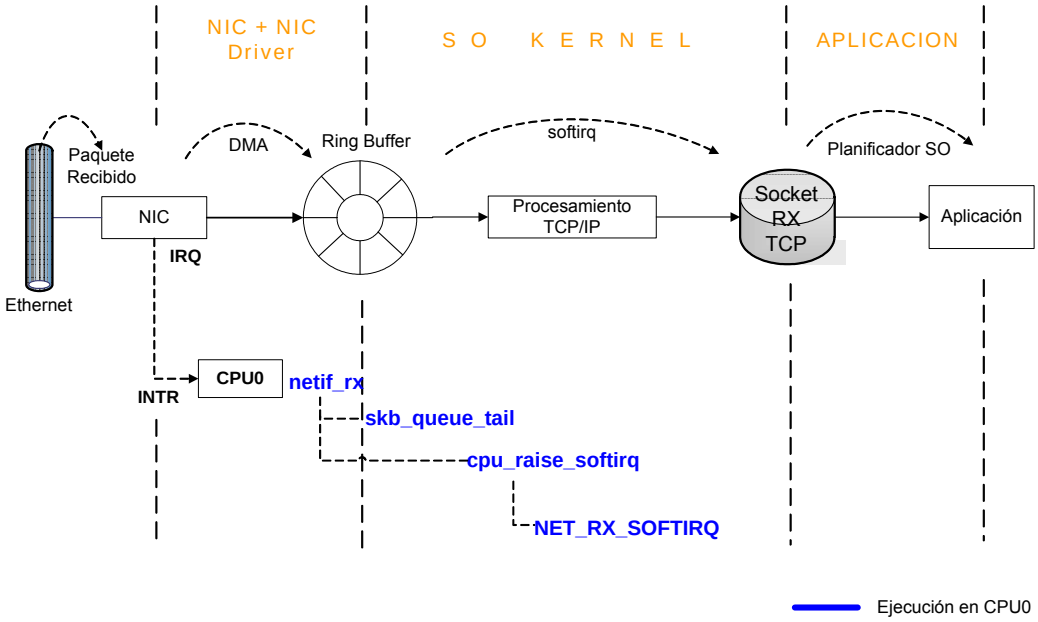


Figura 3.14. Proceso de recepción de un paquete en Linux

Cuando un paquete llega a la tarjeta de red, esta copia el paquete recibido a un *buffer* circular (*Ring Buffer*) utilizando DMA. Dicho buffer circular consiste en una serie de descriptores de paquete que se utilizarán para determinar cuando se puede copiar un paquete en la memoria principal, de forma que éste quede accesible por el núcleo del sistema operativo para el procesamiento en capas superiores de la pila de protocolos (tradicionalmente, la tarjeta de red y su correspondiente driver implementan sólo las funciones de las capas 1 y 2 del modelo OSI). Sólo se copiará un paquete a memoria principal, si existe un descriptor de paquete libre. En la memoria principal, se utilizan estructuras *sk_buff* [BEN05] que son preparadas mediante la función *skb_queue_tail* para copiar los paquetes. Por tanto, cuando se habla de copia de los paquetes al buffer circular, en realidad se está indicando la copia de dichos paquetes a las estructuras *sk*_buff anteriormente citadas. Las características de dichas estructuras dependerán del protocolo utilizado (a través del atributo *skb→protocol*) así como de la MTU (*Maximum Transfer Unit)* utilizada, encontrándose en una zona de la memoria principal compartida por la tarjeta de red y el núcleo del sistema operativo. Una vez que dicha transferencia se ha completado, y que por tanto, los paquetes son están disponibles para el núcleo, la tarjeta de red envía una interrupción a la CPU que ejecuta el *driver* correspondiente. Dicho *driver* deberá informar al núcleo del sistema operativo

de la recepción de nuevos paquetes que están a la espera de ser procesados por capas superiores de la pila de protocolos, siendo la función *netif_rx* [BEN05], definida en la librería */net/core/dev.c de Linux,* la interfaz entre el *driver* y el núcleo. Esta notificación puede realizarse de dos formas. Tradicionalmente, tras la interrupción generada por la tarjeta de red, informando a la CPU de la existencia de un nuevo paquete en memoria principal que está a la espera de ser procesado, dicha CPU ejecuta el driver de la tarjeta de red, y éste a su vez, llama a la rutina *netif_rx* que, en contexto de interrupción, planifica una interrupción software (*softirq).* El arranque de la interrupción software se realiza mediante la función *cpu_raise_softirq* [BEN05, BOV05]. Dicha *softirq* será la encargada del procesamiento de los paquetes en capas superiores a la capa 2, de los paquetes disponibles en memoria principal. Este método implica que el driver genera una interrupción software para cada paquete recibido, lo que supone una sobrecarga considerable para la CPU cuando el ancho de banda es elevado o en situaciones de alto tráfico en la red.

Sin embargo, los nuevos núcleos de Linux, implementan la interfaz conocida como NAPI (New API), introducida en el Capítulo 1 (Sección 1.3.3). El objetivo de NAPI es reducir la sobrecarga generada en la CPU, mediante la mezcla de interrupciones y sondeo (*polling*). De esta forma, si se recibe un nuevo paquete cuando aún no se ha terminado de procesar el anterior, el *driver* no genera una nueva interrupción software, sino que el núcleo continúa procesando todos los paquetes que se encuentren en la cola de entrada de la interfaz de red [SAL01, BEN05, BOV05]. En nuestro trabajo experimental hemos utilizado NAPI, ya que está implementado en las versiones actuales de Linux.

Aunque por defecto, una *softirq* se ejecuta en el mismo procesador en la que se inició, es posible conseguir que se ejecute en un procesador específico [MAT03], mediante la modificación de las librerías */net/core/dev.c* y *kernel/softirq.c* [BEN05, BOV05, JON05]. Esta *softirq* llama a la rutina *net_rx_softirq()* que en función del protocolo de las capas superiores, ejecutará las rutinas necesarias para el procesamiento del paquete (*tcp_v4_rcv* en el caso de protocolo TCP). A partir de aquí, los datos extraídos de la pila TCP/IP se ubican en el *socket* desde donde los podrá recoger la aplicación.

Por tanto, en la recepción de un paquete desde la red hasta la aplicación se pueden distinguir tres fases:

1. Recepción del paquete

2. Copia de los paquetes a las estructuras *sk_buff* de acuerdo a los descriptores del *buffer* circular mediante DMA.
3. Envío de una interrupción hardware (IRQ) a la CPU, para que esta ejecute el driver de la tarjeta de red.
4. Notificación al núcleo del sistema operativo de la recepción de nuevos paquetes disponibles para su procesamiento.
5. Arranca la interrupción software (*softirq*) que continuará con el procesamiento del paquete, fuera del contexto de la interrupción. Al finalizar, la *softIrq* habrá copiado los datos extraídos del protocolo en el correspondiente *socke*t de recepción, quedando disponibles para la aplicación.

En la implementación de la técnica de *offloading* que hemos realizado se aprovecha que disponemos de dos procesadores. El sistema operativo arrancará en la CPU_0/BSP (*Boot system Processor*) que es donde reside el driver de la interfaz de red. Por tanto, la distribución del software se realiza como se muestra en la Figura 3.18.

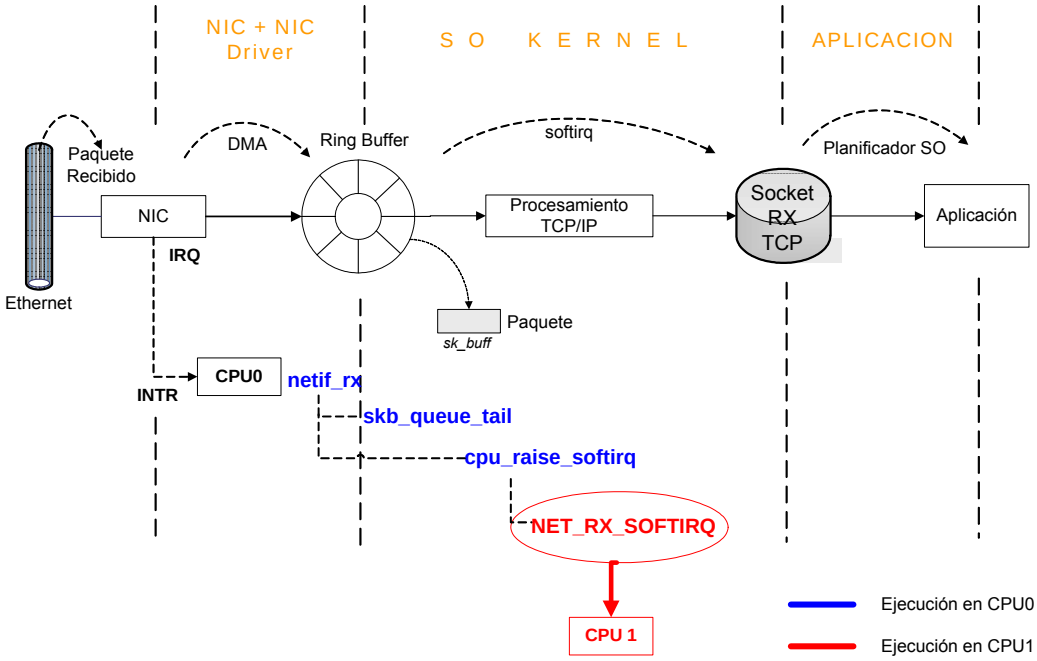


Figura 3.18. Proceso de recepción de un paquete con offloading

En este caso, la interrupción llega a la CPU_0. Esta ejecuta el driver y el paquete se copia al *buffer* circular (*Ring Buffer*). A partir de aquí, se arranca la *softirq* en la CPU_{NIC} en lugar de en la CPU_0. Para ello, es necesario modificar el código de la rutina *netif_rx*

de forma cuando se arranque una *softIrq* para procesar un paquete de red, esta se inicie y procese en la CPU_{NIC} que está siendo utilizada en exclusiva para el procesmiento de los protocolos de comunicación, como se ha descrito en la Sección 3.6.2.

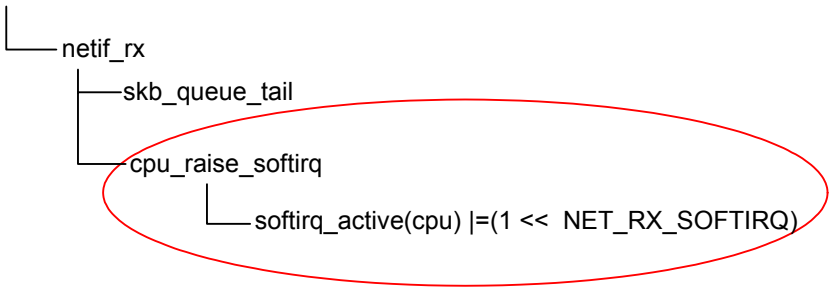


Figura *3.19. Código de ejecución de softirq para recepción de paquetes*

3.7 Modelo de simulación para la externalización onloading

Para evaluar los beneficios que aporta la alternativa de externalización de protocolos de comunicación mediante *onloading* y comparar con los obtenidos mediante *offloading*, se ha desarrollado otro modelo correspondiente a implementación de la externalización mediante *onloading* [REG04a, REG04b, ORT08]. Igual que en la descripción de la Sección 3.5 para el modelo de simulación de la externalización mediante *offloading*, en esta sección se describe el modelo hardware para la simulación de la técnica de *onloading* así como la distribución del software que se requiere.

3.7.1 Modelo hardware para la simulación de *onloading*

Igual que para el caso de utilizar *offloading*, el modelo hardware para la simulación de la técnica de onloading incluye dos sistemas configurados a medida y una red Gigabit Ethernet para conectarlas entre sí, de la misma forma como se haría en el mundo real.

Como en la simulación de la externalización con *offloading*, se han utilizado los diferentes modelos de temporización necesarios para simular las latencias en los buses del sistema. Además, se ha modificado el perfil de interrupciones de forma que sea posible definir diferentes latencias para interrupciones que provienen de diferentes dispositivos.

También se necesita un modelo sin externalizar para poder evaluar las mejoras introducidas por la externalización de protocolos mediante esta técnica. En este caso, utilizaremos el mismo modelo que se ha descrito en la Sección 2.6.

En cuanto al comportamiento de las interrupciones, aunque Simics no soporta la simulación de un nodo con más de un APIC (*Advanced Programmable Interrupt Controller*) de E/S, es posible configurar el APIC correspondientede forma que las interrupciones generadas en el bus PCI se envien directamente a la CPU_1, que en el caso de la externalización con *onloading* es el procesador que se encarga de todas las tareas de comunicaciones incluyendo la ejecución del driver correspondiente.

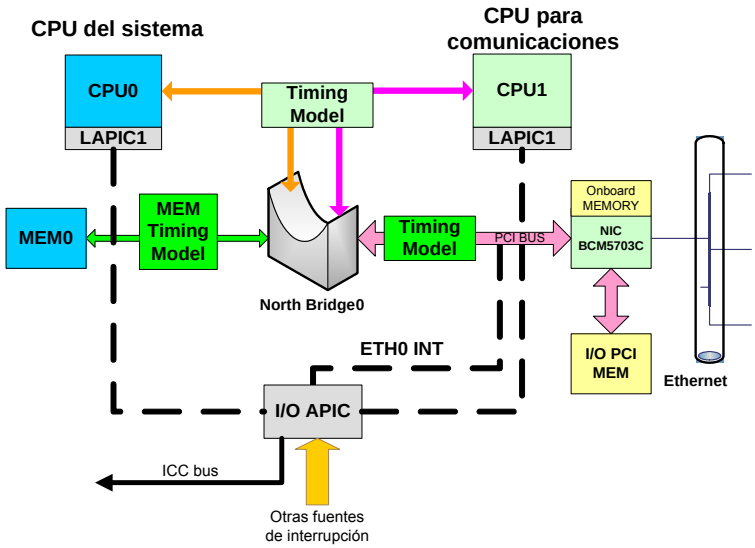


Figura 3.16. Modelo de simulación para Onloading

La Figura 3.16 muestra todos los elementos incluidos en el modelo de simulación con Simics. Las principales diferencias entre los modelos para la simulación de la externalización mediante *onloading* u *offloading* son las siguientes:

1. En el modelo de externalización mediante offloading, la tarjeta de red no genera interrupciones hardware a la recepción de cada paquete. La interrupción hardware se genera después de transferir todo el contenido de la memoria de la tarjeta de red a la memoria principal del nodo. Dicha interrupción llega a la CPU_0 que ejecuta el *driver* sin pasar por el APIC de E/S. En el modelo de externalización mediante *onloading*, la tarjeta de red no tiene capacidad de almacenamiento de paquetes y las

interrupciones tienen que pasar a través del controlador de interrupciones, el bus APIC, etc.

2. En el modelo de externalización mediante *offloading*, el *driver* de la NIC se ejecuta en la CPU_0, es decir, en el mismo procesador que el sistema operativo, mientras que en el modelo de externalización con *onloading* dicho driver se ejecuta en la CPU_1 que ejecuta todo el software de comunicación.
3. En el modelo de externalización mediante offloading, se han eliminado las colisiones en los accesos a memoria desde la CPU_1, dado que se está modelando una memoria en la propia tarjeta de red, y por lo tanto no comparte los buses de E/S del sistema.
4. En ambos casos, las hebras TCP/IP se ejecutan en la CPU_1. No obstante, en un caso el procesador está en la NIC y en otro se tarata de uno de los procesadores que comparten memoria en un SMP.

Así pues, para la simulación de la externalización mediante *onloading* se han utilizado dos procesadores conectados al puente norte, como puede verse en la Figura 3.15. Para llegar a dichos procesadores, las interrupciones tienen que pasar a través del bus del APIC y del APIC de E/S. Esto ocasiona retardos en la propagación de las interrupciones debido al efecto del controlador de interrupciones (APIC) simulado. Además, este controlador tiene que decidir a qué procesador llega finalmente la interrupción. Es posible configurar el sistema de modo que el driver de la NIC se ejecute en la CPU_1, en lugar de en la CPU_0. De esta forma, cada vez que se recibe un paquete, la NIC genera una interrupción que llega al APIC de E/S, y el sistema operativo inicia la ejecución del driver en la CPU_1. Además, la correspondiente hebra TCP/IP se ejecuta en la misma CPU_1. El perfil de interrupciones utilizado para la simulación de onloading se muestra en la Figura 3.17.

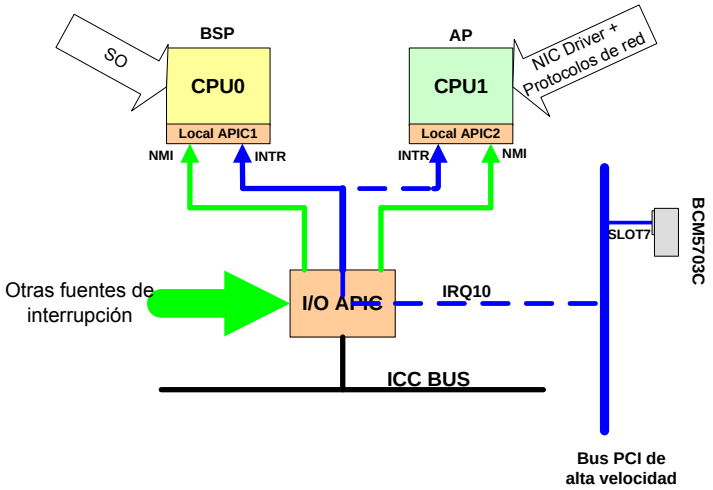


Figura 3.17. Perfil de interrupciones simulado para onloading

En la Figura 3.18 se muestra la operación del modelo de externalización mediante *onloading* para la recepción de un paquete. Inicialmente ① se recibe un número de paquetes en la tarjeta de de red, determinado por las tramas utilizadas (jumbo o no) y su longitud. Una vez recibidos los paquetes de datos, se almacenan en el buffer en anillo ② y la NIC genera una interrupción que llega al APIC E/S ③ y el sistema operativo comienza la ejecución del driver de la NIC en la CPU_1. De esta forma, la hebra TCP/IP correspondiente a la interrupción software que procesará los paquetes TCP/IP, se ejecuta en la misma CPU_1. Una vez que la ejecución de dicha interrupción software ha finalizado, el sistema operativo realiza la copia de los datos a la zona de memoria de usuario ④ y ⑤.

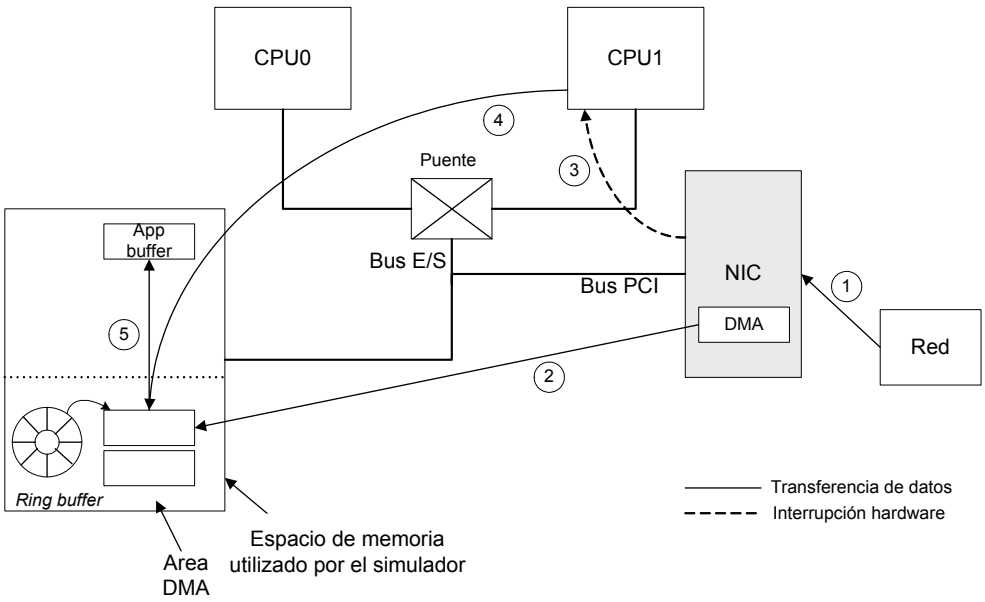


Figura 3.18. Operación del modelo de simulación onloading

3.7.2 Distribución del software en el sistema

En el caso de utilizar la técnica de *onloading*, el procesamiento de los protocolos se ejecuta en uno de los procesadores centrales del sistema que no se dedica a la ejecución de las aplicaciones. Esto presenta el inconveniente de tener que utilizar los buses del sistema para la comunicación entre la interfaz de red y el procesador. No obstante, por otro lado, la ejecución del *driver* en el mismo procesador en la que se externalizan los protocolos de red. Para conseguir esto, se configura el sistema de forma que se asigne el servicio de la interrupción asociada al NIC (IRQ10 en nuestro caso) a la CPU_1 del SMP. Al igual que se hizo en el caso de *offloading*, no se permite la ejecución de ningún otro proceso en la CPU_1.

El proceso de recepción de paquetes en este caso puede verse en la Figura 3.19.

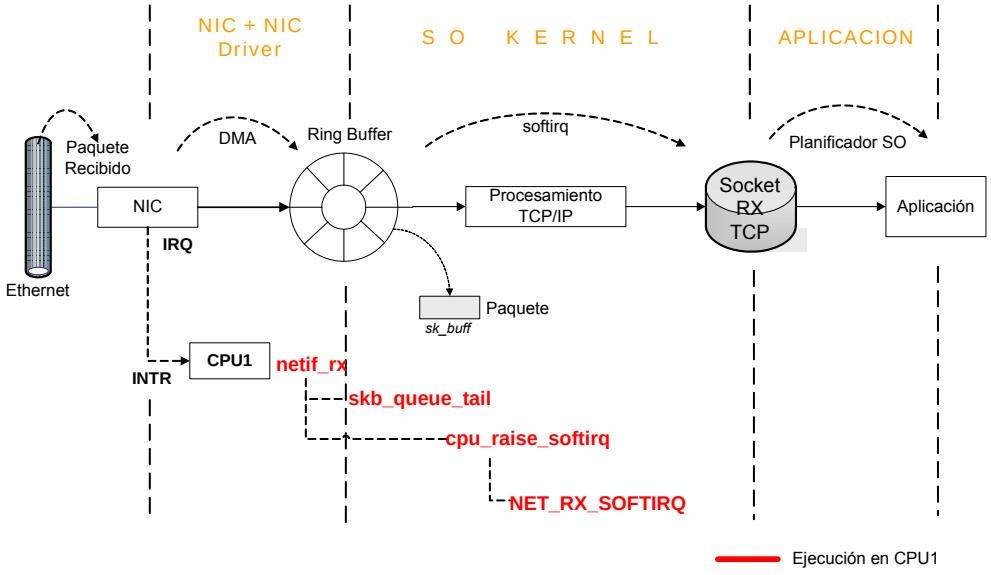


Figura *3.19. Proceso de recepción de paquetes con onloading*

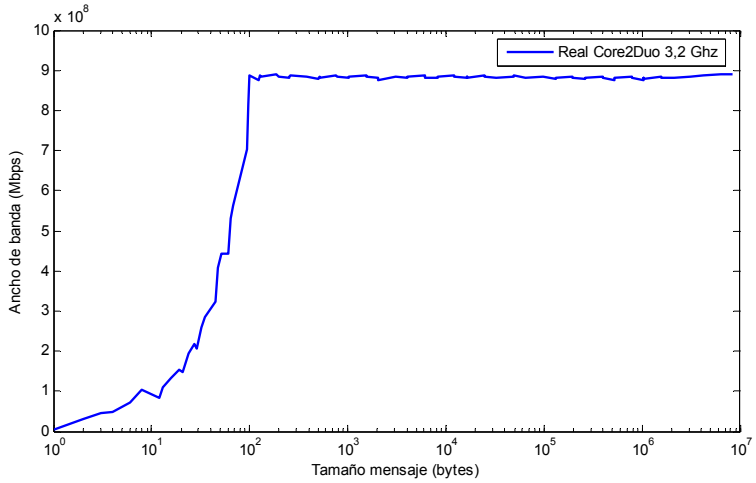
En nuestra implementación de la externalización con *offloading*, la interrupción hardware (IRQ) llega desde el NIC a la CPU_0 donde se ejecutaba el *driver* y posteriormente, se ejecutaba la rutina de servicio a la interrupción software asociada al procesamiento del paquete recibido en la CPU_1. Sin embargo, en el caso *onloading*, la interrupción hardware (IRQ) llega directamente a la CPU_1 del SMP, donde se ejecuta el

driver y posteriormente la rutina de servicio a la interrupción software (*softIrq)*, tal y como se muestra en la Figura 3.19.

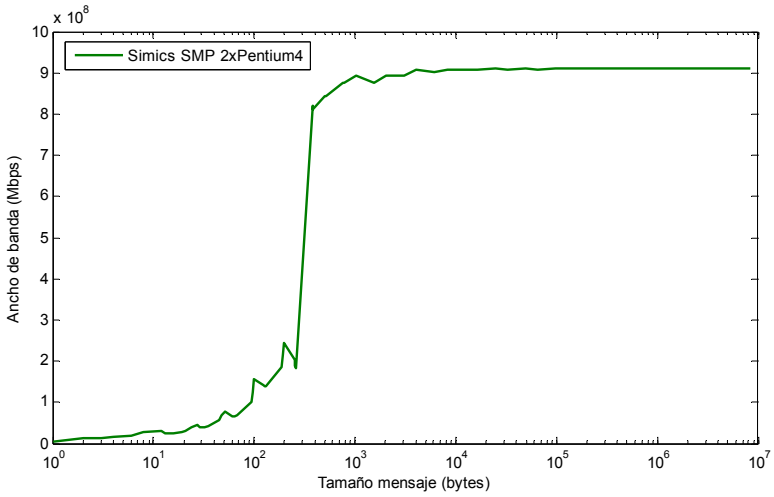
Esta implementación tiene la ventaja de liberar a la CPU_0 de la carga asociada al procesamiento de interrupciones ocasionadas por la llegada de paquetes TCP/IP, ya que el *driver* se ejecutará en la CPU_1. Al mismo tiempo, el procesamiento de los protocolos se realizará también en la CPU_1 ya que la *softIrq* se inicia en ella.

3.9 Validación de los modelos de simulación

Con el fin de validar las prestaciones del simulador y por tanto, la precisión de los resultados obtenidos, se han realizado medidas en una máquina real y se han comparado con los resultados obtenidos mediante simulación. En las Figuras 3.20(a) y 3.20(b) se muestra una comparación de ambos resultados.



(a)



(b)

Figura 3.20. Validación del simulador. (a) Real. (b) Simulación

En la Tabla 3.3 se muestran las características de los sistemas real y simulado utilizados en los experimentos de validación. Tal y como se muestra en dicha tabla, los sistemas real y simulado no son exactamente iguales. Además, así como el sistema real está basado en procesadores Core2Duo los cuales disponen de tecnología SMT, el sistema simulado utiliza procesadores Pentium4 sin SMT. Esto, junto con la diferencia de memoria RAM en ambos sistemas hace que existan diferencias en las prestaciones de ambos sistemas. No obstante, en las Figuras 3.20 y 3.21 se observa como el comportamiento cualitativo es muy similar, así como el ancho de banda máximo alcanzado.

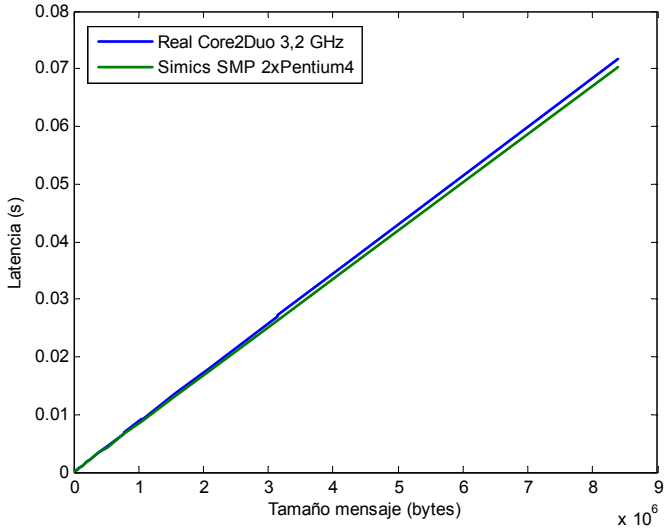


Figura 3.21. Validación del simulador. Latencia

Tabla 3.3. Características de los sistemas utilizados para la validación del simuladore externalización

	Procesador	Memoria	*Cache*	Interfaz de red
Real	Core2Duo 3,2 GHz	2 GB	L1 32Kb, L2 2MB	bcm5703c
Simulado	SMP 2xPentium4 3,2 GHz	256 Mb	L1 32 Kb, L2 2MB	bcm5703c

3.10 Resumen

En este capítulo se ha presentado la externalización de los protocolos de comunicación, como técnica que pretende optimizar el comportamiento de la interfaz de red para que sea posible aprovechar el ancho de banda proporcionado por los enlaces de altas prestaciones, al tiempo que se disminuye la sobrecarga del procesador asociada a los procesos de comunicación. Se han visto las distintas alternativas para implementar la externalización de protocolos y se han modelado convenientemente las distintas alternativas para permitir su simulación a través del simulador de sistema completo Simics. Así, se han construido modelos que permiten obtener resultados realistas, en los que se ha incluido tanto el sistema operativo (Linux en nuestro caso) como aplicaciones (*benchmarks*) que se podrían ejecutar en una máquina real.

Entre las implementaciones presentadas para conseguir la externalización de los protocolos, se han propuesto distintas implementaciones de la externalización mediante *offloading y* mediante *onloading*, y se han descrito los modelos de simulación que hemos desarrollado en cada caso, tanto en lo referente a la configuración hardware como a la distribución del software.

Finalmente, se propone una alternativa de mejora de la interfaz de red basada en la hibridación de las estrategias de externalización mediante *offloading* y *onloading.* Mediante esta alternativa híbrida se trata de aprovechar las ventajas que proporcionan cada una de las técnicas anteriores y evitar algunos de sus inconvenientes, con el fin de mejorar las prestaciones del sistema de comunicaciones.

En el capítulo siguiente, se utilizarán los modelos de simulación desarrollados, para evaluar y comparar las prestaciones ofrecidas por cada técnica de externalización con distintos niveles de carga y *benchmarks*. También se utilizará el modelo LAWS [SHI03] para orientar la justificación de los resultados obtenidos.

Capítulo 4

Estudio experimental de las alternativas de externalización

En el capítulo 3 se han descrito distintas alternativas para la externalización de protocolos de comunicación, junto con dos implementaciones que hemos propuesto correspondientes a la técnica de *onloading* y *offloading* respectivamente. Además, se han mostrado los modelos de simulación desarrollados para el simulador de sistema completo Simics, con el fin de poder evaluar posteriormente las prestaciones que proporciona de cada técnica. Como se ha indicado, actualmente existe cierta controversia respecto a la técnica de externalización a seguir en función de los beneficios aportados por cada una y existen implementaciones comerciales tanto de la técnica de *onloading* [IOA05] como de la de *offloading* [DEL06]. Así, aunque se ha generado un trabajo de investigación muy activo en el que se defiende una u otra alternativa. Faltan trabajos donde se comparen las dos alternativas en condiciones experimentales similares. Esto se debe, en parte, a la falta de herramientas de simulación que permitan generar sistemas para las distintas alternativas.

En este capítulo, tras presentar en la Sección 4.2 los diferentes programas de *benchmark* que se han empleado y sus principales características, se utilizan, en la Sección 4.3, los modelos que hemos elaborado incorporando los modelos de tiempo que se han desarrollado para hacer posible la simulación de sistemas de E/S en Simics y que se describen en el capítulo anterior para la evaluación de las técnicas de *offloading* y de *onloading*. Para realizar dicha evaluación, se realizan diferentes medidas de anchos de banda proporcionados, latencia de red y consumo de CPU, como se muestra en las

Secciones 4.3 y 4.4. Posteriormente, en la Sección 4.5 se realiza un análisis comparativo beneficios aportados por cada alternativa y en la Sección 4.6 se utiliza el modelo LAWS [SHI03] en para estudiar el comportamiento de las dos técnicas de externalización en función de diferentes parámetros del sistema. De esta forma, al comparar los resultados obtenidos experimentalmente con los predichos por el modelo teórico LAWS [SHI03] se puede comprobar la vigencia de dicho modelo. En este ámbito, también se ha propuesto un modelo LAWS [SHI03] modificado para conseguir un mejor ajuste a los resultados experimentales y obtener así una predicción más aproximada a las mejoras que se pueden esperar. Finalmente, en la Sección 4.7 se presentan los resultados obtenidos mediante modelos de simulación a los que se ha incorporado un modelo detallado de *cache* en dos niveles. A partir de los resultados experimentales obtenidos, también hemos realizado propuestas de mejora para las arquitecturas de comunicación.

4.1 Introducción

Como se ha visto en el Capitulo 3, la externalización consiste en la implementación de la interfaz de red en un procesador del nodo distinto al procesador central. De esta forma, al llevar a cabo el procesamiento de los protocolos de comunicación en otro procesador, se liberan ciclos de reloj de la CPU principal que quedan disponibles para la ejecución de las aplicaciones. Este procesador puede estar incluido en la propia NIC (*offloading*), o puede ser uno de los procesadores de propósito general incluidos en un CMP o de las CPU de un SMP (*onloading*). En el caso de la externalización mediante *offloading,* la NIC puede interactuar directamente con la red sin la participación de la CPU del nodo, liberando a este no solo de la sobrecarga asociada al procesamiento de las interrupciones, sino reduciendo también la latencia asociada a mensajes pequeños de control, tales como ACKs, etc., debido a que estos no tendrían necesidad de llegar a la memoria principal y por tanto no atravesarían el bus de E/S. Existen implementaciones comerciales de *offloading* que externalizan diferentes partes de la pila de protocolos TCP/IP [BRO07] TCP/IP o la pila TCP/IP completa [DEL06, BRO07, CHE07, NET07a], mediante los llamados *TCP Offload Engines* o TOE, así como implementaciones comerciales de *onloading* como [IOA05].

Ha existido un interés considerable en la mejora de la interfaz de red, prueba de ello es la multitud de trabajos que: (1) describen técnicas para mejorar las prestaciones del sistema de comunicaciones, y el impacto que tendría el uso de un interfaz de red que

externalice los protocolos de comunicación [GAD07]; (2) proponen optimizaciones hardware y software en la implementación de la externalización mediante *offloading* [STE94]; (3) proponen la externalización mediante el uso de protocolos a nivel de usuario [BHO98]; (4) proponen la utilización de uno de los procesadores de un CMP o SMP para los procesos de comunicaciones [GRO05]; o bien que plantean la externalización de parte de las comunicaciones [KIM06]. Además, también hay trabajos que proponen modelos teóricos que permiten predecir las mejoras introducidas por la externalización en el sistema de comunicaciones como el modelo LAWS [SHI03] para transferencias de streaming o el model EMO [GIL05] para paso de mensajes.

Sin embargo, es difícil encontrar trabajos que proporcionen estudios comparativos de las dos principales técnicas de externalización (*offloading* y *onloading*) bajo las condiciones que definen su espacio de diseño en situaciones equiparables para todas las alternativas y establecer aquellas condiciones en las que es más beneficioso utilizar una técnica u otra.

Además, como se ha dicho en el Capítulo 1, es necesario tener en cuenta el efecto de las aplicaciones, del sistema operativo y de la interacción de los diferentes elementos del sistema (buses de E/S, memoria, etc.) para evaluar las mejoras proporcionadas por la externalización de una forma realista. La dificultad para disponer de simuladores adecuados se ha puesto de manifiesto en el Capítulo 2.

En este capítulo se evalúa el comportamiento de las alternativas de externalización a partir de los modelos de simulación de sistema completo presentados en el Capítulo 3, para comparar las técnicas y extraer conclusiones respecto a su ámbito de aplicación y a los beneficios que aporta cada técnica bajo diferentes condiciones de trabajo. De esta forma, y con las simulaciones realizadas, se puede tener una idea más clara de cuando resulta más beneficioso utilizar una técnica u otra.

4.2 Benchmarks para la medida de prestaciones de red

Existen diferentes programas *benchmarks* utilizados usualmente para la medida de prestaciones que pueden emplearse para comprobar el funcionamiento del modelo de externalización propuesto y, al mismo tiempo, para evaluar la mejora en las prestaciones que proporciona.

Las medidas de prestaciones que vamos a obtener, se refieren fundamentalmente a latencia y ancho de banda de comunicación. Para realizar estas medidas, se han

utilizado dos programas diferentes, Netperf [NET07b] y Netpipe [SNE96], que se describen en las secciones 4.2.1 y 4.2.2, respectivamente. Estos dos programas proporcionan una información similar en cuanto a ancho de banda (*throughput*) y latencia, para transferencias de secuencia de bits (*streaming*) o bidireccionales. Sin embargo, la arquitectura modular de *Netpipe*, independiente del protocolo utilizado, lo hacen más adecuado para nuestros propósitos sobre todo si en un trabajo futuro, se desea analizar las prestaciones ofrecidas por la externalización de protocolos de paso de mensajes como MPI. *Netpipe* de hecho, dispone de un módulo para MPI. Por otro lado, se ha utilizado *Hpcbench* [HUA05] por la con la que se puede modificar el código fuente para modelar distintas cargas de procesamiento y realizar simulaciones de análisis de prestaciones en relación con las predicciones del modelo LAWS [SHI03], como se muestra en la Sección 4.6.

Para realizar medidas con Simics, se han modificado los *benchmarks* añadiéndoles la denominada *magic-instruction* de Simics (ver Apéndice 1) para que el *benchmark* esté totalmente sincronizado con el simulador, y asegurar que todos los experimentos se realicen en las mismas condiciones, ejecutando el mismo número de instrucciones en cada simulación para que los resultados sean comparables. A continuación se describen los *benchmarks* utilizados, cuyo uso se muestra en la Tabla 4.1.

Tabla 4.1 Benchmarks utilizados y su uso

Programa de medida / benchmark	***Uso***
Netperf	Validación inicial de los modelos
Netpipe	Validación inicial de los modelos. Medidas de ancho de banda y latencia
Hpcbench / sysmon	Medidas para análisis modelo LAWS / medidas uso CPU
Oprofile	Medidas uso CPU / interrupciones/s
Inspector de Simics	Estadísticas de memoria / *cache*

4.2.1 Netperf

Netperf [NET07b] es un benchmark que permite evaluar diferentes parámetros relacionados con las prestaciones de la red. Está orientado principalmente a la realización de dos tipos de pruebas:

- Pruebas de secuencias (*streaming*) TCP/UDP (también llamadas pruebas de transferencia masiva de datos o *bulk data transfer*). Dicha transferencia masiva de datos se realiza a través de aplicaciones software que utilizan diferentes técnicas (compresión, buffering, etc.) para optimizar el ancho de banda. Un ejemplo de aplicación para transferencia masiva de datos es FTP que es la más utilizada en Internet para transferir ficheros de gran tamaño. En este caso, las transferencias son unidireccionales. Es útil disponer de un *benchmark* que permita realizar experimentos con *streaming* porque existen multitud de referencias en las que se evalúa el rendimiento de una red en base al ancho de banda máximo para transferencias de *streaming*. Esto es debido a que estas transferencias se utilizan en aplicaciones reales como aplicaciones de distribución de video.
- Pruebas de solicitud/respuesta TCP/UDP. Consiste en enviar de un paquete con un determinado tamaño y esperar a que el sistema remoto lo devuelva. Este *test* es el que habitualmente se utiliza para obtener medidas de latencia de la red y que en algunas ocasiones se denomina *ping-pong.*

Netperf utiliza el modelo de funcionamiento cliente/servidor. Concretamente, el correspondiente esquema de la Figura 4.1, que es el más habitual en los *benchmarks* para TCP o, en general, para cualquier protocolo orientado a conexión.

Esto significa que, para aplicar los *benchmarks*, es necesario ejecutar en una máquina el proceso *netserver*, que abrirá el *socket* correspondiente y permanecerá a la escucha de conexiones por un determinado puerto, y en la otra máquina, ejecutar *Netperf*, que enviará las opciones con las que se haya configurado el *benchmark* a *netserver*. En el caso de transferencias de *streaming*, el sentido del flujo de datos es

desde la máquina servidor (*netserver*) hacia el cliente (*Netperf*). Además, es posible configurar las pruebas para utilizar diferentes tamaños de paquete, diferentes tamaños de *sockets* etc.

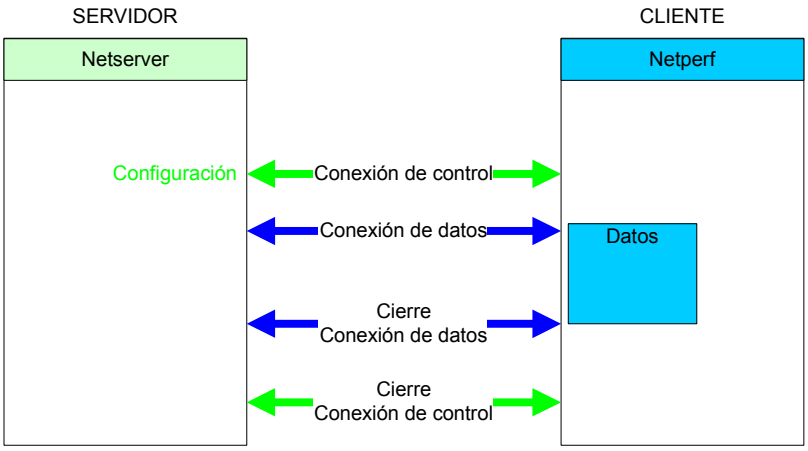


Figura 4.1. Funcionamiento de Netperf

Uno de los problemas de *Netperf* es la carga de trabajo más elevada que suponen los tests especialmente el test de solicitud/respuesta TCP/UDP a diferencia de los programas de *benchmark* como *Netpipe* o *Hpcbench* [HUA05], que están optimizados en ese sentido.

4.2.2 Netpipe

La herramienta *Netpipe* [SNE96] permite para medir las prestaciones del sistema de comunicaciones independientemente del protocolo utilizado. Así, *Netpipe* puede utilizarse tanto para realizar *tests* de *ping-pong* (medida de la latencia de red) como de *streaming* (medida del ancho de banda en transferencia de secuencias). Está diseñado de forma modular de manera que las llamadas a funciones de red (que incluyen los protocolos de comunicación) y las medidas de prestaciones se implementan en módulos diferentes. Gracias a este diseño modular, *Netpipe* es independiente del protocolo utilizado, si se cambia convenientemente el módulo de red. Así, existen versiones de *Netpipe* para TCP y MPI [SNE96, TUR03].

Es posible configurar el tamaño de los mensajes y el algoritmo para incrementar el tamaño de los mismos. También dispone de diferentes opciones para modificar la

configuración de los buffers TCP. En *Netpipe* se incrementa el tamaño de cada bloque transferido de forma exponencial, según el algoritmo descrito en [SNE96]. Para medir el tiempo de transferencia de un bloque de tamaño *c* bytes, se realizan tres medidas. Una primera medida en la que se transfiere un bloque de tamaño *c-p* , otra medida en la que se transfiere un bloque de tamaño *c* y una tercera medida para un bloque de tamaño *c+p*. En estas medidas, *p* es una perturbación de tamaño inicial 3 bytes, que se incrementa exponencialmente en cada experimento [SNE96]. Esta perturbación introducida en el tamaño de cada bloque proporciona una forma de analizar el efecto de tamaños de bloque ligeramente inferiores o superiores a los *buffers* de red [SNE96, TUR03].

Para que las medidas realizadas sean fiables, *Netpipe* repite la medida durante un tiempo configurable que, por defecto, es de 0.5 segundos, asegurándose que el tiempo de medida siempre sea mayor que la resolución de la medida (que dependerá del procesador), siendo la medida final media de todas las medidas realizadas. El número de repeticiones para cada medida puede verse en la ecuación 4.1, donde *tiempo_objetivo* es el tiempo configurable durante el cual se repetirá la medida, *tultimo* es el tiempo empleado en transmitir el último bloque de tamaño *bz1,* y *bz2* es el tamaño del bloque que se va a transferir en el experimento actual. Este método proporciona una medida precisa del tiempo de transferencia de cada bloque [SNE96, TUR03].

$$\text{nrepeticiones} = \frac{tiempo\_objetivo}{(bz1/bz2) * tultimo} \qquad (4.1)$$

En *Netpipe* se utiliza el método de transferencia descrito en [HOC91] para cada bloque, lo que hace posible que en cada experimento se transfiera únicamente un bloque del tamaño indicado, obteniéndose de forma precisa el tiempo de transferencia de dicho bloque. Por tanto, la información derivada de los resultados de *Netpipe* es útil para determinar el efecto producido por un bloque de un tamaño determinado en la red o en el nodo.

En definitiva, *Netpipe* es un *benchmark* de tiempo variable (el tiempo de medida se adapta a la precisión de la CPU para medir tiempo) que puede utilizarse en redes de cualquier velocidad, no quedando obsoleto ni perdiendo precisión con los avances de la tecnología de redes [SNE96, TUR03].

La forma de ejecutar el *benchmark* se muestra en la Tabla 4.2.

