UNIVERSIDAD DE GRANADA

DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGIA DE COMPUTADORES

Análisis, evaluación de prestaciones y mejora de interfaces de red mediante modelos HDL

TESIS DOCTORAL

WASEEM M. HAIDER

GRANADA, Septiembre 2011

Editor: Editorial de la Universidad de Granada
Autor: Waseem M. Haider
D.L.: GR 1072-2012
ISBN: 978-84-695-1078-0

UNIVERSIDAD DE GRANADA

DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGIA DE COMPUTADORES

Análisis, evaluación de prestaciones y mejora de interfaces de red mediante modelos HDL

WASEEM M. HAIDER

TESIS DOCTORAL

DIRECTORES:

Julio Ortega Lopera
Antonio Francisco Díaz García

2011

Dr. D. Julio Ortega Lopera, Catedrático de Universidad, y **Dr. D. Antonio Francisco Díaz García**, Profesor Titular de Universidad, ambos del Departamento de Arquitectura y Tecnología de Computadores de la Escuela Técnica Superior de Ingeniaría Informática y Telecomunicación de la Universidad de Granada.

CERTIFICAN:

Que la presente memoria, titulada "**Análisis, evaluación de prestaciones y mejora de interfaces de red mediante modelos HDL**", ha sido realizada por **D. Waseem M. Haider** bajo nuestra dirección, en el Departamento de Arquitectura y Tecnología de Computadores de la Escuela Técnica Superior de Ingeniería Informática y Telecomunicación de la Universidad de Granada. Esta memoria constituye el trabajo de investigación que **D. Waseem M. Haider** presenta para optar al grado de Doctor.

Granada, 1 de Septiembre de 2011

Fdo: Dr. D. Julio Ortega Lopera
Director de la Tesis

Fdo: Dr. D. Antonio Franciesco Díaz García
Director de la Tesis

UNIVERSIDAD DE GRANADA

DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGIA DE COMPUTADORES

Análisis, evaluación de prestaciones y mejora de interfaces de red mediante modelos HDL

Memoria presente por

WASEEM M. HAIDER

Para optar al grado de

DOCTOR EN INGENIERÍA INFORMÁTICA

Fdo: Waseem M. Haider

Agradecimiento

En el primer lugar mi gustaría dar las gracias a mis directores de este trabajo Dr. Julio Ortega Lopera y Dr. Antonio F. Díaz por dedicar tiempo, conocimiento y paciencia a lo largo del presenta de este trabajo. A mis compañeros del departamento de Arquitectura y Tecnología de Computadores que me han ayudado desde el comienzo de este trabajo.

También quiero dar las gracias a mi mujer, Shams, mi complemento, por su apoyo incondicional siempre.

Dedico este trabajo a mis padres, la luz de mis ojos, cuyos enseñanzas y buenas costumbres han creado en mi la sabiduría y el conocimiento que de lo que soy. Agradezco a mis hermanas el apoyo que siempre me han brindado con su impulso y ánimo.

Al Departamento de Arquitectura y Tecnología de Computadores y a los proyectos de investigación TIC-1395, y TIN2007-60587.

La realización de este trabajo no habría sido posible sin la beca de la Agencia Española de Cooperación Internacional y para el Desarrollo (AECID),…. Muchas Gracias.

شُكروتقدير

فـــي المقام الأول أشكر الله سبحانه وتعـــالى الـذي أعـــــانني على انجاز هذا العمل والوصـــــول الى ما انا عليه ...

أتقدم بالشكر الجزيل والأمتنان الى الســـادة المشرفين د. خوليو اورتيغا لوبيرا و د.انطونيو فرانشسكو غارثسيا ديث لمساعدتهم ودعمهُم لي طيلة فترة دراستي.

أعبر عن شكري وتقديري الى حبيبتي وزوجتي شمس لدعمها اللا محدود لـــــي ولصبرها معي طيلة هذة الفترة. كما اقدم هذا العمل مع شكري وتقديري الى والـــديَّ حيث ان بدعواتهم وحبهم وبدعمهم وصلت الى ما انا عليه الان , وأقدم شكري الى اخواتي وذلك لتمنياتهم لي بالنجاح في جميع مجالات حياتي.

واخيراَ شكري الجزيل الى الوكالة الأسبانية للتعاون الدولي حيث انهُ من غير الممكن ان يكون هذا العمل لولا حصـــولي على المنحة الدراسية.

A Mis padres, Hermanas, Shams y Qamar,

Por su Amor y Simplemente por Todo

ÍNDICE

Índice

Presentación 13

Capítulo 1

Introducción y Fundamentos de las Operaciones de la tarjeta de Red 21

1.1 Introducción 22
1.2 Tareas de la interfaz de red 31
1.2.1 Almacenamiento y control de buffers 32
1.2.2 Control de dispositivos 35
1.2.3 Tarjeta de red y driver de dispositivos 36
1.3 La sobrecarga (*overhead*) de comunicación 38
1.4 Estrategias para aumentar las prestaciones del camino de comunicación 40
1.4.1 Los protocolos ligeros 41
1.4.2 Interfaz de red nivel de usuario 42
1.4.3 Fusión de interrupciones 44
1.4.4 Copia-0 de memoria 45
1.5 La externalización de protocolos de comunicación 45
1.5.1 NIC para redes multigigabits por segundo 46
1.5.2 La comparación entre la pila de TCP/IP y los TOE 49
1.5.2.1 La sobrecarga (overhead) TCP y TOE 50
1.5.2.2 Procesamiento de interrupciones 51
1.5.2.3 Copias de memoria 51
1.5.2.4 Procesamiento del protocolo de comunicación 52
1.6 Modelos de evaluación de prestaciones 53

1.6.1 Evaluación de las prestaciones de comunicación mediante el modelo LAWS 53
1.6.2 Evaluación de las prestaciones de comunicación mediante el modelo EMO 59
1.6.3 Relación entre los modelos EMO y LAWS 64
1.7 Análisis y optimización el sistema de comunicación mediante la simulación 65
1.7.1 El simulador M5 67
1.7.2 El simulador SimOS 68
1.7.3 El simulador SIMICS 68
1.8 Conclusiones 71

Capítulo 2

Modelo en Lenguaje de Descripción de Hardware (HDL) y Optimización de la Interfaz de Red 73

2.1 La simulación HDL del sistema de comunicación 74
2.2 Modelos HDL para simulación del sistema de comunicación 79
2.2.1 El modelo de simulación HDL sin *externalización* 81
2.2.2 El Modelo de simulación HDL para la *externalización* mediante *offloading* 84
2.2.3 El Modelo de simulación HDL para la *externalización* mediante *onloading* 86
2.3 Los módulos de la simulación HDL del sistema de comunicación 91
2.3.1 El módulo CPU 91
2.3.2 El módulo *Chipset* 94
2.3.3 El módulo de memoria y el bus de memoria 97
2.3.4 El módulo del bus de E/S 100
2.3.5 El módulo NIC 101
2.3.6 El módulo Red (módulo Network) 103
2.4 Conclusiones 105

Capítulo 3

Evaluación de los Resultados Experimentales e Identificación de Oportunidades de Mejora 107

3.1 Validación del modelo HDL utilizado 108
3.2 Evaluación y análisis de las prestaciones mediante *offloading* 111
3.2.1 El efecto de los parámetros αβ en la mejora del ancho de banda 112
3.2.2 Efecto del parámetro αβ en la latencia de comunicación 116
3.3 Evaluación y análisis de las prestaciones mediante *onloading* 118
3.3.1 El efecto en la mejora del ancho de banda 118
3.3.2 El efecto de la latencia de comunicación 120
3.4 Comparación entre la mejora del *offloading* y el *onloading* 122
3.4.1 Comparación de las mejoras en el ancho de banda 122
3.4.2 Comparación entre la mejora de la latencia de comunicación 124
3.5 El efecto del bus de E/S en las técnicas de *offloading* y *onloading* 125
3.6 El efecto de la cache en las técnicas de *offloading* y *onloading* 127
3.6.1 Efecto del tamaño de las caches L1y L2 en la mejora del ancho de banda 129
3.6.2 El efecto del tamaño de las caches L1 y L2 en la latencia 132
3.7 Modelo de simulación HDL para interfaz de red que combina las técnicas de *offloading* y *onloading*. 133
3.7.1 Análisis de prestaciones de la interfaz híbrida *offloading*/*onloading*. 136
3.7.1.1 Mejora del ancho de banda 136
3.7.1.2 Mejora en la latencia 138
3.8 Conclusiones 138

Capítulo 4

Diseño de Experimentos en el Análisis de los parámetros del camino de comunicación 141

4.1 El diseño experimento 142
4.1.1 El diseño factorial completo 2^k 143
4.1.2 El diseño factorial fraccionales 145
4.2 El análisis de varianza ANOVA 146
4.2.1. ANOVA de un factor 147
4.2.2. ANOVA factorial 148
4.3 Diseño de 2 parámetros para el análisis de la interfaz de red 149
4.4 Diseño de tres parámetros para el análisis de la interfaz de red 152
4.5 Diseño experimental de más de tres parámetros 156
4.6 El diseño ANOVA 158
4.7 Conclusión 160

Capítulo 5

Conclusiones Y Trabajo Futuro 163

5.1 Aportaciones 164
5.2 Trabajo Futuro 170

Apéndice I

Módulos *Verilog* para la simulación del sistema de comunicación 173

1. Módulo de la tarjeta de red NIC 173
2. Módulo de la memoria principal 178
3. Módulo del procesador principal 181

Referencias 185

Lista de figuras

1.1 Tarjeta de interfaz de red Myrinet para bus PCI.....28
1.2 Ejemplo de sistema conectado a partir de conmutadores Infiniband.....30
1.3 Esquema de los elementos hardware que intervienen en la interfaz de red (se considera una NIC con procesador incluido).....31
1.4 Formato de un paquete..... 33
1.5 Una Cabecera de IP.....34
1.6 Una Cabecera de TCP..... 34
1.7 NIC genérico como dispositivo de E/S con acceso estándar a nivel de de sistema operativo..... 37
1.8 Comparación de las capas de CLIC y TCP/IP [DIA03].....41
1.9 Caminos de datos hacia y desde la aplicación.....46
1.10 Arquitectura de controlador de Ethernet de 10Gb/s.....48
1.11 La comparación de TCP/IP estándar y TOE-permitidas de la pila de TCP/IP..... 50
1.12 Implementar TCP/IP al usar un TOE..... 52
1.13 Modelo LAWS (a) antes del *offloading*, (b) después del *offloading*..... 55
1.14 Porcentaje de mejora de ancho de banda máximo según el modelo LAWS.....57
1.15 Beneficio de offloading en función del coeficiente de aplicación (γ) y para varias valores de p, y con α, β, σ iguales a 1..... 58
1.16 Beneficio de la técnica de offloading completo en función de coeficiente de aplicaciones y para varios valores de β, con α=2..... 58
1.17 Modelo EMO para el *offloading*..... 60
1.18 Esquema de operación de la técnica de *Splintering* de TCP..... 61
1.19 Composición de la sobrecarga según el modelo EMO..... 63
1.20 Composición de la latencia según el modelo EMO..... 63
1.21 Los módulos del simulador M5..... 67
1.22 El envaramiento del simulador SimOS..... 68
1.23 La arquitectura del simulador SIMICS..... 69
1.24 La arquitectura del simulador GEMS..... 70
2.1 Distribución el tiempo de procesador de un servidor apache [RAN02]..... 75
2.2 La ubicación de la tarjeta de red [BIN05].....76

2.3 Recepción sin externalización (a); y dos opciones de offloading (b) y (c)..... 80
2.4 El modelo de simulación HDL de todo el camino de comunicación (Sin *externalización*)..... 83
2.5 El modelo de simulación HDL de todo el camino de comunicación para la *externalización* (*offloading*)..... 85
2.6 El modelo de simulación HDL de todo el camino de comunicación para la *externalización* (*onloading*)..... 87
2.7 Diagrama de tiempo para las tres alternativas (a) sin-*externalización* , (b) *offloading*, y (c) *onloading*..... 89
2.8 Módulos y señales de control del Módulo CPU..... 92
2.9 Diagrama de tiempo para las señales del módulo de CPU..... 93
2.10 Diagrama de flujo del funcionamiento de cache..... 93
2.11 Módulos y señales de controles del módulo Chipset en la emisión..... 95
2.12 Diagrama de tiempo para las señales del módulo de Chipset (a) leer datos del procesador, (b) leer datos, y (c) escribir datos..... 96
2.13 Módulos de memoria y bus de memoria (a) Bus de memoria (b) Memoria del emisor (c) Memoria del receptor..... 98
2.14 Diagrama de tiempo para las señales del módulo de memoria en el caso de (a) emisión, y (b) recepción..... 99
2.15 Módulos y señales de controles del módulo de bus de E/S (a) recepción (b) emisión..... 100
2.16 Módulos y señales de controles de NIC (a) recepción (b) emisión..... 101
2.17 Diagrama de estado del movimiento de datos en el módulo NIC..... 102
2.18 Módulos de red de (a) productor de datos (b) consumidor de datos..... 104
2.19 Diagrama de tiempo para los módulos de red..... 104
3.1 Comparación entre el ancho de banda obtenido cuando no hay *externalización* mediante el simulador SIMICS y el HDL para varios tamaños de mensajes..... 109
3.2 Comparación entre el ancho de banda obtenido con el *offloading* mediante el simulador SIMICS y el HDL para varios tamaños de mensajes..... 110
3.3 Comparación entre el ancho de banda con *onloading* mediante el simulador SIMICS y HDL para varios tamaños de mensajes..... 110
3.4 Comparación entre el ancho de banda cuando hay *offloading* y no hay ($\alpha\beta$=1.01, 64 bits)..... 112

3.5 Mejora del ancho de banda con *offloading* para varios valores de αβ y enlace de 1Gbps……113
3.6 Mejora del ancho de banda con *offloading* para varios valores de αβ y 10Gbps……114
3.7 Comparación de la mejora del ancho de banda para 1Gbps y 10Gbps y para varios valores de αβ……114
3.8 El número de los paquetes en el fichero de trazas (*Ethernet at the Bellcore Morristown Research*) [INT10]…… 115
3.9 Comparación de la mejora del ancho de banda para un fichero de trazas [INT10] y para varios valores de αβ……115
3.10 El ancho de banda con *offloading* y sin *externalización* para varios tamaños de mensajes (αβ=1.01)……116
3.11 Latencias para *offloading* y sin *externalización* con varios valores de αβ……117
3.12 Comparación de latencias para varios tamaños de mensaje con y sin *externalización* (mediante *offloading*)……117
3.13 Comparación entre la mejora del ancho de banda para varios velocidades de la memoria principal……118
3.14 Comparación entre el ancho de banda cuando hay *onloading* y no hay (αβ=1.01)……119
3.15 Ancho de banda con *onloading* y sin él para varios tamaños de mensajes……120
3.16 Comparación entre la latencia cuando hay *onloading* y no hay……120
3.17 Latencias para *onloading* (αβ=1.01) y sin él con varios tamaños de mensaje……121
3.18 Comparación entre el ancho de banda cuando hay *offloading*, *onloading* y sin *externalización* (αβ=1.01)……122
3.19 Comparación de la mejora del ancho de banda del *offloading* y *onloading* para un fichero de trazas [INT10] y para varios valores de αβ…… 123
3.20 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* con varios tamaños de mensajes y αβ=1.01……124
3.21 Comparación entre la latencia del *offloading*, *onloading* y sin *externalización*……124
3.22 Comparación entre la latencia observada con *offloading*, *onloading* y sin *externalización* para varios tamaños de mensajes (αβ=1.01)…… 125