Tabla 4.2. Ejecución de Netpipe

	Ejecución	***Uso***
Cliente	NPtcp –r –s	Receptor en el caso de Straming TCP
	NPtcp	Ping-pong TCP
Servidor	NPtcp –h <ip_cliente> -s -o <fichero_salida>	Emisor en el caso de Streaming TCP
	NPtcp –h <ip_cliente> -o <fichero_salida>	Ping-pong TCP

4.2.3 Hpcbench

Hpcbench [HUA05] es un programa de medida de prestaciones del sistema de comunicaciones diseñado para ser utilizado en entornos de altas prestaciones como multicomputadores o sistemas con enlaces de red de gran ancho de banda como es el caso de redes Gigabit Ethernet, Myrinet o QsNET. Este test mide la latencia y el ancho de banda pico entre dos extremos. Al estar pensado para entornos de altas prestaciones, dispone de la utilidad *sysmon* que proporciona información de seguimiento del núcleo, útil para conocer el estado de la máquina durante la realización de los *tests*, tales como tests de uso de la CPU, de uso de memoria, del número de interrupciones por segundo, de cambios de contexto, y diversas estadísticas de red como por ejemplo número de paquetes enviados, recibidos, perdidos, etc. *Hpcbench* puede utilizarse con UDP, TCP o MPI y dispone además de varias opciones para cambiar el tamaño de los buffers, etc.

En la Sección 4.6 donde se trata el modelo LAWS, se aborda con más profundidad el funcionamiento de este *benchmark* ya que se ha utilizado, modificándolo convenientemente, para generar una carga variable en la CPU principal del nodo, y se ha adaptado a nuestro modelo de simulación. De esta forma se puede comprobar si los resultados que se obtienen experimentalmente se ajustan al mencionado modelo LAWS.

Lo mismo que *Netpipe*, la arquitectura de *Hpcbench* es del tipo cliente-servidor. La forma de ejecutar el test se muestra en la Tabla 4.3.

Tabla 4.3. Ejecución de Hpcbench

	Ejecución	***Uso***
Cliente	tcpserver	Receptor en el caso de Straming TCP
Servidor	tcpserver –h <ip_cliente> -m <tamaño_msg>	Emisor en el caso de Streaming TCP
	Tcpserver –h <ip_cliente> -m <tamaño_msg> -a	Ping-pong TCP

4.2.5 Oprofile

La herramienta *Oprofile* [OPR07] permite inspeccionar el núcleo del sistema operativo (para tener información del tiempo de CPU empleado en cada función, llamadas al sistema, cambios de contexto, etc.), o la ejecución de ficheros binarios, para medir el tiempo de CPU empleado en cada tarea. Así, *Oprofile* se utiliza habitualmente en el ámbito de estudio de la arquitectura de computador para identificar cuellos de botella y depurar código optimizado para una determinada arquitectura.

Para funcionar, *Oprofile* necesita el correspondiente soporte del *núcleo* que, por tanto, debe ser compilado incluyendo las librerías correspondientes. La forma de ejecutarlo se muestra en la Tabla 4.3.

Tabla 4.3. Ejecución de Oprofile

Ejecución	***Uso***
opcontrol –vmlinux=<ruta_vmlinux>	Configuración del demonio opreportd
opcontrol –start-daemon	Arranque del demonio opreportd
opremort –l vmlinux=<ruta_vmlinux>	Generación de informe

4.3 Análisis de las prestaciones de la externalización mediante *offloading*

En las secciones siguientes, se muestran los resultados experimentales obtenidos mediante la simulación con Simics de las implementaciones que hemos desarrollado para *offloading*. También se comparan las prestaciones de las dos alternativas de externalización mediante el uso del modelo LAWS, bajo diferentes condiciones de carga del sistema.

Los resultados experimentales, obtenidos utilizando el modelo de simulación descrito en la Sección 3.5 y los *benchmarks* de la Sección 4.2, ponen de manifiesto que se consiguen mejoras en el sistema de comunicaciones al implementar la externalización de protocolos. Con los diferentes experimentos realizados, no sólo se pueden comparar las prestaciones de los distintas alternativas, también se pueden poner de manifiesto qué parámetros influyen en el rendimiento de dichas alternativas. Concretamente, hay cuatro aspectos fundamentales que afectan directamente al rendimiento del sistema de comunicaciones: la latencia en los accesos a memoria, el tiempo de procesamiento del protocolo, la utilización de los buses de E/S del sistema y la carga de la aplicación.

En cuanto a los experimentos a realizar, en primer lugar, se realiza un experimento utilizando *Hpcbench* y *Oprofile* para medir el uso de la CPU del sistema, ejecutando el sistema operativo y las aplicaciones. A continuación, mediante *Netpipe* se llevan a cabo diferentes experimentos utilizando *Netpipe* destinados a realizar medidas de ancho de banda de transferencia (mediante transferencias de streaming) y latencia (mediante *ping-pong*). Además, se comprueba experimentalmente el efecto de la latencia de acceso a memoria descrito en [STE94, CHA01, MAR02, BIN05b, GAD07]. En las medidas anteriores se trata de establecer el máximo ancho de banda que el sistema puede proporcionar. En ellas, la carga de la aplicación se debe únicamente a la ejecución del *benchmark*. Esa carga es prácticamente despreciable debido a que los *benchmarks* utilizados están optimizados en ese sentido.

Una vez realizadas las medidas de prestaciones anteriores, se lleva a cabo un análisis del modelo LAWS mediante la simulación de sistema completo. Concretamente, se realizan experimentos para determinar la variación de la mejora en las prestaciones del sistema de comunicaciones al variar la carga de la aplicación (parámetro γ en el modelo LAWS descrito en la Sección 1.7.1).

En los experimentos anteriores, se modela el efecto de la memoria *cache* a través del tiempo medio de acceso, suponiendo una determinada tasa de aciertos de *cache* es decir, solamente mediante modelos de tiempo. En estos experimentos los modelos de simulación los denominamos *modelos sin cache*.

Posteriormente se analiza el efecto que la inclusión de una jerarquía de memoria *cache* más detallada, de dos niveles, con un primer nivel L1 separado para datos e instrucciones y un nivel L2 unificado. Así, se analiza el efecto de la memoria *cache* en las prestaciones del sistema de comunicaciones, dado que la influencia de la memoria

cache en las prestaciones del sistema de comunicación ha sido cuestionada en artículos como [NAH97] donde se muestran ventajas de la utilización de memoria *cache*, así como en [MUD05] donde se afirma que la inclusión de memoria *cache* no beneficia sustancialmente el sistema de comunicaciones. En estas circunstancias, se realizará el mismo tipo de experimentos que se ha llevado a cabo an los modelos de simulación sin *cache*.

4.3.1 Carga de la CPU del sistema

Puede decirse que una de las principales consecuencias de la externalización del procesamiento de los protocolos de comunicación es que se reduce la sobrecarga de la CPU principal del sistema (CPU_0), ya que, fundamentalmente, dicha CPU solo ejecutará la aplicación que genera los datos. Si la CPU_0 tuviera que generar los datos y ejecutar los protocolos de comunicación, la carga de la misma aumentaría por encima del 90% en redes de anchos de banda altos (del orden de los Gbps), como se muestra en [STE94, FOO04, GAD07] y como hemos comprobado experimentalmente. La disminución de la carga en la CPU_0 debida a la liberación de ciclos para la ejecución de otras tareas se ilustra en la Figura 4.2, en la que se muestra la ocupación de la CPU del sistema para los casos sin externalización y con externalización mediante *offloading*, utilizando TCP como protocolo de transporte. Así, en la Figura 4.2, la CPU tiene que realizar tanto el procesamiento de la aplicación como de las comunicaciones en el caso "sin externalización". Así, CPU_0 estaría ocupada y quedarían muy pocos ciclos disponibles para la ejecución de otras tareas (como la propia aplicación).

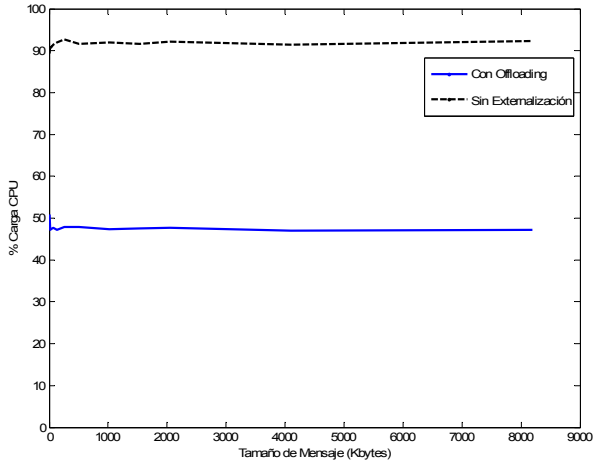


Figura 4.2. Gráfico de comparación de carga de la CPU_0 *(con Hpcbench)*

Las curvas de la Figura 4.3 corresponden al porcentaje de interrupciones a la CPU0 (en el caso no externalizado) que se evitan si se externalizan los protocolos de comunicaciones. En la gráfica de la Figura 4.3, un 50% significa que, cuando el protocolo se externaliza, la CPU0 recibe la mitad de interrupciones que sin externalización, y un 0% significa que con externalización, la CPU_0 recibiría el mismo número de interrupciones que sin externalización.

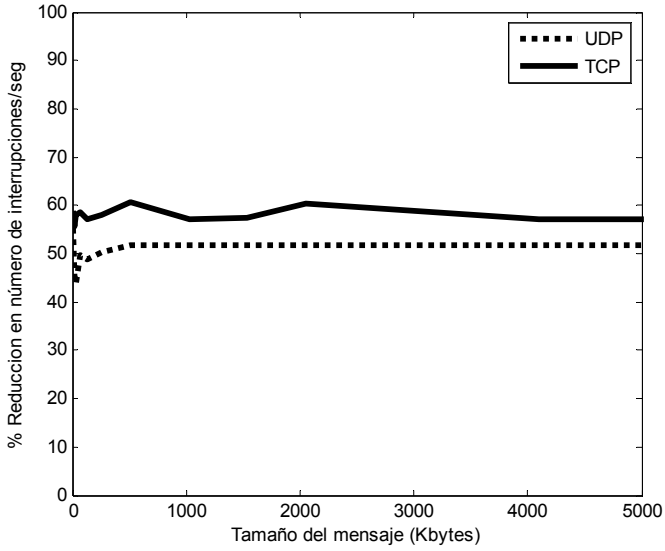


Figura 4.3. Descenso en el número de interrupciones por segundo (con Oprofile)

El descenso en el número de interrupciones por segundo obtenido con la externalización mediante *offloading* es de alrededor del 60% cuando se utiliza TCP como protocolo de transporte y de alrededor del 50% cuando se utiliza UDP. Además, estos porcentajes no varían sustancialmente al variar el tamaño del mensaje. Así pues, se corrobora lo que se indica en [PRA04], donde se muestra que al externalizar la reducción del número de interrupciones por segundo respecto al sistema sin externalización es del 60% aproximadamente para TCP, y del 50% para UDP.

En cuanto al efecto de la externalización en las prestaciones globales del sistema, cuantos más ciclos de CPU se necesiten para procesar el protocolo, mayor es la mejora en el tiempo dedicado a las rutinas de servicio de las interrupciones (menos interrupciones generadas y en consecuencia menos tiempo de CPU dedicado a procesarlas). Como el protocolo TCP requiere un procesamiento más costoso que UDP

en número de ciclos, los beneficios que se podrían obtener en el caso de TCP son mayores.

En la gráfica de la Figura 4.3 también se pone de manifiesto que para tamaños de mensaje pequeños, donde el número de interrupciones generadas es más alto, la variación en la reducción del número de interrupciones con el tamaño del mensaje es mayor en el caso de TCP.

4.3.2 Latencia de acceso a memoria

Trabajos previos como los descritos en [MIN95, NAH97, BIN05, IOA05, BIN06] ponen de manifiesto la importancia de la latencia de acceso a memoria en las prestaciones del sistema de comunicaciones. Ya se ha comentado en la Sección 3.6.1 que en este aspecto influye de forma decisiva la localización de la interfaz de red dentro del sistema [BIN05]. La importancia que tiene el tiempo de acceso a memoria se debe al gran número de transacciones de memoria que son necesarias durante el procesamiento de un paquete TCP/IP. Este número será mayor cuanto mayor sea el tamaño del paquete TCP. A la vez, un elevado número de transacciones de memoria ocasiona *overheads* elevados para el sistema debido a la contención en el bus de memoria y a los ciclos que el procesador debe esperar hasta que se completa la transacción (*processor stalls*).

En la Tabla 4.4 puede verse el porcentaje de tiempo de CPU empleado en cada una de las tareas involucradas en el procesamiento del protocolo TCP. Estos porcentajes han sido medidos en un sistema real, para un procesador de 1,73 GHz que transfiere 2 Gbytes en paquetes de 1500 bytes, utilizando para ello la herramienta *Oprofile* [OPR07] y, como ya se ha comentado en la sección 3.5, un núcleo 2.6 de Linux.

Tabla 4.4. Tiempo de proceso para la recepción de 2GB de datos

Tarea	***Tiempo***
Copias de datos	20.3%
Procesamiento TCP/IP	7.2%
Driver	2.82%
Sistema Operativo	5.89%

Como puede verse en la Tabla 4.4 los accesos a memoria debidos a las copias de datos suponen un porcentaje mayor de tiempo que el resto de las tareas.

La latencia en los accesos a memoria hace que la CPU del sistema se detenga hasta que se completa la transacción, lo que hace que usualmente el tiempo empleado en las copias de datos sea considerable. Los procesadores comúnmente utilizados en los servidores actuales funcionan una frecuencia de reloj entorno a los 4 GHz y la tecnología actual de memoria RAM permite buses de memoria de unos 400 MHz [REG04b, IOA05]. Es decir, la frecuencia de reloj de la memoria RAM más rápida actualmente (2007) es de una décima parte del reloj del procesador. Esto hace que cuando el procesador necesita acceder a memoria RAM para cargar un dato que no está en la memoria *cache* [NAH97], para que la transferencia de memoria se complete deberá esperar el tiempo de latencia para acceder al bus y posterioremente al menos 10 ciclos de reloj para que el dato llegue hasta el procesador. Durante estos ciclos, que incluso pueden llegar a ser cientos, no se está ejecutando la aplicación. Los procesadores multihebra simultánea (*Simultaeous Multithreading* [DEA95]) pueden ser insuficientes para solucionar este problema sobre todo en situaciones en las que la aplicación ocasione una sobrecarga alta en la CPU, ya que las hebras de la aplicación deberán competir con las hebras de los procesos de comunicaciones tanto por los recursos compartidos dentro del microprocesador como por el bus de acceso a memoria. Este problema está también presente en los microprocesadores actuales de propósito general y doble núcleo que no utilizan técnicas o buses de acceso a memoria que mejoren adecuadamente las prestaciones (como por ejemplo, *Hypertransport* [HYP06]). Por tanto, la posible mejora del sistema de comunicaciones utilizando procesadores multihebra o doble núcleo dependerá del perfil de comunicaciones de la aplicación. Este problema es cada vez más grave, ya que la velocidad de reloj de los procesadores crece mucho más deprisa que la de las memorias RAM [NAH97, HEM95].

En los siguientes apartados, se analiza experimentalmente mediante los modelos de simulación desarrollados en el Capítulo 3 y posteriores, la forma en que la latencia de acceso a memoria afecta al sistema de comunicaciones y qué cambios se producen con la externalización.

4.3.3 Efecto de la latencia de memoria en el ancho de banda efectivo de la red.

En las Figuras 4.4 y 4.5 se muestran los resultados obtenidos con *Netpipe*. Las gráficas proporcionan el ancho de banda para diferentes tamaños de mensaje así como la máxima tasa de transferencia que se podría alcanzar mediante externalización.

La Figura 4.4 ilustra la mejora que podría alcanzarse en el caso ideal si la NIC se pudiera comunicar a través del bus sin ninguna latencia. Es decir, la NIC que procesa los protocolos está conectada a un bus del sistema sin latencia alguna, sólo de una forma funcional y, por tanto, podría acceder a la memoria sin ningún retardo. Como se muestra en la Figura 4.4, el ancho de banda obtenida en este caso es prácticamente igual al ancho de banda del enlace cuando se utiliza externaliza en la NIC (*offloading*). En caso contrario el ancho de banda máximo es bastante menor.

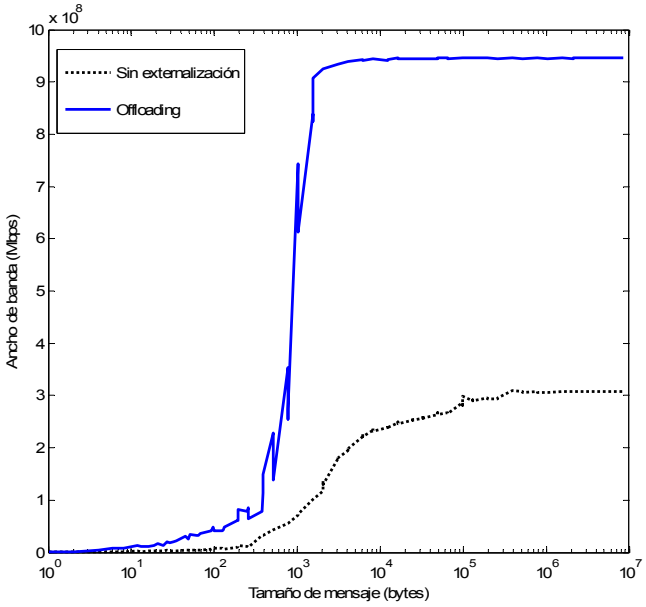


Figura 4.4. Comparación de tasas de transferencia (con Netpipe)

Para obtener los resultados de la Figura 4.5, se ha modelado el comportamiento real de la conexión entre la CPU_0 y la memoria a través del bus del sistema y de la NIC y la memoria a través del bus de E/S y el chipset. Así, se han introducido los correspondientes modelos de temporización en el bus de E/S que está conectado la NIC y en los accesos a memoria desde el procesador de la NIC. En la Figura 4.5, se muestran

los anchos de banda obtenidos para diferentes valores de latencia en los accesos desde la NIC. En la notación de la figura, *Offload x*, x es el número de retardos en los accesos al bus de E/S desde la CPU_1 con respecto a un valor de referencia que puede ser configurado por el usuario en el modelo de simulación. De esta forma, *Offload 2* significa un retardo doble en el acceso a memoria desde la CPU_1 que *Offload 1*. *Offload 1* se corresponde a un retardo de 1 ciclo de reloj.

Las condiciones en las que se han realizado las simulaciones para obtener las Figuras 4.5 y 4.6 se resumen en la Tabla 4.5.

Tabla 4.5. Condiciones de simulación. Hardware

Procesador principal	Intel Pentium 4
Frecuencia reloj	400 Mhz
Memoria principal	128 Mbytes
Tiempo medio de acceso a memoria principal	10 ciclos
Tamaño sockets tx /rx	64 Kbytes
Tarjeta de red	bcm5703c [BRO07]
Ancho de banda del enlace de red	1000 Mbps
Memoria NIC	128 Kbytes
Tamaño MTU	9000 bytes

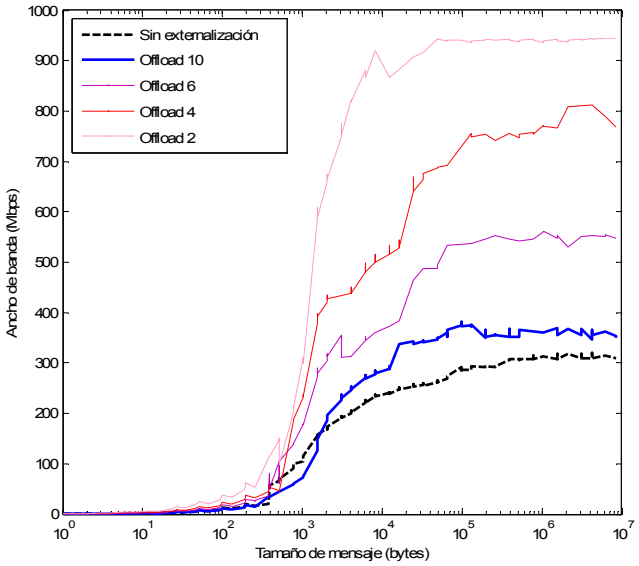


Figura 4.5. Ancho de banda pico para diferentes latencias de memoria (con Netpipe)

Aunque en los experimentos realizados hasta este momento, no se ha incluido un modelo detallado de memoria *cache* en la jerarquía de memoria, la utilización de 10

ciclos de retardo en la latencia de acceso a memoria considera el tiempo medio de acceso a memoria teniendo en cuenta una tasa de aciertos de *cache* del 90%, un tiempo de acceso a *cache* de 1 ciclo y un tiempo medio de acceso a memoria de 91 ciclos aproximadamente. Por tanto, podemos calcular la latencia media de acceso teniendo en cuenta el efecto de la memoria *cache* con la expresión 4.1.

latencia de acceso media = ta + (1-ta)*latencia de acceso (4.1)

Donde, *ta* es la tasa de aciertos de *cache*. Por tanto, el valor seleccionado de 10 ciclos de latencia media implicaría:

10 ciclos lat. media = 0.9 * 1 + 0.1 * lat. de acceso ➔ lat. de acceso = 91 ciclos.

En la Sección 4.7 se incluye en el modelo de simulación de Simics un modelo detallado de la jerarquía de memoria que incluye la *cache* y se estudia su efecto en las prestaciones del sistema de comunicaciones.

En la Tabla 4.5 se pone de manifiesto que se han utilizado *sockets* de 64 KB. Esto se debe a que este tamaño de *sockets* permite evitar el problema de la caida de el ancho de banda de alrededor del 80% en mensajes con un tamaño comprendido entre 32 KB y 64 KB, observado en los diferentes experimentos realizados debido al uso de una *ventana de congestión TCP* [STE94b] demasiado pequeña. Este efecto de caida de el ancho de banda, así como su solución basada en el incremento del tamaño de los *sockets* se muestra en [FAR00].

Por otro lado, en todos los experimentos realizados con *Netpipe* se han utilizado los parámetros indicados en la Tabla 4.6, de acuerdo a los algoritmos descritos en la Sección 4.2.2 para el cálculo del número de repeticiones y el incremento del tamaño del mensaje.

Tabla 4.6. Condiciones de simulación. Estadísticas

Benchmark	Netpipe 3.7.1
Tamaño mensaje en cada simulación	$c,\ c \pm p$ (bytes)
Algoritmo incremento tamaño mensaje	Exponencial [SNE96]
Tiempo de simulación	2 segundos
Número de repeticiones	$\text{nº repet} = \frac{2}{(\text{bsz1}/\text{bsz2}) * \text{tultimo}}$
Tamaño máximo de mensaje	2 Mbytes
Tipo transferencia	*block ping-pong* [HOC91]

En las simulaciones realizadas para la obtención de la Figura 4.5, la única sobrecarga en la CPU debida a la aplicación es la del propio *benchmark Netpipe*. Es decir, la Figura 4.5 muestra el ancho de banda pico alcanzable sin tener en cuenta la seguramente mayor sobrecarga que produciría una aplicación real que generase los datos.

En la Figura 4.5 también puede observarse que el ancho de banda pico crece conforme la latencia de acceso a memoria decrece. Sin externalización, solo se alcanza un ancho de banda de alrededor de 300 Mbps, mientras que al externalizar mediante *offloading* las comunicaciones en un interfaz de red que pueda acceder a la memoria deteniendo el procesador menos ciclos que los que necesita el procesador nodo, se obtienen mejoras en el ancho de banda. Sin externalización, vemos que aproximadamente se cumple la conocida regla de 1 MHz de velocidad del procesador por cada 1 Mbps de ancho de banda de la red. En la simulación realizada para generar el gráfico de la Figura 4.5, se ha considerado que la memoria requiere un tiempo medio de acceso de 10 ciclos de reloj (lo que implicaría un tiempo de acceso a memoria de unos 90 ciclos como se ha indicado anteriormente en esta sección), para completar una transacción hacia la CPU de un interfaz de red [IOA05].

Estos resultados corroboran lo expuesto en [NAH97, MIN95, BIN05, IOA05] en referencia a la importancia del tiempo de acceso a memoria y el cuello de botella que éste supone. Por tanto, la latencia de los accesos a memoria juega un papel decisivo en las prestaciones obtenidas y en los efectos de la externalización mediante *offloading*.

Así, sin sobrecarga de aplicación en la CPU principal, la externalización proporciona una mejora en torno al 35 %, en el caso de un nodo saturado (sin ciclos de CPU libres), que no pueda aprovechar todo el ancho de banda del enlace de red.

Una vez visto el efecto de la latencia de acceso a memoria en las prestaciones del sistema de comunicaciones, se supondrá una jerarquía de memoria que requiera de una media de 10 ciclos de CPU para completar un acceso, de forma similar a como se muestra en [IOA05]. Los valores concretos de configuración del modelo de temporización utilizados en las simulaciones que determinan las latencias de acceso a memoria, buses de E/S y la penalización por colisión se muestran en la Tabla 4.7.

Tabla 4.7. Configuración del modelo de temporización

Tiempo de acceso a memoria principal (incluyendo el efecto de la *cache*)	10 ciclos
PCI-X (64 bits, 33 MHz)	2.12 Gbps
Penalización por colisión	20 ciclos

4.3.4 Efecto de la tecnología de la NIC

En la Figura 4.6 se muestra el efecto de la tecnología del procesador de la NIC en las prestaciones obtenidas por la externalización de protocolos con *offloading*. De hecho, este es uno de los principales argumentos que se han utilizado para cuestionar los beneficios que puede proporcionar la externalización de protocolos [MOG97]. Esto queda corroborado mediante nuestro análisis con los modelos de simulación de sistema completo.

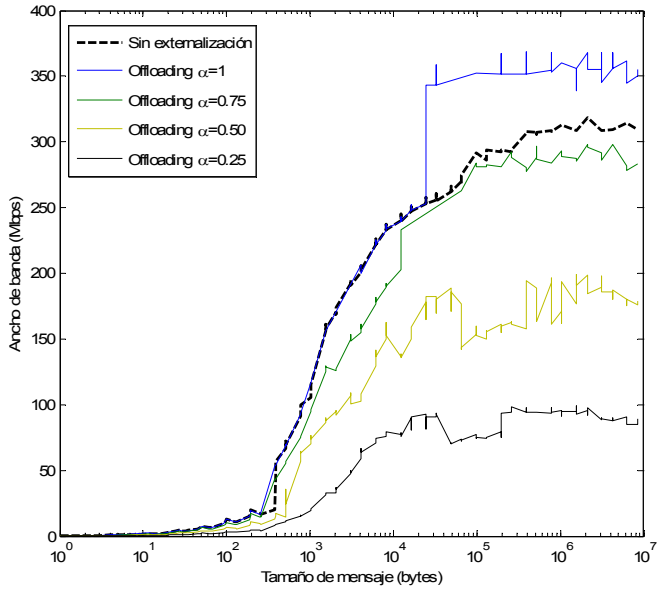


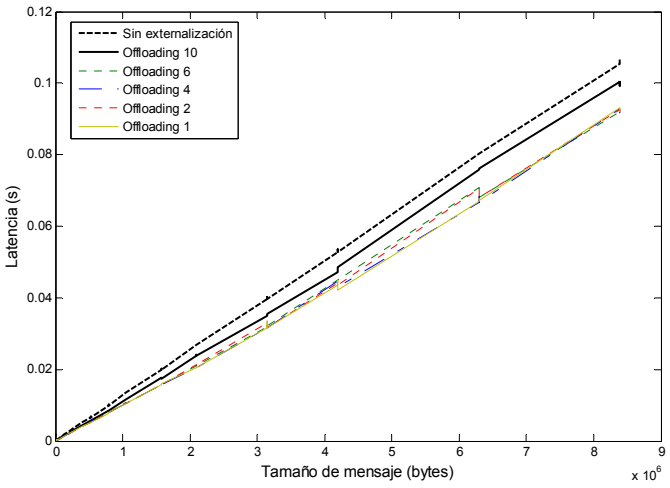
Figura 4.6. Ancho de banda (streaming) para diferentes velocidades de procesador de la NIC (con Netpipe)

Las curvas de la Figura 4.6 muestran los anchos de banda obtenidos utilizando procesadores en la NIC cuya velocidad es respectivamente del 75%, 50% y 25% de la velocidad del la CPU principal del nodo (CPU_0), de acuerdo al parámetro α del modelo LAWS que se presentó en la Sección 1.3. Como puede observarse, la velocidad del procesador de la NIC (CPU_1) afecta de forma decisiva al ancho de banda obtenido, que Este disminuye a medida que la velocidad del procesador de la NIC se reduce. Además, en el caso de tener un NIC con un procesador muy lento, la externalización de protocolos puede incluso empeorar las prestaciones proporcionadas por el sistema no externalizado [SHI03, MOG03]. Por tanto, es evidente que la externalización de protocolos mejoraría las prestaciones del sistema de comunicaciones sólo si la CPU incluida en la NIC (CPU_1) es lo suficientemente rápida comparada con la CPU principal del nodo (CPU_0).

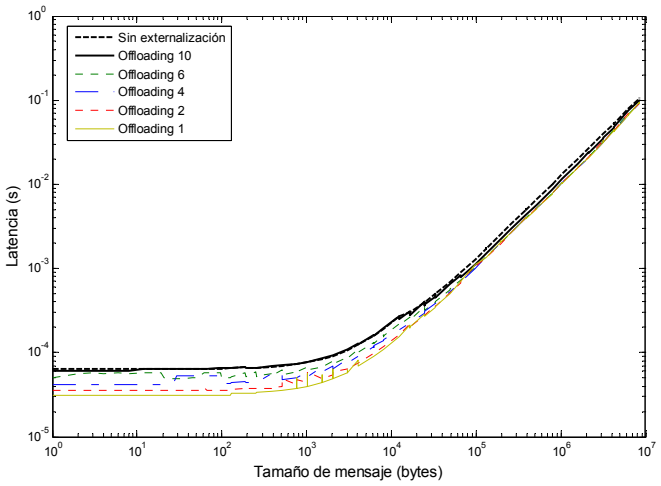
4.3.5 Efecto de la latencia de memoria en la latencia de la red.

Las Figura 4.7a y 4.7b muestran la latencia media a partir del denominado RTT (*Round Trip Time*), que es el tiempo de ida y vuelta de un mensaje para diferentes tamaños de mensaje, con escalas lineal y logarítmica, respectivamente para el ancho de banda y el tamaño del mensaje. Al mismo tiempo, la Tabla 4.8 proporciona la latencia

de red en función de la latencia media de acceso a memoria que se han medido utilizando *Netpipe*. Como se ha comentado en la Sección 4.2.2, *Netpipe* puede utilizarse para medir tanto el ancho de banda como la latencia de la red. De hecho, los diferentes experimentos realizados con *Netpipe* proporcionan la información necesaria para obtener dichas medidas de prestaciones [SNE96, TUR03]. En los experimentos realizados se han utilizado los parámetros de simulación de la Tabla 4.2a y 4.2b, salvo para el tipo de experimento que, en el caso de la medida de latencia de red, usa la técnica de petición/respuesta. Así, esta medida es más precisa al hacer que el mismo mensaje viaje en ambas direcciones ya que, en las medidas de envíos de secuencia, los mensajes pequeños (que son los que se utlizan para determinar la latencia de la red) podrían enviarse en una única transferencia.



(a)



(b)

Figura 4.7. Latencia en Gigabit Ethernet con y sin externalización (a) escala lineal y (b) escala logaritmica (con Netpipe)

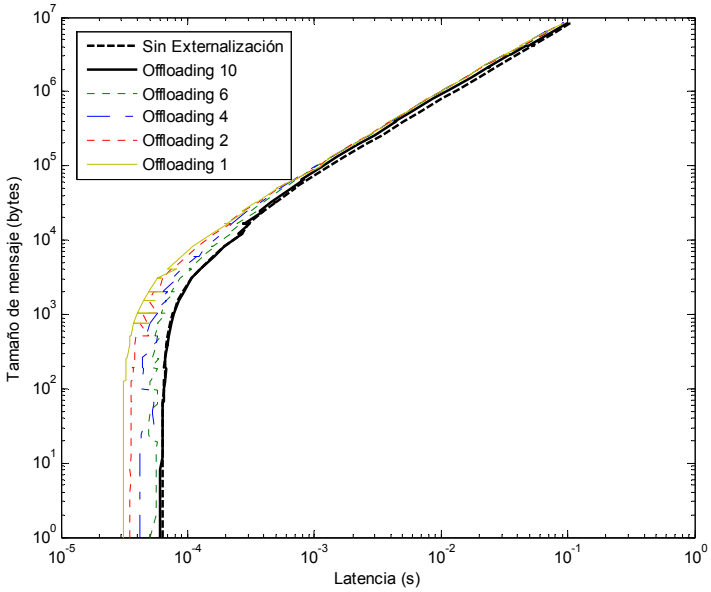
En la Figura 4.7 se ha utilizado la misma notación que en las gráficas relativas al ancho de banda (Figura 4.5). Así, en *offload x*, el valor *x* es el número de ciclos de retardo en los accesos a memoria desde la CPU_1 con respecto a un valor de referencia dado que puede fijarse en el modelo de simulación. De esta forma, *Offload 2* significa un retardo doble en el acceso a memoria desde la CPU_1 que *Offload 1*. Para offload 1 se ha tomado un retardo de 1 ciclo de reloj.

Los experimentos para medir la latencia se han realizado con una red Gigabit Ethernet utilizando TCP como protocolo de transporte. Evidentemente, tener un ancho de banda elevado no implica tener una latencia baja, dado que el sistema puede comportarse mejor con paquetes de gran tamaño que con paquetes pequeños [CAR00]. De hecho, en TCP la latencia se ve afectada por varios factores dependientes de la configuración de los parámetros del protocolo, como son el uso de algoritmos que retrasan el envío del ACK o que evitan enviar paquetes pequeños por separado. Esto puede verse en la Figura 4.7. Además, en la Tabla 4.8 se pone de manifiesto la mejora conseguida en la latencia al aplicar la técnica de *offloading* para distintos valores del retardo de acceso a memoria. La Figura 4.8a muestra el punto de saturación (punto a partir del cual, un incremento en el tamaño del bloque supone un incremento lineal en el tiempo de transferencia) para TCP, y considerando tanto la externalización mediante *offloading* como la no externalización. Se puede apreciar como el punto de saturación depende de las características de la externalización. En el gráfico de firma de la red que representa la evolución de el ancho de banda frente al tiempo (Figura 4.8b) puede observarse la mejora en la latencia. La latencia de la red corresponde al primer punto de dicho gráfico.

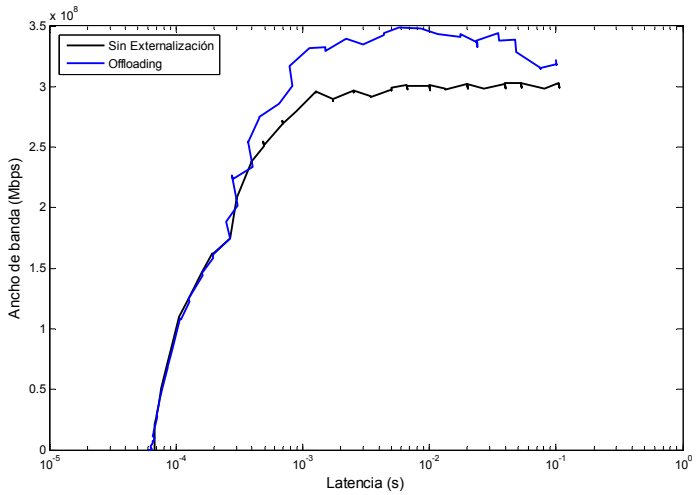
La latencia y el punto de saturación tienen una fuerte dependencia de la configuración de los parámetros TCP, tales como el tamaño de los *buffers*, el uso del algoritmo de *Nagle* [STE94b, FAR00] para mejorar las prestaciones en una red TCP en cuando hay retardos considerables al enviar paquetes pequeños, o el uso de procedimientos que permiten retrasar el envío de ACK un número de paquetes dado [FAR00, MOG01]. Así, mediante el algoritmo SACK (*Selective ACK*) [STE94b] se evita el reconocimiento de paquetes duplicados, lo que supone una mejora de la eficiencia de las redes TCP al minimizar el número de reconocimientos enviados por el receptor.

Tabla 4.8. Latencia de memoria / Latencia de red (con Netpipe)

Tasa de aciertos de *cache*	**Latencia de memoria (ciclos)**	**Latencia de red (μs)**
1	1	31
0.98	2	35
0.96	4	42
0.94	6	50
0.91	10	60
	Sin externalización	68



(a)



(b)

Figura 4.8. (a) Grafico de saturación para diferentes alternativas de externalización (con Netpipe) (b) Firma Ethernet para offloading (con Netpipe)

En cualquier caso, e independientemente de los parámetros TCP, mediante la simulación de sistema completo de nuestro modelo de máquina se pueden observar los siguientes efectos del *offloading*:

1. La latencia disminuye con la externalización mediante *offloading*. Este efecto se debe a que el procesamiento de los protocolos se realiza en una CPU, diferente a la CPU del sistema, que se dedica en exclusiva a la interfaz de red, y a la ubicación de esa CPU, en una posición más cercana a la red.
2. Al externalizar, el punto de saturación se alcanza con paquetes de mayor tamaño. Esto se traduce en que el tamaño del paquete que maximiza el *throughput* es mayor con *offloading* y por tanto, la externalización mediante *offloading* resulta adecuada para la transferencia de paquetes de gran tamaño.

Estos efectos observados en la externalización mediante *offloading* corroboran los resultados obtenidos en otros trabajos, como por ejemplo en [MOG03].

Tabla 4.9. Latencia de red y punto de saturación para offloading

Externalización	**Latencia de red (μs)**	**Punto de Saturación (KB)**	**Ancho de banda (Mbps)**
Sin Externalización	68	2	230
Offloading	60	4	340

En la Tabla 4.9 se muestra la latencia de red para el caso con externalización mediante *offloading* y sin externalización, así como el punto de saturación de la red en cada situación. En dicha tabla se puede ver como la latencia de red al externalizar disminuye en torno al 18%. Las simulaciones con las que se han obtenido los resultados de las Tablas 4.5 y 4.6 se han realizado utilizando los modelos de simulación para *offloading* descritos en la Sección 3.5 con los parámetros de las tablas 4.5 y 4.6. Esta disminución en la latencia se debe al funcionamiento del modelo de externalización mediante *offloading* descrito en la Sección 3.5.1, ya que, en dicho modelo, la tarjeta de red incluye los elementos necesarios para interaccionar con la red directamente.

4.4 Análisis de las prestaciones de la externalización mediante *Onloading*

En esta sección se presentan los resultados de las simulaciones realizadas utilizando nuestro modelo de simulación para la externalización mediante *onloading*, presentado en la Sección 3.6. A partir de dicho modelo se analizan las prestaciones que alcanza nuestra implementación de interfaces de red con *onloading*, igual que se ha hecho para el caso de usar *offloading*.

4.4.1 Carga de la CPU del sistema

En esta sección se analiza la carga de la CPU que procesa la aplicación al implementar los protocolos de comunicación, externalizados en una de las CPU del sistema. Concretamente, en la Figura 4.9 se observa la carga de la CPU del sistema que procesa la aplicación cuando recibe datos con el máximo de ancho de banda del enlace Gigabit Ethernet, sin externalización y con externalización mediante *onloading*. Las medidas se han realizado utilizando la utilidad *sysmon* de *Hpcbench*, igual que se realizó la medida análoga para la interfaz de red con *offloading* en la Sección 4.3.1 (Figura 4.2).

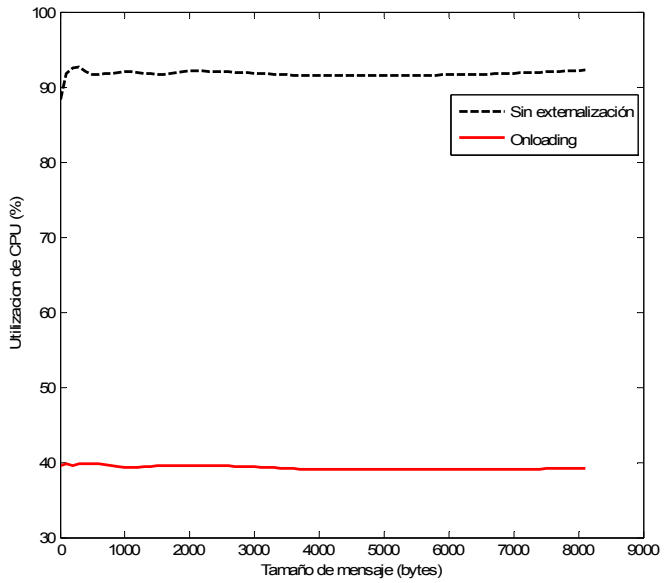


Figura 4.9. Carga de CPU con onloading (con Hpcbench)

Para aprovechar todo el ancho de banda del enlace si no hay externalizacion, la CPU del sistema dedicaría más del 90% de los ciclos de reloj a los procesos de comunicación, quedando en la práctica, colapsada. Al externalizar los protocolos de comunicación mediante *onloading*, la carga de la CPU que ejecuta la aplicación desciende hasta el 40% de los ciclos de reloj. Esta disminución de la carga también está relacionada con el número de interrupciones que recibe la CPU que ejecuta la aplicación por unidad de tiempo debidas a la recepción de paquetes. Como vimos, para el caso de la interfaz de red con *offloading* se tenía una carga algo superior (alrededor del 50%) debido a que la CPU_0 tenía que ejecutar además de la aplicación y el sistema operativo, el *driver* de la tarjeta de red.

4.4.2 Ancho de banda y latencia

A continuación, se presentan las prestaciones que ofrece la técnica de *onloading* en cuanto a ancho de banda pico y a latencia. Para ello se ha utilizado el *benchmark Netpipe* igual que se ha hecho anteriormente para la externalización mediante *offloading*.

En las Figuras 4.10 y 4.11 se puede ver el efecto de la latencia de acceso a memoria desde la CPU que procesa los protocolos de comunicación, en las mismas condiciones descritas en la Sección 4.3 para la externalización con *offloading.* Aquí, también se pone de manifiesto la importancia del acceso a memoria en las prestaciones de la externalización mediante *onloading.*

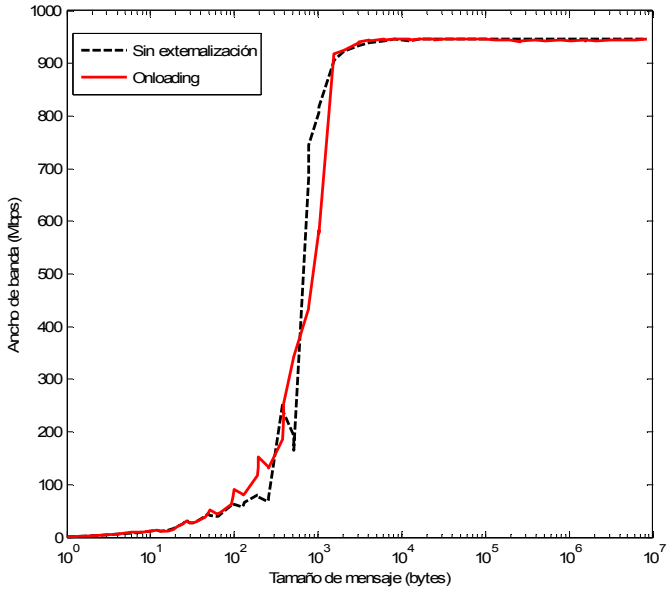


Figura 4.10. Ancho de banda onloading (streaming). Caso ideal (con Netpipe)

La Figura 4.11 proporciona el ancho de banda pico que se puede alcanzar mediante la externalización con *onloading*, en el caso de que la latencia de acceso a memoria, para la CPU que procesa los protocolos sea la misma que en el caso no externalizado. Se puede observar que la técnica de *onloading* proporciona beneficios evidentes en este caso.

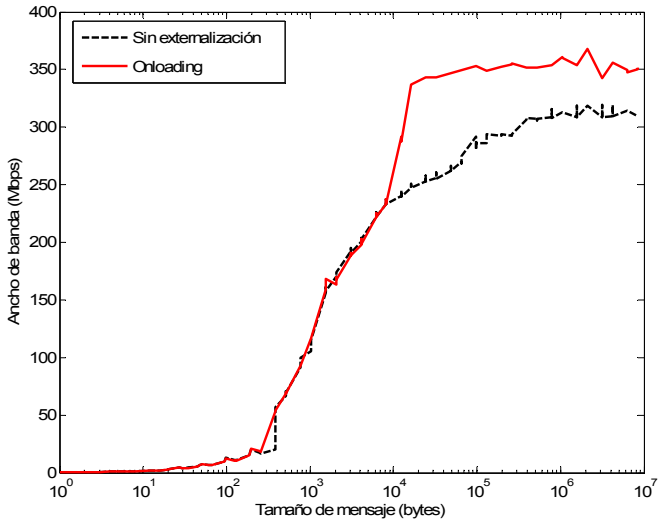
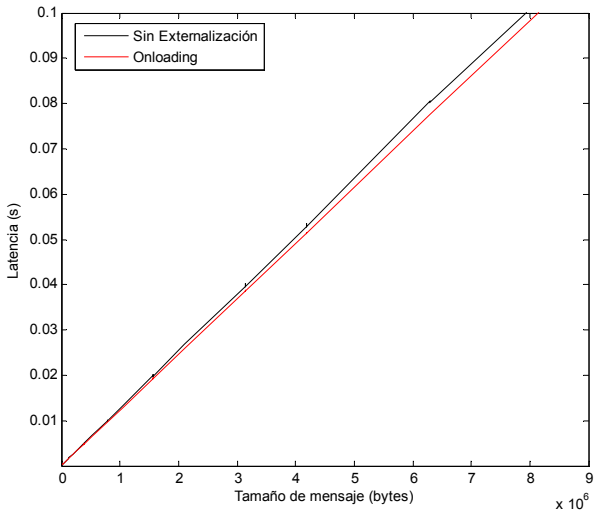


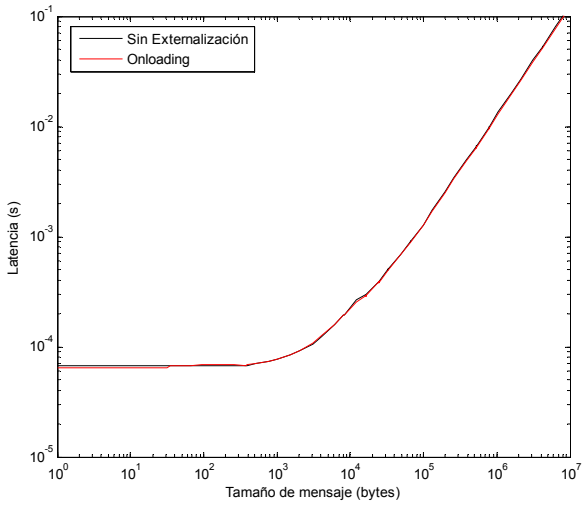
Figura 4.11. Ancho de banda pico con el modelo onloading (streaming)

En la Figuras 4.12a y 4.12b se puede ver como la latencia de red es menor al externalizar mediante *onloading* que al no externalizar. Con la implementación de *onloading* que se ha hecho, la latencia se debe a que las interrupciones generadas por la NIC a la llegada de un paquete son atendidas directamente por la CPU_1, que está dedicada en exclusiva al procesamiento de protocolos. Sin embargo, dicha disminución en la latencia no es muy notable sobre todo para paquetes pequeños ya que, en ese caso, el overhead de comunicación es mayor. Hay que tener en cuenta que, en el caso de la externalización mediante *onloading*, se están compartiendo más recursos del nodo entre la CPU que ejecuta la aplicación y la CPU que realiza el procesamiento de las comunicaciones (como el bus de acceso a memoria, por ejemplo), que en el caso de la externalización mediante *offloading*.

En el caso de no tener externalización alguna, las interrupciones generadas por la interfaz de red son atendidas por la misma CPU que ejecuta la aplicación. Esta, además, puede recibir interrupciones de otros dispositivos.



(a)



(b)

Figura 4.12 Latencia para diferentes tamaños de mensaje con onloading. (a) escala lineal y (b) logarítmica (con Netpipe)

En la Figura 4.13 podemos ver el grafico de saturación de la red, cuando no se implementa externalización en el sistema comparado con el de externalización con *onloading*.

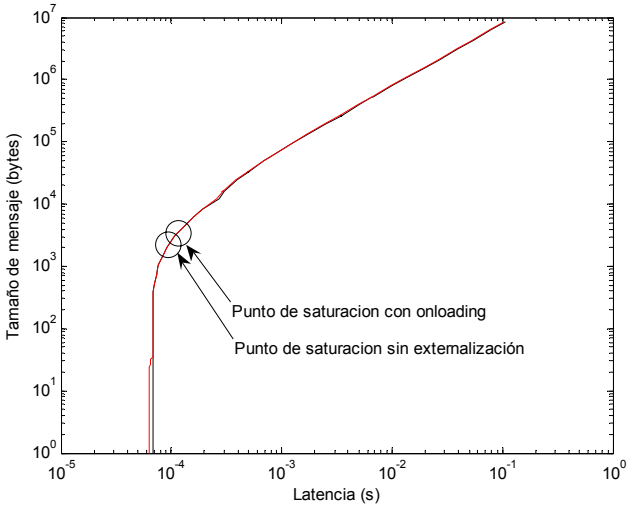


Figura 4.13. Punto de Saturación con onloading

En la Tabla 4.3 se pueden ver los valores de latencia y punto de saturación para ambos casos. Los puntos de saturación, que se encuentran en el codo de la curva, se han calculado mediante *matlab*, teniendo en cuenta que a partir de dicho punto, el ancho de banda crece linealmente con el tamaño del mensaje.

En el caso de la externalización mediante *onloading*, el punto de saturación se alcanza ligeramente antes que en el caso de no tener externalización. Esto significa que el tamaño de bloque a partir del cual, un incremento en el mismo supone una mejora significativa en la latencia es menor (mayor tiempo de saturación). Aunque los puntos de saturación para el caso de externalización mediante *offloading* y mediante *onloading* se alcanzan para tamaños de bloque similares (4 Kbytes), la latencia proporcionada por la externalización mediante *offloading* es aproximadamente un 20% menor que el caso de onloading. En la Sección 4.5 se realiza un estudio comparativo de las prestaciones alcanzadas por las dos técnicas de externalización *onloading* y *offloading*.

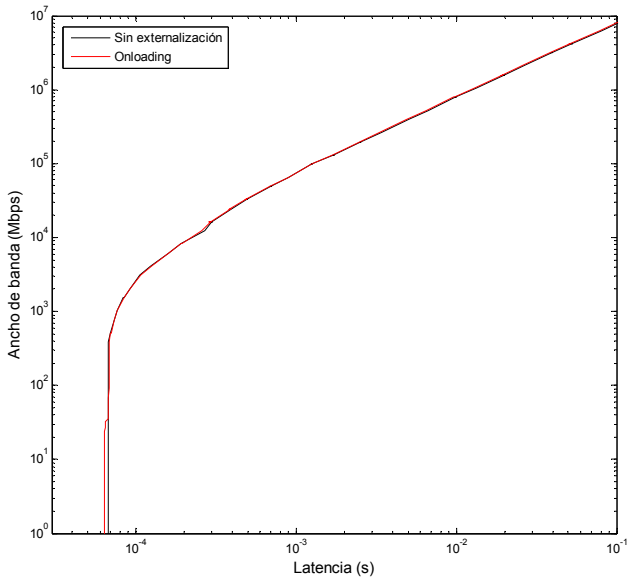


Figura 4.14. Firma Ethernet para Onloading (con Netpipe)

En la Figura 4.14 se muestra el gráfico de firma Ethernet. A partir de esa figura se puede extraer la latencia de la red, que se corresponde con el primer punto en el eje de tiempo. De esta forma se han obtenido los resultados de la Tabla 4.10, en la que se puede ver que la latencia de red es 4 μs menor en el caso de la externalización mediante *onloading* que sin externalización. Este dato corresponde a tamaños de mensaje muy pequeños, que generan una gran sobrecarga en el sistema al tener que atravesar sus buses en numerosas ocasiones. Además, la recepción de cada uno de estos mensajes genera una interrupción. La sobrecarga asociada al sistema operativo y a la gestión de las interrupciones mediante el APIC, hacen que la ganancia en latencia para mensajes muy pequeños no sea considerable.

Tabla 4.10. Latencia de red y punto de saturación para onloading

Externalización	Punto de Saturación (Kb)	Latencia (μs)	Ancho de banda (Mbps)
Sin externalización	2	68	230
Onloading	4	64	345

4.5 Comparación de prestaciones *offloading/onloading*

A continuación, se proporciona un análisis comparativo de las técnicas de externalización mediante *offloading* y de *onloading*, a partir de los resultados obtenidos en las simulaciones realizadas con ambas.

4.5.1 Utilización de CPU

La Figura 4.15 proporciona la utilización de la CPU del nodo para los casos de externalización mediante *offloading*, *onloading,* y sin externalización, para distintos tamaños de mensaje con la utilidad *sysmon* del benchmark *Hpcbench*. Como puede verse, el porcentaje de uso de la CPU del nodo dedicada a la aplicación (CPU_0) es menor en el caso de la externalización mediante *onloading*, ya que es posible ejecutar el driver de la interfaz de red en la CPU_1, como se mostró en la Sección 3.6 y por tanto, reducir la sobrecarga de la CPU_0.

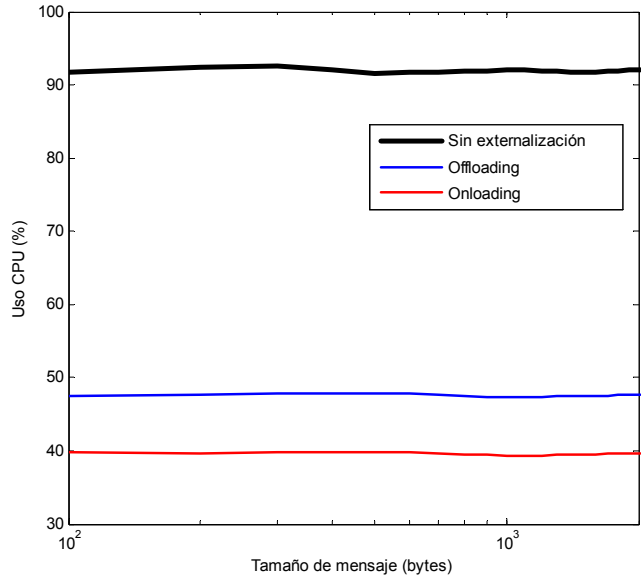


Figura 4.15. Comparación del uso de CPU (con Hpcbench)

Estos resultados se han obtenido mediante la utilidad *sysmon* [HUA05] del *benchmark Hpcbench.* En el caso de la externalización con *onloading,* al ser posible la ejecución del driver de la NIC en la CPU_1, la carga en CPU_0 es incluso menor que en el caso del *offloading*, como puede verse en la figura 4.15. Si se utiliza *offloading*, el driver tiene que ejecutarse en CPU_0, causando una sobrecarga mayor en la misma.

En la Figura 4.16, se muestran las interrupciones por unidad de tiempo atendidas por la CPU_0 considerando diferentes tamaños de mensaje. Como se observa en la Figura 4.16 en el caso de la externalización mediante *offloading*, y en el sistema de partida (sistema base) sin externalización, las interrupciones por segundo recibidas por CPU_0 decrecen conforme el tamaño de mensaje crece dado que el número de mensajes que llegan en un tiempo determinado disminuye.

Respecto a las interrupciones recibidas por la CPU_1 en el caso de utilizar *onloading*, hay que tener en cuenta que, aunque según lo dicho en la Sección 3.6 esta CPU se ha dedicado al procesamiento de las comunicaciones, es necesario que los dos procesadores del sistema estén en un estado consistente. Para ello, la CPU_1 debe atender las interrupciones inter-procesador o *cross-calls.* Dicho efecto se pone de manifiesto en la Figura 4.16 donde puede apreciarse un pequeño número de interrupciones por segundo en el caso de *onloading*.

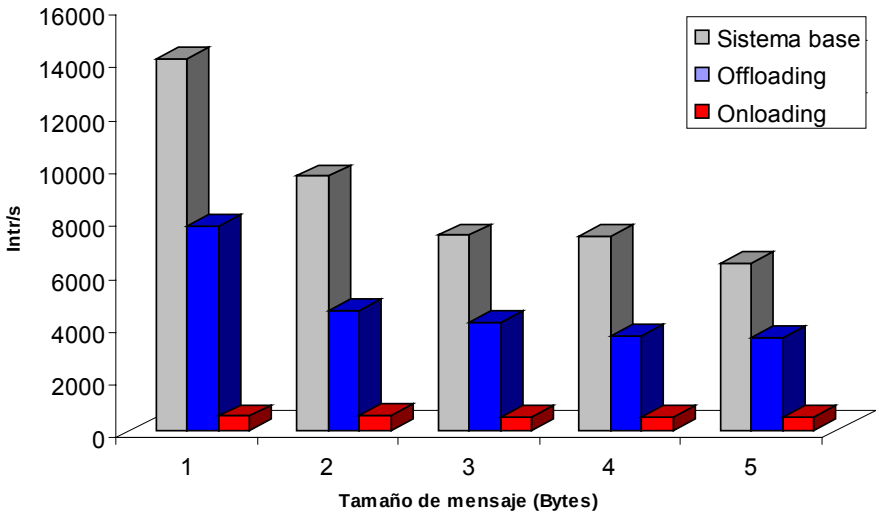


Figura 4.16. Comparación del número de interrupciones generadas por unidad de tiempo para diferentes tamaños de mensajes TCP (con Simics)

4.5.2 Comparación de anchos de banda y latencia

En la Figura 4.17 se muestra una primera comparación entre el ancho de banda máximo proporcionado por las simulaciones de las técnicas de externalización mediante *offloading* y *onloading*, comparado con el punto de partida sin externalización. Estos resultados se han obtenido utilizando el *benchmark Netpipe* [SNE96] y TCP como protocolo de transporte. Como se ha indicado en la Sección 4.2, *Netpipe* es un *benchmark* independiente del protocolo, que mide las prestaciones de la red.

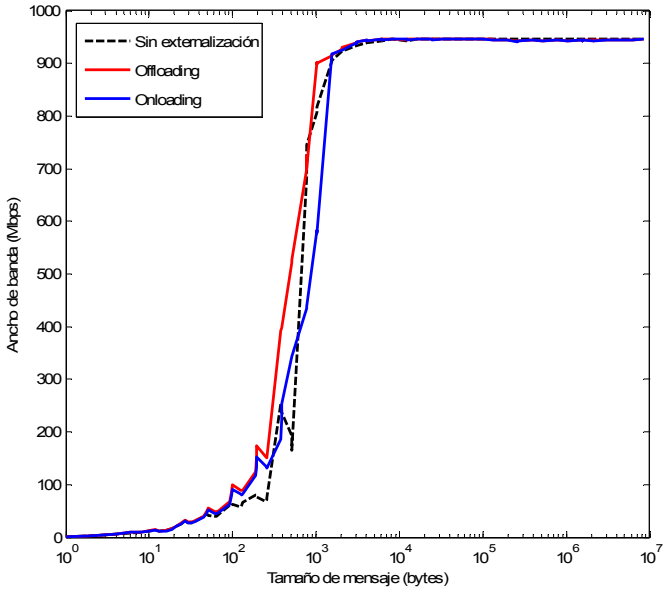
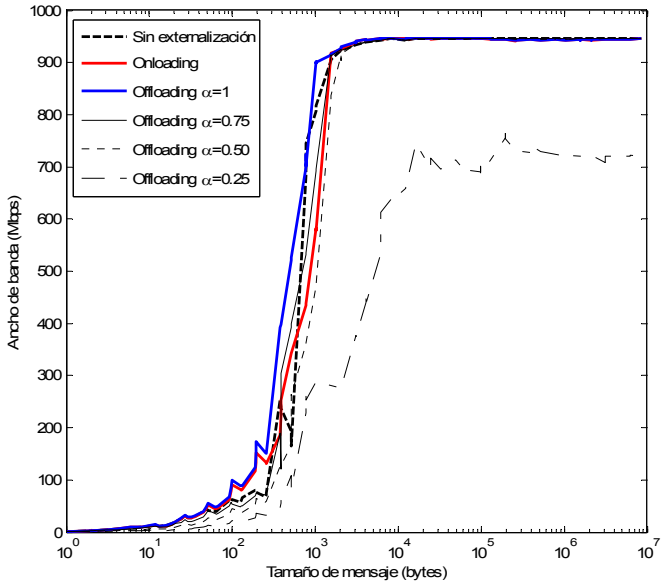


Figura 4.17. Simulación sin retardo en los buses (streaming)

Como punto de partida para nuestro análisis, la Figura 4.17 muestra el ancho de banda máximo obtenido mediante la externalización con *onloading* y con *offloading* en el caso ideal en el que no hay retardos en los accesos a memoria por parte de las distintas CPU. Como puede verse en la Figura 4.17, sin retardos en el subsistema de E/S ni en los accesos a memoria, hay ciertas diferencias entre el ancho de banda obtenido utilizando externalización con *offloading* y con *onloading*, para mensajes de tamaño intermedio, aunque el comportamiento es bastante irregular en el sentido de que el uso de onloading es, a veces, peor que la situación en la que no se externaliza. Estas se deben a la diferencia en la forma de gestionar las interrupciones en uno y otro caso y a la diferente latencia que ello implica de acuerdo a los modelos descritos en el Capítulo 3. En el caso de mensajes de tamaño grande, prácticamente no hay diferencia entre los anchos de banda pico obtenidos por ambas técnicas y por el sistema sin externalización, debido a que en este caso, el efecto dominante es el de las copias de datos, y éstas no son el cuello de botella. Al mismo tiempo, dado que la sobrecarga asociada a la aplicación es suficientemente baja, se puede alcanzar el ancho de banda de la red sin utilizar ningún tipo de externalización.



(a)

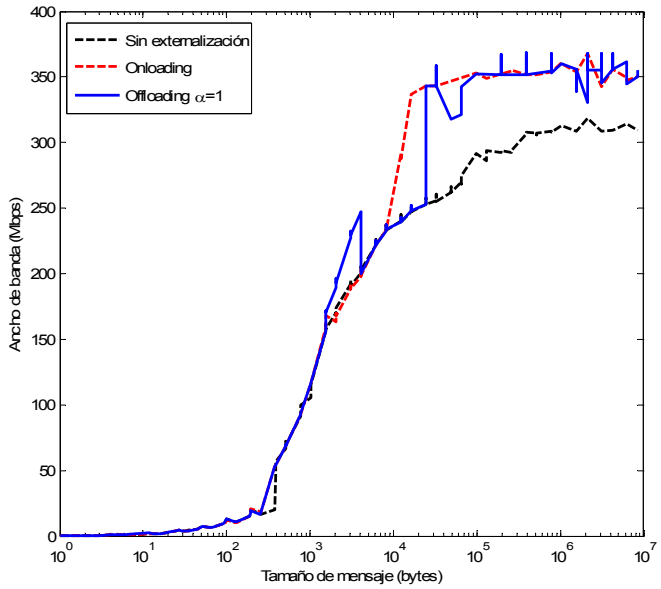


Figura 4.18(a) Efecto del parámetro α en un sistema sin retardos en los buses (streaming)
(b) Simulación con retardos en los buses (streaming)

En la Figura 4.18a se muestra el efecto de la diferencia de velocidad entre la CPU del nodo y la CPU de la NIC. En el caso de onloading, se está utilizando una de las CPUs de un SMP o CMP para la NIC y por tanto la velocidad de ambas CPUs es la misma (caso de α=1). Sin embargo, en el caso de *offloading*, la CPU que incorpora la NIC puede trabajar a una velocidad diferente, y esto se reflejará en las prestaciones del sistema de comunicaciones. En la Figura 4.18a, α=1 se corresponde con el caso en que

la velocidad de la CPU de la NIC es cuato veces inferior a la velocidad de la CPU del nodo.

En cualquier caso, un sistema en el que se consideran los buses ideales (sin retardos y en los que no existen colisiones por accesos concurrentes), es capaz de proporcionar todo el ancho de banda de un enlace Ethernet gigabit (1 Gbit/s) si la CPU dispone de un número de ciclos libres suficiente para procesar la pila de protocolos y puede realizar dicho procesamiento a la velocidad suficiente, dado que como ya se ha comentado, los buses de E/S del sistema suponen un cuello de botella muy importante en el camino de comunicación.

Así pues, los resultados anteriores solo pueden utilizarse como una prueba de corrección de los modelos presentados y como se ha dicho, sólo son una primera aproximación que ilustra el comportamiento diferente en el caso de la externalización con *offloading* y con *onloading*. Sin embargo, si se consideran condiciones de trabajo realistas, y por tanto, retardos en los buses y en los accesos a memoria, se obtienen los resultados experimentales que se muestran en la Figura 4.18b. Para obtener dichos resultados experimentales se ha utilizado un tiempo de acceso medio a memoria de 10 ciclos (como se ha comentado en la Sección 4.3.3) y un bus PCI de 64 bits de ancho de datos con un ancho de banda de 2.1 Gbit/s. Además, se ha utilizado un tiempo medio de penalización por colisión en los accesos a los buses de E/S de 20 ciclos. Como puede verse en la Figura 4.18b, el ancho de banda proporcionado por los sistemas sin externalización y con externalización es similar para mensajes de tamaño pequeño o intermedio, hasta 32 Kbytes aproximadamente. Esto se debe a la elevada sobrecarga generada en el procesador que recibe las interrupciones de la tarjeta de red, en el procesador que realiza el procesamiento de los protocolos, y en los buses de E/S con tamaños de mensaje pequeños. Al incrementar el tamaño del mensaje, las técnicas de externalización comienzan a producir mejoras en el ancho de banda respecto al sistema sin externalización. No obstante, hay que tener en cuenta que las simulaciones realizadas para obtener las Figuras 4.17 y 4.18, se han realizado con el *benchmark Netpipe* que, como se ha indicado anteriomente introduce una sobrecarga baja. Por tanto, estas simulaciones siguen siendo útiles sólo para validar los modelos de simulación y realizar un primer análisis del comportamiento de los modelos.

Un análisis más realista, debe incluir además, el efecto de la carga de la aplicación en el procesador principal del nodo. Dicho análisis se realiza en la Sección 4.6, en la que se utiliza el modelo LAWS descrito en el Capítulo 1. Como se ha visto , el modelo LAWS

incluye otros parámetros para caracterizar el nodo además de la carga de la aplicación, y poder alcanzar una primera aproximación para entender las mejoras introducidas por la externalización mediante *offloading* u *onloading*.

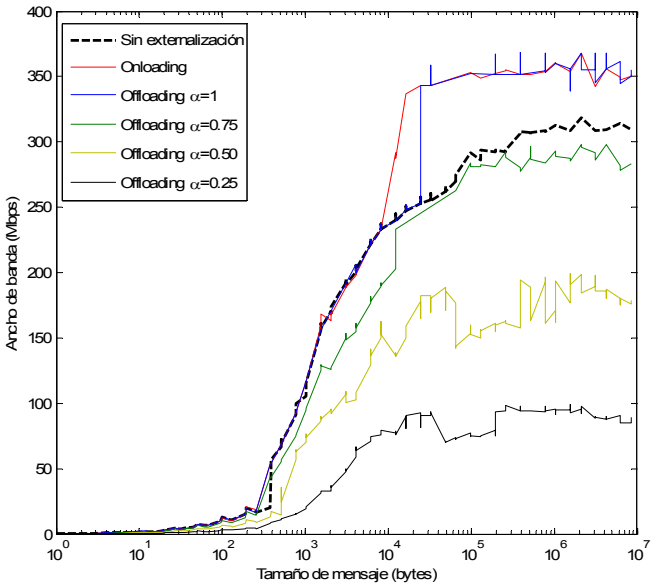
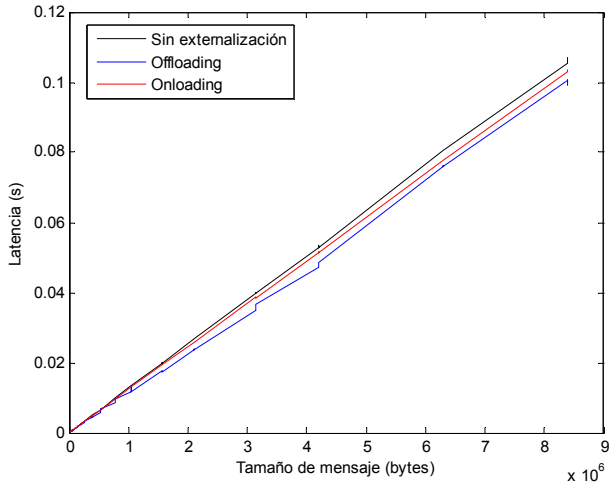


Figura 4.19. Efecto del parámetro α del modelo LAWS en el ancho de banda (streaming)

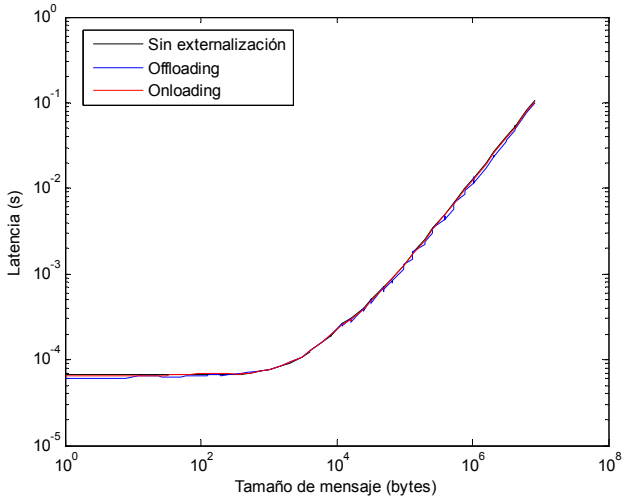
En la Figura 4.18, las simulaciones de externalización con *offloading* se han realizado considerando que el procesador de la NIC (CPU_1) y del nodo (CPU_0) son iguales. En un análisis de las dos técnicas de simulación desde el punto de vista del modelo LAWS, el parámetro α (**L**ag Ratio) se ha considerado la razón entre la velocidad de la CPU del nodo y de la NIC, fijándolose a 1. A medida que la CPU_1 se hace más lenta que la CPU del nodo (CPU_0), el parámetro α decrece (es decir, α=0.5 significa que la CPU del nodo es dos veces más rápida que la CPU de la NIC) y las mejoras obtenidas mediante la externalización con *offloading* también decrecen.

Los beneficios que proporciona la externalización mediante *offloading y onloading* afectan no sólo al ancho de banda sino también a la latencia. En la Figuras 4.20a y 4.20b se muestra la latencia medida para diferentes tamaños de mensajes, en el caso del sistema externalizado con *offloading*, con *onloading*, y para el sistema sin externalización. Como se puede ver en dicha figura, las latencias son menores para el sistema externalizado mediante *offloading* que para el sistema externalizado mediante *onloading*. Esta menor latencia en el caso de usar *offloading*, se debe a que dado que el protocolo ha sido externalizado en la interfaz de red, este puede interactuar con la red

con un menor número de transferencias a través de los buses de E/S del sistema. De aquí que la externalización mediante *offloading* proporcione una menor latencia.



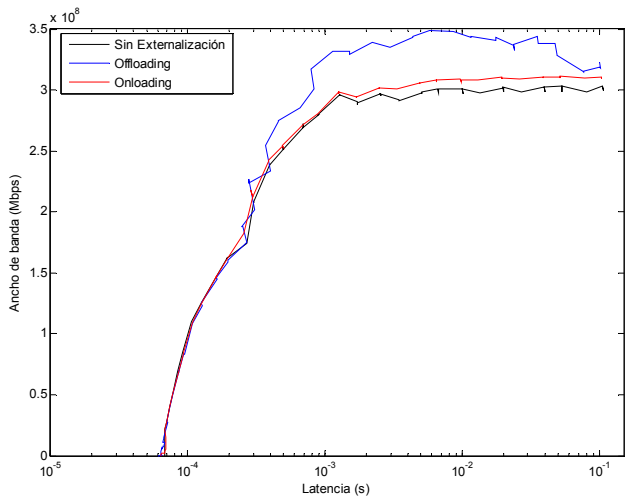
(a)



(b)

Figura 4.20. Latencia para la externalización con onloading y con offloading. (a) Escala lineal y (b) escala logarítmica

En la Figura 4.21a y 4.21b se muestra el gráfico de firma de red para las dos implementaciones de la externalización. Esta figura nos permite conocer la latencia de la red, que se corresponde con el primer punto en el eje del tiempo.



(a)

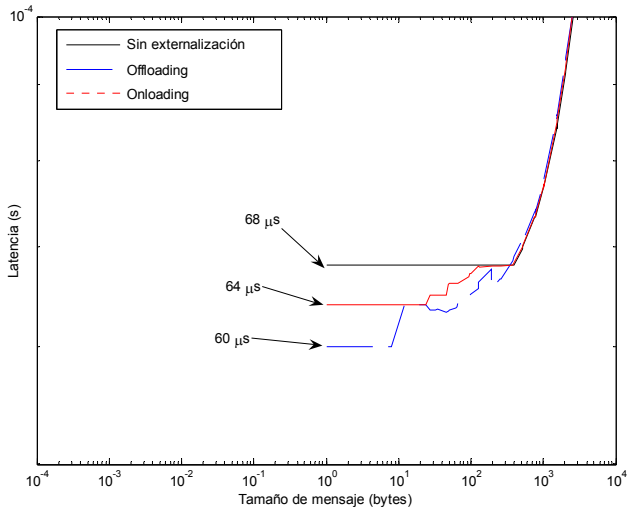


Figura 4.21. (a) Comparación firma Offloading / Onloading. (b)Detalle latencia

Teniendo en cuenta el primer punto de la gráfica de la Figura 4.21, y resumiendo los resultados de las Tablas 4.9 y 4.10, se tiene la Tabla 4.12.

Tabla 4.12. Comparación latencias de red offloading / onloading

Externalización	**Latencia (µs)**
Sin Externalización	68
Offloading	60
Onloading	64

Por tanto, la latencia de red es menor para la implementación de la externalización mediante *offloading* que para la externalización mediante *onloading*. Por otro lado, las latencias son muy similares en el caso del sistema sin externalización y del sistema con

externalización mediante *onloading* para mensajes de tamaño pequeño debido a la interacción con el sistema operativo y cambios de contexto y a la sobrecarga debida a las interrupciones generadas por esos mensajes pequeños. En cualquier caso, en la Figura 4.20a se muestra que al incrementar el tamaño del mensaje se consigue una mayor mejora en la latencia al externalizar, tanto con *offloading* como con *onloading*.

Los experimentos realizados hasta este momento, ponen de manifiesto la utilidad de la técnica de *offloading* en aquellos casos en los que la latencia sea importante. Por otro lado, los experimentos realizados con el objetivo de analizar la influencia de disponer de un procesador más lento en la NIC que el procesador principal del nodo, ponen de manifiesto que la técnica de *onloading*, aunque proporcione peores valores de latencia, puede estar más indicada si no se dispone de una NIC de última tecnología o si se quiere disponer de un sistema con externalización a un coste inferior que el de *offloading*. En cualquier caso, como ya se ha comentado, es necesario analizar la influencia de la carga de la aplicación en el procesador principal del nodo para comparar las técnicas de *offloading* y de *onloading* en un escenario más realista.

4.6 Estudio de la aplicabilidad del modelo de prestaciones LAWS

Hasta ahora se ha realizado un primer análisis de los resultados de ancho de banda y latencia obtenidos con los modelos de simulación que implementan las técnicas de externalización mediante *offloading* y *onloading* sin incluir un modelo funcional de memoria *cache* en la jearaquía de memoria. Además, para disponer de una descripción suficientemente representativa de las condiciones en las que se puede esperar más o menos nivel de mejora gracias a la externalización, es necesario tener en cuenta algunos parámetros importantes que todavía no se han considerado como la sobrecarga generada por la aplicación, o parámetros estructurales de la implementación que se haga de la externalización. Estos parámetros afectan de forma decisiva a los beneficios que pueden esperarse de la externalización y por tanto, se han de tener en cuenta en el análisis de prestaciones. El modelo LAWS, presentado en la Sección 1.7, proporciona un marco para estimar la mejora máxima que prodría proporcionar la externalización, teniendo en cuenta parámetros como son la carga de trabajo de la CPU del nodo, el modelo de externalización utilizado y su implementación, o las características tecnológicas del

nodo. A continuación, además de validar el modelo LAWS mediante nuestros modelos de simulación de sistema completo, lo utilizamos para orientarnos en el estudio del espacio de diseño de la externalización.

4.6.1 Simulación con Simics

Como se vió con detalle en la Sección 1.7 (Capítulo 1), el modelo LAWS expresa la mejora proporcionada por la externalización de protocolos, en función de cuatro parámetros, el parámetro tecnológico, α (***L**ag ratio*), el parámetro de aplicación, γ (***A**pplication ratio*), el parámetro de ancho de banda, W (***W**ire ratio*) *y* el parámetro estructural, S (***S**tructural ratio*). La Expresión 4.2, ya discutida con detalle en el Capítulo 1, muestra la mejora δB introducida en un sistema en el que se ha externalizado una porción *p* de los procesos de comunicación, y en el que se tiene una sobrecarga de comunicación *o*, una sobrecarga de aplicación *a*, y un enlace de red con ancho de banda *B*.

$$\delta B = \frac{\min\left(B, \frac{1}{(aX+(1-p)oX}, \frac{1}{poY\beta}\right) - \min\left(B, \frac{1}{aX+oX}\right)}{\min\left(B, \frac{1}{aX+oX}\right)} \qquad (4.2)$$

Para simular las condiciones que plantea el modelo LAWS y hacer posible la variación de las condiciones definidas por los parámetros L, A, W, y S de manera que se puedan comparar los resultados de las simulaciones con los del modelo teórico, es necesario generar cargas variables correspondientes a los cambios en el parámetro γ. Por tanto, se debe utilizar un test adecuado para cambiar las cargas de trabajo en el sistema. Esto requiere implementar un mecanismo de sincronización entre los procesos generadores de carga y las hebras TCP/IP que simule un determinado nivel de carga en el procesador principal del nodo a la vez que se procesan los paquetes TCP/IP. Así, se puede simular la carga que una aplicación genera en el procesador principal para producir nuevos datos a enviar o procesar datos recibidos en transferencias anteriores.

El *benchmark* elegido ha sido *Hpcbench* [HUA05], (Sección 4.2.3), dado que proporciona todas las medidas de prestaciones que hemos de realizar y, además, es de

libre distribución y podemos modificarlo para incluir los mecanismos de sincronización y las rutinas de generación de carga que necesitamos. Como se ha indicado, mediante esos mecanismos se consigue que las hebras TCP/IP que se ejecutan en la CPU de la tarjeta de red estén sincronizadas con la aplicación (y el sistema operativo) que se ejecuta en la CPU_0.

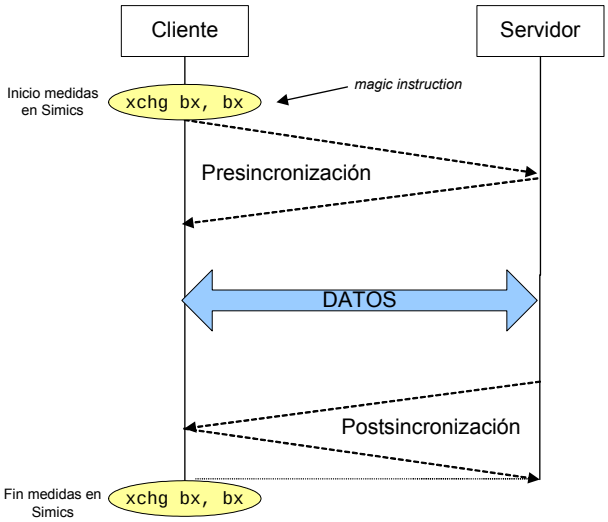


Figura 4.22. Funcionamiento Hpcbench y sincronización con Simics

El funcionamiento de *Hpcbench* se ilustra en la Figura 4.22. El *benchmark*, tras un proceso de presincronización en el que el cliente recibe los parámetros de simulación del servidor (tamaño máximo de mensaje, tipo de test, tamaño de los sockets, número de medidas, tiempo de simulación, etc.), inicia la transferencia de cada mensaje y realiza los cálulos necesarios para estimar el throughput y la latencia en cada transferencia. Por otra parte, *Hpcbench* realiza una estimación del número de veces que tiene que repetir una transferencia para asegurarse que el tiempo de medida no sea menor que la resolución, de forma similar a *Netpipe* [HUA04, HUA05], tal y como se ha explicado en la Sección 4.2.2. El tamaño de mensaje se incrementa exponencialmente en cada transferencia, partiendo de un tamaño inicial de 1 byte [HUA04].

Como se ha comentado en la Sección 4.3, la sincronización con el simulador se realiza mediante la instrucción *magic* como se muestra en la Figura 4.22. Dicha instrucción se ejecuta al iniciar y al finalizar el *benchmark,* deteniendo el simulador y asegurando así que todas las estadísticas se recogen en las mismas condiciones (como por ejemplo, el número de fallos de *cache*). Para generar una carga variable, se ha

modificado *Hpcbench* de forma que arranque una hebra que llamaremos hebra de generación de carga. Esta hebra estará esperando a que el proceso principal de *Hpcbench* le de permiso para comenzar a ejecutarse. La modificación implementada permite que, si originalmente el procesamiento de los datos recibidos era prácticamente inocuo para la CPU del nodo, ahora consume un tiempo de CPU gracias a un proceso generador de carga, como se muestra en la Figura 4.23. Así modelamos el comportamiento de una aplicación, en la que se genera carga en la CPU al procesar los datos de los *buffers de streaming*, como por ejemplo en el caso de la decodificación de datos MPEG2 en secuencias de video. Si esta carga fuese muy elevada, por ejemplo debido al uso de una técnica de codificación/decodificación que requiriese mucho tiempo de CPU (es decir, $\gamma >> 1$ en el modelo LAWS) podría ocurrir que el nodo no sea capaz de absorber el flujo de datos y la *red* debería esperar a que se terminaran de procesar los datos y se vacíen los *buffers* de streaming para seguir proporcionando datos. En otros sistemas comerciales de transmisión de video *Streaming* mediante TCP, se implementa un mecanismo denominado "*stop and wait*" [WAN04], de forma que cuando un paquete determinado no ha llegado en el tiempo de visualización del video, ésta se detiene hasta que no llega el paquete. Esto, junto con las características propias del protocolo TCP, hace que no se pierdan datos, como ocurriría si se utilizase UDP.

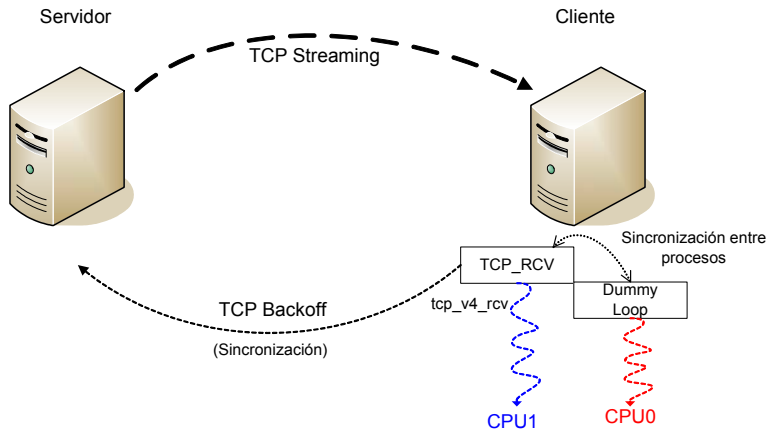


Figura 4.23. Funcionamiento de Hpcbench modificado

4.6.2 Evaluación experimental del modelo LAWS

Como ya se ha comentado, los modelos LAWS [SHI03] y EMO [GIL05] son un buen punto de partida para conseguir modelos cuantitativos aproximados que permiten extraer conclusiones cualitativas acerca de las condiciones bajo las que externalización puede resultar ventajosa. Para conseguir una validación experimental adecuada de estos modelos, se necesita un abanico de aplicaciones o *benchmarks* que pongan de manifiesto las condiciones en las que se extraen las expresiones matemáticas correspondientes. En el Capítulo 3 se han descrito los modelos de simulación de sistema completo desarrollados [ORT06, ORT07] para el análisis experimental de la externalización. Estos serán los que usemos para validar el modelo teórico LAWS. En esta sección, se van a comparar los resultados de simulación y las predicciones del modelo LAWS. Además, existen otros trabajos como [DIA05, ORT08d] que proponen la utilización de modelos HDL para simular los efectos de la externalización de protocolos y validar estos modelos teóricos.

La Figura 4.24 muestra dos curvas correspondientes al incremento del ancho de banda pico frente al parámetro γ (*Application ratio*) o razón entre la carga generada por la aplicación y la sobrecarga generada por los procesos de comunicaciones. Una de las curvas corresponde a la predicción del modelo LAWS y la otra ha sido obtenida experimentalmente mediante simulación para unas condiciones prefijadas ($p=0.75$, $\alpha=1$, $\beta=1$ y $B=1Gbps$) con mensajes de 2 Mbits. A pesar de que existe cierto parecido en el comportamiento cualitativo puede verse que existen diferencias cuantitativas importantes entre ambas curvas, no solamente en el valor máximo de la mejora en el ancho de banda sino también en la ubicación de dicho máximo, etc. Esto no es de extrañar, ya que el modelo LAWS simplemente proporciona una cota superior, como efectivamente ocurre en la Figura 4.24.

Como se ha indicado (Capítulo 1), el modelo LAWS tiene en cuenta la distribución de la carga de CPU asociada a la aplicación y a las comunicaciones antes y después de la externalización y proporciona una estimación del ancho de banda pico del camino de comunicación segmentado de acuerdo al ancho de banda del correspondiente cuello de botella activo en unas circunstancias determinadas (el enlace, la NIC o la CPU del nodo). También introduce los posibles efectos de la implementación de la externalización, a través del cambio en la carga total de trabajo, representado mediante

el parámetro β (*Structural Ratio*). No obstante, hay también efectos en el tiempo total de CPU consumido por las aplicaciones debido a los perfiles específicos de uso de la jerarquía de memoria y otros elementos del subsistema de entrada/salida. Por tanto, aunque en el caso de la predicción de anchos de banda pico, el camino de comunicación puede considerarse segmentado, existen dependencias que afectan a los beneficios que aporta la externalización. No obstante el modelo LAWS ofrece una información cualitativa importante, aunque la cota superior que proporciona puede estar bastante lejos de los resultados experimentales.

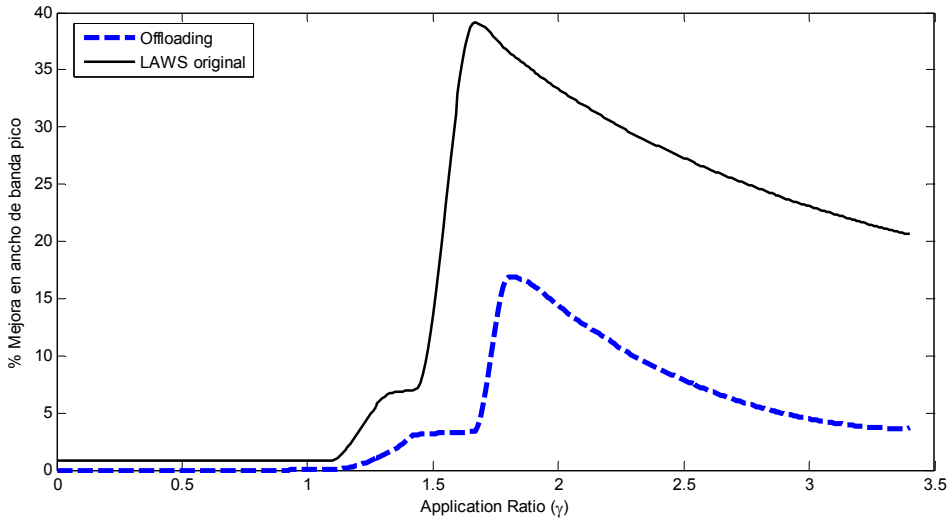


Figura 4.24. Comparación entre la mejora de ancho de banda pico predicha por el modelo LAWS y la obtenida mediante simulación para offloading

La Figura 4.24 muestra la variación de la mejora en el throughput con el parámetro de aplicación γ. No obstante, es necesario tener en cuenta los demás parámetros del modelo LAWS así como el tamaño del mensaje utilizado en los experimentos. Para ello, se han realizado diferentes experimentos cuyos resultados se muestran en las secciones posteriores. Además hemos propuesto un modelo LAWS modificado con el fin de conseguir una mejor predicción de dichos resultados experimentales y extraer algunas conclusiones acerca de los factores que intervienen más decisivamente en las discrepancias entre el modelo LAWS y la experiencia.

4.6.3 Propuesta de un modelo LAWS modificado

Para acercar más los resultados experimentales a un modelo cuantitativo que nos ayude a entender el efecto de los factores que determinan las prestaciones, se han añadido tres nuevos parámetros δ_a, δ_o y τ al modelo LAWS presentado en [SHI03]. El significado de esos parámetros, se puede entender a partir de la expresión del ancho de banda pico, de la ecuación 4.3.

$$\delta b = \frac{\min\left(B, \frac{1}{(a(1+\delta_a)X + (1-p)o(1+\delta_o)X)(1+\tau)}, \frac{1}{\beta o(1+\delta_o)poY}\right)}{\min\left(B, \frac{1}{aX+oX}\right)} - 1 \quad (4.3)$$

Los parámetros δ_a y δ_o representan la porción de cambio tras externalizar en el trabajo por unidad de datos para la aplicación y el *overhead* de comunicación respectivamente. El parámetro τ representa la proporción de cambio en la carga de trabajo de la CPU después de externalizar debido a la sobrecarga que genera la comunicación entre la CPU y la NIC a través del subsistema de E/S. De esta forma, *a* pasa a ser $a+a\delta_a$, *o* pasa a ser $o+o\delta_o$ y la carga de la CPU después de externalizar pasa de W a W+Wτ.

La Figura 4.25 muestra que es posible obtener aproximaciones mejores a los resultados experimentales utilizando la expresión de la Ecuación 4.3, y valores adecuados para los parámetros δ_a , δ_o y τ. En la Tabla 4.13 pueden verse los valores de estos tres parámetros, de forma que se obtienen las curvas respectivas LAWSmod(1) y LAWSmod(2).

Tabla 4.13. Valores de los parámetros δ_a, δ_o y τ en las curvas LAWSmod

Parámetro	LAWSmod(1)	LAWSmod(2)
δ_a	$5\cdot10^{-5}$	$5\cdot10^{-5}$
δ_o	0	$-2.6\cdot10^{-6}$
τ	0.05	0.147

Para ajustar estos parámetros se ha realizado una minimización de la función error entre el modelo LAWS modificado y los resultados experimentales, utilizando *matlab*

[MAT07] y los parámetros del modelo LAWS (p, α, β, B) original. Los valores que minimizan dicha función error se ha utilizado como punto de partida para encontrar el mejor ajuste con los datos experimentales.