3.23 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* cuando hay bus de E/S y no, para varios tamaños de mensajes y αβ=1.01 126
3.24 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* con cache (L1=32KB, L2=1024KB) y sin cache 128
3.25 Comparación entre la latencia de *offloading* y *onloading* con cache (L1=32kbs, L2=1024kbs) y sin cache 128
3.26 Comparación entre la mejora del ancho de banda del *offloading* (αβ=0.3), con varios tamaños de L, y el tamaño de los paquetes 64 Kbyte 129
3.27 Comparación entre la mejora del ancho de banda del *offloading* (αβ=0.3), con varios tamaños de L2, y el tamaño de los paquetes 64 Kbyte 130
3.28 Comparación entre la mejora del ancho de banda del *onloading*, con varios tamaños de L1, y el tamaño de los paquetes 64 Kbyte 131
3.29 Comparación entre la mejora del ancho de banda del *onloading*, con varios tamaños de L2 y cuando el tamaño de paquetes 64 Kbyte 131
3.30 Comparación entre la latencia del sin-*externalización*, *onloading* y *offloading* con memoria cache de dos niveles 132
3.31 El modelo de simulación HDL del camino de comunicación completo para la propuesta híbrida *offloading*/*onloading* 134
3.32 Comparación entre el ancho de banda para *offloading*, *onloading*, *offloading*/*onloading* y sin *externalización* (αβ=1.01) 136
3.33 Comparación entre el ancho de banda para *offloading*, *onloading*, *offloading*/*onloading* y sin *externalización* con varios tamaños de mensajes y αβ=1.01 137
3.34 Comparación entre la latencia observada con *offloading*, *onloading*, *offloading*/*onloading* y sin *externalización* para varios tamaños de mensajes 138
I.1. Los módulos *Verilog* para la simulación HDL de todo el camino de comunicación 174

Lista de Tablas

2.1 Distribución de las tares de comunicación..... 90
3.1 La distribución de las tares de comunicación entre los elementos de la simulación con la propuesta híbrida..... 135
4.1 Matriz de experimentos para los diseños factoriales completos (a) 2^2, (b) 2^3, y (c) 2^4..... 144
4.2. Matriz completo del diseño fraccional 2^K por K=3..... 146
4.3 Expresiones para el cálculo del ANOVA de un factor..... 148
4.4 Factores y dominio experimental (a) y (b)..... 149
4.5 Diseño factorial completo 2^2 para los factores $\alpha\beta$ y γ, (a) para la tabla 4.4 (a) y (b) para la tabla 4.4 (b)..... 151
4.6. Factores y dominio experimental..... 153
4.7 El plan experimento del diseño de 3 factores..... 153
4.8 Matriz de experimentos de un diseño factorial completo 2^3 y respuestas medidas..... 154
4.9. Los seis factores y el dominio experimental..... 156
4.10 Matriz de experimento de un diseño factorial completo 2^{6-3} y respuesta medida..... 157
4.11 Los resultados experimentales de la mejora del ancho de banda..... 158
4.12 Análisis de varianza de dos factores..... 159
4.13Análisis de varianza..... 159

Presentación

Los simuladores son herramientas indispensables para el análisis y la evaluación de las propuestas en los distintos niveles de diseño de las arquitecturas de computadores y a lo largo de las distintas etapas del ciclo de diseño. La investigación en arquitectura de computadores necesita de la simulación para estudiar el efecto de las distintas ideas y alternativas, y corroborar las expectativas a un coste limitado. La existencia de simuladores adecuados para estudiar determinado ámbito de investigación facilita el trabajo en dicho ámbito y fomenta la obtención de resultados en el mismo [BUR10]. De ahí que el estudio de las características y requisitos que deben satisfacer los simuladores del camino de comunicación interesante de cara a proporcionar herramientas que faciliten la investigación en el desarrollo de interfaces de red eficientes que puedan afrontar las necesidades de comunicación de las aplicaciones, en entornos de red con enlaces de anchos de banda cada vez mayores.

Precisamente, el ritmo de mejora del ancho de banda de los enlaces plantea retos importantes en el diseño de arquitecturas de comunicación eficientes que impidan que el cuello de botella de comunicación pase de la red a los nodos [HEN95, FOO03]. Por ejemplo, una interfaz de red para Ethernet dúplex a 10 Gb/s debería ser capaz de ejecutar instrucciones a una velocidad sostenida de alrededor de 435 MIPS y mantener 4.8 Gb/s de ancho de banda para procesar los protocolos de comunicación [SCH07]. Estos requisitos pueden provocar la saturación de la CPU, que debería dedicarse por completo a las tareas de comunicación. Por tanto, la interfaz de red, entendida como el conjunto de software y hardware que se encarga de ajustar los requisitos de comunicación y el hardware de comunicación, tiene una importancia crucial en el diseño de arquitecturas eficientes para las plataformas de cómputo actuales, altamente interconectadas y ejecutando aplicaciones distribuidas en mayor o menor medida. En un computador, las aplicaciones, usualmente a travésd el sistema operativo, utilizan la interfaz de red para enviar y recibir paquetes desde y hacia la memoria principal aprovechando el trabajo cooperativo de la tarjeta de red NIC **(***Network Interface Controller*), que tiene acceso a los enlaces de red, y del driver correspondiente del sistema operativo, que accede a las estructuras del mismo. Teniendo en cuenta los recursos de procesamiento disponibles en las plataformas actuales, las interfaces de red deberían diseñarse de forma que fueran capaces de aprovechar el paralelismo disponible al existir varios procesadores, tanto en el caso de plataformas SMP (*Symmetric Multiprocessors*), como en el de microprocesadores multinúcleo CMP (*Chip Multiprocessors*), o en controladores de red programables (NIC *programables o* NIC *inteligentes*). Además, en el diseño de una interfaz de red hay que tener en cuenta la forma en que se distribuye el software de comunicación y la interacción del mismo con los distintos elementos hardware (*buses, memorias, chipsets*,…), con el sistema operativo, y con el software de las aplicaciones.

Se han desarrollado muchos trabajos buscando minimizar la sobrecarga (*overhead*) de comunicación [CLA89, CHE05]. Entre las alternativas que se han considerado frecuentemente está la *externalización* de los protocolos de comunicación [DIA05, ORT07, ORT09], que consiste en la ejecución de parte (o la totalidad) de dichos protocolos en un procesador distinto a la CPU del computador. Entre las opciones más importantes que se han propuesto para la *externalización* están el *offloading* [WES04, MOG03] y, posteriormente, el *onloading* [REG04a]. En la primera alternativa, la externalización se lleva a cabo utilizando la tarjeta de interfaz

de red, NIC, mientras que en la segunda, se hace uso de otro procesador presente en el computador, que tiene los mismos privilegios desde el punto de vista de acceso a memoria y de ejecución del sistema operativo que la CPU o el núcleo en donde se ejecutan las aplicaciones.

Como se ha comentado más arriba, la investigación en el diseño de computadores con enlaces de red de gran ancho de banda requiere herramientas de simulación adecuadas que incluyan, entre otras cosas, modelos de tiempo de la actividad de DMA y un modelo coherente y preciso del sistema de memoria [BIN03]. Algunos ejemplos de simuladores que poseen estas características son M5 [M5S10], SIMICS [MAG02], y SimOS [ROS97], y otros como GEMS [GEM08] y TFsim [MAU02] basados en SIMICS. Por otra parte, la simulación HDL (*Hardware Description Language*) permite tener una idea de la implementación de nuevas propuestas hardware, complementar los resultados obtenidos por simuladores funcionales, y generar estimaciones de la velocidad de reloj que puede alcanzar el hardware si se elabora un modelo HDL sintetizable. También hacen posible la identificación de caminos críticos y aspectos de diseño del hardware que pueden mejorarse. Por todas estas razones, los modelos HDL pueden ser de gran ayuda en la evaluación de interfaces de red al permitir, tanto análisis cualitativos, como análisis cuantitativos aproximados para llegar a conclusiones acertadas para la toma de decisiones en el diseño de interfaces eficientes. No obstante, aunque los modelos HDL permiten estudiar, a un nivel próximo al hardware, el efecto de los retardos en las distintas etapas del camino de comunicación, difícultan la simulación de códigos realistas y su interacción con el sistema operativo y con el hardware del sistema. También es necesario disponer de procedimientos para la generación de conjuntos de trazas que modelen adecuadamente la interacción entre aplicaciones, hardware y sistema operativo en las simulaciones HDL. En este sentido, al analizar las posibilidades que ofrecen los modelos HDL, la presente tesis (cuarta tesis que se ha realizado en el grupo de investigación CASIP en el ámbito de la mejora de las arquitecturas de comunicación y las interfaces de red [DIA01, ORT08, CAS10] constituye un complemento al trabajo que se describe en la tesis "*Alternativas de externalización para la interfaz de red. Análisis y optimización mediante simulación de sistema completo*", de Andrés Ortiz [ORT08], donde se utilizaba la simulación de sistema completo mediante SIMICS para analizar el efecto de las alternativas de externalización de la interfaz de red en las prestaciones de comunicación.

En el presente trabajo se comparan, mediante modelos de simulación HDL, distintas alternativas para el aprovechamiento del paralelismo en la optimización de las arquitecturas de comunicación. El análisis de los resultados y la exploración del espacio de diseño realizado en nuestros experimentos se ha realizado tomando como referencia el modelo LAWS, previamente propuesto en [SHI03]. Este modelo pretende caracterizar la mejora que proporciona la *externalización* de la comunicación en la NIC, y es aplicable fundamentalmente en el caso de aplicaciones relacionadas con servicios de Internet u otras aplicaciones que implican comunicación de secuencias de datos (*streaming data applications*). El modelo LAWS proporciona una estimación de la mejora máxima de ancho de banda (*peak throughput*) partiendo de un modelo segmentado del camino de comunicación, en el que el cuello de botella (que puede estar en el enlace, el NIC, o el procesador central) determina el ancho de banda del sistema antes y después de la externalización.

El trabajo aquí presentado considera las dos alternativas de *externalización* a las que nos hemos referido más arriba el *offloading* y el *onloading*. La primera de ellas corresponde a la implementación de las tareas de comunicación en un procesador ubicado en la tarjeta de red NIC, y es la alternativa para la que se propuso el modelo LAWS. En la segunda alternativa se contempla la asignación de uno de los procesadores de un CMP (*Chip Multiprocessor*) o un SMP (*Symmetric Multiprocessor*) para dichas tareas de comunicación. Además, se propone una alternativa nueva que hibrida las técnicas *offloading* y *onloading* para tratar de aprovechar los beneficios de cada una de esas técnicas reduciendo, en la medida de lo posible, sus efectos menos deseables.

Objetivos de este trabajo

Teniendo en cuenta lo expuesto en la sección anterior, los objetivos de este trabajo de investigación son los siguentes:

1. El modelado HDL del sistema de comunicación de un computador para permitir el análisis de las mejoras posibles en el diseño de interfaces de red, en particular el estudio de nuevas alternativas que, como las tecnicas de

externalización (*offloading* y *onloading*), buscan explotar la concurrencia a nivel de tarea en las comunicaciones, aprovechando los procesadores disponibles en distintas ubicaciones dentro del computador (SMP, CMP, o en la NIC). Para alcanzar este objetivo habrá que analizar el efecto de las aplicaciones (caracterizada en términas de la frecuencia y tamaño de los mensajes intercambiados), la latencia y el ancho de banda de memoria, la latencia y el ancho de banda de los buses, el tamaño de los buffers, la influencia de la cache, y la velocidad, la ubicación, y la carga de los procesadores disponibles en el nodo.

2. La elaboración de un entorno basado en el modelado HDL del camino de comunicación, para la simulación de benchmarks de comunicación, incluyendo la generación de trazas de aplicaciones realistas que pongan de manfiesto la interacción entre sistema operativo y software de aplicación a través de simulaciones HDL.

3. El estudio y la validación de los resultados experimentales extraídos mediante la simulación HDL de las alternativas a considerar en el diseño de interfaces de red eficiente. En ese trabajo se hará uso de modelos teóricos que, como LAWS, proporcionan descripciones del espacio de diseño en términos de un número relativamente reducido de parámetros para facilitar el enfoque del trabajo experimental en las zonas más interesantes de dicho espacio de diseño.

4. La propuesta, simulación, y análisis experimental de los resultados obtenidos para nuevas alternativas que permitan aumentar las prestaciones de las interfaces de red, aprovechando la disponibilidad, en los nodos de los servidores actuales, de mejores recursos en cuanto a integración de buses de E/S, jerarquías de memoria cache, núcleos de procesamiento, y NICs .

Resumen de los capítulos de la presente memoria

A continuación se presenta una breve reseña de los contenidos de cada uno de los capítulos de esta memoria.

Capítulo 1: **Introducción y fundamentos de las operaciones de la interfaz de red**. Este capítulo proporciona una introducción a los conceptos y las líneas de trabajo en el ámbito de la mejora de prestaciones de las arquitecturas de comunicación. Así, se revisan algunas cuestiones relacionadas con la clasificación de distintas aproximaciones al diseño de interfaces de red eficientes, y se presentan las características de las tarjetas de red (NIC) y cómo han evolucionando con la velocidad de los enlaces (NIC de Gigabits por segundo), y de la red *Ethernet* y algunas de las redes de altas prestaciones más utilizadas. Además, se describen los distintos factores que afectan a la sobrecarga de comunicación, y se hace una breve descripción de las técnicas de *externalización* (incluyendo las alternativas de o*ffloading y onloading*) y las propuestas de utilizar TOE (*TCP Offload Engines*) para reducir la sobrecarga de comunicación en el procesador en el que se ejecutan las aplicaciones. También se describen los modelos teóricos LAWS y EMO, que ayudan a organizar la exploración del espacio de parámetros a considerar en el análisis de la *externalización* y proporcionan aproximaciones cuantitativas de la máxima mejora de prestaciones que cabe esperar al aplicar esta técnica. Además, se describen brevemente algunos simuladores de sistema completo, como M5, SimOS, SIMICS, GEMS, TFsim. La aproximación que se sigue en esta memoria para la simulación del camino de comunicación se basa en el uso de modelos HDL, y por lo tanto es distinta a la que ofrecen los simuladores de sistema completo, que permiten utilizar modelos más realistas de la ejecución de las aplicaciones y el sistema operativo y que, precisamente por esta razón se han utilizado (en concreto el simulador SIMICS) para validar los resultados de nuestras simulaciones HDL.