El primer conjunto de parámetros LAWSmod(1) proporciona un mejor ajuste del modelo a los resultados experimentales para valores bajos del parámetro γ (Application ratio), y el segundo conjunto de parámetros LAWSmod(2) para valores altos del parámetro γ.

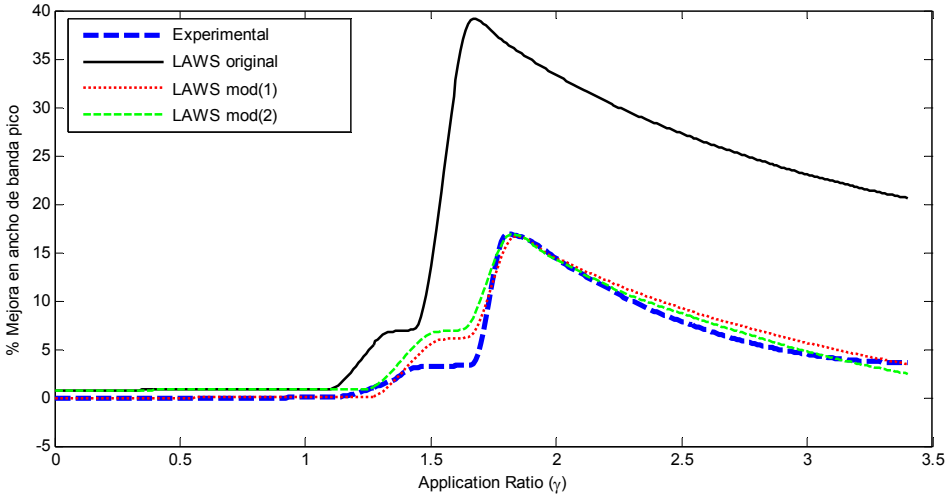


Figura 4.25. Aproximación a los resultados experimentales mediante el modelo LAWS modificado

Si se modifican las características del sistema y de la aplicación, se modifican los valores de estos parámetros de la misma forma que se modifican los valores de los parámetros del modelo LAWS original. Con estos parámetros se busca expresar de una forma algo más realista el efecto de la externalización en los parámetros medidos por el modelo LAWS y la desviación con respecto al comportamiento de cauce segmentado que presupone el modelo LAWS.

Con el modelo LAWS modificado, es posible por tanto, conseguir una mejor aproximación que prediga los resultados experimentales, pudiendo obtener información más precisa acerca sobre los valores de γ donde la curva experimental de mejora toma valores mayores que cero y el valor de γ donde dicha curva presenta su máximo. De esta forma, de acuerdo con los valores de los parámetros δ_a, δ_o y τ se pueden obtener las siguientes conclusiones:

1. Después de externalizar, la carga de trabajo de la CPU (que incluye el tiempo de la aplicación más el de comunicación no externalizado) del nodo aparentemente necesita más tiempo de ejecución que el que supone el modelo LAWS ($\tau>0$).
2. La carga de trabajo asociada a la aplicación se incrementa con respecto a la que supone el modelo LAWS ($\delta_a>0$).
3. La sobrecarga de comunicación decrece ($\delta_o<0$) respecto a la que presupone el modelo LAWS, debido a que, en realidad, el driver de la NIC se sigue ejecutando en la CPU_0, junto con el procesamiento de las interrupciones. Esto no se tiene en cuenta en el modelo LAWS cuando *p=1*.

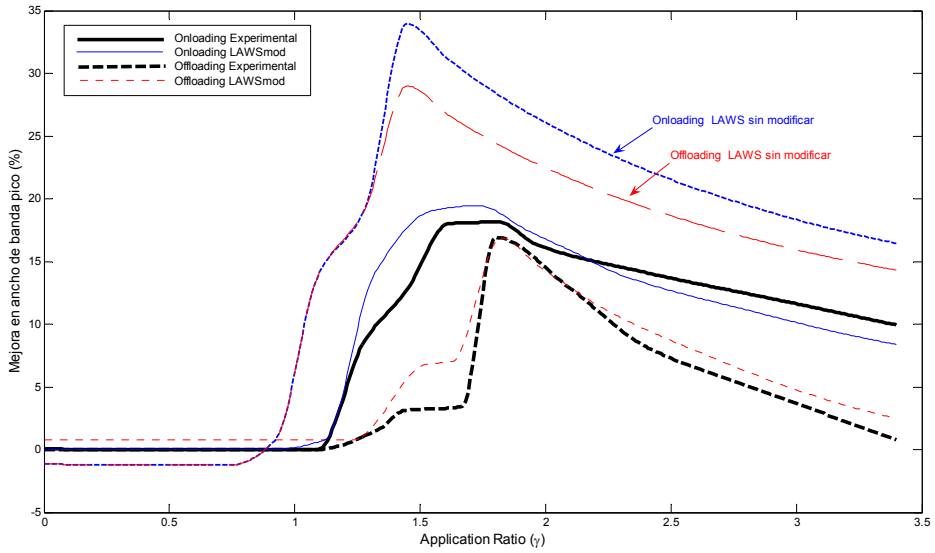
Por tanto, mientras que el parámetro τ está relacionado con las características de los buses de E/S y con la ubicación de la NIC (Sección 1.2), los parámetros δa y δo están relacionados con la implementación concreta que se haga de la externalización y con la integración en el sistema operativo (Sección 1.3).

Por otro lado, se puede ver que el decrecimiento de la mejora del ancho de banda pico conforme γ crece se produce de una manera más rápida al principio y luego más lentamente en los resultados experimentales que en los predichos por el modelo LAWS (correspondiente a la ley 1/γ). Probablemente, con la inclusión de más parámetros en un nuevo modelo LAWS modificado se pueda explicar este efecto. Esto además, proporcionaría una mejor comprensión de los elementos que afectan a las prestaciones ofrecidas por la externalización.

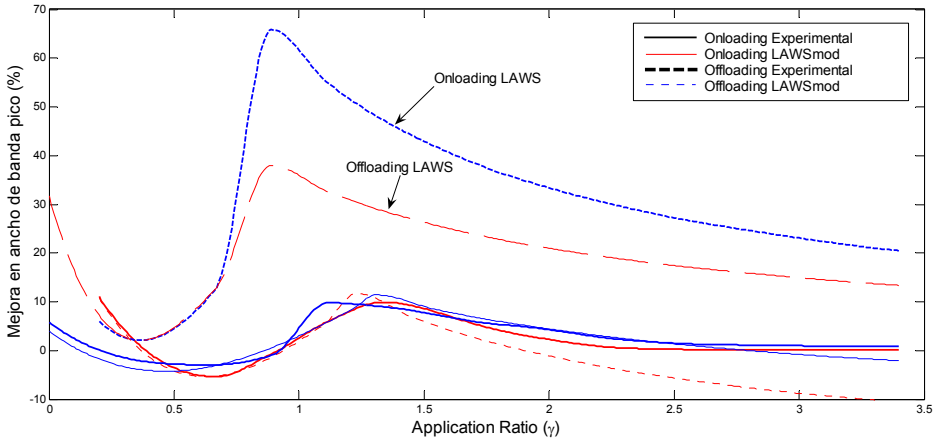
Las Figuras 4.26a y 4.26b muestran las curvas correspondientes a la mejora en el ancho de banda pico frente al parámetro γ, que modela la relación entre la sobrecarga asociada a los procesos de la aplicación y la sobrecarga asociada a los procesos de comunicación, tantopara la externalización mediante *offloading* como mediante *onloading*. Para los experimentos se ha considerado una comunicación unidireccional con TCP como protocolo de transporte y transfiriendo mensajes de 2 Mbits y 8 Kbits, para las Figuras 4.26a y 4.26b respectivamente. Se han incluido además las curvas correspondientes a la mejora en el ancho de banda pico que predice el modelo LAWS para los casos de uso de *offloading* y *onloading*.

También en estos casos una cosa está clara: mediante el modelo LAWS se predicen mejoras mayores que las observadas: el modelo LAWS es una cota superior a las prestaciones.

El comportamiento observado al variar γ se parece bastante al comportamiento experimental, si bien los valores exactos para los puntos canónicos cambian. Como en el caso anterior, esto se explica porque el modelo LAWS considera un sistema que funciona como un cauce segmentado perfecto, y para aplicaciones que realicen transferencias de secuencias (*streaming*). En la realidad no se tiene ni un cauce perfecto ni aplicaciones que se comporten exactamente como se necesita.



(a)



(b)

Figura 4.26. Comparación de resultados de simulación con la predicción del modelo LAWS para (a) mensajes de 2Mbits y (b) mensajes de 8 Kbits

En la Tabla 4.14 se resumen los valores de los parámetros del modelo LAWS para las implementaciones de la externalización correspondientes a la Figura 4.26 y en la Tabla 4.15, los valores de los parámetros del modelo LAWS modificado que se han

usado utilizados para ajustar el modelo LAWS a los resultados experimentales en cada caso.

Tabla 4.14. Parámetros del modelo LAWS en las simulaciones con Simics

Parámetro LAWS	Offloading	Onloading
$\boldsymbol{p}$	0.55	0.75
$\boldsymbol{\beta}$	0.4	0.45
$\boldsymbol{B}$	1 Gbps	1 Gpbs
$\boldsymbol{X}$	1	1
$\boldsymbol{Y}$	1	1

Tabla 4.15. Valores de los parámetros en el modelo LAWS modificado (Figuras 4.26a y 4.26b)

Tamaño de Mensaje	Parametro LAWS modificado	Offloading	Onloading
2 Mbits	δ_a	$3.11 \cdot 10^{-5}$	$8.6 \cdot 10^{-5}$
	δ_o	$-3.9 \cdot 10^{-5}$	$-7.3 \cdot 10^{-5}$
	τ	0.165	0.034
8 Kbits	δ_a	$1 \cdot 10^{-4}$	$1.43 \cdot 10^{-4}$
	δ_o	$-1.7 \cdot 10^{-4}$	$-1.8 \cdot 10^{-4}$
	τ	0.36	0.15

En la Tabla 4.15 puede verse como al utilizar mensajes de tamaño pequeño (8 Kbits) el parámetro τ crece, debido a que en este caso, se realiza un uso más intensivo de los buses de E/S que en el caso de mensajes de tamaño grande (2 Mbits). De la misma forma, los parámetros δ_a y δ_o incrementan un orden de magnitud su valor debido a la mayor influencia de la implementación de la externalización y de su integración en el sistema operativo. Además, de la Tabla 4.15 pueden extraerse las siguientes conclusiones:

1) δ_a (*onloading*) > δ_a (*offloading*) : se produce una desviación de la carga de trabajo asociada a la aplicación mayor en el caso de *onloading* que en el caso de *offloading*.

2) δ_o (*onloading*) < δ_o (*offloading*) : la desviación de la carga de comunicación respecto a la que presupone el modelo LAWS es mayor en offloading que en onloading.

Esto se debe a que en offloading, en realidad, se sigue ejecutando el driver de la NIC en la misma CPU que se está ejecutando la aplicación.

3) τ (*onloading*) < τ (*offloading*) : debido a que en el caso de offloading existe una mayor interacción entre la CPU_0 y la NIC (por ejemplo, el driver de la NIC se sigue ejecutando en la CPU_0).

Evidentemente, si se modifica la implementación de la externalización varian los parámetros de la Tabla 4.14 y por tanto, el ajuste de los parámetros de la Tabla 4.15 (modelo modificado) también varía.

No obstante, no es objeto de este trabajo hacer un estudio completo de la validez del modelo LAWS. Se pretende ver que los comportamientos son cualitativamente parecidos y que se pueden aproximar bastante. Esto sirve de validación del modelo LAWS, pero también de validación de nuestros modelos de simulación con Simics. De hecho, las Figuras 4.17, 4.18 y 4.19 muestran valores de anchos de banda máximo similares y correspondientes a valores del parámetro γ (*Application ratio*) pequeños.

De las Figuras 4.26a y 4.26b se puede concluir que ambas técnicas de externalización, *offloading* y *onloading,* proporcionan casi la misma mejora para bajas cargas de carga de trabajo asociadas a la aplicación (valores bajos de γ). Conforme el parámetro γ crece (mayor sobrecarga asociada a la aplicación con respecto a la asociada a los procesos de comunicaciones) se puede observar lo siguiente:

1. La curva de mejora para el caso de *onloading* crece más rápidamente que para el caso de *offloading*.
2. El valor máximo de la mejora proporcionada por la externalización es mayor en el caso de usar *onloading* que en el caso de usar *offloading*.
3. En el caso de valores altos de γ (valores altos de la sobrecarga asociada a la aplicación respecto a la asociada a los procesos de comunicaciones), la mejora del ancho de banda pico decrece más rápidamente para el caso de *onloading* que para el caso de *offloading*. Esto significa que la sobrecarga asociada a la aplicación necesaria para colapsar el nodo, puede ser mayor en el caso de tener una externalización implementada mediante *onloading* que en el caso de una implementación de la externalización con *offloading*.

Comparando la Figura 4.26a con la Figura 4.26b, se pone de manifiesto la clara influencia del tamaño del mensaje con la mejora en el ancho de banda pico. Además, la la variación de los nuevos parámetros al cambiar el tamaño del mensaje pone de manifiesto la importancia de la implementación que se haga de la externalización y de la integración en el sistema operativo, como ya se comentó en el Capítulo 1. Esta influencia es mayor cuanto menor es el tamaño del mensaje debido a la sobrecarga generada en los procesadores y en los buses de E/S. Los resultados experimentales mostrados en la Figura 4.26a muestran como el pico en la mejora del ancho de banda se mueve hacia valores más bajos del parámetro γ . La Figura 4.26b, muestra además, que la forma de las tendencias de las curvas de mejora para el caso de *onloading* y *offloading* son muy similares. Por tanto, si se quiere que la externalización proporcione mejoras para tamaños de mensaje pequeños, la implementación de la externalización y la arquitectura del sistema son más críticas, en cuanto a los buses de E/S, acceso a memoria y la ubicación de la NIC.

4.6.4 Efecto del parámetro σ (*wire ratio*).

En el análisis de las prestaciones proporcionadas por la externalización, es necesario tener en cuenta el efecto de la tecnología del nodo. Dicho de otra forma, considerar el ancho de banda que el nodo es capaz de proporcionar sin externalización. Este efecto se hace mediante el parámetro σ (*Wire ratio*).

En esta sección se va a analizar el efecto de la variación del parámetro σ (*Wire ratio*) en las simulaciones del modelo LAWS. A medida que el parámetro σ es menor, la porción de ancho de banda que el nodo puede proporcionar sin externalización es también menor. Por tanto, según el modelo LAWS, la mejora proporcionada por la externalización debe ser mayor cuanto menor sea el parámetro σ.

En la Figura 4.27 se muestra la variación teórica de la mejora para cambios en γ, y distintos valores del parámetro σ, según el modelo LAWS (Ecuación 4.4). En dicha figura puede apreciarse la dependencia de la mejora en ancho de banda con $1/\sigma$, de forma similar a lo observado en los resultados experimentales presentados en las Figuras 4.28 y 4.29. De esta forma, y de acuerdo a los resultados experimentales

presentados, se tiene una clara constatación de la validez del modelo LAWS y también de nuestros modelos de simulación con Simics.

$$\delta b = \frac{\min\left(\frac{1}{\sigma}, \frac{1}{\gamma+(1-p)}, \frac{1}{p\alpha\beta}\right) - \min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)}{\min\left(\frac{1}{\sigma}, \frac{1}{1+\gamma}\right)} \tag{4.4}$$

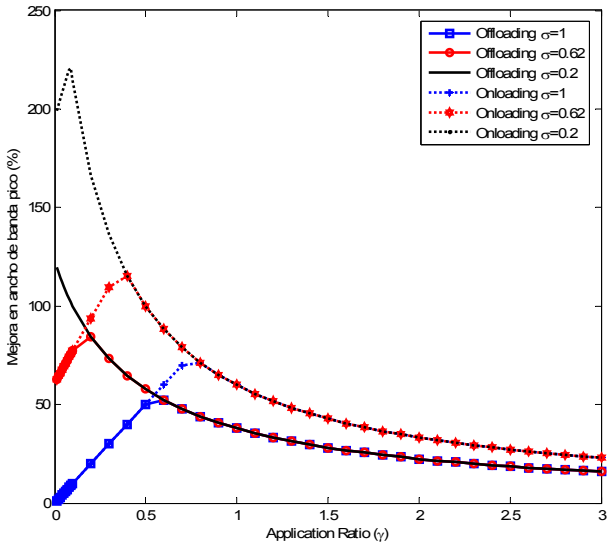


Figura 4.27. Variación teorica de la mejora pico según el modelo LAWS para diferentes valores de σ

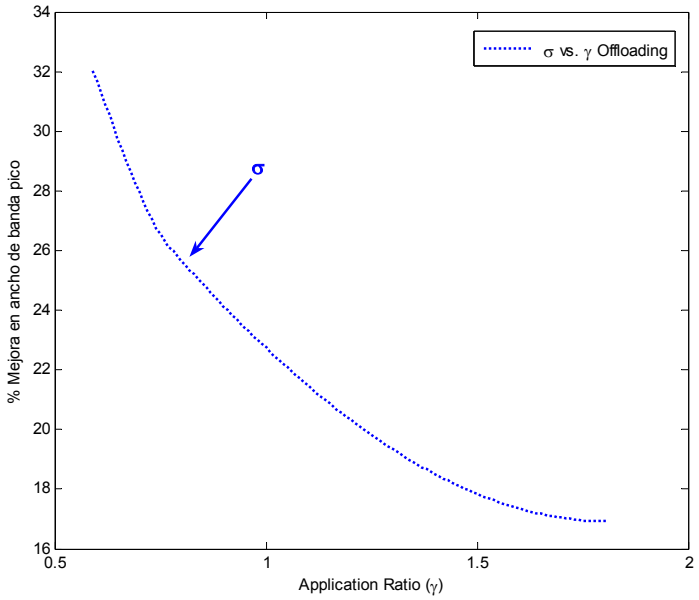


Figura 4.28. Efecto observado experimentalmente de la variación del parámetro σ en la externalización mediante offloading

En la Figura 4.28 se muestran los resultados de los experimentos que ponen de manifiesto el efecto de la variación del parámetro σ en el caso de la externalización mediante *offloading*. Para σ=1, el nodo es capaz de proporcionar todo el ancho de banda del enlace sin externalización, de acuerdo con el modelo LAWS. Es el caso en que menos mejora se debe esperar de la externalización de protocolos. Así, como se muestra en la Figura 4.28, la máxima mejora que puede proporcionar la externalización mediante offloading es del 16.92 %, para un γ=1.81. Para valores de σ inferiores a 1, el nodo no es capaz de proporcionar todo el ancho de banda del enlace de red sin externalización y, como se puede apreciar en la Figura 4.28, para valores de σ inferiores a 1, la ganancia máxima crece, a la vez que dicho máximo se desplaza hacia valores más bajos de γ. Este comportamiento, predicho por el modelo teórico LAWS, indica que cuanto más limitado está el nodo, las mejoras que proporciona la externalización son mayores. Al mismo tiempo, la sobrecarga debida al procesamiento de la aplicación tiene un mayor impacto en dicha mejora. Por tanto, se obtienen mejoras para valores más bajos del parámetro γ (*Application ratio*).

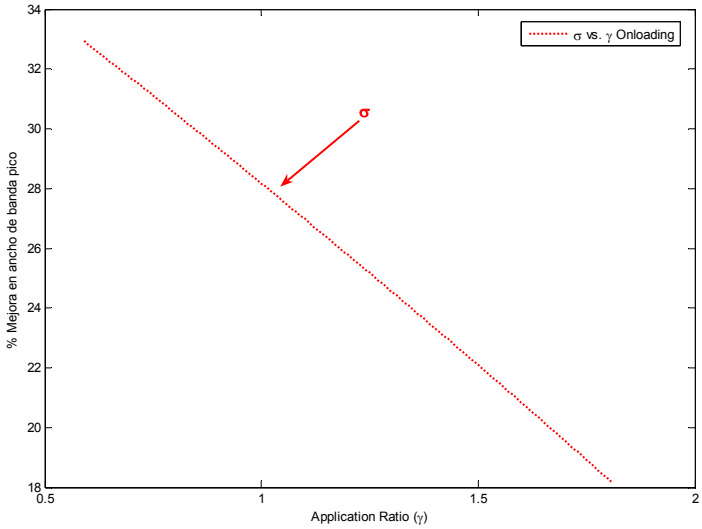


Figura 4.29. Efecto observado experimentalmente de la variación del parámetro σ en la externalización mediante onloading

En la Figura 4.29 se muestran los experimentos realizados para poner de manifiesto el efecto de variar el parámetro σ en el caso de la externalización con *onloading*. Ahora, para σ=1 se obtiene una mejora del 18.27% para un valor de γ = 1.80. Es decir, la externalización mediante *onloading*, proporciona una ganancia superior que la que se obtiene mediante *offloading* dándose el valor máximo en la mejora, aproximadamente para el mismo valor de γ en ambos casos. Sin embargo, al disminuir el valor de σ, la Tabla 4.16 muestra como la mejora proporcionada por la externalización con *onloading* es mayor. Por un lado esto es debido a la menor sobrecarga que, desde el punto de vista del sistema operativo, supone que las interrupciones de la NIC lleguen directamente a la CPU_1, donde se realiza el procesamiento del protocolo (ya que el driver, se ejecuta en la CPU_1). Por otro lado, en el caso de *onloading*, la CPU que realiza el procesamiento de los protocolos está conectada al resto del sistema a través del puente norte, evitando el bus PCI con la correspondiente latencia que esto supone.

Tabla 4.16. Efecto del parámetro σ en el modelo LAWS (experimentales)

Externalización	σ	Mejora del pico de ancho de banda (%)	γ
Offloading	1	16.92	1.81
	0.62	26.14	0.77
	0.2	32.03	0.59
Onloading	1	18.27	1.8
	0.62	30.53	0.8
	0.2	32.82	0.6

4.7 Efecto de la memoria cache en la externalización de protocolos

Hasta ahora, se han realizado simulaciones de sistema completo con los modelos presentados en la sección 3.5. Estos modelos consideran una arquitectura con todos los elementos que están presentes en un computador real. No obstante, el efecto de la memoria *cache* se ha modelado a través de los tiempos medios de acceso a la jerarquía de memoria. Es importante modelar el efecto de la memoria *cache* de la manera más precisa posible para tener la certeza de que se han alcanzado unos resultados suficientemente realistas, como se muestra en [NAH97]. En este apartado, se describe el modelo de memoria *cache* de dos niveles por CPU que hemos introducido en nuestros modelos de simulación con Simics que se describieron en el Capítulo 3 respectivamente para un sistema sin externalización, con externalización mediante *offloading* y con externalización mediante *onloading*. Ahora, a cada uno de estos modelos se ha conectado el modelo de memoria *cache* de dos niveles que se muestra en la Figura 4.30. En una etapa previa al primer nivel, se ha implementado un divisor (*splitter*) para separar los datos y las instrucciones en distintas para el nivel 1. En la memoria *cache* de primer nivel de datos se utiliza una política de escritura inmediata (*write-through*), y en la memoria *cache* de nivel 2, una política de post-escrituta (*write-back*). En un segundo nivel se ha utilizado una memoria *cache* unificada para datos e instrucciones.

En la Figura 4.31 se muestra la implementación de una memoria *cache* para la externalización mediante *offloading*. La arquitectura incluye una memoria *cache* conectada a la CPU del sistema (CPU_0), mientras que la CPU, en la NIC, no dispone de

memoria *cache*. Por tanto, el modelo para la NIC es el que se ha descrito en el Capítulo 3.

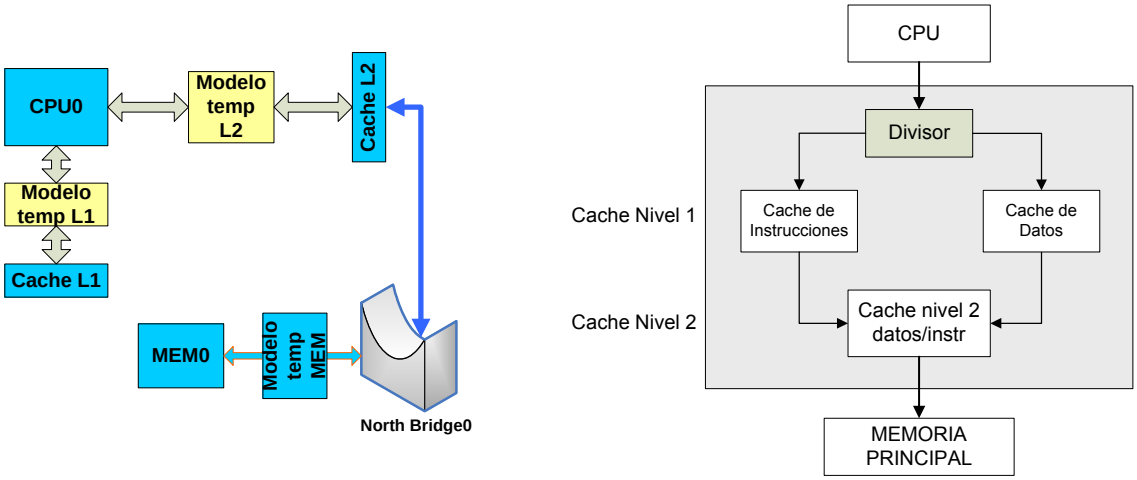


Figura 4.30. Conexión de la memoria cache al modelo sin externalización y detalle de la implementación en dos niveles

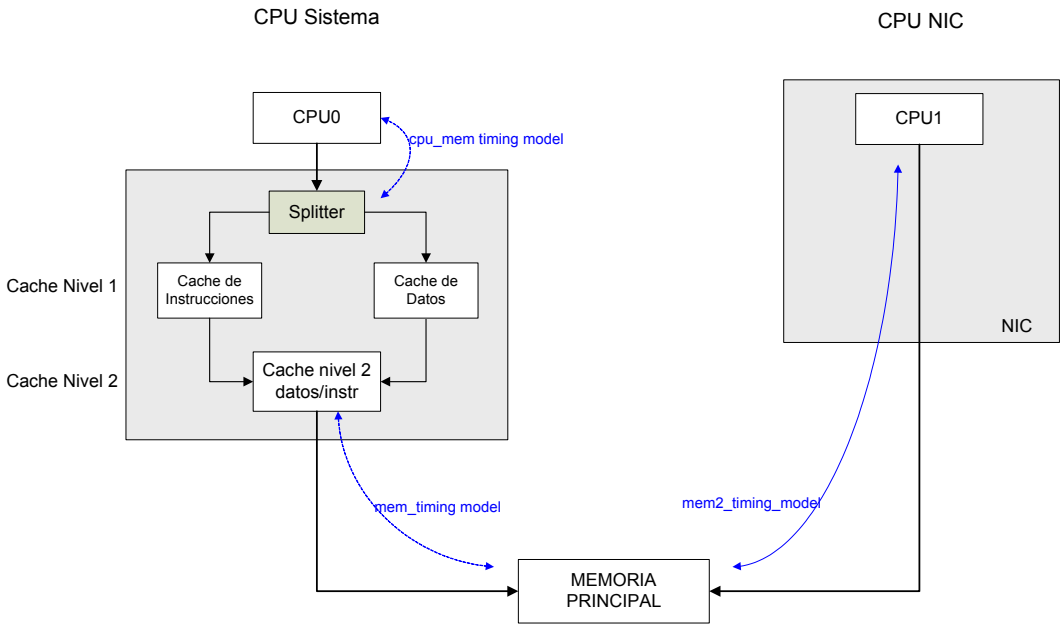


Figura 4.31. Memoria cache en el modelo de externalización mediante offloading. Conexión de los modelos de temporización

En la Tabla 4.17 se resumen las características de las memorias *cache* implementadas.

Para construir la jerarquía de memoria correctamente se necesita definir un modelo de tiempo adecuado. Para ello se han encadenado los modelos de temporización tal y como como muestra en la Figura 4.31. El modelo de memoria *cache* de la Figura 4.31 se compone de tres partes. Una primera parte corresponde al divisor que permite separar instrucciones y datos en memorias *cache* de nivel 1 diferentes, la segunda parte que incluye el modelo de tiempo que controla las transferencias entre el procesador y el divisor, y finalmente la tercera parte corresponde a un segundo modelo de tiempo para controlar las transferencias entre la *cache* de nivel 2 y la memoria principal.

Estos modelos de tiempo permiten configurar los parámetros de las memorias *cache* así como sus políticas de escritura, reemplazo y coherencia de la Tabla 4.17.

En la Figura 4.32 se muestra la implementación de la memoria *cache* en el caso de externalización mediante *onloading*. Se ha utilizado el protocolo MESI [MES04] para mantener la coherencia de memoria.

La jerarquía de memoria que se ha creado al añadir la memoria *cache* hace que ahora sean necesarios menos ciclos de reloj para completar los accesos a memoria cuando el dato buscado se encuentra en la *cache*. En los modelos previos se consideran accesos con tiempos promedio según una jerarquía con una determinada tasa de fallos de *cache*.

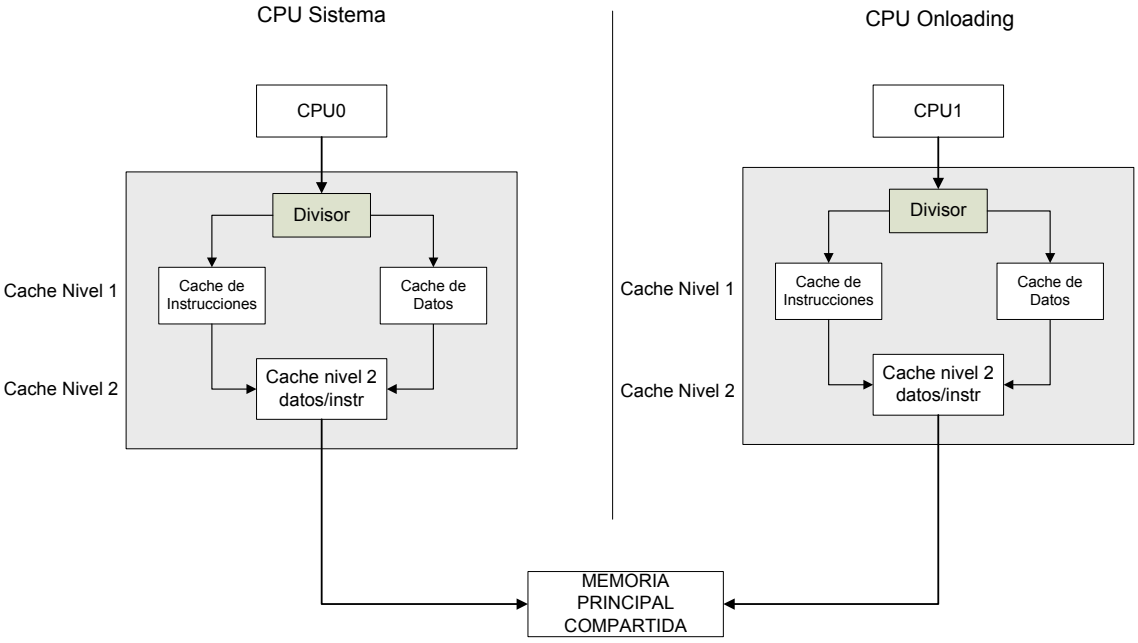


Figura 4.32. Memoria cache L2 compartida en el modelo de externalización mediante onloading

Tabla 4.17. Configuración de la memoria cache

	L1 Instrucciones	**L1 Datos**	**L2**
Política de escritura		*Write-through*	*Write-back*
Numero de líneas (líneas)	256	256	8096[†]
Tamaño líneas (bytes)	64	64	128
Tamaño *cache* (Kbytes)	16	16	1024
Asociatividad (líneas)	2	4	8
Write back (líneas)			1
Write allocate (líneas)			1
Politica reemplazo	LRU	LRU	LRU
Penalización lectura (ciclos)	2	3	5
Penalización escritura (ciclos)	1	3	5
Penalización lectura siguiente (ciclos)	0	0	0
Penalización escritura siguiente (ciclos)	0	0	0

Evidentemente, la mejora que puede proporcionar la memoria *cache* depende del número de aciertos al buscar un dato o una instrucción en la memoria *cache*, y esto dependerá a su vez de la localidad espacial y temporal de los datos e instrucciones de la aplicación que se esté ejecutando en el nodo y de las características de los códigos que implementen los protocolos de comunicación.

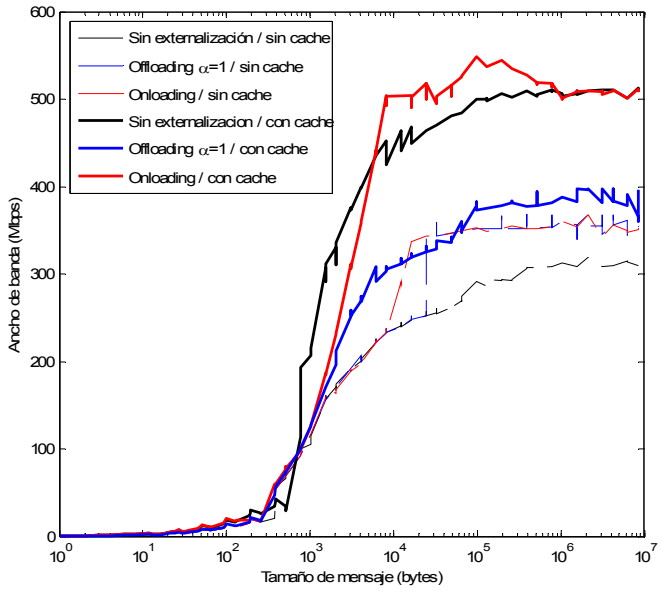


Figura 4.33. Ancho de Banda con memoria cache (streaming)

[†] Memoria *cache* L2 unificada para CPU0 y CPU1

En los experimentos realizados se ha utilizado el *benchmark Netpipe.* A partir de dichos experimentos se podrán comparar los resultados obtenidos con los presentados en las Secciones 4.3 y 4.4. En la Figura 4.33 se muestra el ancho de banda para las configuraciones sin externalización, con externalización mediante *offloading,* y con externalización mediante *onloading* con memoria *cache* compartida por las dos CPU en este último caso, según el esquema de la Figura 4.32. En la Figura 4.33 se muestra como para paquetes pequeños (de alrededor de 2 Kbytes), los anchos de banda proporcionados por la externalización, son similares en el caso de externalización mediante *onloading* y *offloading*, debido a que en este caso, la interacción con el sistema operativo y la contención en el bus de memoria y de E/S. Además, para mensajes de alrededor de 2 Kbytes, el ancho de banda del bus de E/S que ofrece la rutina *memcpy* utilizada por *Netpipe* para llevar a cabo las transferencias de memoria es relativamente bajo, como se muestra en [TUR03].

Por otro lado, al conectar un modelo de memoria *cache* a la CPU que realiza el procesamiento de los protocolos de comunicación se observa que el ancho de banda mejora (Figura 4.33). De hecho, al externalizar los protocolos de comunicación mediante *offloading*, utilizando una NIC cuya CPU no dispone de memoria *cache* empeoran las prestaciones respecto del sistema sin externalización. Puede encontrarse una explicación a este comportamiento desde el punto de vista del modelo LAWS descrito en la Sección 1.3, ya que este caso correspondería con un valor del parámetro α menor que 1 (es decir, la CPU del sistema con su jerarquía de memoria tiene unas prestaciones superiores a la CPU de la NIC). Existen trabajos como [KIM05] donde se propone la inclusión de una memoria *cache* en la interfaz de red para mejorar las prestaciones de la misma y reducir el tráfico en los buses de E/S.

Para ver más claramente el comportamiento del sistema sin externalización y con externalización mediante *offloading* y mediante *onloading*, en la Sección 4.7.1 se describe el análisis que hemos realizado con el modelo LAWS [SHI03] para poner de manifiesto la dependencia de las prestaciones de los sistemas en función de la carga que la aplicación genera en la CPU_0 (como se ha visto esta carga se expresa a través del parámetro γ).

En el caso de mensajes de tamaño alrededor de 2 Kbytes, la Figura 4.33 muestra como las prestaciones ofrecidas por el sistema sin externalización pueden superar a las del sistema con externalización, debido a la sobrecarga generada por los procesos de comunicaciones, tanto en la CPU del nodo como en los buses de E/S. En el caso de

externalización mediante *offloading* no se está aprovechando la ventaja de la memoria *cache* en el procesamiento de los protocolos y el uso de la memoria local de la NIC dependerá de la arquitectura interna de esta y de las características de la memoria local. En nuestra implementación, la memoria *cache* conectada al procesador principal del nodo es mucho más rápida que la memoria local de la NIC, cuyas características son similares a las de la memoria principal del nodo. En términos del modelo LAWS modificado, esta situación se corresponde con un valor del parámetro δ_a menor que el correspondiente cuando no existe *cache* (como si hubiera menor carga debida a la aplicación). Al final de este capítulo se muestran los resultados de experimentos realizados en los que se ha incluido una *cache* local en la NIC en la externalización mediante *offloading*. Dichos experimentos ponen de manifiesto que la *cache* local en la NIC hace que se obtenga una menor latencia que en el caso de la externalización mediante *onloading* y un ancho de banda similar.

En el caso de la externalización mediante *onloading*, los mensajes de tamaño pequeño (alrededor de 2 Kbytes) pueden ser alojados en la memoria *cache* de nivel 1. Esto también ocurre en el sistema sin externalización. Por tanto, en el caso de que la carga generada por la aplicación sea baja (aproximadamente, $\gamma < 0.5$), la sobrecarga generada en el sistema con externalización mediante *onloading* cuando este se encuentra sometido a una avalancha de mensajes de tamaño pequeño, estará principalmente provocada por las llamadas interprocesador (*cross-call*) y la interacción con el sistema operativo. Estas llamadas interprocesador necesitan del bus del sistema, con lo que las prestaciones en este caso son inferiores respecto a las prestaciones obtenidas para mensajes de mayor tamaño. En el sistema sin externalización, los mensajes pueden alojarse en la memoria *cache* de nivel 1, y no es necesario un uso intensivo del bus del sistema para la sincronización con el sistema operativo. En [MOG03] se hace referencia también a este efecto.

Si se incrementa el tamaño del mensaje hasta aproximadamente 100 Kbytes, de forma que estos mensajes ya no pueden alojarse totalmente en la memoria *cache* de nivel 1, aunque sí en la memoria *cache* de nivel 2. Esto supone bajar un nivel en la jerarquía de memoria con la correspondiente perdida de ciclos (penalización por fallo en el nivel 1 de *cache*). Sin embargo, en este caso, por un lado se tiene que el tamaño de los mensajes hace que la rutina *memcpy* (la cual implementa transferencias *byte* a *byte* y está optimizada para transferencias de bloques de tamaño pequeño) proporcione el ancho de banda máximo (aproximadamente 40 Mbps en nuestra implementación) en las

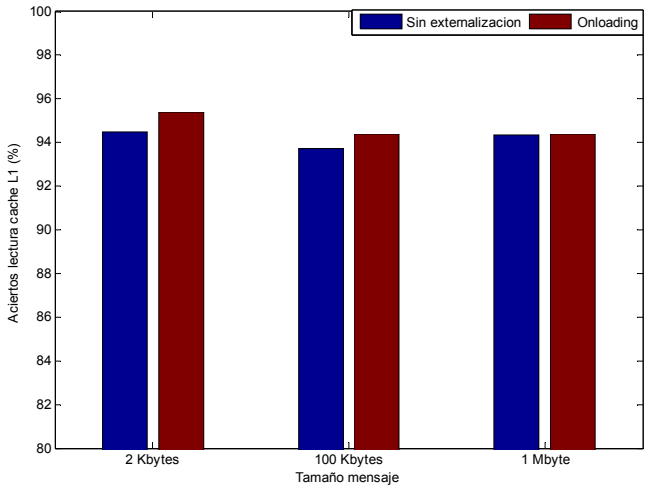
copias de datos de memoria [TUR03] y por otro lado, la sobrecarga generada tanto en la CPU como en el bus del sistema por mensajes de alrededor de 100 Kbytes es mucho menor que en el caso de mensajes de alrededor de 2 Kbytes. Así, este mejor aprovechamiento de los buses del sistema y de la rutina de copia de datos (usualmente optimizadas para transferencias de bloques de memoria entorno a 64 Kbytes) se traduce en una mejora de las prestaciones respecto del sistema sin memoria *cache*, tal y como se observa experimentalmente en la Figura 4.36.

Si se vuelve a incrementar el tamaño del mensaje, este tampoco puede alojarse por completo en la *cache* de nivel 2 del sistema sin externalización (con un tamaño de 512 Kbytes en los experimentos). Así, en la Figura 4.33 puede verse, que un incremento del tamaño del mensaje no produce una mejora en las prestaciones a pesar de que dicho incremento reduce la sobrecarga asociada a los procesos de comunicación. En el sistema con externalización mediante *onloading*, podemos observar (Figura 4.33) que al acercarse el tamaño del mensaje al de la memoria *cache* de nivel 2, las prestaciones comienzan a disminuir. Como el sistema se encuentra bajo una condiciones de baja carga debida a la aplicación (parámetro γ bajo en el modelo LAWS), no se obtienen las ventajas de la externalización de protocolos de acuerdo a lo que se indica en [SHI03].

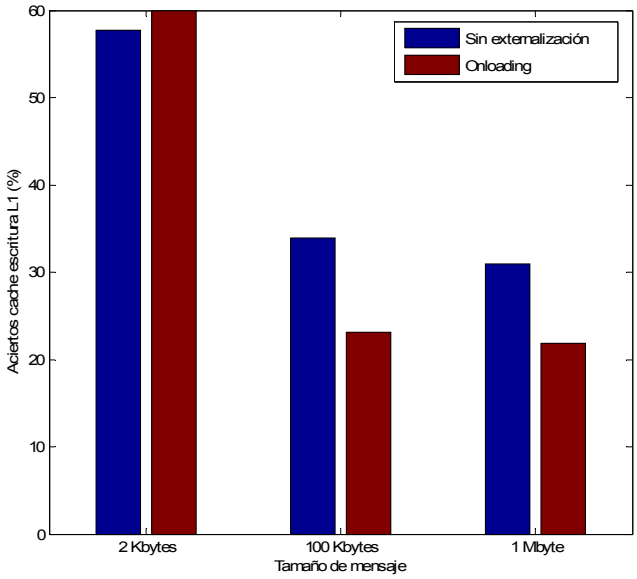
En la Tabla 4.18 y en la Figura 4.34, se muestra la tasa de aciertos en lectura y en escritura de la memoria *cache* L1 del procesador encargado de los procesos de comunicaciones. En dicha figura puede verse como la tasa de aciertos en escritura es relativamente baja. Este efecto se debe a que cuando un paquete llega al extremo receptor, la probabilidad de que los datos recibidos estén en la memoria *cache* es muy baja [NAH97], lo que explica la baja tasa de aciertos en escritura. Por otro lado, la tasa de aciertos disminuye al incrementar el tamaño del mensaje hasta 100 Kbytes, debido a que mensajes con un tamaño superior a 16 Kbytes no caben en la memoria *cache* de nivel 1. En lectura sin embargo, la tasa de aciertos es superior al 90%, debido a que el procesador está manejando datos que ya se encuentran en memoria principal y por tanto dispondrá de bloques de la misma en su memoria *cache*.

Tabla 4.18. Estadísticas aciertos de memoria cache para las diferentes implementaciones de la externalización (Simics). Cache L1 de 32Kbytes, cache L2 de 512 Kbytes (1 Mbyte en onloading)

	Nivel *cache*	2 Kbytes Lect / Escr (%)	100 Kbytes Lect / Escr (%)	1 Mbyte Lect / Escr (%)
Sin externalización	L1 (datos)	94.47 / 57.69	93.74 / 33.95	94.32 / 30.96
	L2	77.28 / 99.8	70.85 / 99.79	69.81 / 98.59
Offloading	L1 (datos, CPU_0)	99.62 / 99.68	99.23 / 99.41	96.77 / 95.55
	L2	90.61 / 99.99	94.12 / 99.94	86.65 / 99.87
Onloading	L1(datos,CPU 1)	95.38 / 59.96	94.39 / 23.14	94.39 / 21.83
	L1(datos,CPU 0)	95.35 / 94.24	95.68 / 94.87	95.75 / 95.43
	L2	87.98 / 99.97	78.86 / 99.94	72.51 / 98.78



(a)



(b)

Figura 4.34. Estadísticas de aciertos de cache L1(a) en lectura y (b) en escritura

En el caso de la memoria *cache* de nivel 2, la tasa de aciertos es mucho más alta tanto en lectura como en escritura, debido a que la política de actualización de la *cache* de nivel 1 (*Write-through*) hace que se actualicen los bloques en la *cache* L2 ante fallos de escritura.

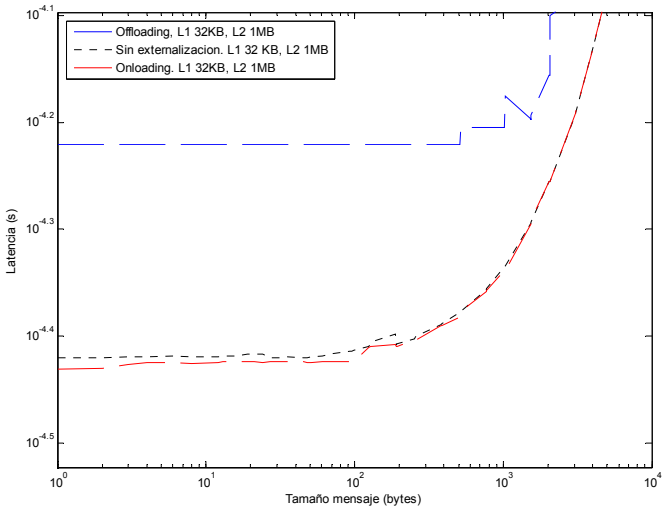


Figura 4.35. Latencia con memoria cache

En la Figura 4.35 se muestra el efecto de la memoria *cache* en la latencia de red, y en la Tabla 4.19 podemos ver los valores de latencia en cada caso.

Tabla 4.19. Valores de latencia con y sin modelado de cache (L1 16 Kbytes, L2 512 Kbytes)

Externalización	**Latencia (μs) sin modelo de *cache***	**Latencia (μs) con *cache***
Sin Externalización	68	40
Offloading	60	60
Onloading	64	36

Como se muestra en la Figura 4.35 y en los valores de la Tabla 4.19, sin modelo de memoria *cache*, la externalización mediante *offloading* proporciona una latencia menor que cuando no se utiliza externalización y que en el caso de externalización con *onloading*. Esto se debe a que a igualdad de condiciones en cuanto a tiempos de acceso a memoria por parte de la CPU principal del nodo, la técnica de *offloading* permite interactuar con la red de una forma más directa ya que la comunicación entre el procesador que realiza el procesamiento de los protocolos y el resto de la interfaz de red no necesita del bus de E/S del sistema. En cambio, al utilizar un modelo preciso de *cache*, la externalización mediante *offloading* proporciona una mayor latencia que en el caso de *onloading,* en el que la utilización de la memoria *cache* permite reducir el uso del bus de memoria. Además, al añadir una memoria *cache* con las características que se muestran en la Figura 4.10 (L1 de 16 Kbytes, L2 de 512 Kbytes), el acceso a la memoria desde la CPU del nodo encargada del procesamiento de los protocolos (CPU_0, en el caso sin externalización y CPU_1en el caso de externalización mediante onloading) puede realizarse en menos ciclos que los que necesita la NIC en la externalización mediante *offloading*. Dicho de otra forma, y desde un análisis basado en el modelo LAWS presentado en la Sección 1.7 y utilizado en la Sección 4.6, al añadir memoria *cache* al procesador principal del nodo se estaría incrementando el valor del parámetro α, dado que al añadir memoria *cache* al sistema con externalización mediante onloading, se mejora el acceso a la memoria desde la CPU que realiza el procesamiento de los protocolos, con la consiguiente mejora de prestaciones en el procesamiento de las tareas con respecto al sistema sin modelo preciso de memoria *cache*.

4.7.1 Efecto del tamaño de la memoria *cache*

En esta sección se analiza el efecto del tamaño de la memoria *cache* de nivel 1 en las prestaciones finales de los sistemas con externalización mediante *offloading* u *onloading*.

En los experimentos realizados en esta sección, se ha considerado la misma arquitectura que la utilizada para obtener los resultados de la sección anterior y los correspondientes modelos de simulación, pero se ha variado el tamaño de la memoria *cache* de nivel 1 para analizar su influencia en las prestaciones del sistema de comunicaciones.

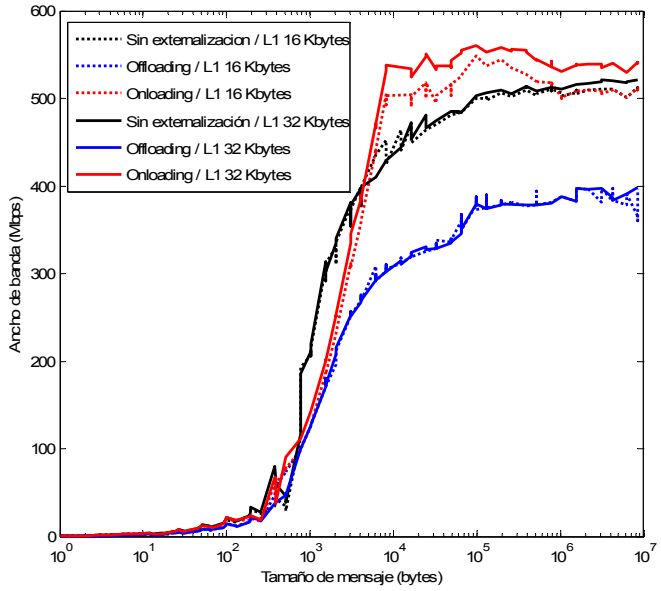
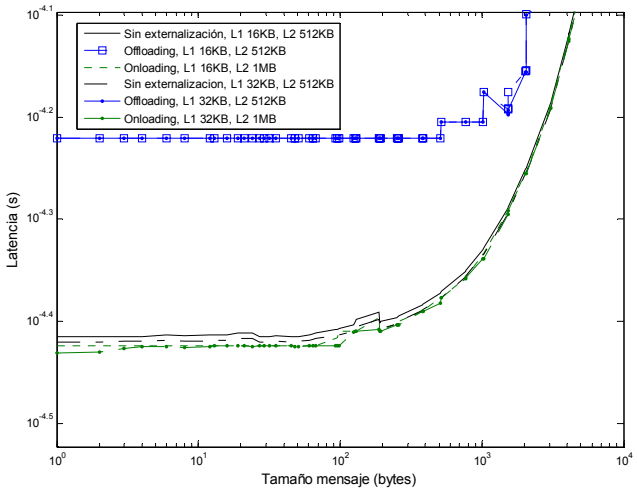


Figura 4.36. Efecto del tamaño de la memoria cache de nivel 1 en el ancho de banda (streaming)

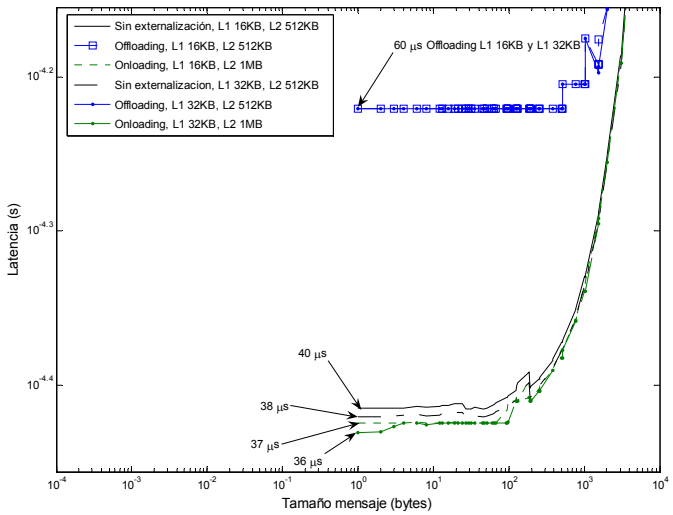
En la Figura 4.36 se puede ver que las prestaciones de la externalizació mediante *onloading* no mejoran sustancialmente al incrementar el tamaño de la memoria *cache* de nivel 1. En la Tabla 4.20, donde se resumen las estadísticas aciertos de la memoria *cache* para 16 Kbytes y 32 Kbytes, se puede ver como el número de aciertos de *cache* no mejora al incrementar el tamaño de la memoria *cache*. Un incremento en la tasa de aciertos supondría un menor número de ciclos perdidos debido a la penalización por fallos y por tanto una mejora en las prestaciones.

Tabla 4.20. Resumen de estadísticas (promedio) de cache para los modelos de simulación

Externalización	Nivel *cache*	L1 16 Kbytes Lect / Escr (%)	L1 32 Kbytes Lect / Escr (%)
Sin Externalización	L1	94.41 / 40.41	95.81 / 39.48
	L2	74.65 / 98.78	65.47 / 98.78
Offloading	L1	96.76 / 93.79	98.17 / 94.03
	L2	88.74 / 99.82	75.95 / 99.79
Onloading	L1(CPU 1)	94.99 / 34.67	96.13 / 39.15
	L1(CPU 0)	95.65 / 94.64	97.89 / 95.97
	L2	78.25 / 99.05	70.08 / 99.15



(a)



(b)

Figura 4.37. Efecto del tamaño de la memoria cache de nivel 1 en la latencia de red. (a) Escala logarítmica. (b) Detalle latencia.

Para tamaños mayores de 32 Kbytes cabría esperar una mejora en las prestaciones, si bien, esta mejora tendría un límite superior impuesto por las propias características de la implementación del protocolo TCP y los buses de E/S.

Algo similar ocurre con la latencia de red, como se muestra en la Figura 4.37 y más detalladamente en la Tabla 4.21 y en la Figura 4.38.

Tabla 4.21. Latencia de red para los sistemas con y sin memoria cache

Externalización	Latencia sin modelo de *cache* (µs)	Latencia con *cache* (µs)	
		L1 16K	L1 32K
Sin Externalización	68	40	38
Offloading	60	60	60
Onloading	64	37	36

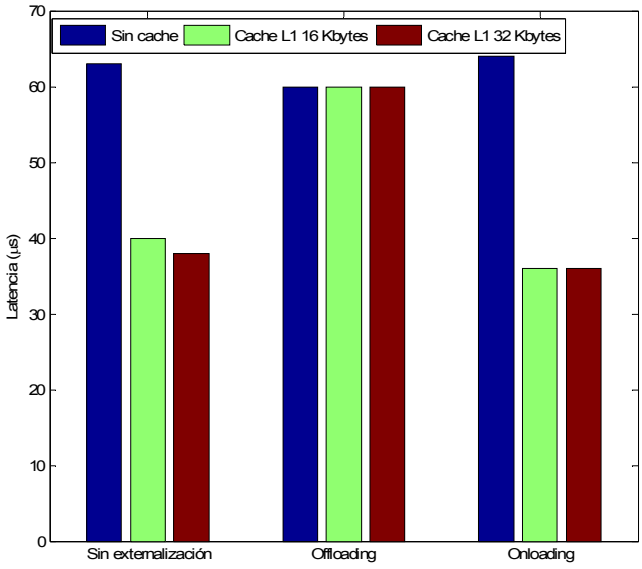


Figura 4.38. Latencia con distintos tamaños de cache

En el caso de la externalización mediante *offloading*, la latencia no mejora al añadir memoria *cache* al sistema, debido a que se está conectando memoria *cache* al procesador principal del nodo pero no a la NIC. Aunque la inclusión de dicha memoria *cache* en la jerarquía de memoria reduce las posibles colisiones en el bus de memoria, dichas colisiones se deberían en su mayoría a los accesos relacionados con el procesamiento de los protocolos y este procesamiento se realiza en la propia NIC en el

caso de *offloading*. Al mismo tiempo, vemos que al tener en cuenta el efecto de la *cache* de forma más precisa en la CPU que realiza el procesamiento de los protocolos las prestaciones mejoran sustancialmente, ya sea sin externalización o con externalización mediante *onloading*. Se puede concluir, entonces, que el modelo preciso de la memoria *cache* en la CPU principal del nodo, no mejora las prestaciones del sistema de comunicaciones en condiciones de alto grado de externalización (es decir, un valor del parámetro $p > 0.5$, o $p \rightarrow 1$ en el modelo LAWS) y baja carga de aplicación (es decir, un valor del parámetro parámetro $\gamma \approx 1$), ya que en este caso la mayor parte del procesamiento se realiza se realiza en una CPU distinta a la CPU principal del nodo. Al mismo tiempo, disponer de un modelo detallado de memoria *cache* en el subsistema de comunicaciones que acelere la jerarquía de memoria, puede producir una mejora sustancial en las prestaciones, dependiendo de la aplicación y de la localidad espacial y temporal de los datos que ésta maneje. Estas conclusiones son análogas a las extraidas de los resultados experimentales obtenidos en la Sección 4.3.5 para la latencia y coinciden con las conclusiones de trabajos como [NAH97] en el que se indica que las prestaciones del sistema de comunicaciones disminuyen drásticamente si las transferencias producen siempre fallos de *cache*, sobre todo en cuanto a la latencia de red. En [NAH97] se hace referencia también a la mejora que puede obtenerse incrementando la asociatividad de la *cache* así como la importancia de la implementación del protocolo TCP. Otros trabajos, como [KIM05] (donde se propone la utilización de una memoria *cache* específica para la interfaz de red), concluyen que dicha memoria *cache* produce una disminución del tráfico en los buses de E/S, dado que la efectividad de una *cache* depende de la localidad espacial y temporal de los datos, disponer de una memoria *cache* en la interfaz de red es espcialmente útil para aplicaciones que realizan transferencias de datos de forma repetitiva. En este sentido, se han realizado experimentos que ponen de manifiesto el efecto de una memoria *cache* en la tarjeta de red. Para realizar estos experimentos se ha modificado el modelo de simulación para la externalizción mediante *offloading* de forma que la CPU de la NIC pueda acceder a su memoria local con menos ciclos de reloj. Para simular este efecto, no se ha incluido una jerarquía de memoria específica en la NIC, sino que se ha modelado un tiempo medio de acceso a la memoria de la NIC de 10 ciclos de reloj. En la Figura 4.39 se muestra el ancho de banda, que proporciona la externalización mediante offloading en este caso, y se pone de manifiesto la mejora en ancho de banda al utilizar una memoria local rápida en la NIC. Del mismo modo, en la Figura 4.40 se

muestra que la latencia que proporciona la externalización mediante *offloading* en este caso es menor que la que proporciona la externalización mediante *onloading*.

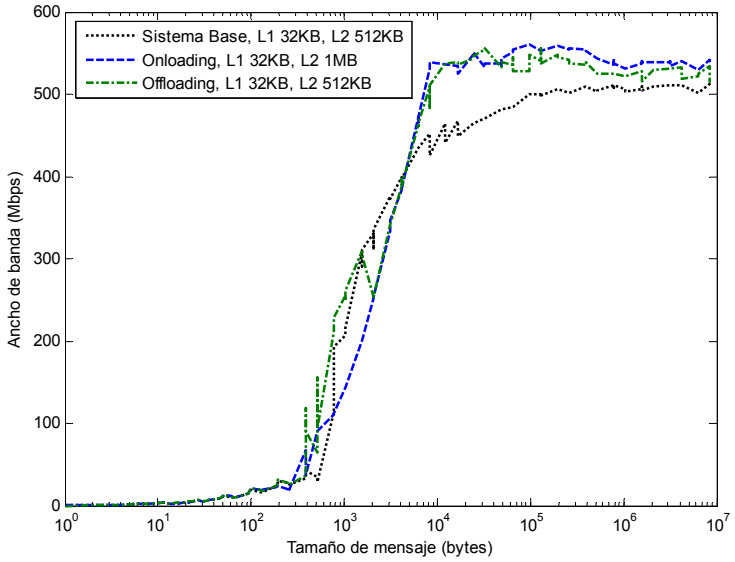


Figura 4.39. Comparación de anchos de banda. Offloading con cache local

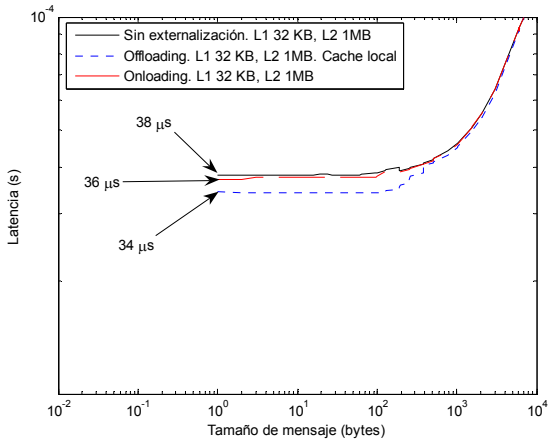


Figura 4.40. Comparación de latencias. Offloading con cache local

Por otro lado, otros trabajos como [REG04a, REG04b] muestran las mejoras en las prestaciones del sistema de comunicaciones al externalizar mediante *onloading* y, por tanto, ponen de manifiesto el efecto beneficioso que tiene utilizar un procesador con memoria *cache* de dos niveles para el procesamiento de los protocolos de comunicación.

4.7.2 Análisis con el modelo LAWS

A continuación se realiza un análisis con el modelo LAWS, de misma forma análoga a como se ha hecho en la Sección 4.6, pero, en este caso, incluyendo un modelo preciso de la jerarquía de *cache* en las simulaciones.

Se han realizado experimentos para ver la influencia del parámetro γ (*Application ratio*) en la mejora proporcionada por la externalización mediante offloading y mediante onloading, en sistemas que incluyen el modelo de memoria *cache* descrito en la Sección 4.7.

En dicha sección se ha puesto de manifiesto que la externalización mediante *offloading* proporciona unas prestaciones peores, incluso que el sistema sin externalización, en el caso de que la carga generada por la aplicación no sea lo suficientemente alta. Sería conveniente realizar un análisis con el modelo LAWS para poner de manifiesto el efecto de la carga generada por la aplicación mediante el parámetro γ. También se realizará el mismo tipo de análisis para el caso de externalización mediante *onloading*.

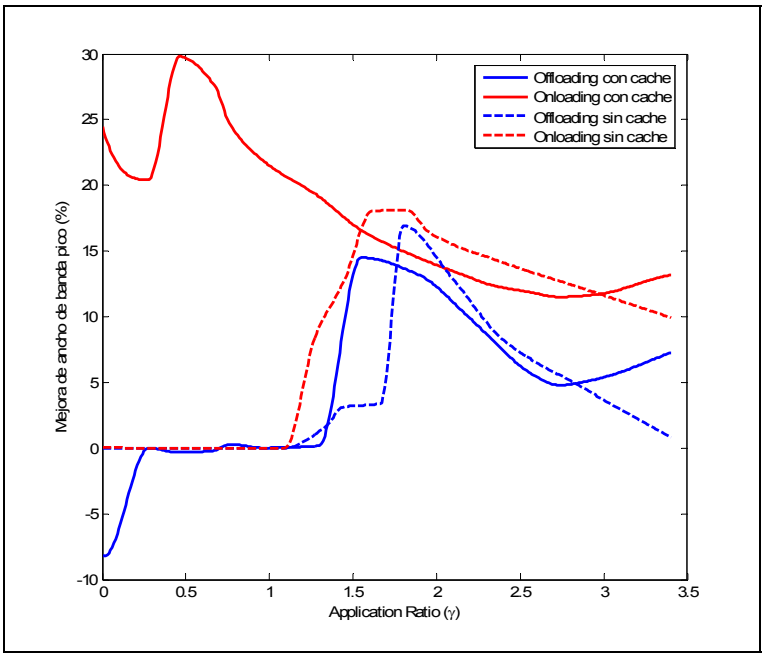


Figura 4.39. Análisis de los resultados experimentales con el modelo LAWS

Como se ha indicado en la Sección 4.6, en los experimentos realizados para el análisis con el modelo LAWS, utilizamos el programa de *benchmark Hpcbench*, con las modificaciones descritas en la Sección 4.6.1.

En la Figura 4.39 se puede ver que la inclusión del modelo de memoria *cache* pone de manifiesto mejoras en el caso del sistema con externalización mediante *offloading* a partir de cierto valor de la carga generada por la aplicación. Este resultado es lógico, si se tiene en cuenta que la CPU que dispone de memoria *cache* es la encargada del procesamiento de la aplicación y, por tanto, disminuye el número de accesos al bus de memoria relacionados con la aplicación así como el número medio de ciclos necesarios para completar un acceso (si hay acierto de *cache*). Además, la mejora en ancho de banda proporcionada por la externalización comienza a crecer para valores del parámetro γ menores que en las simulaciones sin modelo de *cache*. Esto se debe a que la sobrecarga debida a la aplicación es menor en el caso de utilizar memoria *cache* respecto del caso sin *cache* (menor valor del parámetro γ).

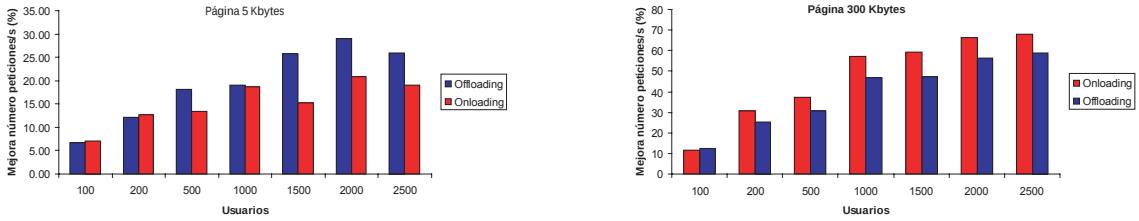
En el caso del sistema con externalización mediante *onloading*, la simulación más precisa de la memoria *cache* tanto en la CPU que realiza el procesamiento de la aplicación como en la CPU encargada del procesamiento de los protocolos de comunicación, pone de manifiesto mejoras en las prestaciones a partir de valores bajos del *factor de aplicación*. Además, la mejora observada en la externalización con *onloading* es mayor que en el sistema sin modelo preciso de memoria *cache*.

El efecto de la latencia de memoria presentado en la Sección 4.3 así como en trabajos tales como [BIN05, IOA05, BIN06], ya hacían pensar en la importancia del efecto de la *cache* en las prestaciones de comunicación. Así se ha puesto de manifiesto a través de nuestras simulaciones con Simics (Figuras 4.5 y 4.7).

4.9 Comparación de las alternativas de externalización para un servidor web

En esta sección se van a presentar las medidas de prestaciones obtenidas al utilizar las propuestas de interfaces de red con externalización mediante offloading y mediante onloading, utilizando una aplicación concreta, con un perfil de comunicación muy concreto de peticiones de tamaño pequeño y respuestas de tamaño variable, como es un servidor web. Para la realización de los experimentos cuyos resultados se muestran en

esta sección se ha utilizado un servidor web *apache 2.0*. Dichos experimentos consisten en la realización de peticiones *http* por parte de un cliente y la respuesta del servidor en forma de página web. De esta forma se ha medido el número de peticiones por segundo que el servidor es capaz de atender, el tiempo medio de respuesta y el ancho de banda, promediando para 100 medidas en cada caso. Además, se han utilizado páginas web de 5Kbytes y 300Kbytes con el objeto de analizar el efecto del tamaño de la respuesta del servidor.



(a) *(b)*

Figura 4.40. Mejora en número de peticiones/s. Página de (a) 5 Kbytes y (b) 300 Kbytes

En la Figura 4.40 se muestra el número de peticiones por segundo atendidas por el servidor para una página de 5Kbytes y de 300 Kbytes. Como puede observarse en dicha Figura, se obtienen mejoras para todas las alternativas de externalización, siendo la interfaz híbrida la que proporciona un mayor valor de la mejora. En el caso de las peticiones por segundo procesadas por el servidor, el mayor valor de la mejora se obtiene para la interfaz híbrida, y esta mejora es mayor para tamaños de página de 300Kbytes que para páginas de 5 Kbytes. Como ya se ha comentado en la Sección 4.7, la mejora proporcionada por la externalización es mayor cuando las prestaciones ofrecidas por el sistema sin externalización son menores.

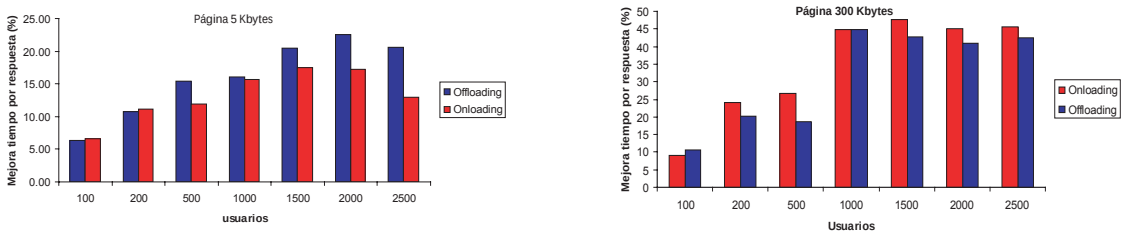


Figura 4.41. Mejora en tiempo por respuesta. Página de (a) 5 Kbytes y (b) de 300 Kbytes

Por otro lado, en el caso de páginas de tamaño pequeño (5Kbytes) la externalización mediante *offloading* es capaz de atender más peticiones por segundo que la externalización mediante *onloading* debido a que en esta aplicación concreta en la

que se realizan peticiones de tamaño pequeño y se reciben mensajes de mayor tamaño que las peticiones y en la que además, el servidor se encuentra en una situación de carga alta (un alto número de peticiones concurrentes deriva en valores altos del *application ratio*, γ) la interfaz de externalización mediante *offloading* no necesita de los buses del sistema para recibir las peticiones. En el caso de la interfaz con *onloading*, puede verse en la Figura 4.40b, como proporciona una mayor mejora para mensajes de 300 Kbytes.

En la Figura 4.41a y 4.41b se muestra la mejora en el tiempo empleado por el servidor para cada petición. Dicha mejora es mayor para la interfaz con *offloading* para el caso de páginas pequeñas y mayor para *onloading* para el caso de páginas de mayor tamaño. No obstante, se observa en las gráficas de las Figuras 4.40 y 4.41 como la diferencia entre la mejora proporcionada por la externalización mediante offloading y la proporcionada por la externalización mediante onloading crece a medida que aumenta el número de usuarios (y por tanto el factor de aplicación, γ).

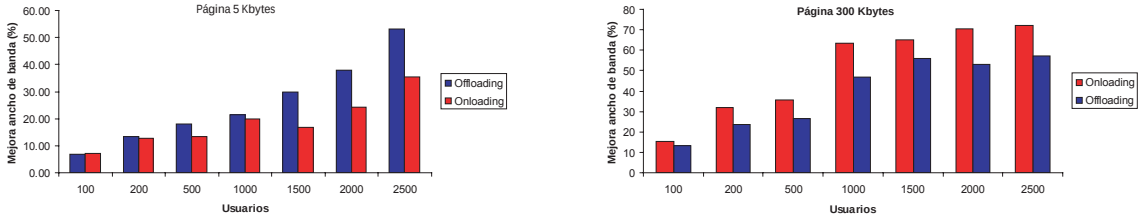


Figura 4.42. Mejora en ancho de banda. Página de (a) 5 Kbytes y (b) de 300 Kbytes.

Por otro lado, el mayor valor de majora para el ancho de banda lo proporciona la interfaz con externalización mediante offloading para páginas de tamaño pequeño (5 Kbytes), y la interfaz con externalización mediante onloading para páginas de mayor tamaño (300 Kbytes). Además, la diferencia en la mejora de ancho de banda proporcionada por ambas técnicas crece al crecer el número de usuarios en el caso de páginas de tamaño pequeño mientras esta diferencia es aproximadamente constante para páginas de mayor tamaño.

Aunque estos experimentos corresponden a un perfil de comunicación concreto, los resultados obtenidos pueden explicarse teniendo en cuenta lo ya comentado en este capítulo, donde se ha visto que la interfaz con *offloading* proporciona un mejor valor de latencia mientras que la interfaz con *onloading* proporciona un mejor valor de ancho de banda y que además, se comporta mejor cuando la carga del procesador principal del nodo es alta (valores altos del parámetro γ en el modelo LAWS).

4.10 Tiempos de simulación

En los experimentos realizados con el simulador Simics, los tiempos de simulación varían considerablemente dependiendo del tipo de experimento realizado y de los parámetros concretos del modelo de temporización. Los tiempos de simulación medios aproximados para los experimentos realizados en este capítulo, en los que se han utilizado los parámetros descritos en la Sección 4.3.3, se muestran en la Tabla 4.22.

Como puede verse en la Tabla 4.22, los tiempos de simulación varían considerablemente dependiendo del modelo de simulación y del tipo de experimento. En general, un incremento del tamaño de mensaje en los experimentos produce un incremento considerable del tiempo de simulación. La complejidad de los modelos de tiempo así como la simulación de latencias altas en los buses de E/S o en los accesos a memoria, también incrementan el tiempo de simulación considerablemente.

Además, como se ha comentado en la Sección 2.6.1, se ha desactivado el *STC* de Simics y se ha configurado el simulador de forma que la conmutación entre CPU para la ejecución de instrucciones se haga ciclo a ciclo, lo que se consiguiendo así una situación más realista. Sin embargo, esto hace que el tiempo de simulación sea aún más elevado. Por tanto, sería útil que el simulador pudiese aprovechar el paralelismo proporcionado por plataformas de altas prestaciones como clusters de computadores o servidores con varios procesadores.

Tabla 4.22. Tiempos medios de simulación para los diferentes experimentos realizados

Experimento	**Benchmark**	**Tiempo sin externalización**	**Tiempo offloading**	**Tiempo onloading**
Medida de ancho de banda. Streaming con tamaño máximo de mensaje de 2 Mbytes. Sin *cache*	*Netpipe*	2 horas	5 horas	6 horas
Medida de latencia. Ping-pong tcp con tamaño máximo de mensaje de 2 Mbytes. Sin *cache*	*Netpipe*	2.5 horas	8.5 horas	10 horas
Simulaciones modelo LAWS con mensajes de 2 Mbits. Barrido del parámetro γ [0..3.5]. Sin *cache*	*Hpcbench*	5 horas	8 horas	8.5 horas
Simulaciones modelo LAWS con mensajes de 8 Kbits. Barrido del parámetro γ [0..3.5]. Sin *cache*	*Hpcbench*	3.5 horas	6 horas	6.5 horas
Medida de ancho de banda. Streaming con tamaño máximo de mensaje de 2 Mbytes. Con *cache*	*Netpipe*	3 horas	7 horas	8 horas
Medida de latencia. Ping-pong tcp con tamaño máximo de mensaje de 2 Mbytes. Con *cache*	*Netpipe*	3.5 horas	10 horas	12 horas
Simulaciones modelo LAWS con mensajes de 2 Mbits. Barrido del parámetro γ [0..3.5] . Con *cache*	*Hpcbench*	6 horas	9 horas	10.5 horas
Simulaciones modelo LAWS con mensajes de 8 Kbits. Barrido del parámetro γ [0..3.5]. Con *cache*	*Hpcbench*	4 horas	7.5 horas	8.5 horas

4.11 Resumen

En el Capítulo 3 se mostraron las diferentes alternativas de implementación de la externalización de protocolos, cuyas ventajas suelen venir asociadas a la liberación de ciclos de CPU utilizables por la aplicación y a la reducción del tráfico en los buses de E/S. En dicho capítulo, se trató la externalización de protocolos de forma general, centrándose después en dos implementaciones posibles para la externalización de protocolos: *offloading y onloading*. En este capítulo, se muestran los resultados experimentales obtenidos con los modelos de Simics que hemos desarrollado y se describen en la Sección 3.5 del Capítulo 3, realizando además una comparación de las prestaciones ofrecidas por ambas técnicas.

Además, se ha utilizado el modelo teórico LAWS para realizar este análisis tomando en consideración diferentes parámetros del sistema, como por ejemplo la carga generada por la aplicación en la CPU. Para conseguir un mejor ajuste de los resultados que predice el modelo LAWS se propone un modelo LAWS modificado, que incorpora otros parámetros del sistema así como desviaciones posibles sobre los parámetros originales debidos al carácter real de nuestro sistema, en el que se está ejecutando un sistema operativo y una aplicación.

Finalmente, se realizan experimentos que incorporan de manera detallada efecto que tiene la memoria *cache* de los procesadores en las prestaciones finales proporcionadas por la externalización de protocolos, con el fin de proporcionar reaultados más realistas. Dichos experimentos ponen de manifiesto la importancia del tipo de aplicación en las posibles mejoras que puede aportar la memoria *cache* a la externalización de protocolos, corroborando los resultados mostrados en trabajos como [NAH97, KIM05] donde se estudia el impacto de la memoria *cache* en un sistema de comunicaciones que utiliza el protocolo TCP.

Además, se han realizado experimentos equivalentes a los hechos para el caso de sistemas sin modelo preciso de memoria *cache* para analizar el efecto del *factor de aplicación,* γ en la mejora proporcionada por la externalización. En este caso se concluye que tanto en el caso de *offloading* como con *onlaoding*, la externalización comieza a producir mejoras para valores más bajos del *factor de aplicación*. Además, la mejora en el ancho de banda pico proporcionado por la externalización mediante *onloading* en un sistema SMP con *cache* en los procesadores es superior a la

proporcionada por *offloading* donde solo dispone de memoria *cache* el procesador principal del nodo.

Finalmente, se ha utilizado una aplicación real con un perfil de comunicación concreto como es un servidor web para determinar la mejora en las prestaciones que pueden obtenerse con las interfaces de red propuestas.

Capítulo 5

Esquema híbrido de externalización

En el capítulo anterior se han evaluado y comparado las prestaciones ofrecidas por dos implementaciones diferentes de la interfaz de red correspondientes a las alternativas de externalización mediante *offloading* y *onloading*. Para ello se han utilizado los modelos de simulación presentados en el Capítulo 3. Como se ha comentado en la Sección 4.10, los resultados obtenidos tras la evaluación de las dos implementaciones de externalización indicadas ponen de manifiesto que se consigue mejoras al utilizar una interfaz de red que incorpore externalización mediante la técnica de offloading cuando la aplicación requiera una latencia de red pequeña, y la técnica de onloading cuando se necesite alcanzar un ancho de banda mayor. Por otro lado, en la Sección 4.7.1, en la que se evalúan interfaces de red con externalización desde el punto de vista del modelo LAWS (Sección 1.7.2), se pone de manifiesto que la externalización se comporta de forma diferente ante diferentes valores de la sobrecarga generada por la aplicación.

Por lo tanto, como el comportamiento relativo de las alternativas de externalización depende de la situación concreta (sobrecarga de comunicación, pila de protocolos, sobrecarga de la aplicación, etc.), sería interesante desarrollar una interfaz de red que incorpore las ventajas de la externalización que ofrece *offloading* y las que ofrece *onloading,* y evaluar las prestaciones que proporciona.

En este capítulo se describe la interfaz de red híbrida que hemos diseñado para aprovechar la arquitectura SMP y proporcionar unas prestaciones superiores a las de las implementaciones de la externalización mediante *offloading* y mediante *onloading* que se presentaron en el Capítulo 4.

5.1 Alternativas para la reducción de la sobrecarga de comunicación

Una alternativa para la reducción de la sobrecarga asociada a los procesos de comunicación, consistiría en desarrollar una pila TCP/IP optimizada, e integrable en Linux en modo usuario [DIN02, PRA04]. De esta forma, podría llevarse el procesamiento de los protocolos a otro procesador del sistema ubicado en la tarjeta de red, CPU_{NIC}, diferente al que ejecuta el sistema operativo para evitar las llamadas al sistema operativo y los correspondientes cambios de contexto. Además, si se hacen llegar las interrupciones a ese mismo procesador, (para que la CPU_0 no tenga que consumir ciclos en atender las interrupciones) se conseguiría una mayor autonomía de la interfaz de red, implementada en el conjunto NIC/procesador, ya que se reduciría aún más la intervención del sistema operativo. Se trataría en este caso, de una implementación en la que se utiliza la técnica de externalización mediante *onloading* y protocolos de comunicación a nivel de usuario [PAK97, PRY98, DUN08]. En cualquier caso, no se evitaría uno de los cuellos de botella más importantes que, como ya se ha comentado en la Sección 1.7, se ocasiona por las copias de datos.

No obstante, una mejora respecto a las implementaciones comerciales actuales de la externalización mediante *onloading*, podría consistir en no utilizar un procesador del sistema de forma exclusiva para procesamiento de los protocolos. En cambio, se podrían implementar mecanismos de DMA mejorados, así como una pila TCP/IP optimizada para reducir la sobrecarga, al mismo tiempo que un mecanismo de reasignación de recursos. De esta forma, cuando los procesos de comunicaciones lo requieran, se podrán utilizar más recursos del procesador, mientras que estos mismos recursos se podrían dedicar para la ejecución de la aplicación cuando no se estén utilizando para las comunicaciones. Por otro lado, para hacer más eficientes las comunicaciones, se podría extender el mecanismo de configuración dinámica de parámetros TCP que actualmente está parcialmente implementado en la pila TCP de Linux [RAV04]. Esta implementación requiere el uso de mecanismos de gestión de recursos de procesador, tales como la CPUSET, ya comentado en la Sección 3.6 y, al mismo tiempo, modificaciones en el *planificador* del *núcleo* de Linux que permitan la reasignación dinámica de recursos [COL05], mientras los procesos de comunicaciones conservan una mayor prioridad. En este sentido, en [NAR07] se analiza la interacción

entre la pila de protocolos TCP/IP y los procesadores presentes en un entorno multinúcleo, argumentando que, en entornos en los que hay más de un núcleo de procesamiento disponible, la gestión tradicional de la pila de protocolos TCP/IP hace que la mayor parte del procesamiento de dicha pila se asigne a la misma CPU o núcleo de procesamiento. Este tipo de gestión, que se corresponde a un modelo "*cpu-bounded"* [BEN05], provoca un reparto de carga desequilibrado entre los núcleos de procesamiento disponibles. Así, en [NAR07] se realiza una propuesta para la gestión de la pila de protocolos y de los procesos de la aplicación, que desequilibra la carga entre los diferentes núcleos de procesamiento disponibles. Este tipo de técnicas resultan especialmente interesantes en redes con enlaces de gran ancho de banda, como las multi-Gigabit Ethernet, en las que los procesos de comunicaciones generan una sobrecarga considerable.

Por otro lado, y una vez analizadas las características de los modelos de simulación de la externalización mediante *offloading* y *onloading* en las Secciones 3.5 y 3.6 y teniendo en cuenta sus prestaciones relativas (Capítulo 4), tiene sentido plantearse un sistema híbrido *onloading/offloading* que aproveche los beneficios que aportan los NIC programables y los procesadores multinúcleo. No obstante, no existen muchas propuetas en este sentido. Cabe citar el trabajo descrito en [SHA06], donde se propone una interfaz de red que combina las ventajas de la externalización mediante *offloading* y *onloading*, utilizando elementos software y un *motor de aceleración hardware* para optimizar las transferencias de datos, con el fin de aprovechar el ancho de banda de los enlaces multi-Gigabit. En dicho trabajo, se argumenta que los interfaces de red que utilizan externalización mediante *offloading* presentan diferentes inconvenientes ya comentados en el Capítulo 1: (1) las prestaciones de la CPU del nodo en poco tiempo mejorarán las de la CPU presente en la tarjeta de red; (2) el cuello de botella que puede suponer el propio interfaz de red; (3) el diferente firmware necesario, adaptado a una aplicación concreta; (3) la gran complejidad de la API; y (4) las dificultades de mantenimiento. Por tanto, aunque la externalización mediante la técnica de *offloading* proporciona ventajas, es necesario tener en cuenta los inconvenientes citados para evaluar la viabilidad del uso de un interfaz de tipo TOE (*TCP Offloading Engine*). Por otro lado, las alternativas de externalización mediante la técnica de *onloading* presentan el inconveniente de no reducir la sobrecarga en el subsistema de memoria.

La arquitectura de una interfaz híbrida propuesta en [SHA06] se muestra en la Figura 5.1 y constituye la referencia representativa del trabajo previo realizado en esta

línea hasta el momento, y por lo tanto define el contexto en el que situar nuestra propuesta de interfaz híbrida. Como se puede ver, la interfaz híbrida de [SHA06] incluye básicamente tres elementos. Un elemento hardware, dedicado a controlar las transferencias de datos y cuyo principal cometido es interactuar con la capa de acceso al medio (MAC) de forma que acepte todos los datos que provienen de la red, independientemente de las aplicaciones y de la carga del nodo. Dicho componente hardware recibe el nombre de *streamer* en [SHA06] y sirve a su vez de interfaz entre los *buffers* utilizados por la pila de protocolos y las aplicaciones. Otro elemento utilizado en la implementación descrita en [SHA06] es el llamado TCE (*TCP Control Engine*) que se encarga del procesamiento de la pila TCP. La interfaz entre el *streamer* (y por tanto, con la pila de protocolos) y las aplicaciones es la llamada interfaz de consumidor (*Consumer interface).*Dicha interfaz no utiliza recursos de la CPU principal del nodo (parte que se externaliza mediante *offloading)* así como elementos software que controlan la pila de protocolos y que se ejecutarían en la CPU principal del nodo (parte que se externaliza mediante *onloading*). Este modelo de externalización, que divide la carga de trabajo asociada a los procesos de comunicaciones entre un hardware específico y la CPU principal del nodo, consigue evitar los problemas de interacción con la aplicación y de mantenimiento existentes en la externalización mediante *offloading* y los problemas de compartición de recursos de E/S que supone *onloading*. Esto es gracias a que solamente se externaliza mendiante *offloading* aquellos componentes que realizan el movimiento de los datos y mediante onloading aquellos componentes que realizan el procesamiento de la pila de protocolos. De esta forma, se aprovecha la parte más eficiente de la externalización mediante *offloading* y la parte más eficiente de la externalización mediante *onloading*.

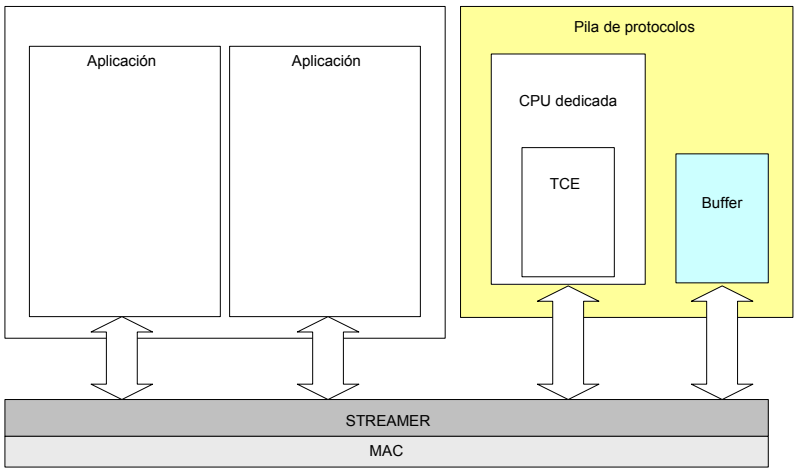


Figura 5.1. Arquitectura híbrida de externalización [SHA06]

5.2 Modelos de externalización híbridos

En esta sección se describe nuestra propuesta para la externalización de la interfaz de red que pretende aprovechar las ventajas de las técnicas de externalización mediante *offloading* y *onloading,* tratando, además, de mejorar las prestaciones proporcionadas por cada una de esas técnicas, y reduciendo la sobrecarga asociada a los procesos de comunicación en la CPU que ejecuta la aplicación. Se dispone así de un modelo *híbrido* de externalización que aprovecha tanto los recursos de las tarjetas de red que incorporan procesadores y memoria, como alguno de los procesadores incluidos en los sistemas multiprocesador (SMP), o en las nuevas arquitecturas de microprocesador que permiten disponer de varios núcleos en un mismo chip (CMP). Igual que en los casos anteriores, se ha utilizado el simulador de sistema completo Simics para implementar el modelo de simulación y para evaluar y analizar las prestaciones ofrecidas (Capítulo 4).

5.2.1 Modelo híbrido de externalización *offloading/onloading*

En las Secciones 3.6 y 3.7 se han descrito las características y los modelos de simulación utilizados para la las propuestas de *offloading* y de *onloading* respectivamente. El modelo de externalización mediante *offloading* utiliza una tarjeta de red que incluye un procesador y su propia memoria. En dicho modelo (Figura 3.10), los paquetes recibidos se transfieren a un buffer en anillo implementado en la tarjeta de red, donde la CPU_{NIC} podrá procesarlos. Además, la interrupción hardware generada cuando

hay paquetes listos para que la CPU_{NIC} pueda procesarlos, llega a la CPU_1, que es la encargada de ejecutar el *driver*. Una vez que los paquetes han sido procesados, la CPU_{NIC} copia los datos al *socket* de recepción, y la CPU que ejecuta el sistema operativo (CPU_0) copia los datos al *buffer* de la aplicación.

En el modelo de externalización mediante *onloading*, cuando se recibe un paquete se genera una interrupción hardware en la CPU_1. Dicho procesador ejecuta el *driver* de la tarjeta de red. Además, cada paquete recibido es copiado a la memoria principal mediante DMA, estando el buffer en anillo implementado en dicha memoria principal, donde la CPU_1 puede procesar la pila de protocolos TCP/IP. Una vez procesados los paquetes, la CPU_1 copia los datos al *socket* de donde los recoge la CPU_0 para copiarlos a los *buffers* de la aplicación.

Del análisis del funcionamiento de las dos alternativas anteriores se puede deducir que hay cuatro aspectos fundamentales que afectan a las prestaciones de la interfaz de red:

1) La atención de la interrupción hardware
2) La ejecución del *driver*
3) El procesamiento de la pila TCP/IP
4) La copia de los datos a los *buffers* de la aplicación

En la Tabla 3.2 se muestra la distribución de los elementos anteriores en los diferentes procesadores presentes en los modelos de externalización mediante *offloading*, *onloading* y en la interfaz *híbrida* que proponemos. En dicha tabla se puede ver como en el caso de la externalización con *onloading* y con *offloading* se hace uso de la CPU_0, que es el procesador que ejecuta la aplicación, y realiza las copias de datos a los *buffers* de dicha aplicación. Como se mostró en la Sección 1.4, las copias de datos necesarias para la transferencia de un paquete, introducen una sobrecarga importante en el sistema de comunicaciones. En la Tabla 3.2, se resumen las copias de datos necesarias para transferir un paquete entre dos extremos *A* y *B* con la implementación actual de la pila de protocolos TCP/IP en Linux. Como se muestra en dicha tabla, son necesarias dos copias de los datos en cada extremo, copias en las que, con las propuestas de externalización *offloading* y *onloading*, al menos en una de ellas, es precisa la intervención de la CPU que ejecuta la aplicación.