Capítulo 2: **Modelo en lenguaje de descripción de hardware (HDL) y optimización de la Interfaz de Red.** En este capítulo se describen los modelos HDL que se han elaborado para evaluar distintas alternativas de mejora de la interfaz de red basadas en la externalización. También se discuten las ventajas e inconvenientes de

este tipo de modelos y se describen los detalles sobre la estructura de los módulos HDL (CPU, NIC, *chipset*, memoria, etc.) que configuran los caminos de comunicación para las interfaces sin *externalización* y con *externalización* (a través de *offloading* y *onloading*) que se van a analizar.

Capítulo 3: **Evaluación de los resultados experimentales e identificación de oportunidades de mejora**. En este capítulo se presentan los resultados obtenidos y se realiza un análisis comparativo de las prestaciones de las técnicas de *offloading* y de *onloading*. En la evaluación y análisis de las prestaciones de comunicación se utilizan como referencia las predicciones cuantitativas del modelo teórico LAWS [SHI03], que facilitan en muchos casos la justificación de los resultados obtenidos

Capítulo 4: **Diseño de experimentos en el análisis de los parámetros del camino de comunicación.** Este capítulo se presentan algunas alternativas para evaluar los resultados experimentales del simulador en el camino de comunicación utilizando las técnicas estadísticas de diseño de experimentos. El diseño de experimentos se ha utilizado para definir una estrategia experimental que permite estudiar simultáneamente el efecto de varios factores en el comportamiento del sistema. Los experimentos realizados posteriormente nos han permitido poner de manifiesto con mayor detalle el efecto de los distintos factores del modelo LAWS.

Capítulo 5: **Conclusiones y trabajo futuro**. Este capítulo presenta las conclusiones y principales aportaciones del trabajo realizado y las líneas de futuras que quedan abiertas para proseguir nuestra investigación. También se recogen aquí las publicaciones que recogen parte del trabajo que se presenta en esta memoria.

La memoria concluye con un apéndice dedicado a una descripción detallado de los modelos HDL más relevantes que se han elaborado, y las referencias a la bibliografía utilizada.

Capítulo 1

INTRODUCCIÓN Y FUNDAMENTOS DE LAS OPERACIONES DE LA TARJETA DE RED

Este capítulo introduce el tema que se aborda en esta tesis incluyendo los objetivos del trabajo de investigación realizado. Se presentan los conceptos fundamentales de las tarjetas de red NICs (*Network Interface cards*) y los tipos de enlaces de alta velocidad que permiten alcanzar altas prestaciones.

Una interfaz de red, NI (*Network Interface*), es el conjunto de elementos hardware/software que se encargan de comunicar el computador a través de una red. Las NI incluyen dispositivos periféricos, las tarjetas de red o NIC (*Network Interface Card*), que transfieren datos entre la memoria principal y la red. Las NIC ocupan la capa de enlace de datos en el sistema operativo. En la pila de protocolos TCP/IP tradicional, las NICs ocupen a la capa que hay debajo de la capa IP, y envían/reciben paquetes a/de la capa IP. Finalmente, desde el punto de vista del software de sistema, las NICs proporcionan un interfaz a la CPU del nodo de modo que el controlador de dispositivos que se ejecuta en la CPU pueda comunicarse con la red.

En este capítulo se describe en detalle el funcionamiento de la NI y la sobrecarga generada por la comunicación. Además, se presenten las estrategias propuestas para aumentar las prestaciones de la comunicación como por ejemplo, los protocolo ligeros (CLIC y GAMMA) [DIA01, CIA01], las interfaces a nivel de usuario como VIA [BHO98], y la *externalización* mediante el *offloading* [WES04, MOG03] y el *onloading* [REG04a]. Finalmente, se explican las características de unos modelos de evaluación (LAWS y EMO) [SHI03, BUC01] para analizar y evaluar las prestaciones del camino de comunicación.

Para presentar todos estos conceptos se ha organizado el capítulo como se indica a continuación. En la sección 1.1 se introduce el problema del aprovechamiento del ancho de banda que proporcionan las redes con enlaces cada vez más rápidos y se descubren los elementos intervienen en la comunicación entre computadores. La sección 1.2 describe las tareas y los elementos de la interfaz de red para pasar a analizar los aspectos de la sobrecarga de comunicación en la sección 1.3 y las principales estrategias propuestas para aumentar las prestaciones del camino de comunicación en la sección 1.4. En la sección 1.5 se introduce el concepto de *externalización* y se presentan las diferentes alternativas para lo mismo. En esta misma sección se describen arquitecturas de tarjetas de red orientada a la *externalización* de la comunicación para poder aprovechar las anchas de bandas de varios gigabits por segundos. La sección 1.6 se dedica a describir los modelos LAWS y EMO para analizar las técnicas de *externalización* de forma cuantitativa aproximada. También se describe la relación entre el modelo LAWS y el modelo EMO. El modelo LAWS será el que fundamentalmente usarnos en este tesis para analizar los resultados experimentales que obtengamos con nuestros modelos de simulación HDL. Finalmente, la sección 1.7 se dedica a analizar los distintos tipos de simuladores y las conclusiones del capítulo se recogen en la sección 1.8.

1.1 Introducción

Las prestaciones de comunicaciones dependen tanto de: (1) Las prestaciones de la red que permiten la interconexión entre nodos; y (2) Las operaciones realizadas en origen y destino para inyectar y recibir mensajes de la red.

Con la disponibilidad de enlaces de ancho de banda elevado (*Gigabit Ethernet*, *GigaNet, SCI, Myrinet, Hippi-6400*) [OME06, CHU80, KEE95, QUA10], el cuello de botella de la comunicación ha pasado de los enlaces a los nodos de la red. Así, las características de las interfaces de red (NI) son cada vez más determinantes en las prestaciones de comunicación y es más importante reducir la sobrecarga (*overhead*) asociado al procesamiento de los protocolos de comunicación, cuyas causas fundamentales se encuentran en los cambios de contexto, las copias múltiples de datos, y los mecanismos de interrupción.

La *externalización* de los protocolos de comunicación (*offloading*) al dispositivo de red es una aproximación atractiva para reducir este sobrecoste de comunicación [DIA05].

Los TOEs (TCP o*ffload Engines*) son tarjetas de red comerciales capaces de ejecutar los protocolos TCP/IP. Los fabricantes afirman que esta capacidad permite manejar el flujo de datos a la velocidad de enlace en redes Ethernet de 10Gbps. Así, los TOE podrían ser una solución interesante para aliviar la carga de trabajo de procesador del nodo. Sin embargo, hay cierta controversia sobre las ventajas de los TOE, aunque un TOE pueda facilitar la eliminación de copias de memoria para aplicaciones directamente construidas sobre los protocolos TCP/IP. La ayuda de un TOE se limita a eliminación de copias de memoria para un protocolo de capa superior ULP (*Upper Layer Protocol*) con sus propias cabeceras y carga de datos útiles [WES04].

Las velocidades del enlace aumentan de manera que el procesador del nodo no puede mantener el ancho de banda de los datos proporcionados por enlaces de 10 Gigabit/s. Por lo tanto, hay una tendencia cada vez más clara hacia tarjetas de interfaz de red programables PNI (*Programmable Network Interfaces*) o tarjetas de red inteligentes INIC (*Intelligent* NIC), incluso con varios procesadores para proporcionar las funciones necesarias y alcanzar los requisitos de prestaciones de comunicación precisas tanto para las aplicaciones actuales como para las emergentes, liberando a la CPU del trabajo relacionado con la comunicación, proporcionando flexibilidad, y reduciendo los tiempos de desarrollo de los sistemas de comunicación [WIL05].

La NIC debería disponer de suficiente capacidad de procesamiento en el NIC para evitar el desfase entre el rendimiento de la red y del procesador del nodo, y mejorar el rendimiento del sistema [BHO98]. Dentro de esta línea se puede incluir el uso de procesadores de red, NP (*Network Processors*) [IXP05]. Los NP son circuitos programables optimizados para el procesamiento de funciones de comunicación a

velocidades elevadas, muy importantes en el diseño de los encaminados actuales. Además, estos procesadores suelen incluir otras componentes hardware para acelerar operaciones comunes en esas funciones de comunicación (como por ejemplo, el procesamiento de CRC) [ORT05].

La implementación de todas las funciones de comunicación en la NIC (*offloading*), a la que nos hemos referido antes, tiene una serie de ventajas que afectan muy directamente a las prestaciones de las aplicaciones [DIA05]:

1. La CPU no tendría que procesar los protocolos de comunicación.
2. La tarjeta de red, al implementar los protocolos de comunicación, podría interaccionar con la red sin la intervención de la CPU, con dos consecuencias importantes:
   2.1 Se reduciría la latencia del protocolo de comunicación correspondiente, ya que mensajes cortos, como los ACK, no tendrían que atravesar el bus de E/S y se reduciría el tráfico en el mismo.
   2.2 La CPU no tendría que procesar interrupciones para cambiar de contexto y atender la llegada de mensajes.
3. Se puede mejorar la eficiencia de las transferencias de DMA de mensajes cortos desde la NIC.
4. La utilización de una NIC inteligente con recurso que permite aprovechar distintos niveles de paralelismo podría mejorar las prestaciones en el procesamiento de los protocolos de comunicación, y ofrece la posibilidad de gestionar dinámicamente los protocolos, utilizando el protocolo más adecuado para construir el mensaje a enviar según los datos a comunicar y el destino.

En cualquier caso, hay factores en la *externalización*, de carácter fundamental, que afectan a la mejora de prestaciones. Entre estos factores se encuentra el aumento de prestaciones de los microprocesadores según la ley de *Moore*, que hace que las prestaciones de la CPU queden bastante por encima de las de los procesadores de propósito específico que se utilizan en la NIC. Así, estos procesadores pasarían a ser el cuello de botella del sistema de comunicación en la *externalización* y limitarían la velocidad a la que se procesa el protocolo. La utilización de procesadores de propósito general en las NIC puede significar un compromiso bastante malo entre coste y prestaciones según algunos autores, que incluso consideran que el coste de

procesamiento de un protocolo como TCP no justifica la *externalización*. Además, las limitaciones en los recursos disponibles en la NIC, fundamentalmente en memoria, pueden ocasionar restricciones en la escalabilidad del sistema [CLA89].

Sí el protocolo de transporte reside en la NIC, debe existir una coordinación entre el sistema operativo (SO) y la NIC para gestionar correctamente recursos como los *buffers*, los números de puerto, etc. En el caso de protocolos como TCP el control de los *buffers* TCP es bastante complicado, reduciendo los posibles beneficios de la *externalización*. Por otro lado, la ineficiencia de las conexiones TCP cortas se debe, en gran medida, a la gestión de los eventos visibles para la aplicación, y éstos no se pueden evitar por la *externilización* del protocolo. Estos problemas se ponen más claramente de manifiesto en el uso que se hace del protocolo TCP en aplicaciones WAN o en aplicaciones LAN que requieren bajo ancho de banda (como *Telenet*) [HYD00].

Así, las razones que se aducen para estos reparos a la *externalización* se refieren, por un lado, a las dificultades que surgen en la implementación, el *test*, y el mantenimiento de la interfaz. Estos aspectos no constituyen argumentos definitivos en contra de las mejoras que puede aportar la *externalización*, pero deben contrapesarse con las posibles beneficios que aporte [MOG03].

A comunicación se consideran brevemente los aspectos que influyen en el rendimiento de la red: el medio físico, la técnica de conmutación, el procedimiento de encaminamiento [LEO02].

A. El medio físico: en líneas generales, hay que tener un cuenta que existen medios fabricados con cable de cobre o con fibra óptica, que previsiblemente será la que sustituirá a los cables dada las limitaciones para la transmisión a grandes distancias y menor ancho de banda de éstos .

B. La técnica de conmutación: permite la conexión y la transferencia de datos entre puertos de entrada y puertos de salida en un etapa de la red. Las dos técnicas más utilizadas en las redes son la conmutación de paquetes, que almacena un paquete completo en el conmutador antes de enviarlo por el puerto de salida correspondiente, y el conmutación segmentado, que envía datos a la siguiente etapa tan pronto como se decodificado la dirección contenida en la cabecera del paquete

C. El procedimiento de encaminamiento: normalmente se utilizan dos procedimientos de encaminamiento, (1) encaminamiento fuente, en el que la dirección de la cabecera del paquete lleva información acerca del camino a tomar en cada conmutador, y (2) encaminamiento con tabla, donde cada conmutador tiene una tabla en la que se indica el puerto de salida para cada destino.

Como se ha descrito anteriormente, las prestaciones de *Ethernet* se han ido mejorando al pasar de un ancho de banda máximo de 10Mbps a 100Mbps con *Fast Ethernet* y al utilizar conmutadores, aunque suponen un moderado incremento en el coste del sistema con respecto al uso de concentradores, mejoran considerablemente el rendimiento del sistema, debido fundamentalmente a la reducción de los conflictos de acceso a la red [HEL03].

A continuación se describen las características más significativas de estas redes que proporcionan anchos de banda que en la mayoría de los casos están en torno de Gbps, o como en el caso de *Infiniband***,** incluso pueden llegar a valores del orden de 10Gbps. Dado que en esta memoria no se pretenden realizar aportaciones en esta área, sólo describiremos algunas de las redes de altas prestaciones disponibles. Concretamente se considerarán *Gigabit Ethernet*, por ser la red sucesora de *Ethernet*; *Myrinet*, que es una de las redes precursoras en alcanzar prestaciones del orden del *Gigabit por segundo* y es una de las más populares en la configuración de clusters, ATM que está dedicado a la telecomunicación global, e Infiniband que es un sistema de comunicación para conectar subsistemas y redes de sistemas.

Gigabit Ethernet (IEEE 802.3z). El siguiente desarrollo a partir de *Fast Ethernet* es *Gigabit Ethernet*. Los enlaces de Gigabit Ethernet transmiten información a 1Gbps en el nivel físico, aunque debido al modelo de codificación que implementan sólo permiten alcanzar 800Mbps útiles (100 Mbytes/s). Aprovechando la casi aceptación universal de la tecnología *Ethernet 10Base-T* que opera a 10Mbps, la tecnología *Fast Ethernet* ha ofrecido una evolución progresiva para obtener 100Mbps. Análogamente, *Gigabit Ethernet* provee un ancho de banda de 1Gbps a un costo más bajo que otras tecnologías de velocidades similares [OLI00]. Los estándares de fibra óptica corresponden a una mejora de la tecnología actual de especificación ANSI (*American National Standards Institute*) para la capa física de *Fiber channel*. Por consiguiente, *Gigabit Ethernet* utiliza el mismo esquema de codificación/decodificación para los datos digitales que *Fiber Channel*, esto es B/10B [OME06].