En ambos casos esto se debe a que no se permite la ejecución de tareas del sistema operativo en la CPU encargada al procesamiento de las comunicaciones, fuera del contexto de interrupción (como se ha comentado en las Secciones 3.5, 3.6 y 3.7).

Tabla 5.1. Distribución de procesos para las diferentes alternativas de externalización

Proceso	Externalización		
	Offloading	*Onloading*	*Híbrida*
Atención Interrupción hardware	CPU_0	CPU_1	CPU_1
Ejecución *driver*	CPU_0	CPU_1	CPU_1
Procesamiento pila TCP/IP	CPU_{NIC}	CPU_1	CPU_{NIC}
Copia a los *buffers* de la aplicación	CPU_0	CPU_0	CPU_1

Tabla 5.2. Copias de datos necesarias para realizar una transferencia entre los extremos A y B

Extremo	**Copia**
A (emisor)	① Espacio de usuario a espacio del núcleo del SO
	② Espacio de kernel a la NIC
B (receptor)	③ NIC al espacio del núcleo del SO
	④ Espacio del núcleo del SO a espacio de usuario

En el modelo híbrido que proponemos, se utilizan dos procesadores para el procesamiento de las comunicaciones aunque sólo uno de ellos (CPU_{NIC}) está dedicado en exclusiva al procesamiento de los protocolos de comunicación. El procesador CPU_1 se encarga de la ejecución del *driver* de la tarjeta de red y será quien reciba las interrupciones. Sin embargo, CPU_1 no se dedica en exclusiva a la ejecución de los procesos de comunicación y por tanto, puede realizar (y realiza) la copia de los datos a los *buffers* de la aplicación, esta realiza en el *contexto de procesos* (Apéndice III). En definitiva, se proponemos una alternativa para explotar el paralelismo que ofrecen los sistemas con más de un procesador (SMP ó CMP) y las tarjetas de red programables, distribuyendo las tareas relacionadas con los procesos de comunicaciones mediante una política *"processor per task"* [BAT06]. En [SHA06] se realiza una propuesta también basada en una interfaz híbrida pero con importantes diferencias respecto a la descrita en esta sección. Así, la propuesta de [SHA06] está basada en el aislamiento de las funciones que realizan las transferencias de datos necesarias en el proceso de

comunicación y en la comunicación asíncrona entre los elementos que implementan dichas funciones, mientras que la nuestra aprovecha el paralelismo presente en un sistema que cuenta con más de un núcleo de procesamiento (SMP ó CMP) y en el que además hay una interfaz de red con un procesador.

Por otro lado, en nuestra implementación híbrida se ha utilizado una tarjeta de red que cuenta, al igual que en el caso de la externalización mediante *offloading,* con memoria para almacenar paquetes (en un múmero determinado por su longitud.) También se pueden utilizan tramas jumbo, con lo que se reduce el número de interrupciones hardware a la CPU que ejecuta el *driver* (CPU_1). Esta implementación tiene además otra ventaja. Como se ha dicho anteriormente, la CPU_1 puede utilizarse para la ejecución de otras tareas del sistema operativo (diferentes de las de comunicaciones) o incluso de determinados procesos de aplicaciones, siempre que se mantenga la prioridad de los procesos de comunicación, es decir, de la gestión de las interrupciones hardware generadas por la NIC. Esto se puede conseguir enviando a la CPU_1 sólo interrupciones hardware procedentes de la NIC.

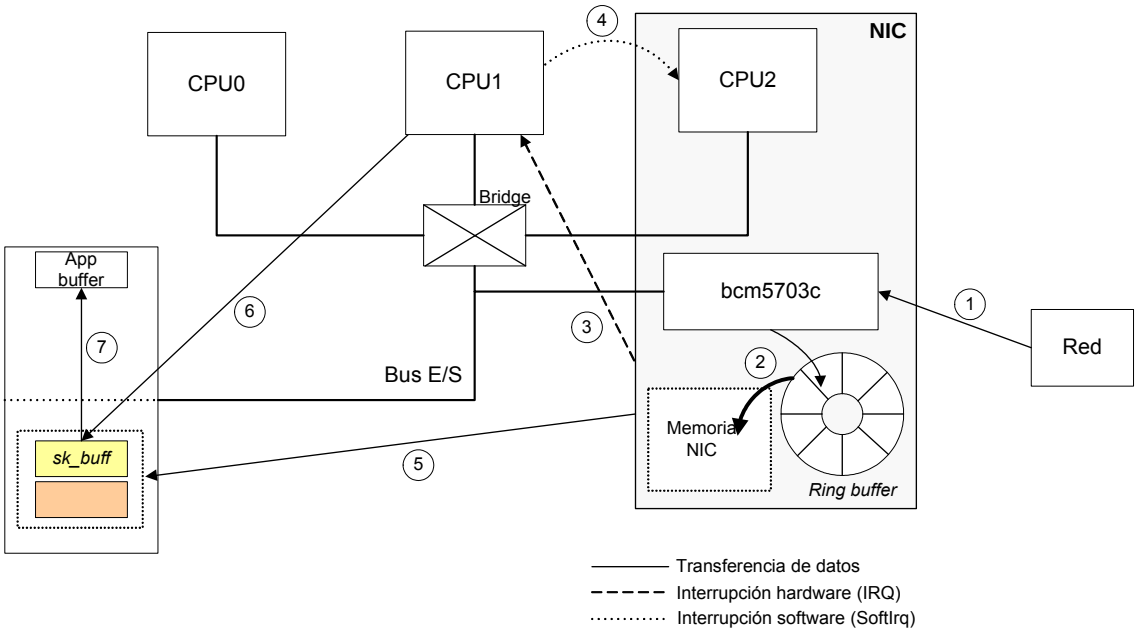


Figura 5.1. Funcionamiento del modelo híbrido de externalización

En la Figura 5.1 se muestra el funcionamiento de la interfaz de red híbrida propuesta. Cuando llega un paquete a la NIC ① se almacena en el buffer en anillo de la tarjeta de red (*ring buffer*) como se ha comentado en la Sección 3.6. Cuando dicho *buffer* se llena o cuando se han terminado de transferir los paquetes, su contenido se

transfiere a la memoria de la NIC ②. Una vez finalizada dicha transferencia, la NIC envía una interrupción hardware (IRQ) a la CPU_1 ③, la cual, ejecuta la función *do_irq*. Dicha función, invoca el gestor de interrupción adecuado, definido dentro del *driver* del dispositivo. De esta forma, dicho gestor de interrupción copia los datos a estructuras *sk_buff* . Después, el *driver* inicia una *softIrq* en la CPU_{NIC} ④, que procesará los paquetes de acuerdo a los parámetros de las estructuras *sk_buff* (parámetros que indican, por ejemplo, el protocolo de capas superiores, IP en nuestro caso). De esta forma, la *softIrq* ejecuta las funciones necesarias para el procesamiento de la pila TCP/IP. Una vez finalizado dicho procesamiento, la CPU_1 (que inició la *softIrq*) copia los datos al *socket* TCP ⑤ y posteriormente a los buffers de la aplicación ⑥ y ⑦, que puede tomar los datos de dichos buffers.

En el esquema anterior, se puede ver que no se envían interrupciones al procesador que ejecuta la aplicación (CPU_0). Además, la tarjeta de red dispone de su propia memoria por lo que se transfiere todo su contenido en una única transferencia de DMA. De esta forma, cuando se informa a la CPU_1 mediante la correspondiente interrupción, hay paquetes para ser procesados. La cantidad de paquetes preparados está determinada por la cantidad de memoria de la NIC. Por otro lado, el inicio de las *softIrq* no afecta al procesador que ejecuta la aplicación (CPU_0).

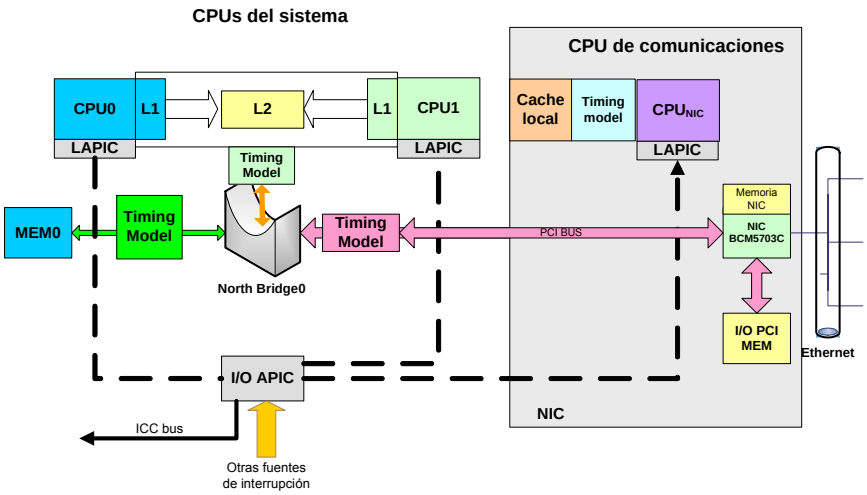


Figura 5.2. Modelo de simulación para externalización híbrida

En la Figura 5.2 se muestra el modelo de simulación utilizado para la propuesta de externalización híbrida *offloading/onloading*. Dicho modelo incluye dos procesadores (CPU_0 y CPU_1) memoria *cache* L1 y una memoria *cache* L2 compartida. Además, un tercer procesador (CPU_{NIC}) que tambíen dispone de su propia memoria

cache. Se han incluido modelos de temporización (descritos en el Capítulo 2), de forma que se pueden configurar los parámetros de latencia para los procesadores CPU_0, CPU_1 y CPU_{NIC}, así como el bus PCI. Además se ha incluido de un controlador APIC de E/S y el correspondiente bus APIC.

Utilizando el modelo de la Figura 5.2 se han realizado experimentos mediante la misma metodología seguida con los modelos propuestos para la externalización mediante *offloading* y *onloading.* En el Capítulo 4 se describen y analizan los resultados para esas alternativas. Ahora pasamos a evaluar la nueva interfaz de red propuesta con el modelo de simulación correspondiente.

5.3 Evaluación del modelo híbrido de externalización

En esta sección se presentan los resultados experimentales de la interfaz de red híbrida propuesta en la Sección 5.2.1. Como se ha comentado en dicha sección, con esta propuesta de externalización híbrida se pretende aprovechar las ventajas de las dos implementaciones de externalización que hemos propuesto, basadas en las técnicas de *offloading* y de *onloading.*

5.3.1 Ancho de banda y latencia de red

En la Figura 5.3 se compara el ancho de banda proporcionado por la alternativa de externalización híbrida, con las otras alternativas de externalización presentadas en el Capítulo 3. Para esta comparación se ha considerado el caso (configuración de la memoria *cache*, etc.) en que las alternativas de *offloading* y *onloading* proporcionan un ancho de banda mayor. Como puede verse en la Figura 5.3, el ancho de banda proporcionado por la alternativa híbrida mejora hasta en un 20% aproximadamente al ancho de banda proporcionado por la externalización mediante *onloading* (la mejor de las restantes opciones). Esto se debe a que, según se ha descrito en la Sección 5.2.1, la alternativa híbrida combina el uso de más recursos de CPU para el procesamiento de los protocolos con la reducción de las transferencias a través de los buses de E/S. El procesamiento de los protocolos se está realizando en una CPU que está aislada del resto, como ocurre en el caso de la externalización *offloading*. Además, es evidente que

la ejecución del *driver* o el procesamiento de las interrupciones por recepción de paquetes afecta en mucha menor medida a la CPU que ejecuta la aplicación (CPU_0) que en el caso de la externalización mediante *offloading*, en la que además de realizar el procesamiento de las interrupciones, la CPU_0 tiene que ejecutar el driver de la tarjeta de red.

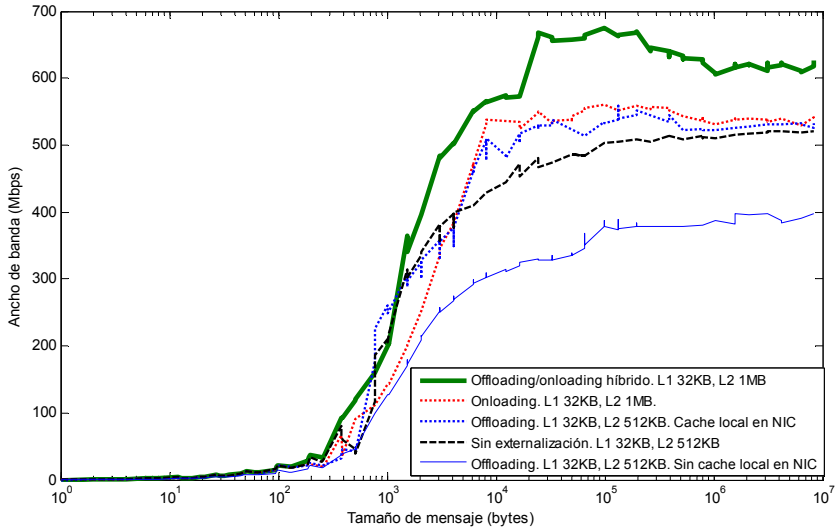


Figura 5.3. Ancho de banda pico para el modelo híbrido de externalización

Por otro lado, en la Figura 5.4 se muestra la latencia de red. Como puede verse en dicha figura, la latencia para el caso híbrido es ligeramente superior al mejor valor de latencia proporcionado por la externalización mediante *offloading*. Esta situación es consecuencia del coste de la sincronización entre las dos CPU del nodo en las que se distribuye el trabajo asociado a la aplicación y el sistema operativo.

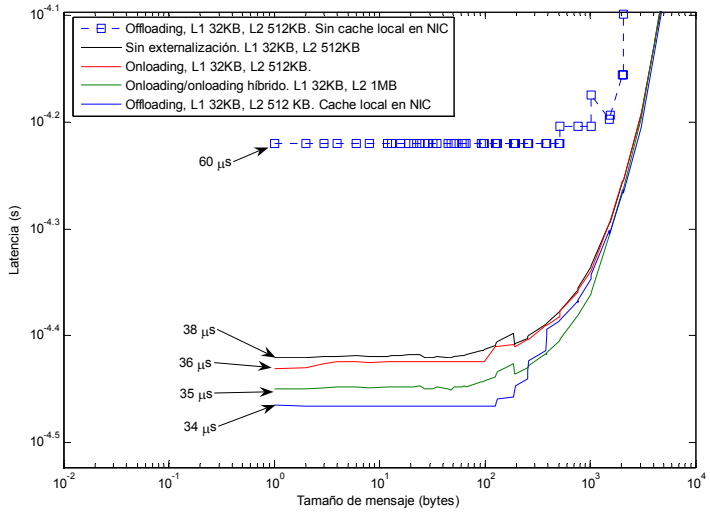


Figura 5.4. Latencia para el modelo híbrido de externalización

Como se ha comentado en la Sección 4.3, en las simulaciones sin modelo detallado de cache se ha tomado una latencia media de acceso a memoria de 10 ciclos. En las simulaciones utilizando un modelo detallado de *cache*, se ha utilizado una latencia de acceso a memoria de 60 ciclos de reloj y el mismo modelo de *cache* que se ha descrito en la Sección 4.7 y cuyas características se resumen en la Tabla 4.17.

5.3.2 Uso de la CPU principal

En las Figuras 5.6 y 5.7 se muestra, respectivamente, el número de interrupciones por segundo procesadas por la CPU principal y el porcentaje de uso de dicha CPU principal (CPU_0) respectivamente, para cada una de las tres técnicas de externalización que se han implementado en este trabajo, y la correspondiente comparación con el sistema base (sin externalización). Dichos resultados se han obtenido utilizando la utilidad *sysmon* del benchmark *Hpcbench* [HUA05], con una carga de trabajo de comunicación intensiva (que trata de utilizar todo el ancho de banda disponible en el enlace).

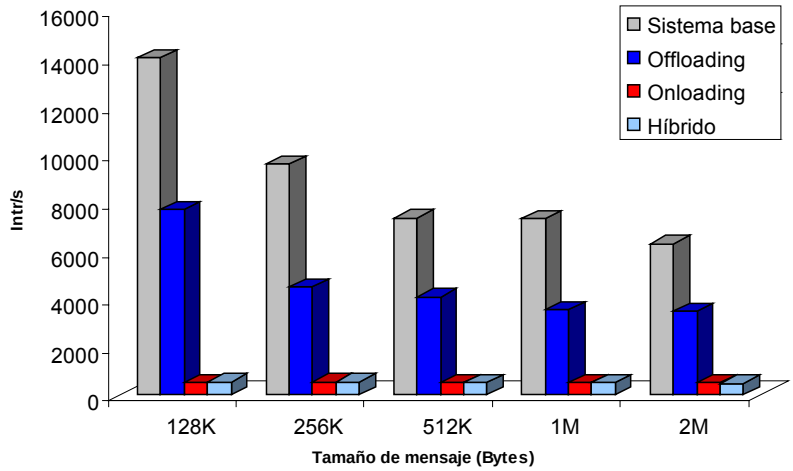


Figura 5.6. Comparación de interrupciones por segundo procesadas por la CPU principal

En la Figura 5.6 se pone de manifiesto la reducción en el número de interrupciones por segundo en la CPU_0 que consiguen las alternativas de externalización que fuerzan el procesamiento de las interrupciones en una CPU diferente a la principal.

Como se he indicado, en la Figura 5.7 se muestra el porcentaje de uso de la CPU principal para las tres alternativas de externalización, así como para el sistema sin externalización para diferentes tamaños de mensaje. Puede verse que la carga en la CPU_0 es menor en el caso de *onloading*, que en el caso de *offloading* debido a que el driver de la NIC se está ejecutando en la CPU_1. En el caso de *offloading*, aunque la pila TCP/IP se está procesando en la CPU_{NIC}, las interrupciones de la NIC llegan a la CPU_0 y es en ésta CPU en la que se ejecuta el *driver*. En el caso de la alternativa híbrida, la CPU_0 queda descarga de de las tareas de comunicaciones de una forma similar a la implementación de *onloading*.

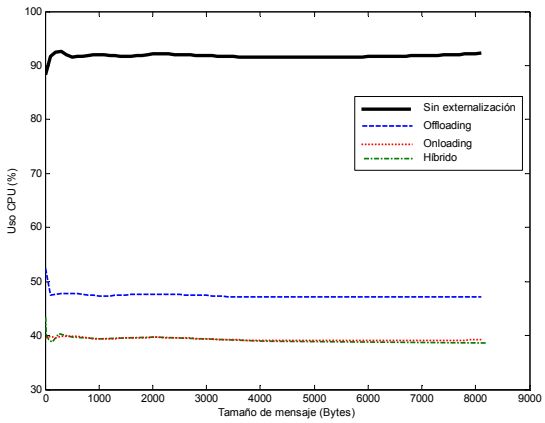
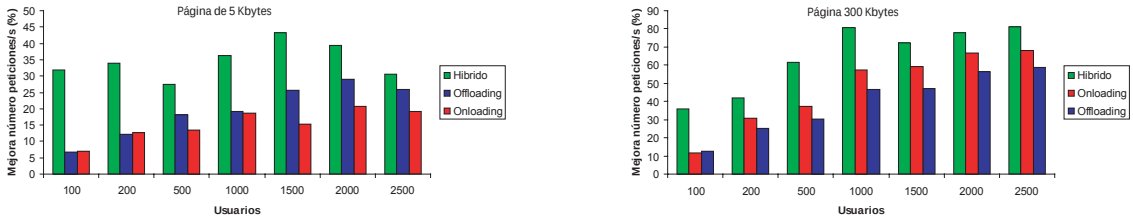


Figura 5.7. Comparación de porcentaje de uso de CPU

5.10 Prestaciones de la interfaz híbrida de externalización en un servidor web

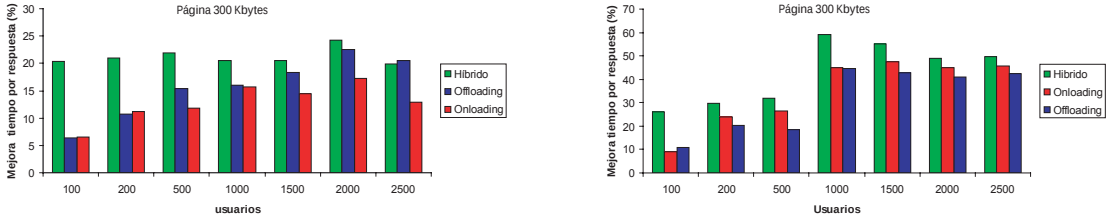
Aunque el uso de benchmarks estándar permite realizar comparaciones directas entre distintas propuestas, a veces queda la duda de si en las aplicaciones reales se observarán esas mejoras debido a que los perfiles de comunicación de esas aplicaciones reales podrían diferir bastante de los de los benchmarks, más o menos sencillos. Es por ello que, en esta sección utilizamos una aplicación real para comparar sus prestaciones bajo las distintas alternativas de externalización implementadas. En concreto, en esta sección se van a presentar las medidas de prestaciones evaluadas utilizando un servidor web apache 2.0 en un nodo con la interfaz híbrida de red que se ha presentado en este capítulo, de forma análoga a lo presentado en la Sección 4.9 para las interfaces de red con externalización mediante *offloading* y *onloading*.



(a) *(b)*

Figura 5.8. Mejora en número de peticiones/s. Página de (a) 5 Kbytes y (b) 300 Kbytes

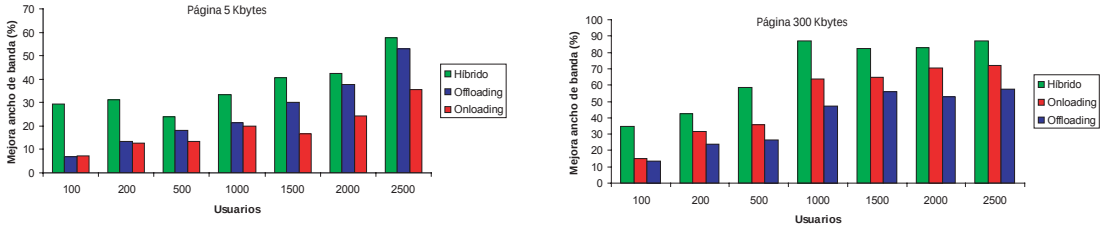
En la Figura 5.8a se muestra el porcentaje de mejora respecto al sistema base proporcionado por la interfaz híbrida en peticiones por segundo considerando distinto número de usuarios y tamaño de página en comparación con las alternativas de *offloading* y *onloading*. Como puede verse en dicha figura, en todos los casos la mejora obtenida es mayor para la interfaz híbrida que para las interfaces con *offloading* u *onloading*. Además, para mensajes de tamaño pequeño la mejora proporcionada por la interfaz híbrida tiende a igualarse con la obtenida por la interfaz con offloading al incrementarse el número de usuarios (y la carga del nodo, por tanto).



(a) *(b)*

Figura 5.9. Mejora en tiempo de respuesta. Página de (a) 5 Kbytes y (b) 300 Kbytes

De forma similar, se observa que la mejora obtenida en el tiempo por petición se iguala con la obtenida por la interfaz con offloading para valores de carga altos.



(a) *(b)*

Figura 5.9. Mejora en ancho de banda. Página de (a) 5 Kbytes y (b) 300 Kbytes

En cuanto al ancho de banda, la interfaz híbrida proporciona mejores resultados en cualquier caso, como ya se mostró en la Sección 5.3.1.

Como se muestra en las Figuras 5.8 y 5.9, al crecer el número de usuarios (lo cual supone un incremento en la carga de la CPU que ejecuta la aplicación y el sistema operativo), la mejora proporcionada por la externalización también crece, debido a que, como se ha visto en la Sección 5.3, cualquier alternativa de externalización reduce la carga en la CPU que ejecuta la aplicación. Esto es, quedan más ciclos libres para la ejecución de la aplicación. Por otro lado, de forma similar a como se mostró en el Capítulo 4, la alternativa de *onloading* proporciona un mayor ancho de banda y esta mejora es más evidente cuando la sobrecarga debida a la apliacación es alta (2500 usuarios en la Figura 5.9). Del mismo modo, la alternativa de *offloading* proporciona una menor latencia y por tanto, un menor tiempo de respuesta, cuando la carga debida a la aplicación no es muy alta. En la Figura 5.8 puede verse como al incrementarse el número de usuarios que piden páginas de un tamaño mediano (300 Kbytes), la alternativa de externalizació mediante *onloading* proporciona un mayor valor de la mejora.

En cuanto a nuestra propuesta de interfaz híbrida de externalización, la mejora proporcionada para el perfil de comunicación correspondiente a un servidor web que envía páginas html estáticas a los usuarios, es mayor que para las otras dos alternativas de externalización (*offloading* y *onloading)* descritas en el Capítulo 3 y analizadas el Capítulo 4.

5.11 Resumen

En este capítulo se presenta una alternativa de interfaz de red híbrida, que combina las ventajas de la externalización mediante *offloading* y mediante *onloading*, utilizando los recursos presentes en una plataforma con varios procesadores (SMP) o con varios núcleos de procesamiento (CMP). Para ello se ha construido un nuevo modelo de simulación de Simics, a partir de los modelos de interfaces de red que incorporan offloading y onloading presentados en el Capítulo 3, y que, por tanto, hacen uso de los modelos de tiempo que se introdujeron también en el Capítulo 3 y que hacen posible la correcta simulación de E/S y de la comunicación. Así, en la Sección 5.2 se describe el modelo de simulación de la interfaz híbrida y su funcionamiento y se analiza

el contexto de las investigaciones previas realizadas con esta misma orientación describiendo otra alternativa propuesta a [SHA06]. En la Sección 5.3 se realiza una evaluación comparativa entre las prestaciones ofrecidas por la nueva implementación híbrida respecto a las ofrecidas por las implementaciones de *offloading* y *onloading*. En dicha comparativa se muestra como la implementación híbrida de la interfaz de red mejora las prestaciones ofrecidas por las alternativas de *offloading* y *onloading*, con un consumo de CPU menor en la CPU principal del sistema.

Finalmente, se ha utilizado un servidor web para comaparar las prestaciones ofrecidas por las interfaces con externalización mediante *offloading y onloading* presentadas en el Capítulo 4 con las ofrecidas por la interfaz híbrida de externalización. Queda claro también el mejor comportamiento de la interfaz híbrida que hemos propuesto con mejoras de latencia de entre un 30% y un 70% y mejoras de ancho de banda de entre un 20% y un 60% dependiendo de la aplicación y de su perfil de comunicación.

Capítulo 6

Conclusiones y Líneas Futuras

En el trabajo que se describe en esta memoria se han abordado aspectos relacionados con la interfaz de red. Los resultados obtenidos son válidos tanto para servidores de computación de altas prestaciones, como para otro tipo de servidores basados en configuraciones de tipo cluster con relativamente pequeñas, así como servidores de ficheros, servidores web, o de almacenamiento. En trabajos anteriores como [MIN95, SNE96, HUA04, KIM06, BIN05, BIN06] se pone de manifiesto la importancia de la interfaz de red para la comunicación entre computadores. La importancia de la optimización de las interfaces de red para superar la diferencia de prestaciones entre los anchos de banda de los enlaces y la capacidad de los procesadores, los buses de E/S y la memoria, ha sido la principal motivación de esta tesis, en la que se han estudiado distintas alternativas posibles para mejorar las prestaciones la comunicación en redes de servidores interconectados por enlaces de anchos de banda cada vez más elevados. Las alternativas consideradas se basan en la idea de la externalización del procesamiento de las tareas de comunicación entendida como la asignación de tareas de comunicación a otros procesadores presentes en el nodo, bien en la tarjeta de red, bien procesadores con la misma capacidad el procesador donde se ejecuta la aplicación en un CMP o en un SMP. Además, dada la actual controversia existente en cuanto a las técnicas de externalización utilizadas para mejorar las prestaciones que se pone de manifiesto en distintas propuestas comerciales (TOEs, Intel IOAT, etc.), es imprescindible disponer de herramientas que permitan realizar un análisis adecuado del espacio de diseño. En nuestro caso hemos optado por la

simulación de sistema completo mediante el simulador Simics, para evaluar el comportamiento del sistema de comunicaciones. Mediante los simuladores de sistema completo es posible una evaluación realista del sistema, incluyendo el sistema operativo y utilizando aplicaciones reales. De esta forma, hemos desarrollado modelos de simulación para realizar una evaluación comparativa de las técnicas de externalización *offloading* y *onloading*, y clarificar las condiciones bajo las que cada técnica resulta más beneficiosa. Además, se ha propuesto una interfaz híbrida que aprovecha las ventajas de las técnicas de *offloading* y *onloading* a la par que evita sus inconvenientes.

Por otro lado, se ha analizado el comportamiento de los sistemas simulados desde el punto de vista del modelo teórico LAWS [SHI03], que considera tanto parámetros del sistema relacionados con la propia arquitectura del sistema, como parámetros relacionados de la red de interconexión, y proporciona una cota superior para la mejora introducida en las prestaciones por la externalización. Este modelo, en su versión original, supone únicamente una primera aproximación a las prestaciones reales. A fin de cuentas es sólo una cota superior lo que proporciona, ya que no tiene en cuenta efectos importantes en un sistema real, aunque también permite explicar cambios cuantitativos en esta cota. Se han incluido algunas modificaciones del mismo, a través de nuevos parámetros que hemos incluido para que, con el nuevo modelo resultante se consigua un mejor ajuste a los resultados experimentales.

Tras introducir la problemática de las comunicaciones a abordar, en el Capítulo 2 se realiza una descripción de la simulación de sistema completo que constituye la base de la metodología de simulación del comportamiento del sistema de comunicación. Ademas, se han comentado las características de los simuladores de sistema completo que existen en la actualidad así como algunas de sus características.

Una vez vistos los simuladores existentes, el trabajo se ha centrado en Simics como el simulador de sistema completo más apropiado para nuestros propósitos, analizando sus ventajas e inconvenientes. Tambíen se describe la estrategia utilizada para superar los problemas y limitaciones que plantea su uso en el ámbito del sistema de comunicación. Dicha estrategia se basa en la utilización de modelos de tiempo que permiten simular el comportamiento temporal del que Simics carece por defecto. Así, se ha desarrollado un modelo de tiempo configurable que puede ser conectado en diferentes partes del sistema y que permite simular diferentes latencias en los accesos a los buses a la memoria o a los dispositivos. Además, se ha implementado en el modelo de tiempo desarrollado, la detección de colisiones en los accesos a los buses o a

memoria, así como la asignación de diferentes latencias en los accesos y una penalización por colisión.

En el Capítulo 3, se ha revisado el estado del arte de la externalización, comentando las técnicas utilizadas en su implementación. Seguidamente se describen las diferentes alternativas para realizar la externalización de protocolos de comunicación que hemos considerado, y se presentan los modelos de simulación desarrollados para simular el comportamiento de la externalización mediante *offloading* y *onloading*. Se describe tanto la arquitectura hardware simulada como la distribución de software que proponemos en cada caso. En el modelo de simulación de la externalización mediante *offloading* se ha utilizado dos procesadores convenientemente conectados y se ha modificado el perfil de interrupciones de la CPU que realiza el procesamiento de los protocolos de red. De esta forma, podemos simular una tarjeta de red que incorpora un procesador y memoria para almacenar los paquetes que provienen de la red. Además, se ha implementado un buffer en anillo en el diseño de la propia NIC, consiguiendo un comportamiento del tipo "una interrupción por evento" en lugar de "una interrupción por paquete". Para poder analizar las distintas alternativas. Además, se ha simulado el efecto de disponer de una jerarquía que acelere el acceso a memoria dentro de la propia tarjeta de red (cache local en la NIC). En el modelo de simulación de la externalización mediante *onloading* se ha utilizado un SMP con dos procesadores, en el que una de las CPU realiza tanto el procesamiento de los protocolos de red como del driver de la interfaz de red, dejando la otra CPU exclusivamente para la aplicación. En este caso, la interfaz de red utiliza la memoria del sistema, donde se ha implementado el buffer en anillo. En ambas alternativas exsisten, por tanto, diferencias eb cuabto al hardware y a la forma en que se ha distribuido el software.

Después, en el Capítulo 4, se presentan los resultados experimentales obtenidos con las dos alternativas de externalización desarrolladas en el Capítulo 3, analizando las prestaciones proporcionadas por la externalización mediante *offloading* y *onloading*. De la misma forma, se presentan resultados que comparan las dos técnicas de externalización, y se utiliza el modelo teórico LAWS para analizar las prestaciones a partir de diferentes parámetros, tanto de la red, como de la implementación de la externalización, y se comparan comparando los resultados experimentales con los que predice dicho modelo teórico. Se propone, además, un modelo LAWS modificado, que incluye nuevos parámetros con el fin de modelar efectos que aparecen en los sistemas

reales y que no tiene en cuenta el modelo LAWS propuesto inicialmente en [SHI03], que supone un sistema de comunicaciones totalmente segmentado.

Como conclusión de los experimentos realizados se puede indicar que el modelo de externalización mediante *offloading* proporciona una menor latencia, y por tanto, mejores prestaciones con paquetes pequeños. Por otro lado, el modelo de externalización mediante *onloading* proporciona un ancho de banda superior y un mejor comportamiento que el modelo de externalización mediante *offloading* bajo niveles de carga de aplicación altos, corroborada a partir del análisis realizado con el modelo LAWS.

Finalmente, en el Capítulo 5 se describe nuestra propuesta de externalización basada en una interfaz híbrida *offloading*/*onloading*, que ofrece unas prestaciones superiores a las que, en los resultados experimentales obtenidos proporcionan las técnicas de externalización mediante *offloading* u *onloading*. Además, se han realizado experimentos utilizando un servidor web Apache 2.0 para poner de manifiesto el comportamiento de las alternativas de interfaces de red propuestas con aplicaciones reales y no sólo con programas de *benchmark*. Las medidas realizadas del número de peticiones por segundo, del tiempo medio por respuesta y del ancho de banda muestran la mejora de prestaciones que puede obtenerse con las interfaces de red propuestas en los Capítulos 4 y 5.

6.1 Contribuciones y resultados

En este trabajo se ha utilizado Simics como herramienta de simulación de sistema completo, que permitiese analizar el comportamiento de la interfaz de red y el efecto de la externalización de protocolos. Para ello ha sido necesario desarrollar modelos de simulación que incorporen todos los elementos presentes en una arquitectura real de un ordenador personal actual (PC), como procesadores, buses, memoria, puentes norte y sur, interfaces de red, etc. Además, para realizar una simulación precisa y poder obtener resultados análogos a los obtenidos en un sistema real, ha sido necesario realizar modificaciones y desarrollos para solventar los problemas que Simics presenta inicialmente para este tipo de simulación en los sistemas de comunicación. Concretamente, para simular el comportamiento temporal que Simics no proporciona por defecto, se han desarrollado modelos de temporización que permiten modelar latencias en los accesos a memoria y a los dispositivos, así como el efecto de las

colisiones. De esta forma, hemos dotado a Simics de la capacidad necesaria para simular de forma configurable el comportamiento temporal de una arquitectura, siendo posible asignar latencias diferentes a los accesos desde diferentes dispositivos y modelar además las colisiones, así como asignar un tiempo variable de penalización por colisión (*tiempo de contención*).

Además, se han desarrollado interfaces de red que incorporan distintas alternativas de externalización, concretamente, la técnica de *offloading*, la técnica de *onloading*, y unainterfaz de red híbrida de externalización que hemos propuesto para aprovechar las ventajas de las técnicas de offloading y de onloading y que evita los inconvenientes de las mismas, proporcionando unas prestaciones de ancho de banda y de latencia mejores que las proporcionadas por *offloading* u *onloading*. Mediante los modelos de simulación y de temporización que hemos desarrollado con Simics, se han estudiado las prestaciones ofrecidas por un sistema con externalización *offloading*, *onloading,* y de la propuesta de interfaz híbrida en comparación con un sistema sin externalización. Las simulaciones realizada han hecho posible analizar con detalle la interacción entre la aplicación, el sistema operativo y los elementos de la interfaz de red tanto hardware como software. Así, por un lado, se han tomado medidas de tasa de transferencia utilizando el protocolo de transporte TCP y comunicación unidireccional (*streaming*) así como la latencia de los mensajes entre dos nodos, mediante experimentos de transferencias tipo *ping-pong* y utilizando benchmarks como *Netpipe, Hpcbench, Sysmon, Oprofile*, etc. y aplicaciones reales como es un servidor web Apache.

Los resultados experimentales muestran mejoras en todos los casos analizados, tanto en ancho de banda, como en latencia. Sin embargo, estas mejoras no son muy evidentes bajo condiciones de carga bajas, cuando el nodo puede gestionar por si mismo el ancho de banda del enlace de red y la carga de procesamiento de los protocolos. Por esta razón es importante realizar experimentos que pongan de manifiesto la variación de la función de mejora en tasa de transferencia, para distintos perfiles de carga de la aplicación. De esta forma, ha sido posible comprobar como, conforme la carga asociada a la aplicación comienza a subir, crece la mejora de ancho de banda hasta un máximo a partir del cual comienza a decrecer, debido al efecto que predice de la *Ley de Amdahl*, dado que el coste relativo de la comunicación es demasiado pequeño en comparación con la carga de la aplicación. La máxima mejora obtenida está entorno al 30% en el ancho de banda con los sistemas simulados tanto para el modelo de externalización mediante *offloading,* como para *onloading,* y con los *benchmark* utilizados.

Así, las técnicas de externalización proporcionan mejoras sobre el sistema sin externalización, en cuanto al ancho de banda, las prestaciones ofrecidas por *offloading* y *onloading* son similares bajo condiciones de carga baja. En cuanto a la latencia, se observa una menor latencia en el caso de la externalización mediante *offloading* debido a que en este caso el proceso de los protocolos se realiza en el propio interfaz de red y éste puede interactuar con la red sin intervención de la CPU del nodo. Al incrementar las condiciones de carga (valores más altos del parámetro γ en el modelo LAWS), puede verse que la externalización con *onloading* proporciona mejores resultados que con *offloading*, poniéndose de manifiesto que un sistema con *onloading* puede mantener mejoras más elevadas en el ancho de banda bajo cargas de aplicación mayores que en el caso de *offloading.*

Aunque el modelo LAWS puede utilizarse para validar los modelos de simulación desarrollados como una primera aproximación al comportamiento real, inicialmente no se obtiene un ajuste muy bueno a los obtenidos mediante simulación. Por tanto, y para tener en cuenta el hecho de que no tenemos un sistema totalmente segmentado, se ha modificado el modelo LAWS introduciendo tres nuevos parámetros que dan lugar a un nuevo modelo. Con éste se consigue un mejor ajuste a los resultados experimentales y, por tanto, una predicción más precisa de los resultados obtenidos en un sistema real. Las modificaciones realizadas al modelo LAWS permiten tener en cuenta desviaciones en la carga de trabajo de las CPU por volumen de datos, debida tanto a la aplicación como a las tareas de comunicaciones. De esta forma, se incluyen en el modelo los parámetros δ_a y δ_o que representan la porción de cambio en el trabajo por unidad de datos, después de externalizar, así como el parámetro τ, que representa la porción de cambio en la carga de trabajo de la CPU después de externalizar, debido a la sobrecarga que genera la comunicación entre la CPU y la NIC a través del subsistema de E/S. De esta forma, a pasa a ser $a+\delta_a$, o pasa a ser $o+\delta_o$, y la carga de la CPU después de externalizar pasa de W a $W+\tau W$. Esto indica que, respecto del modelo original, se produce una desviación δ_a en la carga de trabajo asociada a la aplicación que es mayor en el caso de *onloading* que en el caso de *offloading*, una desviación δ_o en la carga de trabajo asociada a los procesos de comunicación que es mayor en el caso de *offloading* que en el caso de *onloading,* y una desviación, τW, de la interacción entre la CPU_i del nodo y la NIC, menor en el caso de *onloading*.

De la misma forma, los experimentos realizados ponen de manifiesto la reducción del número de interrupciones que llegan a la CPU del *host* siempre que se externalice el sistema de comunicaciones, independientemente de una implementación mediante *offloading* u *onloading* de la externalización. Además, es evidente que tras externalizar, se dispone de un numero mayor de ciclos libres para la aplicación, aportando una clara ventaja con respecto a interfaces no externalizados.

Por otro lado, los experimentos realizados modelos de cache detallados ponen de manifiesto que la memoria *cache* tiene un efecto positivo y una mejora en el sistema de comunicaciones. Este resultado corrobora la importancia de la latencia de acceso a memoria en el sistema de comunicaciones indicado en [CLA89, STE94a, NAH97, CHA01, MAR02, GAD07]. Esta mejora se pone de manifiesto en el ancho de banda, en la latencia de red, y en el comportamiento del sistema ante diferentes niveles de sobrecarga debida a la aplicación. Así, mediante *offloading* se consigue una menor latencia, resultando una alternativa más adecuada para mensajes pequeños. Por otro lado la externalización mediante *onloading* proporciona un mayor ancho de banda y soporta mejor niveles de carga altos de la aplicación, corroborando el modelo LAWS. Además, cabe esperar que la mejora de la interfaz de memoria, así como el uso de buses internos de mayor ancho de banda o el uso de procesadores con más de un núcleo y memorias *cache* de mayor tamaño incrementen las prestaciones de las implementaciones de la externalización mediante *onloading* en un futuro próximo, mientras que en el caso de la externalización mediante *offloading* las mejoras siguen estando limitadas por la interacción con el sistema operativo y su integración en el mismo. Resulta difícil aprovechar totalmente las ventajas de disponer de un procesador en la interfaz de red mientras sigan existiendo cuellos de botella en otras partes del sistema.

También, se ha puesto de manifiesto la importancia de la jerarquía de memoria, no sólo para acceder a la memoria principal sino también en la propia tarjeta de red. Así, añadiendo modelos de memoria cache detallada a los modelos de simulación de las alternativas de externalización, se ha analizado de forma más realista el comportamiento de las interfaces de red, observándose diferencias entre un modelado de la memoria cache simplemente mediante el tiempo medio de acceso y un modelado detallado de una jerarquía de memoria de tres niveles.

Una vez analizadas las prestaciones de las técnicas de *offloading* y *onloading* e identificadas sus ventajas e inconvenientes, se propone un modelo híbrido para la

interfaz de red, que aprovecha las ventajas de las técnicas de *offloading* y *onloading* y evita sus inconvenientes. Los experimentos realizados con la implementación de dicho modelo híbrido de la interfaz de red ponen de manifiesto mejoras respecto a los modelos de externalización mediante *offloading* u *onloading*, tanto en latencia como en ancho de banda. Además, se observa como en el caso de aplicaciones cuyo perfil de comunicación requiera de una baja latencia, las prestaciones de la implementación híbrida tienden a las de *offloading* y en el caso de aplicaciones que requieran de un gran ancho de banda, las prestaciones se aproximan las del modelo de externalización mediante *onloading*.

Todos los resultados experimentales se han obtenido con benchamrks (Netpipe, Hpcbench, Sysmon, etc.) ampliamente utilizados en esta área y finalmente, se han realizado experimentos utilizando una aplicación real como es un servidor web apache para comprobar que se obtienen mejoras en entornos más realistas. Dichos experimentos corroboran nuestra anterior conclusión de que la mejora de prestaciones obtenida con cada interfaz de red depende del perfil de comunicación de una aplicación concreta. En el caso de un servidor web, el cliente realiza peticiones pequeñas (http) y el servidor responde con una página de un tamaño determinado. Los experimentos realizados para páginas de 5 Kbytes y de 300 Kbytes permiten comprobar la variación de la mejora obtenida en función de la carga del servidor (que en el caso del servidor web, se traduce en el número de usuarios o conexiones concurrentes).

En definitiva, podemos concluir que el uso de una técnica de externalización u otra depende del perfil de comunicaciones de una aplicación concreta, y que por lo tanto, no siempre resulta beneficioso utilizar *offloading* u *onloading*. En este sentido, la interfaz híbrida de externalización supone la mejor alternativa, válida para aplicaciones con un perfil de comunicaciones mixto, en el que se realicen transferencias tanto de paquetes pequeños como de paquetes grandes.

6.2 Líneas futuras

A partir del trabajo realizado en esta memoria, han surgido diferentes líneas interesantes para continuar la investigación en esta línea.

1. **Mejora de la herramienta de simulación.**

- Desarrollar nuevos modelos de simulación de Simics
- Desarrollar un modelo hardware de interfaz de red externalizado (TOE), que implemente en un principio TCP, pudiendo ampliarse a otros protocolos como MPI. De esta forma se podría simular una implementación totalmente hardware y externalizada de la interfaz de red. En este caso, sería necesario desarrollar el correspondiente driver para la interfaz de red. De esta forma, se ampliaría el repertorio de elementos hardware disponible en Simics.
- Aprovechar futuras mejoras de Simics que permitan acelerar la simulación.

2. Desarrollo de nuevos modelos teóricos más precisos que puedan corroborarse con los resultados de simulación.

- Estudiar detalladamente el modelo LAWS y proponer nuevas modificaciones, o modelos alternativos que permitan un mejor ajuste a los resultados experimentales proporcionando una información más precisa del comportamiento de un sistema real, ya que LAWS no tiene en cuenta el solapamiento existente en las diferentes etapas: como se ha dicho en la memoria, un sistema real no puede considerarse totalmente segmentado. Podrían tomarse como referencia trabajos como [CUL93] o [WAN98].

Por otro lado, resultaría interesante realizar un análisis similar al que se ha realizado con el modelo LAWS, pero utilizando otros modelos como el EMO [GIL05] que no solo sean aplicables a transferencias mediante *streaming*.

- Utilizar los modelos propuestos para estudiar el comportamiento y las prestaciones ofrecidas por las técnicas de externalización mediante offloading y onloading, para el caso de aplicaciones paralelas que utilizan bibliotecas de paso de mensajes tales como MPI o PVM. Para ello pueden utilizarse los mismos benchmarks (Netpipe, Hpcbench) que se han utilizado en esta memoria, aunque incluyendo los módulos MPI correspondientes en nuestros experimentos. De esta forma podría analizarse si las propuestas realizadas son directamente aplicables a protocolos de paso

de mensajes, y estudiar que parámetros de las arquitecturas propuestas afectan a las prestaciones en este caso. Además, al igual que en la memoria se ha utilizado el modelo LAWS para comparar los resultados experimentales con los proporcionados por un modelo teórico, en este caso podría utilizarse el modelo EMO.

- Completar los modelos de simulación propuestos en este trabajo con otros modelos de cache (existen trabajos que proponen modelo detallados de *cache* para Simics como el modelo estadístico de cache [BER06]). De esta forma se podría estudiar el efecto de diferentes políticas de caché en las prestaciones ofrecidas por las técnicas de externalización.

3. Ensayar nuevas técnicas e ideas para seguir mejorando la interfaz de red. En particular, técnicas que permitan aprovechar el paralelismo en la interfaz de red.

- Implementar posibles mejoras en el modelo híbrido de externalización que mejore las prestaciones del modelo presentado en esta memoria.
- Estudio de las prestaciones ofrecidas por las interfaces de red propuestas con diferentes aplicaciones para identificar el perfil de comunicación al que se adapta mejor cada interfaz de red.
- Implementación en otras redes de los modelos de externalización aquí descritos.

6.4 Principales publicaciones

Congresos Internacionales

1. Título: **Comparación de técnicas para la externalización de protocolos**. Memorias de la 7ª Conferencia Iberoamericana en Sistemas, Cibernética e Informática (CISCI2008). Junio 2008, Orlando (EEUU).
   Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
   Referencia: [ORT08b]

2. Título: **Comparison of offloading and onloading strategies to improve network performance.** 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2008). Toulouse. Febrero 2008.
   Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
   Referencia: [ORT08a]

3. Título: **Analyzing the Benefits of Protocol Offload by Full-System Simulation.** 15th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2007). Nápoles. Febrero 2007.
   Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
   Referencia: [ORT07a]

4. Título: **Protocol Offload Evaluation Using Simics.** IEEE International Conference on Cluster Computing (CLUSTER 2006). Barcelona, Septiembre 2006
   Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
   Referencia: [ORT06b]

Revistas Internacionales

5. Título: **Protocol Offload Análisis by Simulation**. Journal of Systems Architecture. Editorial Elsevier. Aceptado para publicación. Manuscript number JSA-D-07-00082R1. Paper SYSARC_884
Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Pablo Cascón, Alberto Prieto
Referencia: [ORT08d]

6. Título: **Modeling network behaviour by full-system simulation.** Journal of Software. Issue 2/2007. pp. 11-18
Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
Referencia: [ORT07c]

7. Título: **Network Interfaces for Programmable NICs and Multicore Platforms**. Computer Communications. Editorial Elsevier (En revisión)
Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
Referencia: [ORT08e]

Congresos Nacionales

8. Título: **Alternativas para la interfaz de red en servidores multiprocesador**. XVIII Jornadas de Paralelismo. Septiembre 2008
Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
Referencia: [ORT08c]

9. Título: **Comparación de prestaciones onloading/offloading mediante Simics.** XVIII Jornadas de Paralelismo. Septiembre 2008. Zaragoza.
Autores: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto
Referencia: [ORT07b]

10. Título: **Simulación de la externalización de protocolos de comunicación mediante Simics.** XVII Jornadas de paralelismo. Septiembre 2005. Albacete.
Autores: Andres Ortiz, Antonio F. Díaz, Julio Ortega, Alberto Prieto.
Referencia: [ORT06a]

11. Título: **Análisis de la externalización de protocolos mediante simulación HDL.** XVI Jornadas de paralelismo. Septiembre 2005. Granada.
Autores: Antonio F. Díaz, Julio Ortega, Alberto Prieto; Andrés Ortiz
Referencia: [DIA05]

Conclusions

In this thesis, we have proposed and analyzed two implementations of protocol offloading alternatives, onloading and offloading, in order to take advantage of different processors available in the node to improve the communication performance. In this way, we have built simulation and timing models and implementations of both offloading techniques to evaluate and compare their behavior by using the full-system simulator SIMICS. Moreover, we have studied the conditions in which, according to the LAWS model, these two techniques improve the communication performance. Thus, the usefulness of the LAWS model to drive the experimental work and to analyze the obtained results is also shown.

With respect to the behavior of onloading and offloading, we have shown that, although both strategies contribute to reduce the number of interrupts received by the host CPU and the host CPU utilization by communication tasks, the best results correspond to onloading.

The relative improvement on throughput offered by offloading and onloading depends on the rate of application workload to communication overhead of the implementation, the message sizes, and on the characteristics of the system architecture. It is also shown that, in our implementations, onloading provides better throughputs than offloading whenever realistic conditions are considered with respect to the application workload (as the values of the LAWS parameter γ grow).

Offloading gives the best results with respect to the message latencies in the simulations done without cache, and cache does not suppose an improvement in latency when offloading is used, as the processor in the NIC does not use it. Nevertheless, cache produces an important reduction in the latencies for onloading and in the base case. This reduction implies that even the base case (without offloading) is better than offloading with respect to latency. This circumstance demonstrates the relevance of the interaction between the network interface and the memory hierarchy.

Thus, we have also proposed a hybrid network interface that tries both to take advantage of the better features of onloading and offloading and to avoid their drawbacks. Our simulation results show that this interface improves both the throughput improvements achieved by either onloading or offloading schemes and presents latencies almost as good as the best results, obtained by using the network interface through the implementation the onloading alternative.

We consider that, thanks to the SIMICS simulation and timing models which we have developed, it is possible to collect detailed information about the different system events to make more accurate analyses of the different improvement strategies that would be proposed in the future. In this way, we plan to proceed with our analysis of network interfaces by using more benchmarks and other real applications.

Summary of contributions

In order to extend SIMICS with the timing behavior which is not provided by default, we have developed a configurable timing model that can be connected to different parts of the system architecture. This provides latency modeling in the CPUs or devices memory accesses as well as north bridge contention effect. The configuration capabilities of the developed timing model allow allocate different latencies to different parts of the architecture and also contention penalties in clock cycles. This timing model can be chained with others in order to simulate more complex hierarchies.

Moreover, we have developed network interfaces that incorporate offloading and onloading techniques to improve network performance and we have proposed a hybrid network interface which takes advantage of the offloading and onloading benefits and avoids their drawbacks. Thus, the proposed hybrid interface is intended to provide better bandwidth and latency results than offloading or onloading.

Thus, we have implemented the proposed network interfaces by using simulation and timing models. Each implementation involves a hardware model development, our timing model, and software developments on the operating system layer.

In this way, we have investigated and compared the performance provided by each implementation (offloading, onloading, and the hybrid interface) regarding to a base system (a system which does not implement any offloading technique). The performed simulation allows us to analyze the hardware, operating system, and application interactions in detail as well as the different hardware or software elements of the network interfaces.

The experiments performed in order to show the improvements that offloading techniques can provide, involve throughput measurements using TCP as transport protocol and streaming transferences and latency measurements using ping-pong tests.

The experimental results show improvements for all analyzed cases in throughput and latency. Nevertheless, these improvements are not very marked under low application load if the node is capable to provide the full link bandwidth by itself. For this reason, we have performed experiments to show the variation in the improvement function using streaming TCP/IP transferences. In this way, it has been possible to realize that offloading techniques do not provide high improvements under low or very low application overheads. However, as the application overhead increases, the improvement increases until a maximum level is reached. The improvement then starts to decrease according to the Amdahl's law.

With the above experiments, the obtained improvement is up to 30% with the simulated systems for the offloading and the onloading interfaces with the benchmarks used.

As commented before, offloading and onloading techniques provide improvements regarding the system without offloading under low application overhead (low application rate parameter values in the LAWS model). As application overhead increases, the improvement provided by onloading is higher than for offloading.

The results obtained by the LAWS model can be used to validate our simulation models, since they provide a qualitative prediction and so, a first approximation to the real behavior. A better quantitative prediction can be achieved by considering the influence of a not fully pipelined communication path and so, to get a better fit to the experimental results. In this way, we have modified the original LAWS model by adding three new parameters. The new LAWS model provides a better fit to the experimental results. These parameters added to the LAWS model represent deviations in the application overhead, communication overhead, and residual overhead after offloading with respect to the original LAWS model.

Other performed experiments show the reduction in interrupts per second delivered to the host's CPU for offloading and onloading. Thus, there are more CPU idle cycles available after offloading and this may be considered the main advantage of the offloading techniques.

On the other hand, we have included detailed cache models to our simulation models for offloading and onloading in order to simulate the cache effects not only through a mean access time, but through a detailed three level memory hierarchy. The experiments performed with the models that include two level cache memory show improvements in throughput as well as in latency with respect to a less detailed hierarchy corroborating the importance of the memory access time in the communications subsystem.

From the obtained results, we can conclude that offloading provides lower latencies and so, it is more useful for small messages. On the other hand, onloading provides a higher throughput and can withstand higher application overheads according to the LAWS model. Moreover, it is predictable that improvements in the memory and I/O subsystems, the use of multicore processors, and an improved cache system can increase the performance provided by the onloading technique in a near future. In the case of the offloading technique, the improvement is still limited by the interaction and the integration with the operating system. Thus, it is not possible to exploit the advantages of having a processor in the network interface card if there are bottlenecks in other system elements.

Once the performance provided by the proposed offloading and onloading interfaces is analyzed, we propose a hybrid interface that takes advantage of offloading and onloading techniques and avoids their drawbacks. The experiments performed with this hybrid network interface show improvements with respect to the offloading and onloading performance in bandwidth and similar latency values than offloading. Moreover, it shown that the hybrid interface meets the requirements for applications with low latency communication profile as well as for high bandwidth profile applications.

Finally, we have used an Apache web server to load the node and analyze the improvements provided by each offloading technique and by the hybrid interface. This experiment is interesting because a web server is an application with a communication profile that depends on the size of the requests from the client and responses from the server. In this way, we can conclude that a hybrid interface is an interesting alternative for applications that use an undefined communication profile.

Main Publications

International Conferences

1. Title: **Protocol Offload Evaluation Using Simics.** IEEE International Conference on Cluster Computing (CLUSTER 2006). Barcelona, Septiembre 2006

   Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

2. Title: **Analyzing the Benefits of Protocol Offload by Full-System Simulation.** 15th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2007). Nápoles. Febrero 2007.

   Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

3. Title: **Comparison of offloading and onloading strategies to improve network performance.** 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2008). Toulouse. Febrero 2008.

   Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

4. Title: **Comparación de técnicas para la externalización de protocolos**. Memorias de la 7ª Conferencia Iberoaméricana en Sistemas, Cibernética e Informática (CISCI2008). Junio 2008, Orlando (EEUU).

   Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

International Journals

5. Title: **Modeling network behaviour by full-system simulation.** Journal of Software. Issue 2/2007. pp. 11-18

Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

6. Title: **Protocol Offload Analisis by Simulation**. Journal of Systems Architecture. Accepted for publication. Manuscript number JSA-D-07-00082R1. Paper SYSARC_884. (Journal impact factor: 0.490)

Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Pablo Cascón, Alberto Prieto

7. Title: **Network Interfaces for Programmable NICs and Multicore Platforms**. Computer Communications. (Under review). (Journal impact factor: 0.391)

Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

National Conferences

8. Title: **Análisis de la externalización de protocolos mediante simulación HDL.** XVI Jornadas de paralelismo. Septiembre 2005. Granada.

Authors: Antonio F. Díaz, Julio Ortega, Alberto Prieto; Andrés Ortiz

9. Title: **Simulación de la externalización de protocolos de comunicación mediante Simics.** XVII Jornadas de paralelismo. Septiembre 2005. Albacete.

Authors: Andres Ortiz, Antonio F. Díaz, Julio Ortega, Alberto Prieto.

10. Title: **Comparación de prestaciones onloading/offloading mediante Simics.** XVIII Jornadas de Paralelismo. Septiembre 2008. Zaragoza.

Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

11. Title: **Alternativas para la interfaz de red en servidores multiprocesador**. XVIII Jornadas de Paralelismo. Septiembre 2008

Authors: Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Alberto Prieto

Apéndice I

Características y Configuración de Simics

A.1 Características de Simics

A continuación, se resumen las principales características de Simics, así como diferentes aspectos relativos a la configuración del simulador.

A.1.1 Arquitecturas soportadas por Simics

Simics permite simular plataformas, que pueden ser monoprocesador, multiprocesador simétrico o SMP, basadas en un total de nueve procesadores diferentes. Sin embargo, la versión académica utilizada para obtener los resultados descritos en esta memoria, no tiene soporte completo para todas las plataformas.

- **Alpha**

  Simics puede modelar maquinas basadas en el procesador Alpha 21164 (también conocido como EV5) y el chipset DEC 21174 (también conocido como "Pyxis"). De esta forma, pueden simularse máquinas similares al AlphaPC 164LX. Sobre esta arquitectura podemos instalar Linux o Tru64 como sistema operativo.

- **ARM**

  Modelado de arquitecturas basadas en el procesador ARMv5. Se pueden simular arquitecturas con un procesador RISC StrongARM y un controlador de memoria. Sobre esta arquitectura es posible ejecutar una versión limitada de Linux. En esta arquitectura hay dispositivos con un soporte limitado, al menos en las versiones actuales de Simics, como gpio (General purpouse IO), ic (interruptcontroler) ost (OS timer), ppc (peripheral pin controller), rtc (real time clock), y UART (Puerto serie).

- **PowerPC**

  Arquitecturas basadas en los procesadores PowerPC 750 y 7450 y 440GP. Sobre estas arquitecturas es posible instalar Linux con versiones del kernel a partir de la 2.4.

- **IA64**

  Modelado de arquitecturas basadas en la familia de procesadores Intel Itanium (Itanium e Itanium2) y el chipset Intel 460GX. Este chipset soporta hasta un máximo de 4 procesadores. Sin embargo, el modelo de chipset 460GX que incluye Simics permite la configuración de máquinas de hasta 32 procesadores. El único sistema operativo que es posible instalar sobre esta arquitectura es Linux, el cual, soporta la configuración de máquinas de hasta 32 procesadores Itanium. Un ejemplo de máquina con arquitectura Itanium arquitectura es la estación de trabajo HPi2000. Los dispositivos del chipset que están modelados totalmente son el controlador de direcciones 82461GX, puente E/S y firmware 82468GX y el controlador programable de interrupciones UPD66566S1.

- **MIPS**

  Modelado de arquitecturas basadas en procesadores MIPS-4Kc y MIPS-5Kc en modo *little endian*. Aunque el soporte que ofrece Simics para este tipo de arquitecturas es limitada, es posible ejecutar sobre ellas un Kernel de Linux sin modificar, y sin necesidad de un *bootloader*.

- **x86**

  Arquitecturas de la familia x86. Es posible simular desde un 486 hasta un Pentium4 y en la versión comercial es posible activar *Hiperthreading*. Se pueden simular SMP procesadores con la familia x86 hasta de 15 procesadores. Algunos procesadores como Pentium III solo están soportados en la versión comercial de Simics.

- **x86-64:**

  Simulación de procesadores AMD de 64 bits (AMD x8664). Simics no soporta la simulación de SMP con procesadores de 64 bits.

- **UltraSparc II**

  Es posible simular completamente máquinas UltraSparcII hasta de 30 procesadores, simulando los servidores Sun Enterprise 3000-6500.

- **UltraSparc III / UltraSparc IV**

  Para la simulación de servidores con chasis SunFire 3800, 4800, 4810 y 6800, o estaciones de trabajo Sun Blade 1500. Estos chasis se pueden configurar de diferentes formas, ocupando sus Slots con placas de CPU, I/O y/o memoria

  En la práctica, es posible simular hasta un máximo de 16 servidores, y un máximo de 64 procesadores en total.

- **UltraSparc T1**

  Modelado de arquitecturas UltraSparc T1, como por ejemplo el servidor SunFire T2000, sobre el que se puede instalar el sistema operativo Solaris.

- **PPC64**

  Modelado de arquitecturas basadas en el procesador PPC970FX, sobre las que se puede ejecutar Linux. El soporte de Simics para esta arquitectura es limitado.

Tomando estas arquitecturas como base, se pueden construir diferentes configuraciones, utilizando los elementos hardware para los cuales Simics proporciona modelos, o incluso, con modelos hardware hechos a medida.

A.1.2 Configuración de Simics

La configuración de Simics se realiza mediante el llamado *Sistema de Configuración de Simics,* el cual está orientado a objetos, de forma que la máquina simulada (*target*) queda descrita mediante un conjunto de objetos que interaccionan entre sí y proporcionan una determinada salida, modifican determinadas direcciones de memoria, etc., al transcurrir ciclos de reloj. Esta es la forma de incluir CPU, memoria, interfaces de video, interfaces de red, relojes de tiempo real (*RTC)*, etc. Como en todo sistema de descripción orientado a objetos, cada objeto tiene una serie de atributos, los cuales definen las características de cada objeto. El caso más típico es el atributo que define la frecuencia de trabajo de una CPU (llamada *freq_mhz* en Simics).

Este sistema de configuración que proporciona Simics da una gran flexibilidad, ya que la creación o borrado de objetos puede incluso realizarse dinámicamente, no teniendo que limitarse a una configuración estática desde el arranque del simulador.

Además, la configuración de una máquina puede realizarse de tres formas diferentes:

a) Mediante fichero *.conf* : es el método más básico de configuración de Simics utilizando las clases que proporciona Simics. Simics proporciona un lenguaje orientado a objetos para definir las configuraciones. Así, un fichero *.conf* contiene la descripción del conjunto de objetos que se van a utilizar en la máquina *target*. Cada objeto estará dentro de alguna clase y todos los objetos disponen de una serie de atributos que describen o configuran las características o el comportamiento del objeto en cuestión.

b) Mediante Python: también es posible definir configuraciones con este lenguaje de *scripting*. Simics dispone de funciones definidas en este lenguaje, de forma que se pueden inicializar objetos. Además, Python se puede utilizar en Simics para la creación de *scripts* para automatización de simulaciones, cambio de algunos parámetros de objetos en la simulación, etc.

c) Directamente desde la línea de comandos de Simics. La línea de comandos de Simics es una de las funciones implementadas por la API de Simics. Podrían introducirse los comandos *python* que hacen referencia a los objetos directamente e ir creando la configuración de la máquina paso a paso. De todos modos, la línea de comandos se suele utilizar, bien para modificar algún atributo de algún objeto, accediendo al espacio de configuración (*conf name-space*) o para enviar comandos al simulador, con propósitos de inspección, generación de perfiles (*profiling*), etc. utilizándose en la práctica *scripts* de configuración que lee el simulador para configurar una máquina.

A.1.3 Objetos de configuración

En Simics, los objetos de configuración (CPUs, memoria, dispositivos, etc) se organizan en los llamados *namespaces*. Las interfaces con estos namespaces, definen comandos que se invocan como *objeto.comando*. Por ejemplo:

Simics>@CPU0.print-status

Nos mostraría el estado actual del objeto CPU0, mediante el comando *print-status*.

O

Simics > phys_mem0.map

Nos muestra el mapeo de memoria del objeto phys_mem0

Los objetos de configuración son accesibles a través del *conf_namespace*, pudiendo modificar atributos de dichos objetos. De esta forma, si por ejemplo quisieramos modificar el contenido del registro EAX de la CPU0 lo haríamos de la siguiente forma:

Simics>conf.cpu0.eax=10

Habríamos cargado el registro EAX de la CPU0 con el valor entero 10.

A.1.4 Componentes

Otro concepto importante en Simics es el de componente. Un componente es la mínima unidad hardware que puede ser utilizada para configurar el hardware de la

máquina simulada. Ejemplos de componentes son un disco duro, una tarjeta de red, un chipset determinado, un backplane, etc. Los componentes se implementan normalmente utilizando varios objetos de configuración. Suponen un nivel de abstracción superior de manera que solo aparezcan objetos y atributos que tengan sentido en la configuración concreta que se esté simulando.

Por tanto, los objetos implementan la funcionalidad de uno o varios componentes.

Cuando se crea un componente, (para ello se utilizará el comando *create-<componente>*, estará en un estado no-instanciado, es decir, aunque el componente existe, no tiene asociado ningún objeto de configuración que implemente la funcionalidad. Por tanto, para que un componente pueda ser utilizado, este debe ser instanciado.

Al mismo tiempo existen componentes que pueden ser instanciados de modo *standalone*, como ocurre con los enlaces Ethernet.

A.1.5 Conectores

Los conectores en Simics proporcionan la funcionalidad necesaria para conectar unos componentes con otros. Un conector se puede definir como unidireccional o como bidireccional (*up, down o any)*. La dirección en el conector indica el nivel de jerarquía con que extiende el objeto que se conecta: por ejemplo, en el caso de un bus ISA, la conexión de un dispositivo ISA al un slot ISA se realizará mediante un conector con dirección *up*, al conectarse a un nivel de jerarquía superior. Salvo en el caso de componentes standalone, los conectores deben conectarse a los componentes antes de ser instanciados.

También existen conectores *hotplug*, que permiten conectar y desconectar componentes después de ser instanciados. Es el caso de la red Ethernet (*Ethernet link)* o de una consola serie (*uart)*. En la figura 2 se muestra la jerarquía existente en una configuración de Simics.

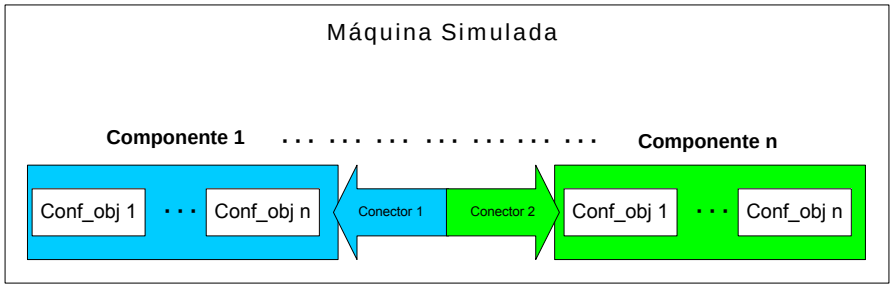


Figura A.1.1. Jerarquía de configuración en Simcs y conexión de componentes

A.1.6 Instanciación de componentes

Una vez que se crea una jerarquía de componentes (por ejemplo: bus pci → slot pci → dispositivo pci), estos deben ser instanciados para poder ser incluidos de forma funcional en la configuración de una máquina. Para ello se utiliza el comando *instansiate-components*. Este comando instancia todos los componentes que estén por debajo del que se pasa como argumento.

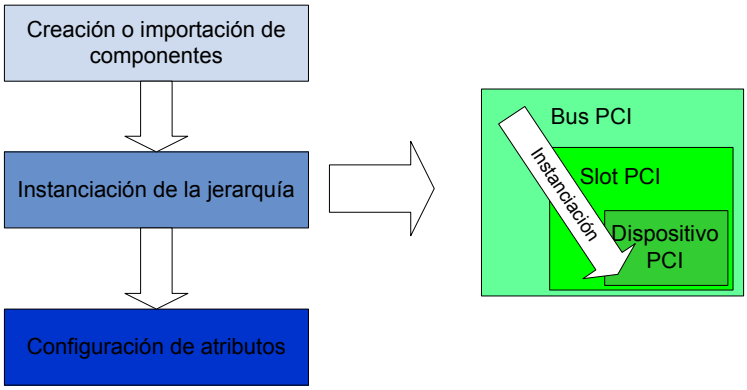


Figura A.1.2. Proceso de instanciación de objetos en Simics

A.1.7 Utilidades de Simics

Simics proporciona una serie de utilidades que no tienen otros simuladores, que lo hacen más versátil y por tanto, obtener obtener resultados de las simulaciones así como realizar cualquier desarrollo requiere menos tiempo. Además, es posible acceder mediante inspección a un gran nivel de detalle en la ejecución de las simulaciones.
Las principales utilidades de Simics son:

A.1.8 Checkpointing

La funcionalidad checkpointing de Simics permite guardar no solo la configuración de una máquina sino también el estado del hardware (contenido de los registros del procesador, contenido de las posiciones de memoria), etc. De esta forma, desde el punto de vista de simulaciones de sistema completo, se pueden disponer de varios *checkpoint* cada uno en un punto determinado de la simulación (por ejemplo, tras el arranque del sistema operativo, tras la ejecución de alguna tarea, etc). Esto facilita enormemente la fase de prueba de los dispositivos, en la que pueden producirse inestabilidades en el sistema.

A.1.9 HindSight *o simulación hacia atrás*

Esta utilidad permite volver hacia atrás un numero determinado de ciclos en una simulación. Mediante la creación de punteros de tiempo (*bookmarks*), se van marcando instantes de la simulación a los cuales es posible volver de forma inmediata. Un ejemplo claro de la utilidad HindSight se puede ver en el caso del borrado de un fichero en la máquina simulada. Podríamos volver hacia atrás, a un instante anterior al borrado del fichero.

Al mismo tiempo es posible activar la grabación de todos los eventos enviados desde el host al simulador y de este a la máquina simulada (por ejemplo, todas las pulsaciones del teclado o las coordenadas del ratón).

A.1.10 SimicsFS

Simics proporciona un sistema de ficheros llamado SimicsFS que permite montar en la máquina simulada (*target*) una partición existente en el host. De esta forma, se puede trabajar con la máquina simulada pero utilizando una partición del host, como si fuese una máquina real.

Para poder montar una partición SimicsFS es necesario disponer en la máquina simulada del kernel 2.4 ó 2.6 e instalar el módulo correspondiente que proporciona Simics.

A.1.11 Depuración

Mediante la funcionalidad *debugging*, es posible depurar desde el simulador el código que se esté ejecutando en la máquina simulada. Para ello, Simics necesita disponer de un mapeado entre direcciones y números del línea del código fuente del ejecutable que se está depurando, el cual lo obtiene directamente del ejecutable.

El módulo de Simics *symtable* contiene los comandos relacionados con la depuración simbólica, de forma, que una vez cargado el ejecutable o los ejecutables que queramos depurar en la *symtable*, disponemos de ciertas instrucciones especiales llamadas *instrucciones mágicas* (*magic instructions*), que serán de gran utilidad en la depuración. Las *magic instructions* no tienen efecto alguno en la máquina real, pero realizan acciones dentro de Simics. Un ejemplo de uso, es la detención de la simulación tras la ejecución de un *magic-break*. Una vez detenida la simulación, justo en el punto que hayamos decidido del código fuente, podemos realizar un desensamblado de la instrucción máquina en ese punto, volver hacia atrás, conocer el contenido de posiciones de memoria o registros del procesador, el estado de la pila o simplemente mover desde el simulador la posición del *magic-break*.

A.1.12 Seguimiento

Mediante la utilidad de *traceo* se puede ver lo que está ocurriendo en la máquina simulada, durante la ejecución de una simulación. Es posible observar el contenido de los registros del procesador (o solo observar los cambios), de posiciones de memoria, de la memoria *cache*, las llamadas al sistema, o la escritura/lectura de un dispositivo, en tiempo de ejecución.

A.1.13 Interfaz de línea de comandos de Simics (CLI)

Simics proporciona un interfaz de línea de comandos (*CLI*) que permite ejecutar comandos python, enviar comandos de control, inspección, generación de perfiles, visualización de estadísticas al simulador o ejecutar scripts (realizados en python).

Mediante los comandos de línea se puede hacer que la simulación se realice a nivel de instrucción, ciclo a ciclo, o que se ejecuten un número determinado de ciclos, por ejemplo. También pueden utilizarse comandos que permiten acceder al contenido de los registros del microprocesador (para leer o escribir en ellos), direcciones de memoria o contenido de los registros reservados para los dispositivos como la interfaz de red. Al mismo tiempo, se pueden visualizar estadísticas como por ejemplo, de la memoria *cache* (aciertos, fallos, etc.), paquetes recibidos o enviados a la red, etc. Además, pueden ejecutarse comandos de sistema operativo (solo comandos UNIX) en la máquina host directamente desde la línea de comandos.

Las funciones antes descritas, se activan, desactivan y controlan desde la línea de comandos.

A.1.14 API de Simics

Simics proporciona una API (*Application Programming Interface*) que permite el acceso a toda la funcionalidad de Simics desde un lenguaje de programación. Gracias al API de Simics podemos realizar *scripts* que automaticen simulaciones o describir configuraciones. Incluso es posible utilizar diferentes lenguajes como Python, C/C++, y aunque el API es único, los nombres de las funciones y su sintaxis difiere de un lenguaje a otro.

La API de Simics permite, tanto el acceso a toda la funcionalidad de Simics desde un lenguaje como C/C++ o de *scripting* como Python. Lenguajes como C/C++ permiten desde el modelado de dispositivos hasta la lectura de determinados parámetros de la simulación. Por otro lado, mediante Python, se pueden crear nuevos comandos en Simics, que permitan proporcionar nueva funcionalidad de la que Simics no disponga. Esto puede resultar especialmente útil a la hora de realizar una inspección muy concreta de alguna característica de la máquina simulada, creando comandos a medida que proporcionen exactamente la información que necesitemos.

La API de Simics permite el acceso a la funcionalidad ofrecida por los diferentes módulos de Simics:

A.1.15 API: funciones del núcleo del simulador

También conocidas como *core API*, son una serie de funciones nombradas como *SIM_* las cuales proporcionan el acceso a la funcionalidad del núcleo del simulador. Esto incluye:

- Acceso a los atributos de los objetos: Es posible leer o modificar el valor de los atributos de los objetos. Como cada atributo tendrá un valor asignado a una variable de un tipo determinado, existen diferentes funciones, para los diferentes tipos de atributos (por ejemplo: Bolean, list, integer, etc)
- Breakpoints: estas funciones permiten insertar breakpoints en la cola de tiempo, refiriéndose siempre a un número determinado de ciclos de reloj, de forma que se puede detener la simulación en un instante determinado. Como los breakpoints se insertan a nivel de ciclo, estas funciones son util cuando se trabaja con modelos de temporización o cuando se modifica el parámetro *cycle/step (ciclos por paso)*.
- Configuración: como alternativa a la utilización de configuraciones mediante ficheros. Por ejemplo, la lectura de una configuración desde un fichero. Además, las funciones de esta clase permiten crear objetos, instanciarlos y configurarlos, así como leer parámetros de su configuración o determinar la clase del objeto, etc.
- Errores y excepciones: devuelven el error producido o la excepción, por ejemplo, tras la ejecución de un determinado comando en un script Python.
- Acceso al log del simulador: Mensajes de log (podemos bien generar nuevos mensajes en el log o bien recoger lo que se esté mostrando)
- Memoria: es posible leer o escribir cualquier posición de memoria incluso bloques completos. También podemos tener información de que tipo de transacciones de memoria se están realizando en cada momento (si se trata de transacciones de datos o instrucciones, y si la transacción procede de una CPU, de la *cache*, de un dispositivo, etc.).
- Funciones de gestión de módulos: en Simics es posible cargar o descargar dinámicamente módulos, bien como librerías .dll en el caso de que el host se ejecute sobre Windows, bien mediante .py (ficheros python).

- Gestión de Simics: Se pueden modificar parámetros del simulador como por ejemplo, directorios de búsqueda de funciones, ficheros en general, etc.
- Procesador: mediante estas funciones se pueden realizar operaciones sobre el procesador: leer ó escribir un determinado registro, habilitar o deshabilitar un procesador, recoger excepciones, o desensamblar el código que se esté ejecutando, o que esté en caché.
- Control de las simulaciones: para poder arrancar, continuar o detener una simulación, al mismo tiempo que es posible saber si actualmente el simlador está corriendo o no. Es posible también vaciar el contenido de la caché de simulación (STC / Simics Internal *Cache*s).
- Stalling: Para poder dejar un procesador o una transacción paralizada durante un número de ciclos determinado, o hasta que ocurra un evento determinado (por ejemplo, la ejecución de una determinada instrucción)
- Control de pasos y eventos: estas funciones devuelven el número de ciclos que han pasado entre dos eventos, o los ciclos que faltan para la ocurrencia de un evento próximo.
- Funciones de interfaz de usuario: permiten acceder al interfaz de usuario desde un programa. De esta forma se pueden ejecutar comandos del mismo modo que se estos se hubiesen introducido desde el teclado en la línea de comandos.

A.1.16 API: Funciones PCI

Las funciones PCI del API de Simics están pensadas para facilitar a tarea de describir nuevos dispositivos PCI. Mediante estas funciones, se pueden controlar las transferencias que queramos realizar hacia o desde el bus PCI, por ejemplo, envío de mensajes, interrupciones, registros de configuración, etc.

Más adelante, se describirá de forma resumida, la manera de escribir nuevos dispositivos PCI en Simics.

A.1.17 Modelos de temporización y ejecución de instrucciones

Simics es un simulador dirigido por eventos, con una resolución máxima de un ciclo de reloj. A continuación se definirán conceptos que se manejan Simics y que son importantes para comprender el funcionamiento del simulador.

A.1.18 Conceptos de evento, paso y ciclo

- **Ciclo:** Es la resolución máxima en una simulación. La duración de un ciclo de reloj es configurable mediante el parámetro *freq_mhz* el cual hace referencia a la frecuencia en MHz del reloj del procesador. El utilizar frecuencias de reloj demasiado altas, puede provocará que la simulación genere una carga de trabajo excesiva para el host y que la simulación sea muy lenta.
- **Evento**: Se entiende por evento en Simics, un suceso tal como la ejecución de una instrucción, una interrupción o una excepción.
- ***Step* (Paso):** En Simics se llama paso a una instrucción cuya ejecución se ha completado, una instrucción que ha generado una excepción o una interrupción procedente de algún dispositivo.

Los conceptos anteriores están íntimamente relacionados con el tiempo de ejecución de una instrucción. No es trivial en Simics conocer el tiempo de ejecución sobre todo si se utilizan modelos de temporización o si se están simulando arquitecturas multiprocesador: en el caso de arquitecturas con un solo procesador, el tiempo de ejecución de una instrucción vendrá determinado por el modo de ejecución y las interfaces de medición de tiempo. Sin embargo, en el caso de un sistema multiprocesador, para mejorar el rendimiento de las simulaciones, desde el punto de vista del funcionamiento interno del simulador, las instrucciones se ejecutan en los procesadores uno a uno, es decir, se asigna una ranura de tiempo a cada procesador, en la cual, solo un procesador podrá ejecutar instrucciones. Después, continúa el siguiente procesador, es su ranura temporal. La duración de la ranura temporal asignada a cada procesador se puede configurar y vendrá expresada en ciclos de reloj (recordemos que en el simulador, el mínimo tiempo medible es un ciclo). Al mismo tiempo, la duración de esta ranura no tiene por qué ser igual en todos los procesadores.

Como ya se ha comentado, Simics es un simulador dirigido por eventos, con una resolución máxima de un ciclo de reloj. Estos eventos, que son ejecutados conforme avanza el tiempo, ciclo a ciclo, incluyen interrupciones de dispositivos, actualizaciones de estado, o ejecuciones de pasos. Un paso es la unidad en la que se divide en Simics la ejecución de un flujo de instrucciones hacia el procesador. Por tanto, los pasos son los eventos más comunes.

Es importante, para comprender el funcionamiento de Simics, distinguir entre ciclos y pasos. En este sentido, Simics diferencia dos tipos de eventos, según estén asociados a un ciclo o a un paso específico. Para cada ciclo, los eventos se disparan en el orden siguiente:

1) Todos los eventos destinados a ese ciclo, excepto los pasos
2) Para cada paso previsto para ejecución en este ciclo:
    2.a) Todos los eventos previstos para este paso específico
    2.b) La ejecución de un paso

Todos los eventos que pertenecen a la misma categoría son ejecutados en orden FIFO.

Simics soporta dos modos de funcionamiento, en cuanto a la temporización en la ejecución de las instrucciones: ejecución en-orden y ejecución fuera de orden, como se puede ver en la figura 2.

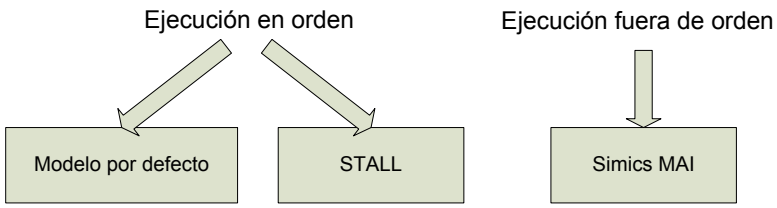


Figura A.1.3. Modos de ejecución de Simics

El modelo usado por defecto en Simics es el que implementa la ejecución en orden o *normal*. En este modelo, las instrucciones se ejecutan discreta y secuencialmente. En este modo, normalmente se ejecuta un paso por cada ciclo, aunque como después se verá, es posible modificar la razón pasos/ciclo.

A.1.19 Modos de Simulación o ejecución del simulador

Los modos de simulación o de ejecución en Simics se utilizan para controlar ciertas características de la simulación que afectarán al comportamiento de la máquina simulada: perfiles de datos e instrucciones, modelos de temporización de la memoria y la interfaz de micro-arquitectura (*Micro-Architectural Interface MAI*). Simics dispone de tres modos de simulación o ejecución:

- Normal
- Micro Architecture
- Stall

La implementación eficiente del simulador, hace que para cada modo, exista un modelo de procesador diferente. Esto hace que el modelo de procesador utilizado esté optimizado para cada modo de simulación, haciéndola lo más eficiente posible.
El modo más rápido es el llamado *Normal*, el cual no implementa ningún tipo de modelo de temporización. Por tanto, no es posible utilizar modelos de temporización conectados a ningún elemento (normalmente CPUs o memoria) cuando Simics esté corriendo en este modo. Además al tratarse de modelos diferentes, implementados por código diferente en el núcleo de Simics, no es posible cambiar de modo de simulación en tiempo de ejecución.

El modo que normalmente utilizaremos cuando queramos conectar modelos de temporización a una CPU o a la memoria será el modo *Stall*.

A.1.20 Simulación en modo micro-arquitectura

El modo micro-arquitectura es un modo de simulación que soporta la simulación de micro-arquitecturas de procesadores. Actualmente, esta totalmente desarrollado para procesadores SPARC y dispone de un soporte limitado en el caso de procesadores x86 ó x86-64. Este modo es el que más carga de trabajo supone para el host durante la

simulación, y solo se suele utilizar cuando se está trabajando sobre la micro-arquitectura del procesador. Sin embargo, describiremos brevemente en qué consiste dada su gran relevancia a la hora de utilizar Simics para investigar en nuevas micro-arquitecturas o estudiar los efectos de la modificación de las existentes.

Los modelos de procesadores que incluye Simics, son funcionalmente muy parecidos a los procesadores reales. Es por esto por lo que pueden construirse modelos de simulación con máquinas que pueden ejecutar sistemas operativos comerciales y aplicaciones sin modificación alguna. Sin embargo, para conseguir que un sistema operativo comercial o que una aplicación se pueda ejecutar en una máquina, normalmente no es necesario disponer de un modelo de temporización exactamente igual al de la máquina real; solo se requiere la funcionalidad de la máquina real.

Simics es un simulador funcional, que considera la ejecución de un evento (instrucción, expeción, interrupción, etc) como una operación única en el proceso de simulación, tomando exactamente para ello, un ciclo de reloj. Sin embargo, esto puede no ser suficiente en algunos casos, por lo que sería muy conveniente, poder modificar en Simics los modelos de temporización, de forma que podamos tener control sobre por ejemplo, cuantos ciclos de reloj requiere una determinada transacción de memoria, o con cuantos ciclos se penaliza un fallo de memoria *cache*, etc. Simics proporciona un módulo que permite introducir modelos de temporización: el *MAI* (*Micro Architectural Interface*).

El concepto que introduce *MAI* es la capacidad de poder decidir cuando va a ocurrir un evento, y la posibilidad de retrasarlo cierto número de ciclos, esperar a que otro evento ocurra, etc. En el caso de un procesador, esto se traduce en introducir un módulo que modifique la temporización del mismo, de forma que la ejecución ya no sea una única operación a nivel de simulación sino que mediante este módulo se pueda modificar cuando cargar la instrucción, cuando decodificarla, cuando ejecutarla, cuando cargar datos de memoria, etc. Utilizando la terminología de Simics, se trata de poder controlar la progresión de las diferentes fases en las que queda dividida la ejecución de una instrucción, la cual vendrá delimitada por los eventos de inicio y de finalización. (Por tanto, podemos tener un control total sobre todas las acciones que van implícitas en un ciclo de reloj, en las diferentes fases de la ejecución de una instrucción. Siempre, debe ser posible la ejecución de Sistemas operativos o aplicaciones sin modificar, por lo que en el caso de que hayamos introducido un módulo de temporización que vaya a

realizar alguna operación ilegal, Simics avisará de ello. Por tanto, MAI es necesario para simular modelos de cpu con ejecución fuera de orden.

Es posible introducir modelos de temporización tanto en los procesadores x86 como en los SPARC, aunque la interfazMAI está mucho más maduro para los procesadores SPARC.

Simics además, no solo permite la modificación de las diferentes fases en la ejecución de una instrucción, sino que da la posibilidad de introducir nuevas fases, añadir modelos de segmentación de cauce, etc. Esto es muy importante en el estudio de nuevas arquitecturas de procesador, aunque una modificación muy grande en el comportamiento del mismo, que derive en una diferencia a nivel funcional hará posiblemente, que no sea posible la ejecución de software comercial.

Para la implementación de MAI, Simics proporciona una API, con diferentes funciones que hacen posible implementar la funcionalidad antes descrita. Existe una función llamada manejador de ciclo o *cycle handler* que se invoca con cada ciclo de reloj, y que contiene el código que modela la microarquitectura del procesador.

En este modo de ejecución, no solo puede controlarse el número de ciclos que se necesitará para ejecutar cada paso (*step_rate*), sino que dispondremos de un control total sobre la ejecución de los pasos. Aunque los eventos asociados a un ciclo de reloj se ejecutan de una vez en cada ciclo, no se ejecutará un paso a menos que lo permita el modelo de temporización conectado al procesador. Este modelo de temporización, por tanto, será llamado una vez cada ciclo.

El modo de ejecución *micro-architecture* proporciona las herramientas necesarias para simular correctamente la ejecución paralela y/o especulativa de instrucciones, a la vez que da la posibilidad de introducir modelos de temporización y así poder simular con precisión modelos de procesadores complejos.

La posibilidad de simular una ejecución de instrucciones paralela y especulativa, hace que a este modo se le llame también modo de ejecución *fuera de orden*. Para la ejecución fuera de orden, el código que implementa el comportamiento de la CPU deberá realizar un seguimiento de todas las dependencias en el flujo de instrucciones, de forma que no se realicen operaciones en un orden erróneo. El ejemplo típico es la dependencia entre registros internos de la CPU.

El inconveniente de utilizar este modo de simulación, es la lentitud de la misma. Si utilizamos modelos muy precisos, que permitan tener un control exhaustivo de la temporización, la simulación será muy lenta.

Dependiendo de la aplicación que se esté ejecutando en la máquina *target*, el utilizar un modo demasiado preciso puede hacer que la simulación inviable. El escoger un modo adecuado, será alcanzar un correcto equilibrio entre precisión y velocidad.

A.1.21 Modo de ejecución *stall*

En este modo, la ejecución en orden del modelo por defecto o normal, puede extenderse conectando modelos de temporización. El modo de ejecución stall es el que permite esto. Añadir modelos de temporización hace posible el contro de las operaciones de acceso a memoria. La diferencia fundamental de funcionamiento al introducir modelos de temporización, está en que ahora los pasos no son operaciones indivisibles y que se realizan de una vez, cada ciclo y cuya ejecución no ocupa tiempo en la máquina simulada. En el modo stall, un paso en el que se ejecute un acceso a memoria, se puede paralizar durante un número determinado de ciclos de reloj. De esta forma conseguiremos que la memoria trabaje a una velocidad diferente a la CPU. Sin embargo, los eventos asociados a un ciclo se seguirán produciendo, ya que los ciclos de reloj siguen avanzando. Por tanto, aunque los pasos se siguen ejecutando de una vez, es posible, utilizando diferentes modelos de temporización conectados a los diferentes elementos de la arquitectura, hacer que pasos asociados a diferentes eventos necesiten un tiempo diferente para ser ejecutados.

Es importante notar que al utilizar Simics en modo stall, la ejecución de instrucciones máquina sigue produciéndose en orden.

A.1.22 Step Rate

El *step_rate*, también llamado IPC (Instrucciones por ciclo), indica en Simics, el número de pasos que se ejecutarán cada ciclo, o lo que es lo mismo, el número de ciclos de reloj que es necesario esperar para que se complete la ejecución de paso. La configuración del *step_rate* es totalmente independiente de los modelos de

temporización, y se corresponde a un concepto diferente, que tendrá efectos diferentes en la máquina simulada.

La configuración del *step_rate* se realiza mediante 3 parámetros:

- p : Pasos
- q: Ciclos
- r: Fase relativa entre los contadores de pasos y ciclos. Permite tener precisión de más de un ciclo de reloj.

Por ejemplo:

```
@conf.<objeto>.step_rate=[p,q,r]
```

Donde r=0 normalmente, ya que habitualmente, no es necesario disponer de una precisión de más de un ciclo.