ATM (*Asynchronous Transfer Mode*). ATM está orientado al sector global de telecomunicaciones, partiendo de diversos estándares previos como TCP/IP y el modelo OSI. Se trata de una tecnología de comunicación de paquetes *Store/Forword* orientada a conexión y encaminamiento por tabla, que emplea celdas de longitud fija formadas por una cabecera de 5bytes y un campo de datos de 48bytes. El estándar ATM define una arquitectura que puede crecer con la tecnología y se puede adaptar y ajustar a los requisitos de diferentes aplicaciones, como las de tiempo real, multimedia, etc. que necesitan que la red pueda ofrecer flujos constantes de datos. Estos requisitos se engloban bajo el concepto de QoS **(***Quality of Service*) [ATM59, CIS00].

Myrinet. Es una red desarrollada por *Myricom*, que surge a partir de los proyectos de investigación Mosaic y Atomic LAN. Se trata de una red de área local basada en la tecnología para comunicación de paquetes en computadores masivamente paralelos. Así, utiliza conmutación segmentada (*Wormhole*), y encaminamiento basado en fuente. Un enlace *Myrinet* está constituido por dos canales *full-duplex* que proporcionan 1.28 Mb/s, y también está disponible una mejora que proporciona 2.56 Gb/s. La transmisión es síncrona enviando caracteres de 9 bits (un byte de datos o uno de los cinco caracteres de control que existen). Se tiene en cuenta que la tasa de errores en la transmisión es muy baja al disponer de un procedimiento de control de flujo con poca sobrecarga. Conforme los datos se van enviado de un nodo a otro, si el receptor sobrepasa un umbral superior en sus *buffers* interiores, envía un *bit* de parada al trasmisor para que bloquee la transmisión. Tan pronto como vuelve a pasar otro umbral inferior, el receptor informa al sistema que puede continuar.

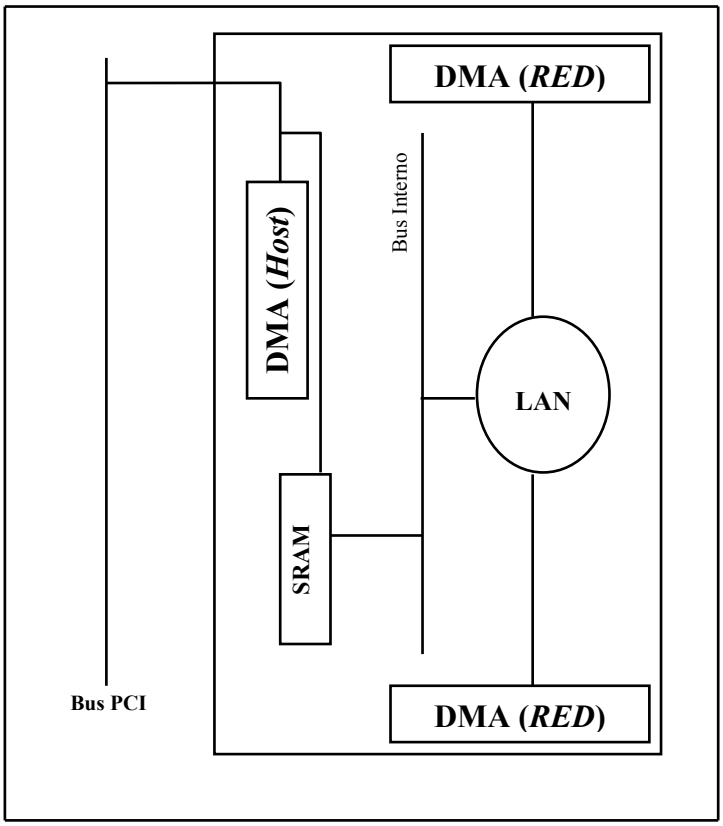

Figura 1.1 Tarjeta de interfaz de red *Myrinet* para bus PCI

En la Figura 1.1 se muestra un diagrama de bloques de un versión de tarjeta de interfaz para conectar la red Myrinet a un bus PCI. La tarjeta dispone de procesador de control de 32 bits, denominado, LAN con una memoria SRAM de 1MByte. La SRAM se utiliza para almacenamiento temporal de los paquetes, y para el código y los datos para el procesador LANai. Hay elementos de hardware para poder realizar simultáneamente tres transferencias por DMA: dos permiten la transferencia de los paquetas (de entrada y salida) entre la red y SRAM, y el otro permite la transferencia entre la SRAM y la memoria principal de computador a través del bus PCI. El procesador LANai ejecuta un programa de control LCP que supervisa el funcionamiento de los tres canales de DMA e implementa un protocolo de comunicación de bajo nivel. Este procesador no puede acceder directamente a la memoria principal de computador, sino que lo tiene que hacer a través del correspondiente canal de DMA. El bus interno de la tarjeta funciona a una frecuencia doble de la del procesador permitiendo que dos canales de DMA cooperen (uno introduciendo datos en la tarjeta y otro sacándolos) a plena velocidad. El computador al que está conectada la tarjeta puede acceder a la SRAM mediante E/S programada, y todos los recursos de la tarjeta de red (memoria y registros de control) se pueden direccionar en el espacio de memoria de usuario del

computador. Las transferencias de DMA entre el computador y la interfaz de red pueden iniciarse tanto por parte del procesador LANai como por la CPU del computador, utilizándose en ambos casos el canal de DMA de la interfaz. Para esta red se han desarrollado varios sistemas de mensajes [CHU80].

Infiniband. Es un estándar abierto que propone un sistema común de comunicación para conectar subsistemas, sistemas, y redes de sistemas [MEL03]. Aborda de forma unificada dos situaciones que plantean cuellos de botella: las transferencias de E/S entre el procesador y ciertos dispositivos, y la comunicación en sistemas distribuidos a través de red de área local. La arquitectura de *Infiniband* permite reemplazar el bus de entrada y salida compartido estándar en los computadores actuales para las E/S por un sistema de comunicación (*Switch Fabric*) serie, de alta velocidad, basado en canales para el paso de mensajes, y escalable. Cada cable permite un ancho de banda de 2.5 Gbit/s en una dirección. Los datos se envían en paquetes, existiendo seis tipos de transferencias: conexión fiable, conexión no fiable, datagrama fiable, datagarama no fiable, conexión multidifusión, y paquetes aislados (*raw packets*). En el caso de conexión fiable, el hardware garantiza la entrega ordenada y fiable de los datos. Para ello, genera un código de reconocimiento, ACK, para todo paquete recibido, genera y comprueba los correspondientes números de secuencia, detecta paquetes perdidos, y rechaza los duplicados. Los errores se detectan tanto en el lado que solicita datos como en el que los suministra, y el que los solicita puede recuperar errores e intentar caminos de comunicación alternativos sin que la aplicación tenga que intervenir. Como se muestra en la Figura 1.2, todos los sistemas y dispositivos se conectan al sistema de comunicación a través de adaptadores de canal para computador, HCA (*Host Cannal Adaptor*), o adaptadores de canal para dispositivos, TCA (*Target Cannal Adaptor*). El HCA proporciona una interfaz a la red basada en el estándar VIA, como mecanismo para reducir la sobrecarga *overhead* de comunicación, evitando la intervención del sistema operativo en el envío/recepción de mensajes. El ancho de banda de una red se está convirtiendo en los MIPS en el sentido de *Meaningless Information of Processor Speed* del mercado de las redes. En consecuencia, como se ha indicado anteriormente, las prestaciones de comunicación que aproveche una aplicación dependerán, además del ancho de banda y del retardo de la red, de la sobrecarga asociada a la capa de mensajes en el emisor y receptor, y de los patrones de comunicación, el tamaño de mensajes, y la congestión de la red (c*ontention*) determinadas por la aplicación. Así, si se analiza la influencia en las prestaciones de comunicación de los factores indicados utilizando

diferentes tipos aplicaciones, se puede observar experimentalmente que la sobrecarga (o*verhead*) de comunicación es el factor más dominante en las prestaciones, sobre todo en el caso de aplicaciones con mucha comunicación del tipo petición-respuesta con mensajes cortos [KEE95].

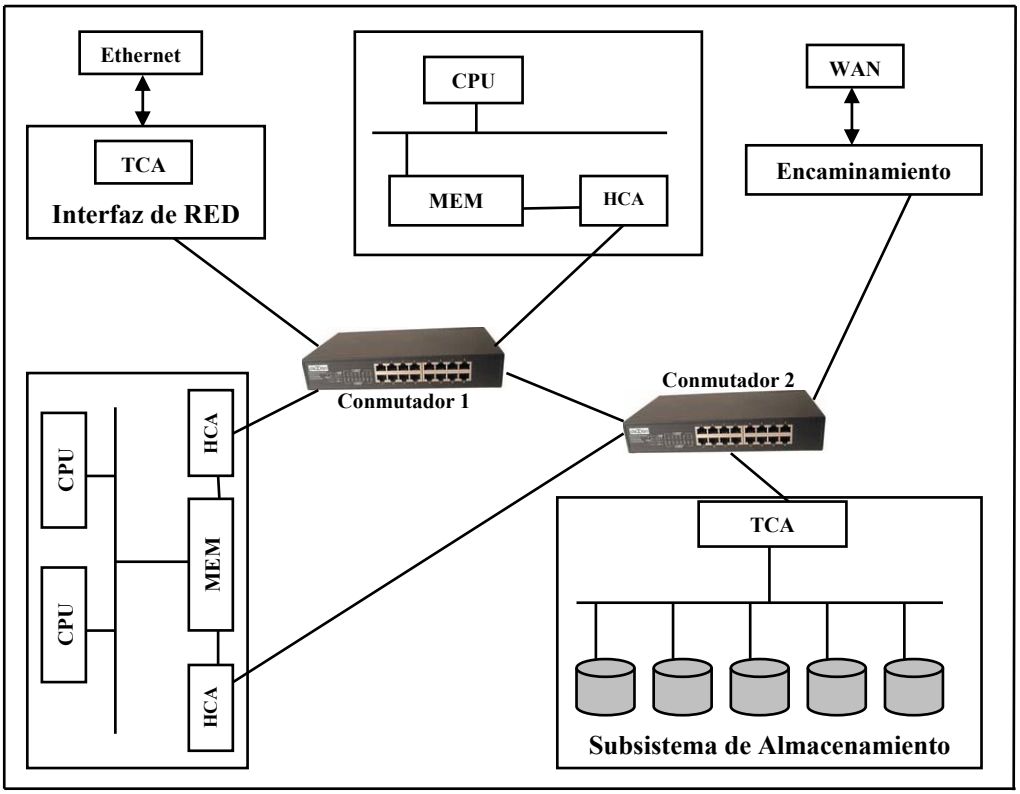

Figura 1.2 Ejemplo de sistema conectado a partir de conmutadores *Infiniband*

1.2 Tareas de la interfaz de red

La interfaz de red puede realizar diferentes tareas para la transferencia de los datos. Entre ellos se encuentran los siguientes [LEO02]:

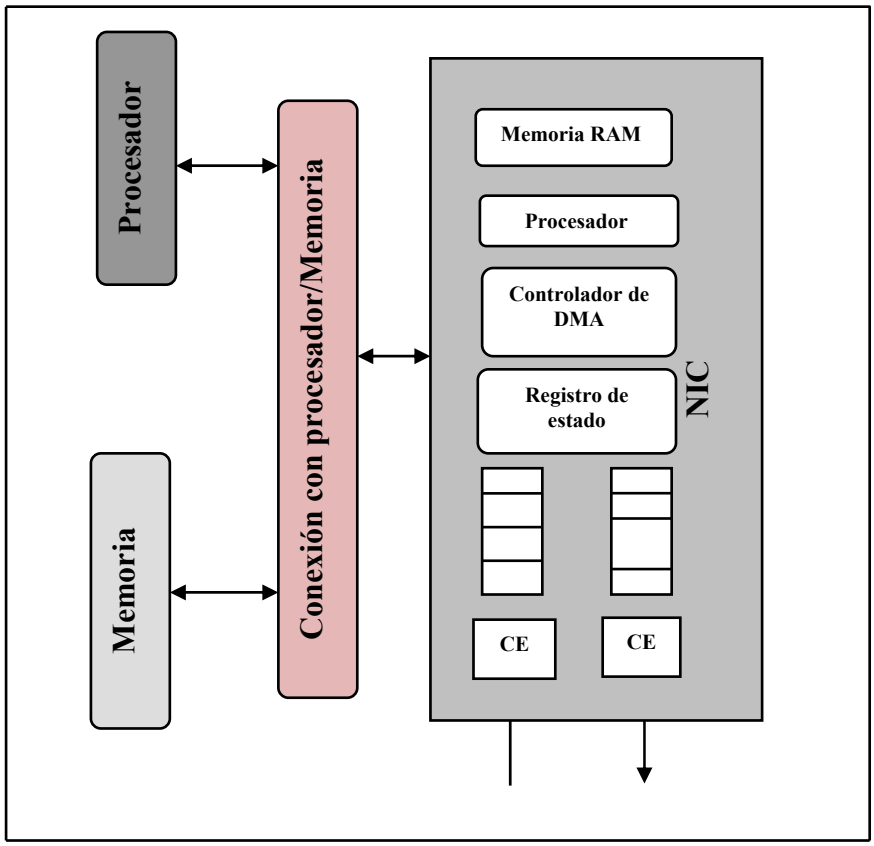

Figura 1.3 Esquema de los elementos hardware que intervienen en la interfaz de red (se considera una NIC con procesador incluido)

1. Almacenamiento (*buffer*) de paquetes entrantes y salientes.
2. Construcción del paquete y extracción de la información de los paquetes recibidos.
3. Copia de datos.
4. Protección y traducción de direcciones.
5. Control de Flujo.
6. Direccionamiento, encaminamiento y reconfiguración

Entre los elementos hardware de la interfaz de red está la tarjeta de red, NIC (*Network Interface card*), como se muestra en Figura 1.3. La NIC se encarga de transmitir y

recibir paquetes a través de dos colas como se muestra en la Figura 1.3 En una de ellas se colocan los paquetes que se van a transferir a la red, y en la otra se van acomodando los paquetes que llegan de la red desde otro nodo. Las tareas implementadas directamente en hardware suelen consumir menos tiempo que las implementadas en software. Además, si la CPU no se encarga de esas tareas puede invertir los ciclos correspondientes en la propia aplicación. Por ejemplo, si la división en paquetes de un mensaje se realiza en la NIC en lugar de que sea el procesador del nodo quien lo haga, ejecutando el correspondiente software de la interfaz dispondrá de más ciclos libres. Hay tareas que usualmente se ejecutan en un hardware específico, como el cálculo de los bits de detección de errores o de la dirección de destino. Las tareas implementadas en software, las ejecuta el procesador del nodo o un procesador incluido en la tarjeta de red (procesador de E/S). Cuantas más tareas realice este procesador de propósito específico más tiempo puede dedicar el procesador del nodo a la aplicación. Probablemente el procesador que ayuda a la comunicación esté situado en la tarjeta de red, como se ha comentado, pero también puede tratarse de un procesador adicional conectado al bus de sistema, como por ejemplo en el Intel la técnica de *onloading* [REG04a]. La memoria de la NIC (figura 1.3) se puede utilizar para almacenar el código a ejecutar por un procesador incluido a la NIC y también puede almacenar la información que hay que enviar o que se recibe. Es importante disponer de distintos componentes que puedan realizar en paralelo diferentes tareas del proceso de comunicación, organizado el camino de comunicación en el computador como un cauce segmentado (*pipeline*). Todas estas cuestiones se analizarán con más detalle en las secciones siguientes y su estudio constituye uno de los aspectos centrales de esta tesis.

1.2.1 Almacenamiento y control de buffers

El almacenamiento para la salida y entrada de paquetes se puede implementar a través de registros o en la memoria de la NIC. Generalmente se gestiona como una cola (por ejemplo con un estructura circular con punteros a la cabecera y a la cola). Si los paquetes no tienen un tamaño fijo, o un conjunto de tamaños fijos, se puede asociar al almacenamiento una cola de descriptores de paquetes con punteros a los datos en la memoria de la NIC. Porciones de paquetes irían pasando entonces en secuencia desde el

almacenamiento de entrada a la memoria de la NIC, o desde esta memoria al almacenamiento de salida. Los controladores de enlace (CE) se encargan de gestionar el envío y la recepción de unidades de transferencia a través del enlace físico, de acuerdo con las características que tenga este medio físico.

La interfaz genera información de control para los datos que se van a transferir a través de la red. Esta información de control se añade al paquete como se muestra en la Figura 1.4, o se envía por líneas de control específicas. En lo que sigue, consideramos que se adjuntan a los datos. La interfaz añade al paquete una cabecera (C) con información sobre el camino que ha de seguir para llegar al destino, o con el identificador del terminal destino. Esta información será utilizada por los enrutadores para determinar por cuál de sus enlaces de salida han de encaminar el paquete con el fin de que éste llegue a su destino.

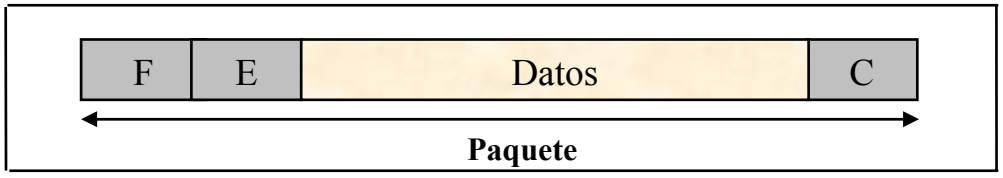

Figura 1.4 Formato de un paquete

Además, la interfaz genera un código para detectar errores en la transferencia del paquete. Esta información (E) se suele añadir al final de paquete. La interfaz de red en el destino, y la propia red, utilizarán estos bits para detectar y corregir errores en la transferencia, y el resultado de la este análisis es utilizado por la función de control de flujo, por ejemplo, para generar un paquete de reconocimiento que se envía a la interfaz origen del paquete. La cabecera puede tener asociados sus propios bits para detección de error. Igualmente, se pueden añadir códigos para detectar el final del paquete (F). Estos códigos se pueden utilizar, por ejemplo, para liberar el camino utilizado en la transferencia. La interfaz de red del destino extrae los datos y utiliza la información de control añadida en la fuente [DAL04].

En la Figura 1.5 se muestra una descripción del aspecto real de un paquete. Hay varias herramientas que permiten analizar qué paquetes están entrando y saliendo en una máquina Linux (*packet sniffers*). El más común es *tcpdump* [TCP10]. Como se ha dicho, el principio de cada paquete dice a dónde va, de dónde viene, el tipo de paquete, y otros detalles para su gestión. Esta parte se denomina cabecera del paquete. El resto

del paquete contiene los datos a transmitir en sí, y normalmente se denomina cuerpo del paquete. Por lo tanto, cualquier paquete IP comienza con la cabecera IP de al menos 20 bytes de largo. La Figura 1.5 se muestra una cabecera de IP. Los campos importantes son el protocolo, que indica si es un paquete TCP (código 6), UDP (código 17) u otro, la dirección IP de Origen, y la dirección IP de Destino.

Versión	**IHL**	**Tipo de Servic.**	**Tamaño Total**
Identificación		**Flags**	**Desplaz. del Fragmento**
Tiempo de Vida	**Protocolo**		**Checksum de la cabecera**
Dirección de Origen			
Dirección de Destino			

Figura 1.5 Una Cabecera de IP

Si el campo de protocolo indica que se trata de un paquete TCP, entonces a esta cabecera IP le sigue inmediatamente una cabecera TCP: la cabecera TCP también tiene al menos 20 bytes de longitud, como se muestra en la Figura 1.6.

Puerto de Origen								**Puerto de Destino**
Número de Secuencia								
Número de Confirmación								
Deplz. de los Datos	**Reservado**	**URG**	**ACK**	**PSH**	**RST**	**SYN**	**FIN**	**Ventana**
Checksum								**Puntero de Urgencia**

Figura 1.6 Una Cabecera de TCP

Los campos más importantes son el puerto de origen y el de destino, que indican el servicio al que está destinado el paquete (o de cual viene, en el caso de que sea un paquete recibido). Los números de secuencia y confirmación (*acknowledgement*) se utilizan para mantener el orden de los paquetes, y decir a los otros extremos cuántos

paquetes se han recibido. Los indicadores (*flags*) *ACK, SYN, RST y FIN* son bits que se utilizan en la negociación de apertura (*SYN*) y cierre (*RST o FIN*) de las conexiones [RUS00].

Siguiendo a la cabecera vienen los datos que la aplicación envía (el cuerpo del paquete). Un paquete normal Ethernet puede tener hasta 1500bytes. Esto significa que el mayor espacio que pueden ocupar los datos es de 1460bytes (20bytes para la cabecera IP y 20 para la cabecera TCP). Es decir, alrededor del 97% del paquete. Frecuentemente se denomina paquete a la unidad de transferencia entre interfaces, independientemente de que se transfiera todo el mensaje de una vez o dividido en partes [RUS00].

1.2.2 Control de dispositivos

Como se ha comentado, el dato o datos que se van a transferir se deben formatear convenientemente antes de ser enviados. Esto puede requerir un almacenamiento provisional de los datos. También deben trasladarse al NIC, donde quedan almacenados hasta que pueden enviarse a la red o mientras no esté garantizado que lleguen correctamente al destino. Los datos a enviar se encuentran inicialmente en posiciones de memoria de usuario. Desde ahí deben pasar a la NIC, donde se almacenan hasta que se envíen a la red. Además, para poder enviarlos a la NIC, ésta debe estar preparada. Por otra parte, los datos recibidos deben terminar en la memoria de usuario del destino. Estos datos recibidos deben quedar almacenados en la interfaz hasta que se pueda disponer de la correspondiente memoria principal. En consecuencia, se necesita almacenar los datos durante un tiempo (espera, formateo) y realizar copias tanto en la recepción como en la emisión [SHI09].

Mediante el control de flujo se garantiza a los niveles superiores de la interfaz de red que los datos se transfieren correctamente entre las interfaces de red fuente y destino. Es decir, se garantiza que las transferencias son fiables. Para ello el control de flujo utiliza paquetes de control y aprovecha el hardware disponible en la NIC. Además, con el control de flujo se puede garantizar cierta calidad en el servicio (QoS) [SAR06]; por ejemplo, no inyectando paquetes en la red si se detecta que está próxima a la saturación.

1.2.3 Tarjeta de red y driver de dispositivos

Normalmente, para acceder a los dispositivos de E/S (y como caso particular, para acceder a una red de comunicación), el *driver* de dispositivo tiene que formar parte del núcleo (*kernel*) del sistema operativo, como se muestra la Figura 1.7. Usualmente el núcleo implementa abstracciones del hardware que proporcionan una interfaz independiente de la implementación concreta del dispositivo. Mediante llamadas al sistema se accede a esa abstracción del dispositivo desde el nivel de usuario.

Para mejorar las prestaciones es preferible transferir bloques de datos entre la memoria y los dispositivos de E/S, en lugar de tener que establecer una conexión a través del SO sólo para un dato individual. Como es conocido, las transferencias entre memoria y dispositivo de E/S se pueden realizar mediante E/S programada, acceso directo a memoria DMA (*Direct Memory Access*), o E/S controlada por interrupción. Con la E/S programada, un procesador del computador realiza la transferencia y se ocupa de detectar la presencia de nuevos datos a transferir. Mediante DMA, el controlador de DMA se encarga de realizar las transferencias sin intervención de ningún procesador. En este caso, mediante una interrupción, se informa al procesador de eventos como la llegada de nuevos datos al dispositivo de E/S. Sí la interrupción indica que hay nuevos datos a transferir, el procesador programa el dispositivo de DMA para que realice la transferencia. El DMA es el procedimiento más conveniente para la transferencia de grandes bloques.

El driver de la tarjeta de red (NIC) puede mejorarse si se tienen en cuenta las características especificas de la red de interconexión, la NIC, y las características del nodo. Concretamente, se podrían:

1. *Eliminar las tareas no necesarias del protocolo.* Por ejemplo, si es alta la fiabilidad en la comunicación a través de la red, se puede implementar una transmisión optimista de una secuencia de paquetes evitando señales de control de comunicación para paquetes no validos.
2. *Aprovechar características específicas de la tarjeta de red.* Así, se pueden implementar en la NIC tareas que realiza usualmente el procesador del computador, incluyendo tareas que corresponden a la interfaz de red. Como por ejemplo, se pueden aprovechar características del hardware, no aprovechadas por el protocolo, a través del *driver*.

3. *Optimizar las tares que usualmente implementa el procesador*. Se trata de aprovechar las características de otros componentes del computador, aparte de la tarjeta de red. Por ejemplo, se pueden optimizar las copias desde memoria a la NIC a través del procesador mediante escrituras combinadas.
4. *Reducir el coste por intervenciones del núcleo***.** Por ejemplo, se pueden usar llamadas ligeras al sistema, que guardan menos registros y no llaman al planificador cuando terminan.

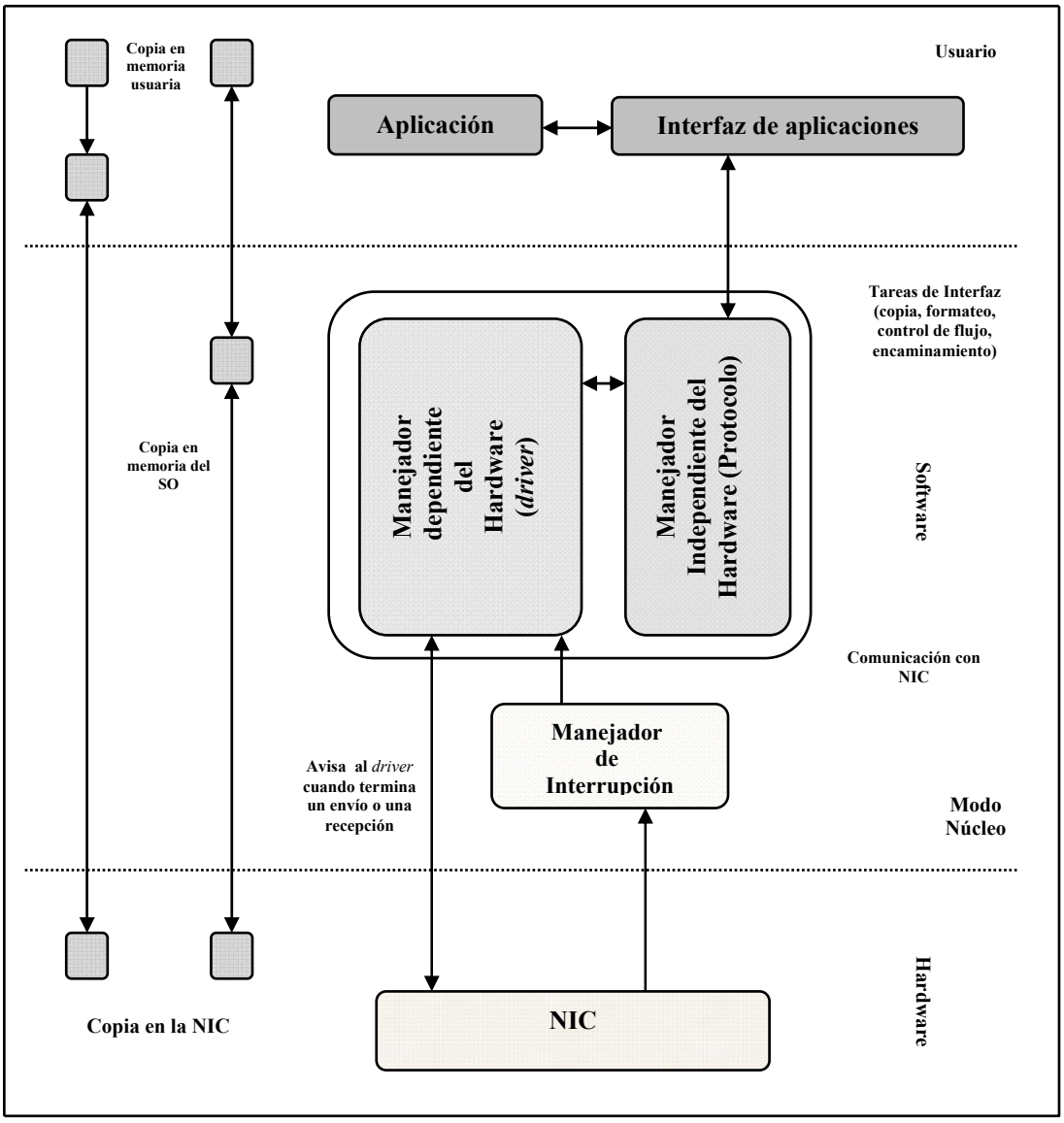

Figura 1.7 NIC genérico como dispositivo de E/S con acceso estándar a nivel de de sistema operativo

5. *Evitar copias de memoria-a-memoria (0-Copia)*. Se trata de evitar copias tanto a la memoria del núcleo como entre memoria a nivel de usuario. Por ejemplo, se

pueden bloquear en memoria de usuario las páginas de memoria a transferir a la NIC mediante DMA [SKE01].

6. *Eliminar costes de interrupción.* Se puede consultar el estado del dispositivo para detectar nuevos eventos (como la recepción de un paquete), y genera una interrupción tras recibir un conjunto de paquetes, en lugar de una para cada uno, o generar la interrupción sí ha trascurrido un tiempo desde la llegada del paquete sin que el procesador lo haya detectado por consulta de estado [GIL05].
7. *Segmentar convenientemente el camino de comunicación***.** Configurar un cauce eficiente en el camino de comunicación [HAL05].

Algunas de estas propuestas se considerarán de forma más detenida en las secciones siguientes, en el contexto de la sobrecarga de comunicación.