A.1.23 Sistema de memoria de Simics

En un sistema simulado mediante Simics, las transacciones de memoria pueden ser ordenadas por una CPU o un dispositivo. Cuando una CPU solicita acceso a una dirección virtual, esta es traducida mediante una unidad de gestión de memoria (MMU) para obtener una dirección física. Al mismo tiempo, las direcciones físicas se organizan en Simics mediante los llamados espacios de memoria (*memory-spaces*). Un espacio de memoria representa una dirección física para un dispositivo que puede aceptar una transacción de memoria, como por ejemplo, un módulo de memoria RAM. Dentro de cada espacio de memoria se pueden definir otros, definiendo una *jerarquía de memoria*. Esto es útil por ejemplo, para definir dos tipos dentro de la memoria física: un espacio para E/S y otro para memoria.

Para que las transacciones de memoria dentro del simulador no tengan necesariamente que provocar acceso a espacios de memoria, lo cual afectaría a la velocidad de las simulaciones, se implementa en Simics el llamado *STC (Simulator Translation Cache)*, haciendo más eficiente el sistema de acceso a memoria. El

concepto es el mismo de cualquier *cache* de memoria: se dejan en una memoria *cache* información relativa a las operaciones o datos que se realizan o acceden con más frecuencia. La implementación del STC en Simics es una de las características que hacen posible una simulación más rápida. Sin embargo, tal y como se describe en el capítulo 4, dependiendo del tipo de simulación puede ser necesario desactivar el STC, debido a que su utilización puede llevar a la obtención de resultados no realistas, sobre todo si se están utilizando modelos de tiempo, ya que las transacciones realizadas a través de STC no están controladas por dichos modelos.

Para que una dirección de memoria concreta se introduzca en el STC, se deben cumplir 3 condiciones:

- Que el mapeo lógico-físico sea válido
- Que un acceso a dicha dirección no afecte al estado de la MMU
- Que no haya breakpoints, callbacks, etc. en dicha dirección.

El comportamiento de STC se puede modificar para objetos a los que se haya conectado un modelo de temporización.

Por defecto, el modelo de memoria que utiliza Simics hace que las transferencias se realicen sin impacto alguno en la temporización, es decir, los accesos a memoria tienen una latencia de 0 ciclos de reloj. Sin embargo, Simics da la posibilidad de crear una jerarquía de memoria así como de conectar modelos de temporización que permitan detener los accesos a memoria durante un número determinado de ciclos.

Los espacios de memoria en Simics, están representados por los objetos de la clase *memory-space*. Dichos objetos implementan las funciones necesarias para la simulación de accesos a memoria, y disponen de los atributos necesarios para realizar el mapeo. Además, los espacios de memoria se pueden definir como memoria RAM, ROM, Flash, o de otro tipo. Además, debe incluirse en el atributo *target* la lista de los objetos que podrán tener acceso a este espacio de memoria. Un ejemplo de definición de espacio de memoria en python podría ser:

```
@mem.map = [[0x00000000, conf.ram0,    0, 0,         0xa0000],
[0x000a0000, conf.vga0,    1, 0,         0x20000],
[0x000c0000, conf.rom0,    0, 0,         0x10000],
[0x000f0000, conf.rom0,    0, 0,         0x10000],
```

```
[0x00100000, conf.ram0,    0, 0x100000, 0xff00000],
[0xfee00000, conf.apic0,   0, 0,          0x4000],
[0xffe81000, conf.hfs0,    0, 0,            16],
[0xffff0000, conf.rom0,    0, 0,         0x10000]]
```

donde se indica por orden: base, objeto al que se mapea, ID para este mapeo en el objeto, offset, longitud.

Además, existen otros atributos opcionales para configurar parámetros como el objeto target o la prioridad del mapeo.

Como con otros objetos, existen comandos que muestran el mapeo actual para un espacio de memoria.

Ejemplo: Simics> phys_io.map

Muestra el mapeo actual del objeto phys_io.

Es importante destacar que el objeto *target* de un espacio mapeo de memoria puede ser a su vez otro espacio de memoria.

A.1.24 Simulación de redes en Simics

Una característica indispensable en un simulador para la realización de las simulaciones que necesitamos en nuestro trabajo, es la capacidad de interconectar diferentes máquinas simuladas *target* mediante una red de interconexión. Simics proporciona las herramientas necesarias para la interconexión de dos o más máquinas en una red local.

Para ello, existe el módulo *Ethernet-central*, que implementa la red Ethernet simulada. Una vez arrancado dicho módulo, la red estará disponible y podremos conectar las diferentes máquinas que estemos simulando, y que se haya provisto en su configuración de un interfaz de red Ethernet, mediante el comando correspondiente.

Además disponemos del módulo *Simics-Central* que además de gestionar la red simulada, actúa como *gateway*, tanto para el caso en que queramos conectar la red simulada a una red real (puente con el host) como para conectar máquinas simuladas que estén corriendo en diferentes instancias de Simics.

Una característica importante de Simics es la capacidad de poder simular diferentes máquinas sobre una misma instancia del simulador. Esto hace posible entre

otras cosas, que no sea necesario el módulo *Simics-Central* para interconectar las máquinas. El problema de utilizar *Simics-Central* es la latencia añadida, que limitará las prestaciones de la red y además, no estaremos simulando realmente un enlace punto a punto.

En nuestro caso, como lo que pretendemos analizar es el rendimiento tanto del protocolo de red como de la interfaz de red, con sistemas que presenten diferentes arquitecturas, es imprescindible el poder simular conexiones punto a punto, sin elementos inermedios que puedan suponer un cuello de botella en el camino de comunicación.

Como se ha comentado, será necesario en las máquinas simuladas, conectar componente interfaz de red. Simics proporciona modelos para diferentes interfaces de red tanto ISA como PCI, con anchos de banda diferentes, desde 10 Mbits/s hasta 1 Gbits/s.

Aunque en nuestro caso, utilizaremos redes Ethernet, Simics es capaz de simular enlaces entre máquinas a través, por ejemplo, de un enlace serie RS-232.

A.1.25 Temporización del enlace. Latencia.

El tiempo que transcurre hasta que se envía un paquete o entre el envío de dos paquetes a la red, se conoce como latencia del enlace. Esta latencia se puede configurar en Simics de forma individual para cada objeto de enlace (en nuestro caso, un *Ethernet-link*). Los objetos de enlace se utilizan para conectar entre si dos dispositivos de red. Debido a la forma en que Simics maneja el *tiempo simulado*, dichos dispositivos de red utilizan relojes diferentes y por tanto, no siempre estarán totalmente sincronizados. Esto puede provocar comportamientos extraños en el comportamiento de la red simulada, y una simulación no-determinista, si se utiliza una latencia lo suficientemente baja como para que un paquete puesto en la red llegue al destinatario en un instante de tiempo que ya ha pasado.

Una forma de evitar esto, es ejecutar las máquinas que se estén simulando sobre la misma instancia de Simics. De esta forma, podremos utilizar latencias bajas sin tener comportamientos extraños. Además, en este caso, Simics ajustará la latencia del enlace con la mínima latencia de red que no provocará comportamientos extraños.

Apéndice II

Modelado del tiempo en Simics

A.2 Modelo de tiempo

Los modelos de tiempo permiten extender Simics de forma que no se comporte sólo como un simulador funcional, sino que sea posible la simulación de latencias en los accesos a memoria o a los dispositivos.

El modelo de tiempo de desarrollado supone una extensión del modelo simple *trans_staller* de Simics, haciendo posible la simulación de latencias en los accesos a memoria y a dispositivos PCI, así como simular la contención en el puente norte.

A.2.1 Modulo *cont_staller*

El modelo de tiempo esta implementado por el módulo *cont_staller*. Una vez cargado en Simics, es posible conectarlo a cualquier componente que disponga de interfaz con modelos de tiempo. No obstante, el módulo *cont_staller* está pensado para ser conectado a CPU, memoria o dispositivos PCI.

A.2.2 Configuración del modelo de tiempo

El modelo de tiempo dispone de los siguientes atributos:

- *stall_time*: latencia en número de ciclos en los accesos a memoria, para transacciones entre la CPU y el nivel de la jerarquía de memoria al que esté conectado el modelo de tiempo.
- *cont_time*: número de ciclos de penalización en el caso de colisión en el puente norte.

- *pci_time*: latencia en número de ciclos en los accesos al bus PCI.

A.2.3 Instalación del modelo de tiempo

1.a) Instalación en Windows: Instalación de la librería dinámica *cont_staller.dll* y configuración del *workspace*.

1.b) Instalación en Linux / Unix: Instalación de la librería dinámica *cont_staller.o* y configuración del workspace.

2) Una vez arrancado Simics cargar el módulo:

```
load-module cont_staller
```

A.2.4 Niveles de log implementados

Con el fin de poder realizar un seguimiento de las operaciones llevadas a cabo por el modelo de tiempo, se han implementado 4 niveles de log:

- Nivel 1: Información de carga del modelo
- Nivel 2: Información sobre accesos desde PCI
- Nivel 3: Información sobre accesos desde CPU y colisiones
- Nivel 4: Información de todos los accesos en los que se llama al modelo de tiempo.

A.2.5 Esquema de funcionamiento del modelo de temporización

En la Figura A.2.1 se muestra el esquema de funcionamiento del modelo de temporización, que se describió con más detalle en la sección 3.4.1.

De la misma forma, en la Figura A.2.2 se muestra el modelado de la contención en el puente norte, debido a los accesos de las CPU y de los dispositivos PCI.

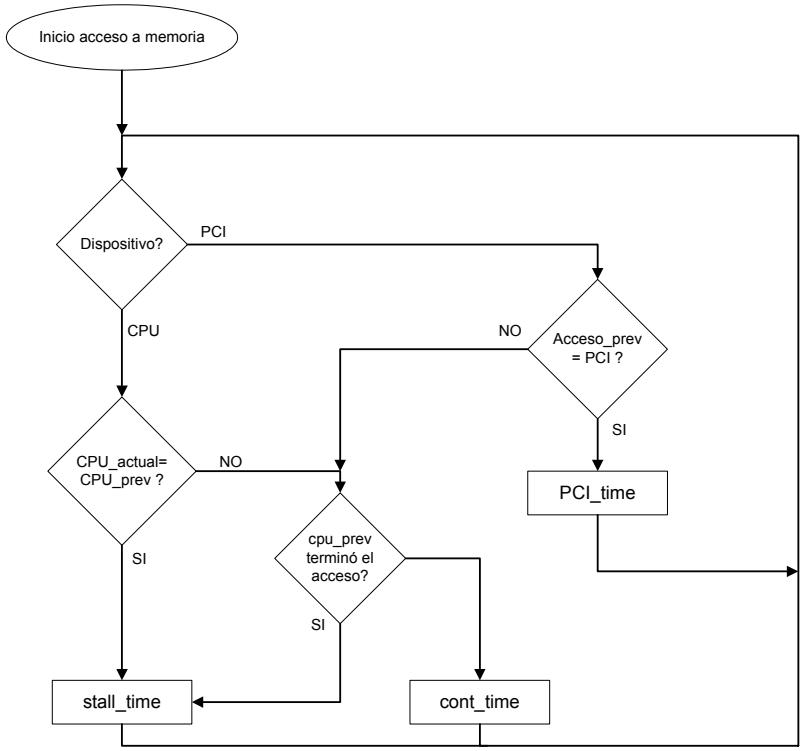


Figura A.2.1 Esquema de funcionamiento del modelo de tiempo

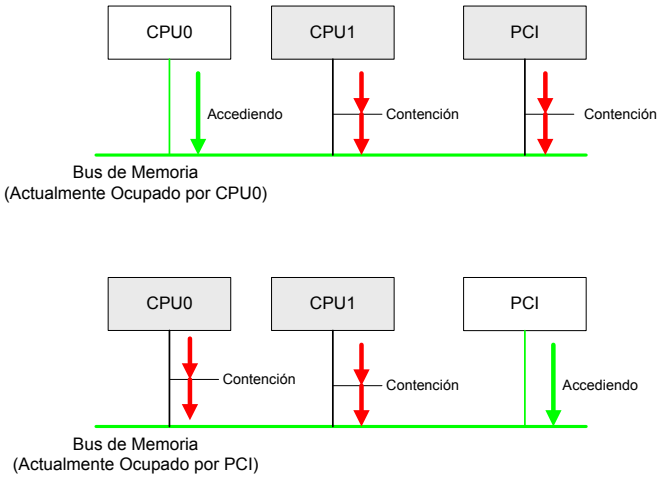


Figura A.2.2. Modelado de la contención

Apéndice III

Funciones de comunicación TCP/IP en Linux

Elemento / Librería → Función	***Tarea***	***Offloading***	***Onloading***	***Híbrido***
NIC	NIC recibe paquete	Firmware NIC	Firmware NIC	Firmware NIC
NIC - DMA	NIC transfiere paquete a memoria principal	Firmware NIC Transferencia del anillo interno (128 Kbytes)	Firmware NIC	Firmware NIC
IRQ a CPU *do_irq()*	Llamada al driver correspondiente dependiendo del nº de IRQ	Driver CPU0 Contexto interrupción	Driver CPU1 Contexto interrupción	Driver CPU1 Contexto interrupción
Driver → /net/core/dev.c → netif_rx()	El driver llama a netif_rx(), recibe un paquete y lo almacena en una estructura sk_buff. Se encola para el procesamiento mediante softIrq	Driver CPU0 Contexto interrupción	Driver CPU1 Contexto interrupción	Driver CPU1 Contexto interrupción
/include/linux/netdevice.h → netif_rx_schedule()	Añade el dispositivo a la lista softnet_Data (poll_list para rx). cpu_raise_softirq → NET_RX_SOFTIRQ	Driver CPU0 Contexto interrupciónerrupción	Driver CPU1 Contexto interrupción	Driver CPU1 Contexto interrupción
NET_RX_SOFTIRQ /net/core/dev.c → net_rx_action()	Inicia softIrq *net_rx_action()* para procesar el paquete (Procesamiento IP → TCP) Llama netif_receive_skb() para cada paquete	Núcleo Línux CPU1 Contexto interrupción	Núcleo Línux CPU1 Contexto interrupción	Núcleo Línux CPU2 Contexto interrupción

Elemento / Librería → Función	***Tarea***	***Offloading***	***Onloading***	***Híbrido***
/net/core/dev.c → process_backlog	Desencola los paquetes para ser procesados por netif_receive_skb()	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/net/core/dev.c → netif_receive_skb()	Rutina principal de recepción dentro de NET_RX_SOFTIRQ. Llama al gestor de paquetes registrado dependiendo del payload. i.e.: ip_recv()	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/net/ipv4/up_input-c →ip_revc()	Rutina principal de recepción de paquetes IP llamada desde netif_receive_skb(). Llama a las funciones de enrutamiento net/ipv4/route.c	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/net/ipv4/ip_input.c → ip_rcv_finish()	Llamada a tcp_v4_rcv() una vez que el paquete es reconocido como TCP	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/include/net/protocol.c → tcp_v4_rcv()	Comprobación cabecera TCP	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half

Elemento / Librería → Función	***Tarea***	***Offloading***	***Onloading***	***Híbrido***
/include/net/protocol.c → tcp_v4_do_rcv()	Rtuina de recepción del paquete TCP. Depende del estado de la conexión	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/incluye/net/protocol.c → tcp_data()	Pone el buffer sk_buff en una lista para ser procesado	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
/include/net/protocol.c → data_Ready()	Informa de que hay datos listos y los copia al socket: *Read (socket, data, len)*	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU1 Bottom-half	Núcleo Linux CPU2 Bottom-half
Sys_read() → sock_Read() → inet_rcvmsg() → tcp_rcvmsg()	Llamada al sistema para copiar los datos del espacio de núcelo al espacio de usuario	Núcleo Linux CPU0 Contexto proceso	Núcleo Linux CPU0 Contexto proceso	Núcleo Linux CPU1 Contexto proceso

Referencias

[ALA03] Alameldeen, A.R. et al.: “Simulating a $2M Comercial Server on a $2K PC”. IEEE Computer, pp.50-57. February, 2003.

[ALL03] Allman, A.: “TCP Congestion Control With Appropiate Byte Counting (ABC)”. RFC3465. http://www.ietf.org/rfc3465.txt

[BAL04] Balaji, P.; Shah, H.; Panda, D.K.: “Sockets vs RDMA Interface over 10-Gigabit Networks: An in-depth Analysis of the Memory Traffic Bottleneck”. Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2004). Sept 20th, 2004, in conjunction with the IEEE Cluster 2004, San Diego, California.

[BAS05] Basto, V.; Freitas, V.: “Distributed QoS Multimedia Transport”. Proceedings of the 1st IEEE Internacional Conference on Distributed Frameworks for Multimedia Applications. (DFMA’05).

[BAT06] Bhattacharya, S.P.; Apte. V.: “A measurement study of the Linux TCP/IP stack performance and scalability on SMP systems”. Proceedings of the 1st International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE), New Delhi, India, January 2006.

[BEO07] http://www.beowulf.org

[BER06] Berg, E.; Zeffer, H.; Hagersten, E.: “A Statistical Multiprocessor *Cache* Model”. IEEE International Symposium Performance Analysis of Systems and Software. March, 2006.

[BEN05] Benvenuti, C.: “Understanding Linux Kernel Internals”. O’Reilly Media Inc. 2005

[BHO98] Bhoedjang, R.A.F.; Rühl, T.; Bal, H.E.: "User-level Network Interface Protocols". IEEE Computer, pp.53-60. Noviembre, 1998.

[BIN03] Binkert, N.L.; Hallnor, E.G.; Reinhardt, S.K.:”Network-oriented full-system simulation using M5”. Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CECW). February, 2003.

[BIN05a] Binkert, N.L.; Hsu, L.R.; Saidi, A.G.; Dreslinski, R.G.; Schultz, A.L.; Reinhardt, S.K.: “Analyzing NIC overheads in Network intensive

workloads". Proceedings of the 8th workshop on computer architecture evaluation using commercial workloads (CAECW'05), 2005.

[BIN05b] Binkert, N.L.; Hsu, L.R.; Saidi, A.G.; Dreslinski, R.G.; Schultz, A.L.; Reinhardt, S.K.: " Performance Analysis of System Overheads in TCP/IP Workloads". 14th International Conference on Parallel Architectures and Compilation Techniques . 2005. (PACT 2005).

[BIN06] Binkert, N.L.; Saidi, A.G.; Reinhardt, S.K.: "Integrated network interfaces for high-bandwidth TCP/IP]. Proceedings of the 12th International Conference con Architectural Support for programming languajes and operating systems. 2006.

[BOI07] Berkeley Open Infraestructure for Network Computing. http:/boinc.berkeley.edu

[BOV05] Bovet, D.P.; Cesati, M.; "Understanding The Linux Kernel". O'Reilly Media Inc. 2005.

[BRA04] Branovic, I.; Giorgi, R.; Martinelli, E.: "WebMIPS: a new web-based MIPS simulation environment for computer architecture education". Proceedings of the 2004 workshop on Computer architecture education. 2004

[BRE06] Brecht, T.; Janakiraman, G.; Lynn, B.; Saletore, V.; Turner, Y.:"Evaluating network processing efficiency with processor partitioning and asynchronous I/O". Proceedings og the 2006 EuroSys Conference. 2006

[BRO07] http://www.broadcom.com, 2007

[BUL04] http://www.bullopenource.org/cpusets

[CAR00] Cardwell, N.; Savage, S.; Anderson, T.: "Modeling TCP latency". IEEE Conference on Computer Communications. Infocom 2000.

[CEB04] Celebioglu, O.: "Optimizing Linux Cluster Performance by Exploring the correlation between Application Characteristics and gigabit Ethernet Device Parameters". Dell Enterprise Solutions Engineering. 2004

[CHA01] Chase, J.S.; Gallatin, A.J.; Yocum, K.G.: "End System Optimizations for High-Speed TCP". IEEE Communications Magazine. pp. 68-74. April, 2001.

[CHA03] Chakraborty, S. et al.:"Permance evaluation of network processor architectures: combining simulation with analytical estimation". Computer Networks, 41, pp.641-645, 2003.

[CHE06] "Time for TOE. The benefits of 10 Gbps TCP Offload". Chelsio Communications. 2006

[CHE07] http://www.chelsio.com, 2007

[CHU96] Yi-Chung, C.; Teorey, T.K: "Modeling and analysis of the Unix communication subsystems". Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research.

[CHU00] Chuanxiong, G.; Shaoren, Z.: "Analysis and Evaluation of the Protocol Stack of LINUX". Proceedings of the International Conference on Communication Technology. 2000. (WCC-ICCT, 2000).

[CIA01] Ciaccio, G.: "Messaging on Gigabit Ethernet: Some experiments with GAMMA and other systems". Workshop on Communication Architecture for Clusters, IPDPS, 2001.

[CLA89] Clark, D.D. et al.:"An analysis of TCP processing overhead". IEEE Communications Magazine, Vol. 7, No. 6, pp.23-29. Junio, 1989.

[CLA01] Claffy, K.; Miller, G.; Thompson, K.: "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone". Proceedings of the INET Conference. 1998.

[COL05] Collier-Brown, D.: "Creating self-balancing solutions with Solaris containers". Sun Blueprints, 2005.

[CRU91] Cruz, R.L.:"A calculus for network delay". IEEE Trans. on Information Theory, Vol.37, No.1, pp.114-141, 1991.

[CUL93] Culler, D.; Karp, R.; Patterson, D.; Sahay, A.; Schauser, K.E.; Santos, E.; Subramonian, R.; Eicken, V.T.: " LogP: Towards a realistic model of parallel computation". Proceedings 4th ACM SIGPLAN Symposium on Principles and Practice of Paralell Programming, May 1993.

[DAV89] Clark, D.; Jacobson, V.; Romkey, J.; Salwen, H.: "An Analysis of TCP Processing Overhead". IEEE Communications Magazine. June 1989. pp. 23-29.

[DEA95] Dean, T.; Eggers, S.; Levy, H.: "Simultaneous multithreading: Maximizing On-Chip Parallelism". Proceedings of the 22rd Annual International Symposium on Computer Architecture, June, 1995.

[DEL06] Dell Technology white paper: " Boosting data transfer with TCP Offload Engine Technology", Agosto 2006.

[DIA03] Díaz, A. F.; Ortega, J.; Cañas, A.; Fernández, F.J.; Anguita, M.; Prieto, A.: ”A Light Weight Protocol for Gigabit Ethernet”. Workshop on Communication Architecture for Clusters (CAC’03) (IPDPS’03). April, 2003.

[DIA05] Díaz, A. F.; Ortega, J.; Ortiz, A.; Prieto, A.: “Análisis de la extgernalización de protocolos mediante simulación HDL”. XVI Jornadas de Paralelismo. Granada. Septiembre 2005.

[DIN02] Dinda, P.; : “The Minet TCP/IP Stack”, Northwestern University Department of Computer Science Technical Report NWU-CS-02-08, January, 2002

[DML07] DML Reference Manual. Virtutech, 2007.

[DOL07] Dolphin Interconect Solutions http://www.dolphinics.no

[DOV01] Dovrolis, C.; Thayer, B.; Ramanathan, P.: “HIP: Hybrid interrupt-polling fir the network interface”. ACM SIGOPS Operating Systems Review, 35(4):50—60. October 2001.

[DUN98] Dunning, D; Regnier, G.; McAlpine, G.; Cameron, D.; Shubert, B.; Berry, F.; Merrit, A.M.; Gronke, E.; Dodd, C.: “The Virtual Interface Architecture”. IEEE Micro.Vol. 18, issue 2.pp. 66-76. Mar. 1998

[DUN08] Dunkels, A.: “lwIP TCP/IP stack”. http://www.sics.se/~adam/lwip/index.html

[DWO97] Dwork, C.; Herlihy, M.; Waarts, O.: “Contention in shared memory algorithms”. Jornal of ACM (JACM). 44 (779-805). November 1997.

[ENG03] Engblom, J. “Full-System Simulation”. European Summer School on e mbedded Systems. (ESSES'03), 2003.

[ERI07] Ericsson. http://www.ericsson.com

[EST99] Estándar IEEE 802.3 http://ieee802.org/3

[EST06] Estándar IEEE 802.3ae

[ETE07] IETF RDDP working group documents: http://www.ietf.org/ids.by.wg/rddp.html

[FAR00] Farnell, P.A.; Ong, H.: “Communication performance over a gigabit ethernet network”. 19th IEEE International Performance, Computing and Communications conference. IPCC 2000. February 20-22, 2000

[FIG96] Figueira, S.; Berman, F.: :" Modeling the effects of Contention Performance of Heterogeneous Applications". IEEE International Symposium on High Performance Distributed Computing (HDPC'96), 1996.

[FOO03] Foong, A.; Huff, T.; Hum, H.; Patwardhan, J.;, Regnier, G.: " TCP Performance Re-Visited". IEEE International Symposium on Performance Analysis on Systems and Software, 2003. ISPASS, 2003.

[FRE07] Freescale. http://freescale.com

[GAD07] GadelRab, S.: "10-Gigabit Ethernet Conevtivity for Computer Servers". IEEE micro, vol. 27, issue 3. pp. 94-105. May 2007

[GAL99] Gallatin, A.; Chase, J; Yocum, K.: "Trapeze/IP: TCP/IP at Near-Gigabit Speeds". Proceedings of the Freenix Track: 1999 Usenix Annual Techniocal Conference. California. June 1999.

[GEA07] General Electric Aviation. http://www.geae.com

[GEO97] George, A.; Phipps, W.; Todd, R.; Rossen, W.: "Multithreading and Lightweight Communication Protocol Enhancements for SCI-based SCALE Systems". Proceedigns of the 7th International Workshop on SCI-based High-Performance Low-Cost Computing. March, 1997.

[GIL02a] Gilfeather, P.; Maccabe, A.B.:"Splintering TCP". http://www.cs.unm.edu/~pfeather/papers/ucf02.pdf

[GIL02b] Gilfeather, P; Macabe, A.B.: "Making TCP Viable as a High Performance Computing Protocol". Proceedings of the 3rd LACSI Symposium, Oct 2002.

[GIL05] Gilfeather, P.; Maccabe, A.B.:"Modeling protocol offload for message-oriented communication". Proc. of the 2005 IEEE International Conference on Cluster Computing (Cluster 2005) , 2005.

[GOP98] Gopalakrishnan, R.; Parulkar, G.: "Efficient User-Space Implementations with QoS Guarantees Using Real-Time upcalls", IEEE/ACM Transactions on Networking, vol. 6, nº4, April 1998

[GRA02] Gray, P.; Betz, A.: "Performance Evaluation of Copper-Based Gigabit Ethernet Interfaces". Proceedings of the 27th Annual IEEE Conference on Local Computer Networks, 2002

[GRO05] Grover, A.; Leech, C.: "Accelerating Network Receive Processing". Proceedings of the 2005 Linux Symposium. Ottawa, 2005.

[GUM03] Gumanow, G.: "Keeping Pace with the rapid adoption of Linux". Dell power solutions. February, 2003.

[GUS92] Gustafson, J.: "The Consequences of Fixed Time Performance Measurement". *Proceedings of the 25th Annual Hawaii International Conference on Systems Sciences*, IEEE Computer Society Press, Vol 3, pp. 113-124.

[HEN95] Hennessy, J.L.; Patterson, D.A.; "A Quantitative Approach". Computer Architecture (2nd. Edition). Morgan Kaufmann Publishers Inc., San Francisco, CA, 1995.

[HOK91] Hockney, R. "Performance Parameters and Benchmarking of Supercomputers". *Parallel Computing*, Volume 17, 1991, pp. 1111-1130.

[HON07] Honeywell. http://www.honeywell.com

[HUA04] Huang, B.: "Network Performance Studies in High performance Computer enviroments". University of Westenr Ontario. London, Ontario, Canada. 2004.

[HUA05] Huang, B.; Mauer, M.; Katchaban, M.: "Hpcbench – a linux-based network benchmark for high performance networks". 19th International Symposium on High performance Computing Systems and applications (HPCS'05), 2005.

[HUG05] Huggahalli, R.; Iyer, R.; Tetrick, S.; "Direct *Cache* Access for High Bandwidth Network I/O". Proceedings of the 23th International Symposium on Computer Architecture, 2005.

[HUG05] Hughes-Jones R.; Clarke, P.; Dallison, S.: "Performance of 1 and 10 Gigabit Ethernet cards with server quality motherboards". Future Generation Computer Systems, v.21 n.4, p.469-488, April 2005

[HYD00] Hyde, D.C.:"Teaching design in a Computer Architecture Course". IEEE Micro, pp.23-27, Mayo_junio, 2000.

[HYP06] HyperTransport Technology Consortium. HyperTransport I/O Link Specification. April, 2006. Revision 3.0

[IBM07a] IBM. http://www.ibm.com

[IBM07b] IBM press release: IBM Unleashes World's Fastest Chip in Powerful New Computer. London, UK, May 2007.

[INF07] Infiniband Trade Association. http://infinibandta.org/

[INT97] Intel MultiProcessor Specification Version 1.4. May 1997

[INT99] Chipset X86-440BX de Intel. http://www.intel.com/design/chipsets/440bx/index.htm

[INT01] Intel 8293AA I/O Advanced Programmable Interrupt Controller (I/O APIC). http://www.intel.com/design/chipsets/specupdt/290710.htm

[INT03a] IPX2800 Intel Network Processor IP Forwarding Benchmark Full Disclosure Report for OC192-POS. Intel Network Processing Forum. October, 2003.

[INT03b] http://www.intel.com/technology/hyperthread

[INT04] Competitive Comparison: Intel I/O Acceleration Technoloy vs. TCP Offload Engine. http://www.intel.com/technology/ioacceleration

[INT07] Familia de procesadores Pentium-4 de Intel. http://www.intel.com/cd/products/services/emea/spa/processors/142763.htm

[IOA05] Intel - "Accelerating High-Speed Networking with Intel I/O Acceleration Technology", 2005

[ISO03] ISO/IEC 14882:2003 Programming Language Standard.

[JAC92] Jacobson, V.: "TCP Extension for High Performance". RFC 1323, LBL, ISI and Cray Research, May 1992.

[JIN05] JIN, H.; Zhang, M.; Tan, P.; Chen, H.; Xu, L.: "Lightweight Real-Time Network Communication Protocol for Commodity Cluster Systems". Lecture notes in Computer Science. Vo. 3824/2005. pp. 1075-1084. Noviembre 2005.

[JON05] Jones, M.T.: "GNU/Linux Application Programming". Thomson Delmar Learning, 2005.

[KAN03] Kant, K.: "TCP Offload Performance fro Front-End Servers", Proc. IEEE Global Telecommunications Conference (GLOBECOM 03), IEEE Press, 2003, pp. 3242-3247

[KIM05] Kim, H; Rixner, S.; Pai, V.: "Network interface data caching". IEEE Transactions on Computers. Vol. 54, No 11. November 2005.

[KIM06] Kim, H.; Rixner, S.: "TCP offload through connection handoff". Proceedings of the 2006 EuroSys conference, 2006.

[LAU05] Lauritzen, K.; Sawicki, T.; Stachura, T.; Wilson, C.E.: " Intel I/O Acceleration Technology improves Network Performance, Reliability and Efficiency". Technology@Intel Magazine. March 2005. pp. 3-11.

[M5S07] M5 Simulator system Source Forge page: http://sourceforge.net/projects/m5sim

[MAG02] Magnusson, P. S.; et al.:"Simics: A Full System Simulation Platform". IEEE Computer, pp.50-58. February 2002.

[MAR02] Markatos, E.P.: "Speeding up TCP/IP: Fast Processors are not Enough". 21st IEEE International Performance, Computing and Communication Conference. April, 2002.

[MAR05] Martin, M.M.; et al.: "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset ". Computer Architecture News (CAN), 2005.

[MAT03] Wilcox, M.: "I'll do it later: Softirqs, Tasklets, Bottom Halves, Task Queues, Work Queues and Timers". Hewlett-Packard. Linux.conf.au, 2003.

[MAT07] Matlab. The language of Technical Computing. http://www.mathworks.com/products/matlab

[MAU02] Mauer, C.J.; et al.:"Full-System Timing-First Simulation". ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, June, 2002.

[MES04] "MESI protocol on L1 and L2 *Cache*s for write protect (WP) memory". Intel, 2004

[MIN95] Minnich, R.; Burns, D.; Hady, F.: "The Memory-Integrated Network Interface". IEEE Micro. Vol. 15, Nº 1. pp.11-20. Febrero 1995.

[MOG97] J. Mogul and K.Ramakrishanan, "Eliminating Receive Livelock in an Interrupt-Driven Kernel", ACM Trans. Computer Systems, Aug. 1997, pp.217-252

[MOG01] Mogul, J.C.; Minshall, G,: "Rethinking the TCP Nagle Alhorithm". ACM Sigcomm Computer Communication Review. 2001.

[MOG03] Mogul, J.C.:"TCP offload is a dumb idea whose time has come". 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.

[MUD05] Mudigonda, J.; Vin, M.H.; Yavaktar, R.: "Mamaging Memory Access Latency in Packet Proccessing". ACM SIGMETRICS Performance Evaluation Review. Colume 33, Issue 1. Junio 2005.

[MUL07] Muller, S.:"Standarizing 40 Gigabit Ethernet". Sun Microsystems, 2007

[MYR07] http://www.myri.com/

[MWA04] Mwaikambo, Z.; Ashok, R.; Russel, R.; Schopp, J.: "Linux Kernel Hotplug CPU Support". Proceedings of the Linux Symposium. Vol.2. July, 2004.

[NAH97] Nahum, E.; Yates, D.; Kurose, J.; Towsley, D.: "*Cache* Behaviour of Network Protocols". Sigmetrics Conference, 1997.

[NAR07] Narayanaswamy, G.; Balaji, P.; Feng, W.: "An Analysis of 10-Gigabit Ethernet Protocol Stacks in Multicore Environments. IEEE International Symposium on High-Performance Interconnects (HotI), 2007, Palo Alto, California.

[NET07a] http:// neterion.com, 2007

[NET07b] Netperf Homepage. http://www.Netperf.org

[NET07c] Nettest homepage. http://dsd.lbl.gov/~boverhof/nettest.html

[NET07d] Netspec homepage. http://www.ittc.ku.edu/netspec/

[NS207] http://www.isi.edu/nsnam/ns/

[ODE02] O'Dell, M. :"Re: how bad an idea is this?". Message on TSV mailing list. Noviembre, 2002.

[OPN07] http://www.opnet.com

[OPR07] Oprofile profiling System for Linux. http://Oprofile.sourceforge.net

[OPP05] Oppermann, A.: "Optimizing the FreeBSD IP and TCP stack". TCP/IP optimization fundraise, 2005.

[ORA07] Oracle Database. http://www.oracle.com

[ORA08] Oracle Grid Computing. http://www.oracle.com/technologies/grid/index.html

[ORT05] Díaz A.F.; Ortega, J; Prieto, A.; Ortiz, A.: "Análisis de la externalización de protocolos mediante simulación HDL". XVI Jornadas de Paralelismo. Septiembre 2005, Granada.

[ORT06a] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: "Protocol offload evaluation using Simics". IEEE Cluster Computing, Barcelona. Septiembre , 2006.

[ORT06b] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: "Análisis de la externalización de protocolos de comunicación mediante Simics". XVII Jornadas de Paralelismo. Septiembre 2006. Albacete

[ORT07a] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: "Analyzing the benefits of protocol offload by full-system simulation". 15th Euromicro Conference on Paralell, Distributed and Network-based Processing, PDP 2007.]

[ORT07b] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: Modeling network behaviour by full-system simulation**.** Journal of Software. Issue 2/2007. pp. 11-18

[ORT07c] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: "Comparación de prestaciones onloading/offloading mediante Simics". XVIII Jornadas de Paralelismo. Septiembre 2007, Zaragoza.

[ORT08a] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.: "Comparison of Onloading and Offloading Strategies to improve Network Performance". 16th Euromicro Conference on Paralell, Distributed and Network-based Processing, PDP 2008.

[ORT08b] A.Ortiz, J.Ortega, A.F.Díaz, A.Prieto; "Comparación de técnicas para la externalización de protocolos"; Memorias de la 7ª Conferencia Iberoaméricana en Sistemas, Cibernética e Informática (CISCI2008), Vol. I, pp. 59-66, ISBN-10: 1-934272-39-6, International Institute of Informatics and Systemics (IIIS), Orlando (EEUU), junio de 2008.

[ORT08c] Ortiz, A.; Ortega, J.; Díaz A.F.; Prieto, A.:"Alternativas para la interfaz de red en servidores multiprocesador". XVIII Jornadas de Paralelismo. Septiembre, 2008.

[ORT08d] Ortiz, A.; Ortega, J.; Díaz A.F.; Cascón, P; Prieto, A.:" Protocol Offload Analisis by Simulación". Journal of Systems Architecture.

[PAP04] Papaefstathiou, I.; et al.: "Network Processors for Future High-End Systems and Applications". IEEE Micro. Septiembre-Octubre, 2004.

[PAK97] Pakin, S.; Karacheti, V.; Chien, A. : "Fast Messages (FM) : Efficient, Portable Communication for Workstation Clusters ans Masively-PArallel Processors".IEEE Parallel and Distributed Technology, Vol.5, No.2. April/June, 1997.

[PET05] Petrini, F.: "Ethernet vs Ethernot", 13th Symposium on High Performance Interconnects (HOTI'05), 2005.

[PLA00] Plagemann, T.; Groebel, V.; Halvorsen, P.; Anshus, O.: "Operating Systtem Support for multimedia systems". The computer Communications Journal, Elsevier, 23(3):267-289, Febfuary 2000.

[PRA04] Pradhan, P.; Kandula, S.; Xu, W.; Shaikh, A.; Nahum, E.:" Daytona: a user level TCP stack". http://nms.lcs.mit.edu/~kandula/data/daytona.pdf

[PRE98] Preston, C.: "Using Gigabit Ethernet to backup six terabytes". Proceedings of the 12th conference on Systems Administration. 1998.

[PRY98] Prylli, L.; Tourancheau, B. : ”BIP : a new protocol designed for high performance networking on Myrinet ». Workshop PC-NOW, IPPS/SPDP98, (Lecture Notes in Computer Science, No 1388), pp.472-485. Abril, 1998.

[QEM07] QEMU web page. http://fabrice.bellard.free.fr/qemu

[QSN07] QsNet High Performance Interconnect http://cuadrics.com

[RAN02] Rangarajan, M.; Bohra, A.; Banerjee, K.; Carrera, E.V.; Bianchini, R.; Iftode, L.: "TCP Servers: Offloading TCP Processing in Internet Servers. Design, Implementation, and Performance". Technical Report DCS-TR-481, Department of Computer Science, Rutgers University, March 2002.

[RAV04] Ravot, S.; Xia, Y.; Nae, D.; Su, X.; Newman, H.; Bunn, J.; Martin, O.: “A Practical Approach to TCP High Speed data Transfers”. Proceedings of GrigNets, 2004. San José, USA. October, 2004.

[REG04a] Reginier, G. et al.:”TCP onloading for data center servers”. IEEE Computer, pp.48-58. Noviembre, 2004.

[REG04b] Regnier et al.: “ETA: Experience with an Intel Xeon Processor as a Packet Process Engine” IEEE Micro, vol.24, nº1, Jan.-Feb. 2004, pp. 24-31

[ROS97] Rosenblum, M.; et al.:”Using the SimOS machine simulator to study complex computer systems”. ACM Tranns. On Modeling and Computer Simulation, Vol.7, No.1, pp.78-103. January, 1997.

[SAL01] J.Salim.: “Beyond SoftNet”, Proc. 5th Ann. Linux Showcase and Conf.; www.linuxshowcase.org/2001/full_papers/jamal/jamal.pdf

[SAL03] Floyd, S.: “RFC3649: High-speed TCP for large congestion windows”. Experimental, December 2003.

[SCO97] Scott, A.P.; Burkhart, L.O.; Kumer, A.; Blumberg, R.M.; Ranson, G.K.; :”Four-way superscalar PA-RISC processors”. Hewlett-Packard journal, 1997.

[SET07] http://www.seti.org

[SHA06] Shalev, L.; Marhervaks, V.; Machulsky, Z.; Biran, G.; Satran, J.; Ben-Yehuda, M.; Shimony, I.: “Loosely Coupled TCP Acceleration Architecture”. Proceedings of the 14th IEEE Symposium on High-Performance Interconnects (HOTI). 2006

[SHI03] Shivam, P.: Chase, J.S.:”On the elusive benefits of protocol offload”. SIGCOMM’03 Workshop on Network-I/O convergence: Experience, Lessons, Implications (NICELI). August, 2003.

[SKA03] Skadron, K; Martonosi, M.; August, D.I.; Hill, M.D.; Lilja, D.J.; Pai, V.S.: "Challenges in Computer Architecture Evaluation". IEEE Computer. Vol 36, issue 8. 2003.

[SKE01] Skevik, K.; Plagemann, T.; Groebel, V.:"Evaluation of a zero-copy protocol implementation". Proceedings of the 17th Euromicro Conference. 2001

[SKO97] Skorupe, J.; Prodan, G.; "Battle of the Backbones: ATM vs. Gigabit Ethernet", Data Communications, April 1997.

[SNE96] Snell, Q.O.; Milker, A.R.; Gustafson, J.L. : "Netpipe: a protocol independent performance evaluator". International conference on intelligent management and systems, 1996.

[SPE05] Specweb05. http://www.spec.org/web2005

[STE94a] Steenkite, P.: "A Systematic Approach to Host Interface Design for High-Speed Networks". Computer, pp. 47-57, vol 27, issue 3. March 1994.

[STE94b] Stevens, W.R.: "TCP/IP Ilustrated Volume 1". 1994, Addison Wesley.

[STE98] Steenkite, P.: "Desing, Implementation, and Evaluation of a Single-Copy Protocol Stack", Software-Practice and Experience, vol.28, nº7, July 1998.

[THI02] Thiele, L. et al.:"Design space exploration of network processor architectures". Proc. 1st Workshop on Network Processors (en el 8th Int. Symp. on High Performance Computer Architecture). Febrero, 2002.

[TOM07] Apache Tomcat Webserver. http://tomcat.apache.org

[TRA94] Tranter, H.W.; Kosban, K.L.:" Simulation of communication Systems". IEEE Communications Magazine, vol32. 1994

[TTC05] TTCP: Benchmarking tool for measuring TCP and UDP Performance. http://www.pcausa.com/utilities/pcattcp.htm

[TUR03] Turner, D.; Oline, A.; Chen, X.; Benjegerdes, T.: "Integrating new capabilities into Netpipe". Euro PVM/MPI conference, September 2003.

[VAJ01] Vajapeyam, S.; Valero, M.: "Early 21st Century Processors", IEEE Computer, vol. 34 nº4, pp. 47-50, abril, 2001.

[VER05] IEEE Standard Verilog Hardware Description Language. IEEE Std. 1364-1995

[VHD93] IEEE Standard VHDL Language Reference Manual. IEEE Std. 1076-1993.

[VID01] http://www.vidf.org/ (Virtual Interface Developer Forum, VIDF, 2001).

[VIL05] Villa, F.J.; Acacio, M.E.; García, J.M.:"Evaluating IA-32 web servers through Simics: a prectical experience". Journal of Systems Architecture, 51, pp.251-264, 2005.

[VIR08a] Virtutech web page: http://www.virtutech.com/

[VIR08b] Virtutech news and events web page. http://www.virtutech.com/news-press/

[VMW07] VMware web page. http://www.vmware.com

[VPC07] Microsoft Virtual PC. http://www.microsoft.com/windows/products/winfamily/virtualpc/overview.mspx

[WAN04] Wang, B.; Kurose, J.; Shenoy, P.; Towsley, D.: "Multimedia Streaming via TCP: an analytic performance study". Proceedings of ACM multimedia. October 2004.

[WAN98] Wang, R.Y.; Krishnamurthy, A.; Martin, R.P; Anderson T.E.; Cluuer, D.E.: "Modeling Communication Pipeline Latency". ACM Sigmetrics Performance Evaluation Review, June 1998

[WAN99] Wang, S.Y.; Kung, H.T.:"A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators", IEEE INFOCOM'99, March 21-25, 1999, New York, USA.

[WEI07] Weinsberg, Y.; Dolev, D.; Wyckoff, P.; Tal, A.: "Accelerating distributed Computing Applications Using a Network Offloading Framework". International Paralell and Distributed Processing Symposium (IPDPS'07), 2007.

[WEN06] Wenji, W.; Crawford, M.; Bowden, M.:"The Performance Analysis of Linux Networking. Packet Receiving". Computer Communications. 30 (2007) 1044-1057. November 2006.

[WES04] Westrelin, R.; et al.:"Studying network protocol offload with emulation: approach and preliminary results", 2004.

[WIL04] Willman, P.; Brogioli, M.; Pai, V.S.: "Spinach: a liberty based Simulator for Programmable Network Interface Architectures". Conference on Languages, Compilers and Cools for Embedded Systems. June 2004. LCTES'04.

[WOL00] Wolf, T.; Franklin, M.:" Commbench – a telecommunications benchmark for network processors". IEEE International Symposium on performance analysis of systems and software" (ISPASS'00), 2000.

http://standards.ieee.org/getieee802/download/802.3ae-2002.pdf

[WU06] Wu, W; Crawford, M.; Bowden, M.:"The Performance Analysis of Linux Networking – Packet Receiving". Computer Communications. Vol. 30, issue 5. pp. 1044-1057. 2007

[WUL95] Wulf, W.A. and McKee, S.A.:"Hitting the Memory Wall: Implications of the Obvious", Computer Architecture News, Mar. 1995, pp. 20-24

[WUN06] Wun, B.; Crowley, P.: "Network I/O Acceleration in Heterogeneous Multicore Processors". 14th IEEE Symposium on High-Performance Interconnects. August, 2006.