1.3 La sobrecarga (*overhead*) de comunicación

En el camino de comunicación existen diferentes elementos que pueden suponer un cuello de botella en las prestaciones del sistema de comunicación. Como se ha indicado en el emisor, los datos se estructuran convenientemente de acuerdo al protocolo de comunicación utilizado, y dichas estructuras se transfieren a la tarjeta de red NIC desde la memoria de usuario, normalmente mediante DMA. En el receptor, se realiza el proceso contrario. Una vez que los datos llegan a la NIC, ésta realiza una transferencia de DMA de los datos recibidos hasta la memoria principal. Una vez terminada la transferencia de DMA, la NIC envía una interrupción indicando al procesador la disponibilidad de los datos en la memoria principal para que pueda usarlos. Las transferencias entre la memoria principal y la NIC se realizan a través de un bus de E/S con un ancho de banda determinado, y las interrupciones generadas por la interfaz de red necesitan tiempo de CPU para ser atendidas. Esto puede suponer una limitación de las prestaciones en el caso de redes con ancho de banda de varios Gigabits/s. El tiempo de comunicación de un paquete de n bits podría expresarse como:

$$T_{comunicacion} = O_s + O_r + (L + {}^{n}/_{BW_{red}}) \qquad (1.1)$$

Donde O_s representa la sobrecarga (*overhead*) asociada al tiempo que tarda el emisor para preparar el paquete a enviar y en trasferir los datos desde la memoria principal hasta la red; O_r representa la sobrecarga (*overhead*) correspondiente al tiempo invertido en el receptor para transferir los datos desde la red hasta la memoria principal donde son accesibles para la aplicación; L (*latency*) es el tiempo de vuelo en la red; y n/BW_{red} el tiempo asociado a la transferencia de n bits en una red con ancho de banda BW_{red}. En la ecuación (1.1), la parte que está entre paréntesis correspondencia al tiempo que tarda la red en transmitir los n bits que conforman el paquete. Tenido 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 del tiempo de sobrecarga del emisor y el receptor pueden expresarse como los tiempos de transferencia entre la memoria y la NIC. Este tiempo puede como:

$$O_s + O_r = M * (n/BW_{red}) + N * (n/BW_{E/S}) + \Delta t \qquad (1.2)$$

Y, por lo tanto, sí se desprecie la latencia L (porque suponemos que el cuello de botella está en los nodos), se tiene que:

$$T_{comunicacion} = M * (n/BW_{red}) + N * (n/BW_{E/S}) + \Delta t + ({}^{n}/_{BW_{red}}) \qquad (1.3)$$

donde M y N representan el número de veces que se atraviesan los buses de memoria y de E/S, respectivamente, y Δt es el tiempo empleado en el procesamiento de los protocolos de comunicación (en los cambios de contexto, en la gestión de interrupción, etc.) que no se solapa con los tiempos de transferencia de información. El ancho de banda efectiva podría expresarse en términos del ancho de banda de la red y , mediante las ecuaciones (1.1) y (1.2):

$$BW_{efictivo} = \frac{n}{T_{comunicacion}} = \frac{BW_{red}}{1+M*(BW_{red}/BW_{memoria})+N*(BW_{red}/BW_{E/S})+\frac{\Delta t*BW_{red}}{n}} \qquad (1.4)$$

De esta expresión se pueden extraer algunas conclusiones. 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 las aplicaciones sea menor que el ancho de banda de la red. Cuanto menor sea el tamaño del paquete, mayor es la importancia del procesamiento de protocolos y de la gestión de las transferencias. Si los anchos de banda, $BW_{memoria}$ y $BW_{E/S}$, son muy elevados frente a $M*BW_{red}$ y $N*BW_{red}$ respectivamente, el término dominante en el denominador de la expresión (1.4), y por tanto en la reducción del ancho de banda efectivo, es $(\Delta t*BW_{red})/n$. A medida que BW_{red} aumenta, es más difícil alcanzar un ancho de banda efectivo $BW_{efecivo}$ próximo a BW_{red}, ya que se necesitarían buses de memoria y de E/S con anchos de banda ($BW_{memoria}$ y $BW_{E/S}$) suficientemente elevados, valores de M y N suficientemente pequeños, y se debe reducir el tiempo Δt, o utilizar valores de n muy grandes esta sería, por ejemplo, la justificación de los trames jumbo (paquete Jumbo) [GAD07]. Así, para poder aprovechar el ancho de banda que proporciona un enlace de red multigigabit (i.e. de ancho de banda de varios Gbits/s), es necesario reducir el tiempo de acceso de memoria, al tiempo que se optimizan las transferencias de datos en bus de E/S y la sobrecarga asociada a los protocolos de comunicación.

A continuación se consideran las principales aproximaciones que se han propuesto para reducir la sobrecarga de comunicación y conseguir interfaces de red más eficientes.

1.4 Estrategias para aumentar las prestaciones del camino de comunicación

Existen varias alternativas para mejorar las prestaciones del camino de comunicación y evitar que sea el cuello de botella. Entre éstas están los protocolos ligeros, que utilizan pilas de protocolos de comunicación para reducir la sobrecarga de comunicación, o las interfaces de red de nivel de usuario. También, se puede utilizar la *externalización* de los protocolos de comunicación para mejorar las prestaciones de comunicación distribuyendo las tareas de comunicación, como se explica a continuación en la sección 1.5. Además, hay otras estrategias que se usan para mejorar las prestaciones del camino de comunicación se explican con detalle en las próximas secciones.

1.4.1 Los protocolos ligeros

Mediante de los protocolos ligeros se pretenden evitar los cuellos de botella que normalmente suponen los buses de E/S y la memoria en el camino de comunicación. Dentro de este línea, además de los protocolos a nivel de usuario y protocolos que intenten mejorar la implementación de TCP/IP, están otros protocolos ligeros, como GAMMA [CIA01] y CLIC [DIA01, ORT05] que sustituyen las capas TCP/IP y mejoran el soporte que el sistema operativo ofrece a las comunicaciones.

El protocolo CLIC. El protocolo CLIC (*Communicaction on Linux Cluster*) está embebido en el núcleo de Linux, y proporciona un interfaz a las aplicaciones de usuario. CLIC se ha desarrollado para optimizar el paso de mensajes en clusters de computadores, minimizando el consumo de recursos del sistema operativo. Está implementado sobre el *kernel* de Linux, y facilita la programación multihebra, presentando un funcionamiento estable tanto para comunicaciones síncronas como asíncronas. Como se muestra en la Figura 1.8, CLIC reduce el número de capas de protocolo respecto a las utilizadas por la pila TCP/IP, disminuyendo la sobrecarga software y el número de copias necesarias. 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. Así, CLIC implementa las funcionas de transporte TCP y red IP, constituyendo una interfaz única entre la aplicación de usuario y el controlador de interfaz de red.

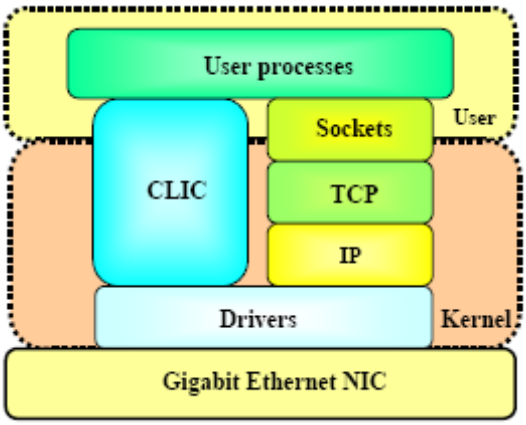

Figura 1.8 Comparación de las capas de CLIC y TCP/IP [DIA03]

Además, CLIC posee gran compatibilidad con los diferentes controladores de las tarjetas de red, permitiendo de esta manera el acceso compartido a la red y la escalabilidad necesaria para utilizar varios dispositivos de red y aumentar el ancho de banda disponible. Además, cuenta con un soporte eficiente para sistemas de paso de mensajes de alto nivel, como MPI y PVM. 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, tal y como se indicará tras presentar GAMMA.

El protocolo GAMMA. GAMMA [CIA01] es un protocolo ligero que está embebido en el núcleo de Linux y puede coexistir con servicios basados en IP en la misma red de área local. Sin embargo, el protocolo se basa en el uso de puertos activos para reducir la sobrecarga generada por la pila de protocolos TCP/IP. El protocolo GAMMA busca reducir el uso de la CPU y de los buses de E/S. Para ello, utiliza mecanismos de *copia-0*. Se consigue que, en la recepción, el uso de CPU sea proporcional al tamaño de paquete. Por otra parte, GAMMA no proporciona control de flujo para evitar la congestión y no garantiza unas prestaciones constantes sino que dichas prestaciones se ofrecen en modo de *best effort*, y no se garantiza una tasa de transferencia determinada. Además, GAMMA no dispone de mecanismos de retransmisión de paquetes. Un inconveniente del protocolo GAMMA es que depende de tarjeta de red, siendo necesario adaptar el conjunto de funciones de bajo nivel que se implementan en la interfaz BIND a cada tarjeta de red [CIA01].

1.4.2 Interfaces de red de nivel de usuario

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 de TCP/IP, no sólo por el propio protocolo, sino también por la integración con el sistema operativo del mismo [BHO98]. Así, se puede disminuir la sobrecarga de comunicaciones mediante interfaces de red de a nivel de usuario en las que se evita la intervención del sistema operativo.

En el diseño de interfaces de red a nivel de usuario [BHO98], surge un problema en direcciones físicas comunes a procesos distintos que se ejecutan concurrentemente,

ya que no existe la protección del sistema operativo. La protección impide que un proceso destruya el estado de otro proceso en la interfaz de red, así como sus datos e instrucciones en memoria principal. Sí varios programas pueden acceder a la tarjeta de red en una implementación con acceso a nivel de usuario, se deben evitar interferencias entre los estados de transferencia de distintos los procesos. Entonces hay que implementar algún mecanismo de protección que multiplexe en el tiempo el uso de la tarjeta o asignarlo cada vez a un proceso. Sí las transferencias entre la memoria del usuario y la NIC se realizan con DMA, hay que tener en cuenta que el controlador de DMA realiza las transferencias utilizando direcciones físicas de memoria, no virtuales. Estos aparecen dos problemas [PET03]:

1. Para realizar la transferencia entre zonas de memoria del usuario y la tarjeta de red se deben *bloquear en memoria* las páginas implicadas.
2. El DMA de la tarjeta de red debe conocer la *dirección física* y, normalmente, a nivel de usuario no se puede pasar la dirección física a la interfaz.

Estos problemas se evitarían utilizando E/S programada, pero entonces no se pueden aprovechar las beneficios del DMA. Se puede mantener en la NIC una cache con las *traducciones de dirección virtual a física* de las páginas bloqueadas en memoria de uso más reciente TLB (*Translation Lookaside Buffer*). Si la tarjeta de red comprueba que una dirección virtual no está en la cache, entonces pide al núcleo que bloquee la página y la proporciona su dirección física. Esta alternativa con TLB proporciona buenos resultados para aplicaciones en las que hay localidad en las páginas utilizadas para envío y recepción de datos [MOS02, BUC01].

Dado un paquete, la interfaz obtiene la dirección en la red del computador destino, e incluso, en algunas implementaciones, encuentra el camino a utilizar en la transferencia. En el primer caso, el camino se va determinado mediante los conmutadores de red por los que va pasando el paquete. En el segundo caso, el camino completo se incorpora en la cabecera del paquete. También pueden soportar funcionas colectivas de comunicación. Si, por ejemplo, se da soporte para la difusión, se evitaría tener que implementarla a niveles superiores, ahorrándose que la NIC tenga que transferir múltiples veces los datos que se difunden.

El fallo de algún componente de la red (enlaces, conmutadores), puede impedir la transferencia de datos entre algunas entradas y salidas del sistema de comunicación. Hay interfaces que reconfiguran el sistema de comunicación tras un fallo para que pueda volver a conectarse cualquier entrada con cualquier salida. Esta propiedad está

relacionada con la disponibilidad. En sistema de comunicación con un alto grado de disponibilidad se debería garantizar que los paquetes lleguen al destino aunque falle dinámicamente algún componente, esto es, durante la transferencia [PET03].

Dentro de esta línea de los interfaces a nivel de usuario se desarrolló un estándar denominado VIA [VID10]. El objetivo de VIA es reducir la sobrecarga de comunicación entre la CPU, la memoria y la tarjeta de red para mejorar las prestaciones en redes de altas prestaciones. Con dichas reducciones en la sobrecarga del sistema de memoria y de la CPU, VIA consigue mejorar la latencia y el ancho de banda entre los nodos de un cluster de computadores [VID10]. 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, y la intervención del sistema operativo. De esta forma, se evitan interrupciones y cambios de de contexto, disminuyendo la sobrecarga en la CPU principal del nodo. Además, VIA proporciona un mecanismo mejorado de comunicación entre procesos en clusters de computadores para disminuir la latencia.

El aprovechamiento de los anchas de banda de gigabit por segundo han dado lugar a algunas técnicas que se han aprovechado para mejorar el rendimiento de las interfaces de red. Se describe a continuación.

1.4.3 Fusión de interrupciones

Mediante esta estrategia la NIC espera que haya un cierto número de solicitudes de interrupción generadas por los paquetes o un intervalo de tiempo establecido antes de enviar una petición de interrupción al procesador del nodo. Esto permite que el sistema reduzca el coste del procesamiento de las interrupciones distribuyéndolo entre varios mensajes. Sobre todo, con este método se reduce la sobrecarga cuando el tamaño de paquete de red es pequeño como por ejemplo en redes Ethernet. Además de reducir la sobrecarga asociada a las interrupciones, la otra ventaja de esta estrategia consiste en que está disponible en las NIC actuales y es relativamente sencilla de utilizar [GIL05].

1.4.4 Copia-0 de memoria

Las técnicas de *copia-0* creadas para evitar las copias en memoria, se usan comúnmente para disminuir la sobrecarga por byte. Se han medido reducciones de la sobrecarga por byte de hasta el 70% con técnicas de c*opia-0*.

La c*opia-0* puede realizarse a varios núcleos. En el primero, se elimina la copia de datos de red en el nodo. En el segundo se eliminan todas las copias de datos de red, incluso la copia final de la memoria desde el k*ernel* a la memoria de usuario. [SKE01].

1.5 La *externalización* de protocolos de comunicación

Hay varias técnicas de externalización de protocolos de comunicación que se usan como implementaciones comerciales para las mismas. La primera alternativa de la externalización es el *offloading*, cuando se implementa todo el procesamiento de los protocolos de comunicación en la tarjeta de red NIC. Por lo tanto, podría tener varios niveles de *externalicación* usando el modelo teórico LAWS. La segunda alternativa para externalizar las funciones de comunicación es el *onloading*, cuando se hace uso de los núcleos del nodo (CMP [WUN06]) o de los procesadores (SMP [IOA06]).

La idea principal de la externalización mediante o*ffloding* [WES04, MOG03] consiste en distribuir las tareas de comunicación entre los elementos diferentes del nodo, en particular entre la CPU del nodo y los procesadores en la NIC. Las tareas que implican interacciones con la red se pueden ubicar en el NIC, liberando ciclos de CPU para otras tareas de propias de la aplicación.

Cuando la CPU tiene que enviar o recibir datos por la red, la memoria principal puede escribir o leer los datos a/desde la NIC. La técnica de o*ffloading* permite la paralelización de las tareas de comunicación y de acceso a los datos en la memoria principal evitando costes de comunicación asociados a la transferencia a través del camino de comunicación completo.

En el caso de la *externalización* mediante *onloading* [REG04a] se asigna el procesamiento de los paquetes a uno de los núcleos de procesamiento en la CPU del nodo o una de las CPUs en un SMP. De esta manera, un procesador de alto rendimiento se encarga de procesar los paquetes. Además, el *onloading* se aprovecha del *chipset* y la

tarjeta de red para manejar las transferencias con la memoria principal del nodo. El movimiento de los datos entre la tarjeta de red y la memoria se gestiona con un DMA dedicado a cada transferencia, de modo que el procesador no tenga que gastar ciclos para hacer la transferencia. Por lo tanto, la realización del protocolo *onloading* reduce la sobrecarga asociada al software. Una realización de la técnica del *onloading* es la I/OAT (I/O Acceleration Technology) de Intel [IC05].

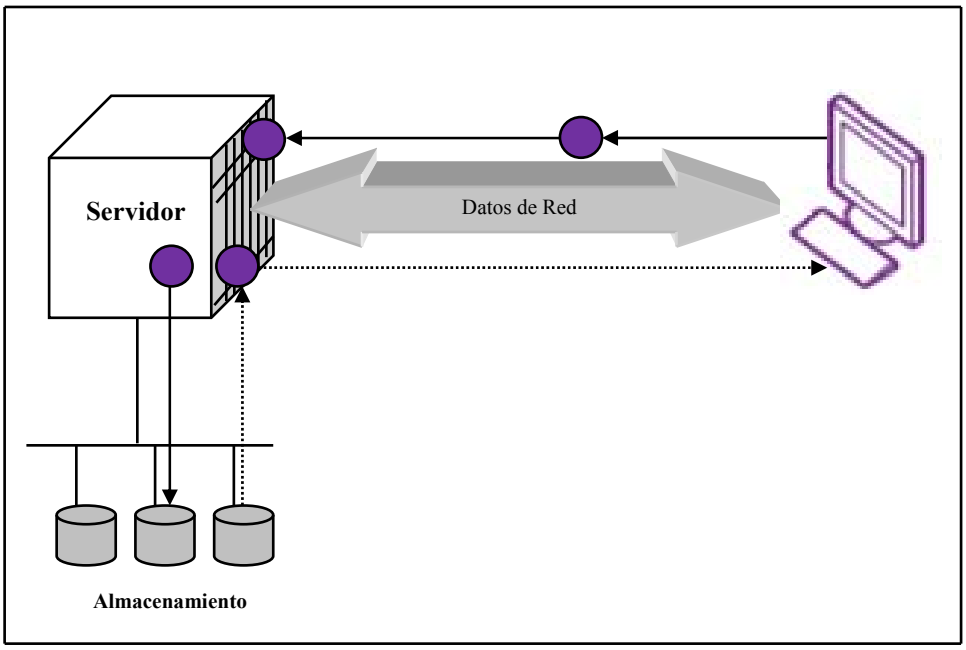

Figura 1.9 Caminos de datos hacia y desde la aplicación

La Figura 1.9 ilustra la posible ubicación de los cuellos de botella del camino de comunicación completo. A través de la técnica I/OAT de Intel se pretende reducir los cuellos de botella de los buses de E/S disminuyendo el *overhead* del sistema, mejorando el acceso a memoria y la implementación de los protocolos TCP/IP. Estas técnicas reducen la utilización de la CPU para tareas de E/S, para el procesamiento de los protocolos [IOA06].

1.5.1 NIC para redes multigigabits por segundo

Las redes de anchos de banda de varios gigabits por segundo necesitan elementos hardwares adecuados. La Figura 1.10 proporciona una descripción simplificada de los

elementos de una tarjeta de red para Ethernet de 10Gbits/s. La arquitectura de la NIC permite el procesamiento paralelo de los datos en los núcleos de procesamiento (*cores*), incluye una jerarquía de memoria relativamente compleja, y el hardware para realizar transferencias de DMA a través del nodo, y los buses para enviar y recibir los datos de *Ethernet* según la MAC de la tarjeta de red. Además, las unidades implicadas con el control de los datos tienen acceso a la memoria y comparten la información con los procesadores sobre los contenidos de las tramas transferidas [WIL05].

Cada núcleo de procesamiento es un cauce segmentado con 5 etapas capaz de finalizar una instrucción por ciclo, y un repertorio de instrucciones que es un subconjunto del MIPS R4000. Para permitir que los núcleos de procesamiento procesen los datos sin interrumpir al procesador central, éstos pueden almacenarse en un buffer. Las instrucciones se almacenan en una memoria de instrucciones de 128KB. El software (*firmware*) para controlar los datos almacenados en la memoria (*scratchpad*) incluida en el chip, tiene una capacidad de 256KB.

La memoria *scratchpad* es globalmente visible por todos los procesadores y el resto de hardware correspondiente a las distintas unidades de la NIC. La memoria de s*cratchpad* también admite datos de baja latencia que comparten los procesadores. Los núcleos de procesamiento y los demás elementos hardware se conectan a la memoria de *scratchpad* mediante una red de barras cruzadas (c*rossbar*). Hay también una red de barras cruzadas para permitir que los núcleos de procesamiento se conecten a la interfaz de memoria externa.

El acceso a la memoria de *scratchpad* tiene una latencia de 2 ciclos: un ciclo para solicitar y cruzar la red y otro ciclo para acceder a la memoria y devolver los datos solicitados. Por la tanto, los procesadores siempre deben parar al menos un ciclo para hacer las cargas. Sí cada núcleo de procesamiento tuviera su propia memoria de *scratchpad*, la latencia de acceso de memoria se podría reducir a un solo ciclo eliminando la red de barras cruzadas. Sin embargo, si cada núcleo de procesamiento necesita acceder a una memoria de *scratchpad* remota (de otro núcleo) necesitaría una latencia mayor.

Los núcleos de procesamiento y las memorias de *scratchpad* de la NIC funcionan a la frecuencia de reloj de la CPU. A esa frecuencia, si los núcleos de procesamiento funcionan al 100%, 4 núcleos y 2 memorias de *scratchpad* podrían reducir el cálculo y disponer de ancho de banda para el acceso a los datos. El interfaz PCI y la unidad MAC comparten un bus de 128 bits para acceder a la SDRAM DDR

externa de 64 bits de ancho. Para la misma frecuencia, tanto el bus como la SDRAM DDR tienen el mismo coste por transferencia. La SDRAM puede transferir dos valores de 64 bits por ciclo. Sin embargo, el tráfico de transmisión introduce una ineficiencia significativa porque requiere dos transferencias por trama (*cabeceras y datas*), donde la cabecera es sólo de 42 bytes [WIL05].

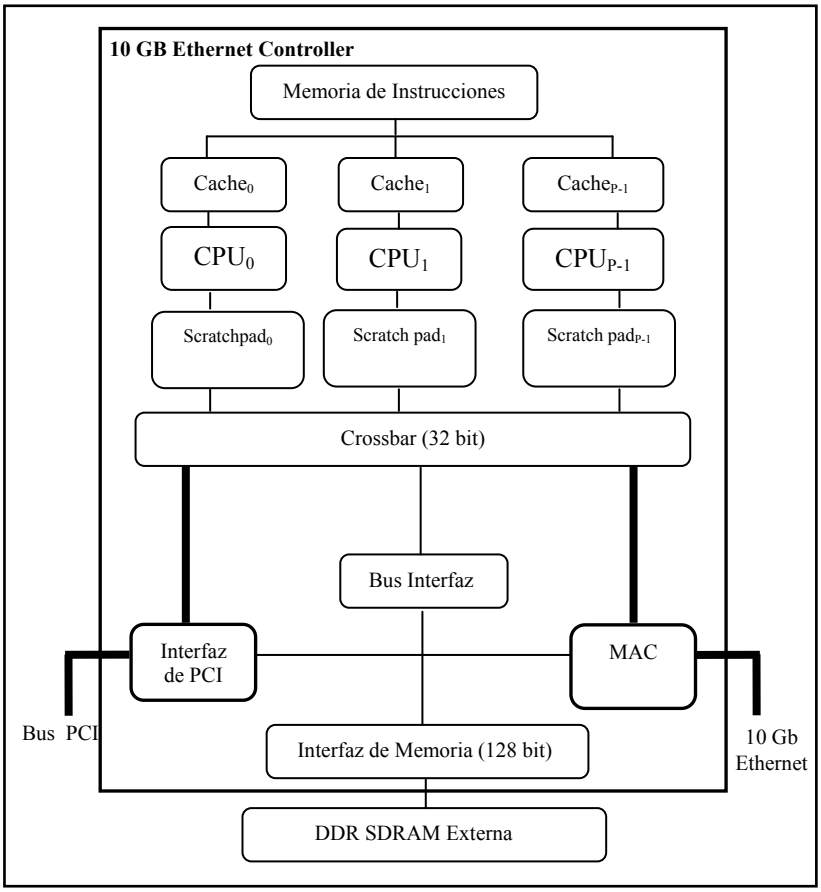

Figura 1.10 Arquitectura de controlador de Ethernet de 10Gb/s

Cada procesador de la familia IXP2XXX de procesadores de red implementa procesamiento multihebra y dispone de un núcleo *XScale* y ocho o dieciséis núcleos denominados *microengines* según la versión IXP2400 o IXP2800, respectivamente. El número relativamente elevado de *microengines* permite aprovechar el paralelismo en el procesamiento eficiente de paquete. El Intel *XScale* puede programarse con lenguajes de programación de alto nivel y ejecutar sistemas operativos estándares. Los *microengines* son procesadores RISC, y cada uno puede ejecutar 8 hebras concurrentemente. El tiempo de interrupción entre las hebras de un *microengine* es mínimo porque el hardware incorpora recurso para cada hebra [PAT00, JOH03].

Los procesadores IXP2XXX disponen de una memoria SRAM externa, memoria DRAM, y bus PCI externo. El núcleo de procesamiento y los *microengine* pueden tener acceso a la memoria externa mediante los controladores de DRAM y SRAM respectivamente. La memoria SRAM es más pequeña que la DRAM y tiene una latencia inferior que la memoria DRAM. Las unidades DRAM realizan las transferencias de datos del RBUF (Buffer de Recepción) y al TBUF (Buffer de Transmisión). El interfaz PCI permite que el núcleo de procesamiento y los *microengines* inicien transferencias de DMA a través del bus PCI [IXP05, GRO05, PAP04].

1.5.2 La comparación entre la pila de TCP/IP y los TOE

Actualmente, la transmisión de la información entre redes usa la familia de protocolos TCP/IP para transmitir datos sobre redes de área local (LAN), redes de área amplia (WAN), e Internet. Como se ha comentado antes y es bien sabido, los protocolos de TCP/IP están organizados en capas. Las capas más relevantes para los TOE (*TCP/IP Offload Engines*) son las capas IP y TCP, tal y como se muestra en la Figura 1.11 [FOO03].

La capa IP se usa para dos objetivos. El primero para transmitir paquetes entre redes por el proceso de encaminamiento; y el segundo para mantener un interfaz a redes diferentes. El IP es un protocolo sin conexión, y eso significa que cada paquete se trata como una entidad separada. En realidad, cada paquete de red conjunto de un corriente de datos, y cada conjunto de datos se comunica a una aplicación. La capa TCP asocia cada paquete de red con sus datos y las capas superiores asocian cada conjunto de datos con su aplicación [SEN04].

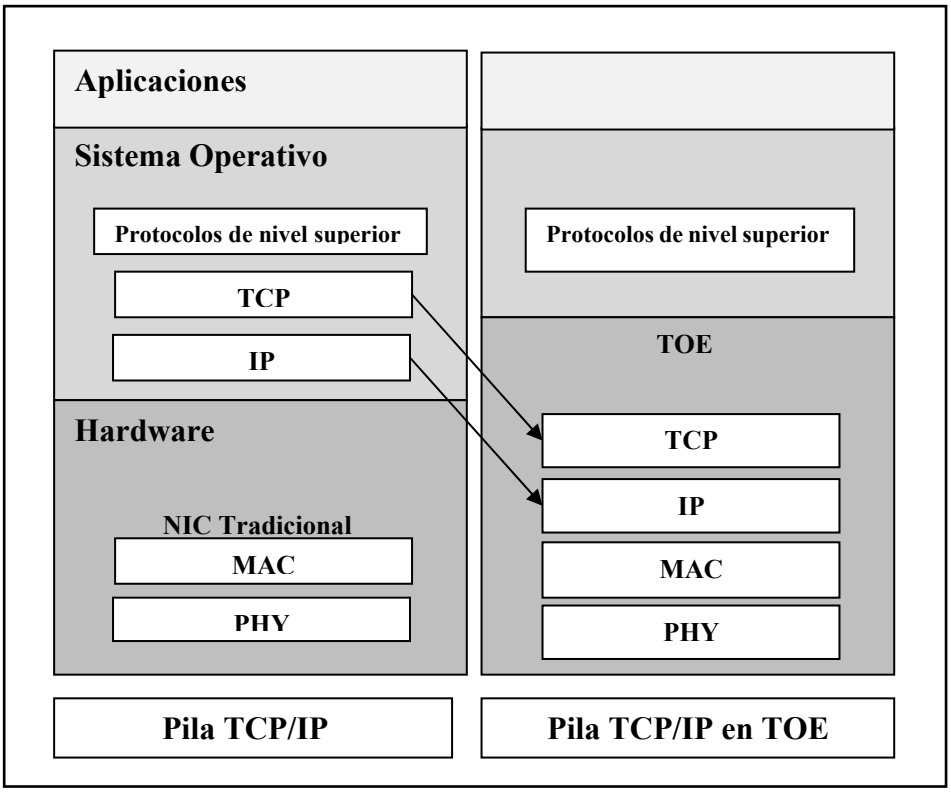

Figura 1.11 La comparación de TCP/IP estándar y TOE-permitidas de la pila de TCP/IP

1.5.2.1 La sobrecarga (*overhead*) TCP y TOE

Los TOE pueden encargarse de todo el procesamiento relacionado con el envío y la recepción de paquetes a la NIC, con lo que la CPU del servidor tendría más ciclos disponible para las aplicaciones. Como los TOE implican al procesador del nodo una vez para cada E/S de la red, los TOE reducen el número de peticiones que la pila debe tratar. En una implementación TCP/IP tradicional, el procesador del nodo procesa los paquetes recibidos. Esto significa que cada paquete recibido pasa por copias múltiples, desde los *buffers* del NIC a la memoria principal. Como un adaptador de red TOE podría realizar todo el procesamiento del protocolo de comunicación. Además, el adaptador puede usar algoritmos de copia cero para transmitir datos directamente de los *bufferes* de la NIC hasta la memoria principal, sin copias intermedias a los *bufferes* de sistema. En cualquier caso, los TOE reducen las tres causas principales de sobrecarga: la sobrecarga de procesamiento de los protocolos TCP/IP, las copias de memoria, y el coste asociado a las interrupciones.

1.5.2.2 Procesamiento de interrupciones

Las interrupciones dan lugar a cambios de contexto que resultan muy costosos. Aunque las técnicas para reducir el efecto de las interrupciones pueden ayudar a reducir la sobrecarga, dichas técnicas no pueden reducir al mismo tiempo el procesamiento de los eventos necesarios para enviar los paquetes. Además, cada transferencia de datos genera una serie de copias de datos desde los *bufferes* de datos de la aplicación a los *bufferes* de sistema, y desde los *bufferes* de sistema a la NIC. Las redes rápidas como *Gigabit Ethernet* obligan a la CPU a procesar un número elevado de paquetes. Para paquetes de 1500 bytes, la pila de SO del host tendría que procesar más de 83.000 paquetes por segundo, o, lo que es lo mismo, un paquete cada 12 microsegundos. El uso TOE puede permitir una reducción dramática de la carga asociada a las transacciones de red. Mediante los TOE, la CPU puede tratar una transacción completa de E/S de la aplicación con una única interrupción. Además, las aplicaciones que trabajan con tamaños de datos que son múltiplos de los tamaños de paquete de red se beneficiarían más de los TOE [SEN04].

1.5.2.3 Copias de memoria

Las NIC estándar necesitan que los datos se copien desde el espacio de usuario de aplicación al núcleo del SO. Entonces, el *driver* de la NIC copiaría los datos de paquetes desde el núcleo de SO al *buffer* de la NIC. Esta técnica se muestra en la Figura 1.12. Cuando los paquetes se reciben desde la red, la NIC copia los paquetes en sus *buffers*, y se pasan a la memoria del host. Finalmente, los paquetes copiados al *buffer* de TCP se mueven al espacio de memoria de usuario que utiliza la aplicación.

En este caso, el total de las copias de memoria sería tres. El TOE puede reducir el número de copias a dos, porque el NIC puede copiar los paquetes al buffer de TCP y luego a los *bufferes* de aplicación directamente.

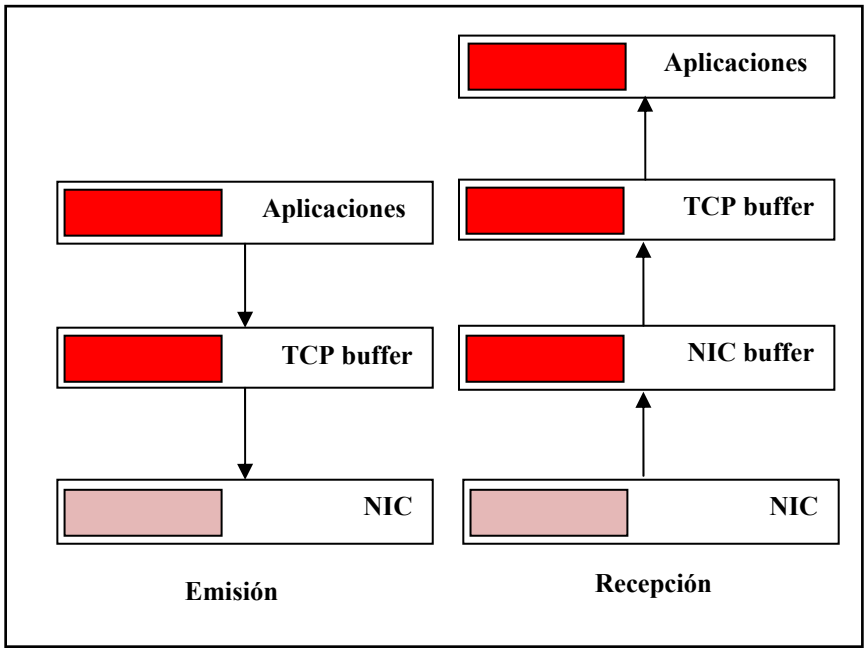

Figura 1.12 Implementar TCP/IP al usar un TOE.

El TOE pueden utilizar RDMA (*Remote DMA*) [ROM05] para usar los algoritmos de copia cero y colocar los datos directamente en los *bufferes* de la aplicación. La definición de RDMA para TCP/IP permite que los protocolos pueden utilizar operaciones de RDMA a través de redes de TCP/IP estándares [SEN04].

1.5.2.4 Procesamiento del protocolo de comunicación

El procesamiento de los protocolos de comunicación consume muchos ciclos del CPU del host en la recepción de paquetes. La NIC estándar debe almacenar los paquetes recibidos en un *buffer*, y luego notificar al procesador del host, mediante una interrupción, que ya han llegado los paquetes. Después de recibir la interrupción, se procesa el protocolo de comunicación para que se pueda tener acceso a la información del paquete. Los datos de paquetes TCP deben asignarse a la aplicación adecuada y, al mismo tiempo, los datos deben copiarse desde el *buffer* del sistema a la memoria de aplicación.

El protocolo TCP/IP usa la información de c*hecksum* de cada paquete, para determinar si el paquete tiene o no tiene errores. Cada una de estas operaciones da lugar

a una interrupción al SO. Por consiguiente, la CPU del host puede saturarse si se producen interrupciones con mucha frecuencia. Existe una regla aproximada que indica que, para poder procesar 1Gbps de datos, la CPU tiene que funcionar a 1GHz. Esto implica, que pueden aparecer cuellos de botella para *Gigabit Ethernet* y sobre todo para *10Gbps Ethernet*. Para estas velocidades de red, quedarían pocos ciclos disponibles para el procesamiento de aplicación ya que la CPU se dedicaría fundamentalmente a procesar paquetes [HYD00, SEN04].

1.6 Modelos de evaluación de prestaciones

Como se ha indicado, para aumentar las prestaciones del camino de comunicación, la interfaz de red debe reducir los costes de comunicación entre la red y la aplicación, al tiempo que se reduce el número de ciclos necesarios para entregar a la aplicación. A continuación se explica los modelos de evaluación en detalle porque se toman como referencia para analizar las prestaciones del camino de comunicación. Desde la controversia surgida acerca de la utilidad de la *externalización*, estos modelos son muy útiles para ayudar a comprender las características en las que puede o no resultas útil alguna de estas propuestas.

1.6.1 Evaluación de las prestaciones de comunicación mediante el modelo LAWS

El modelo LAWS pretende caracterizar la mejora que proporciona la *externalización* de la comunicación en el NIC en aplicaciones relacionadas con los servicios de Internet u otras aplicaciones de transmisión de datos (*streaming data applications*). El modelo LAWS proporciona una estimación de la mejora máxima de ancho de banda (*peak throughput*) partiendo de un modelo segmentado del camino de comunicación en el que el cuello de botella (el enlace, el NIC o el procesador central) determina el ancho de banda del sistema antes y después de la *externalización*. Las aplicaciones que considera el modelo son aquellas cuyas prestaciones están limitadas por el ancho de banda de comunicación, tales como los servidores de Internet. Además, en la presentación del modelo LAWS, se supone que los requisitos de comunicación saturan a la CPU del

nodo antes de que se aplique la *externalización* de las tareas de comunicación (en caso contrario, la *externalización* no aportaría ninguna mejora como corroboraciones en nuestro trabajo experimental).

En la descripción del modelo LAWS que sigue utilizaremos la notación que se emplea en [SHI03]. Así, En el modelo LAWS, las prestaciones de un sistema se caracterizan en términos de cuatro parámetros: *L* (*Lag*), *A* (*Application*), *W* (*Wire*) y *S* (*Structura*l) cuyo significado indica a continuación.

1. El coeficiente de retardo (***Lag,*** representado por $\boldsymbol{\alpha}$). Relación entre la velocidad del nodo y la velocidad del procesador de la NIC.
2. El coeficiente de aplicación (***Application,*** representado por $\boldsymbol{\gamma}$). Relación entre el tiempo de procesamiento de la aplicación y el de procesamiento de comunicación.
3. El coeficiente de conexión ***(Wire,*** representado por $\boldsymbol{\sigma}$). Relación entre el ancho de banda de saturación del nodo y el ancho de banda de red. Es decir, la parte de la ancho de banda de red y el nodo puede aprovechar sin *offloading*.
4. El coeficiente estructural (***Structural,*** representado por $\boldsymbol{\beta}$). Factor de reducción del coste de la comunicación debido del uso de *offloading*.

Antes de la *externalización*, el sistema se contempla como un cauce de dos etapas, el computador y la red. En el computador, para transferir *m* bits, la aplicación da lugar a una carga de trabajo en la CPU igual a *aXm*, y a una carga asociada a las tareas de comunicación de *oXm*. En estos tiempos de procesamiento, *a* y *o* corresponden al tiempo de trabajo de la CPU por unidad de datos, y *X* es un factor de escala utilizado para tener en cuenta las diferencias en capacidad de procesamiento con respecto a un procesador de referencia dado. Por otra parte, el tiempo que la red tardaría en proporcionar esos *m* bits a través de un enlace con un ancho de banda igual a *B*, sería *m/B*. El ancho de banda máximo antes de la *externalización* está determinado por el cuello de botella (la red o la CPU) y vendrá dado por $B_{before}=min(B,1/(aX+oX))$. Después de la *externalización*, tendríamos un cauce segmentado de tres etapas (la CPU, la NIC, y el enlace) y una proporción *p* del *overhead* de comunicación se habrá transferido a la NIC. De esta forma, los tiempos para transferir *m* bits en las distintas

etapas son *m/B*, para el enlace de red, *aXm+(1-p)oXm* para la CPU, y *poYβm* para la NIC.

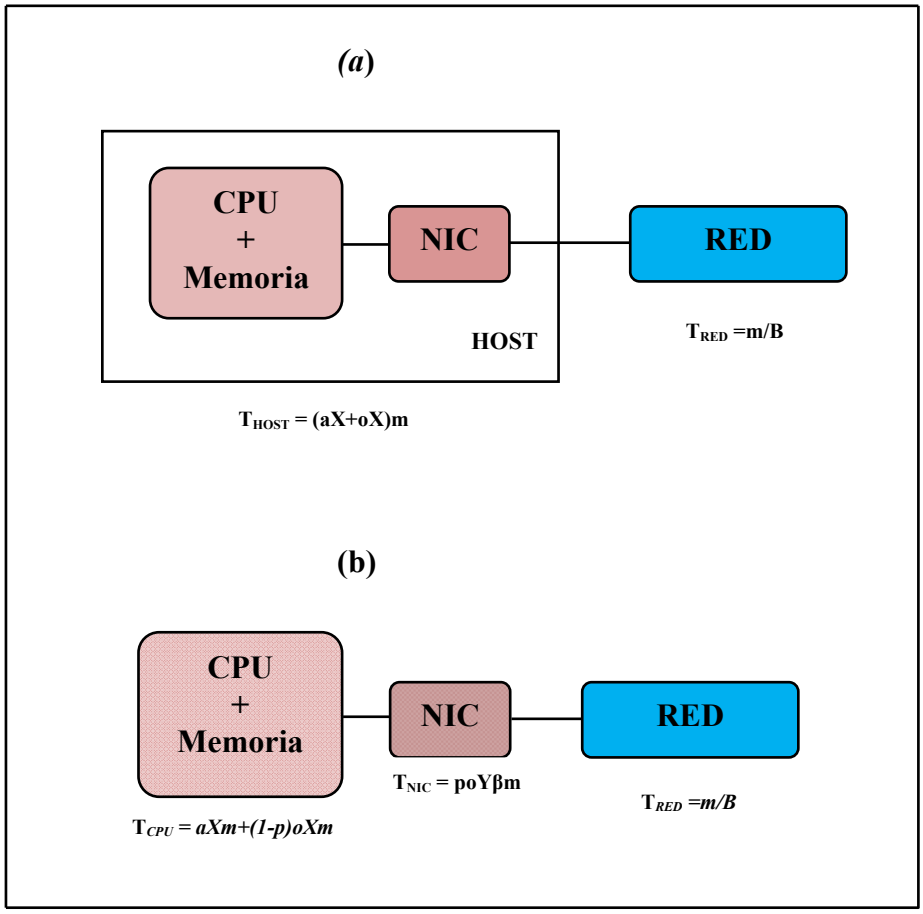

Figura 1.13 Modelo LAWS (a) antes del o*ffloading*, (b) después del o*ffloading*

En la expresión del retardo en la NIC, *Y* es un factor de escala que permite incorporar las diferencias en capacidad de procesamiento entre la CPU y la NIC, y *β* es un parámetro que cuantifica la reducción que se produce en la sobrecarga de comunicación al utilizar la *externalización*, como se muestran en la Figura 1.13. Así, *βo* es la sobrecarga de comunicación normalizada que sigue quedando en el sistema después de realizar una *externalización* con *p=1* (*externalización* completa) [SHI03].

En consecuencia, después de la *externalización*, el ancho de banda máximo vendrá dado por $B_{after}=min(B,1/(aX+(1-p)oX),1/poY\beta)$ y la mejora relativa en el ancho de banda máximo será $\delta b=(B_{after}-B_{before})/B_{before}$. Como se ha dicho, el acrónimo LAWS proviene de los parámetros que se utilizan en el modelo para caracterizar los beneficios de la *externalización*. Así, junto al parámetro *β*, están los parámetros *α=Y/X*, que da cuenta de la relación entre la velocidad de la CPU y el procesador en la NIC; *γ=a/o*, que

indica la relación entre las cargas de cómputo y comunicación que están asociadas a una aplicación determinada; y *σ=1/oXB*, que indica la parte de ancho de banda de red que el sistema proporciona antes de que se produzca la *externalización*.

Sí expresamos en función de estos parámetros α, β, γ, y σ, la mejora relativa del ancho de banda máximo tenemos:

$$BW_{before} = min\ (B,\ 1/\ (aX + oX)) = min\ [(BoX,\ (1/\gamma + 1)) * (1/oX)] \quad (1.5)$$

$$BW_{after} = min\ (B,\ 1/\ (aX + (1\text{-}p)\ oX),\ 1/\ poY\beta) = min\ [((BoX,\ 1/\ (\gamma + (1\text{-}p)),\ 1/\ p\beta\ \alpha)) * (1/oX)]) \quad (1.6)$$

$$\delta b = \frac{BW_{after} - BW_{before}}{BW_{before}} = \frac{\min(\frac{1}{\sigma}, \frac{1}{\gamma + (1-p)}, \frac{1}{p\alpha\beta}) - \min(\frac{1}{\sigma}, \frac{1}{1+\gamma})}{\min(\frac{1}{\sigma}, \frac{1}{1+\gamma})} \quad (1.7)$$

A partir de la expresión (1.7) obtenida del modelo LAWS se pueden extraer algunas conclusiones (Figura 1.14):

- La *externalización* de protocolos proporciona una mejora que crece linealmente para valores bajos de la relación entre coste de computación y coste de comunicación (valores bajos de *γ*). En el caso de aplicaciones intensivas en cómputo (es decir, a medida que crece *γ*), la mejora relativa en ancho de banda está limitada por *1/γ* y tiende a cero a medida que aumenta el coste de computaciónanda (es decir, *γ* crece). La mejora máxima que puede alcanzarse se consigue para γ=max(*αβ,σ*). Además, dado que la pendiente de la función de mejora, (γ+1)/c-1, es 1/c siendo c=max(*αβ,σ*), la mejora del ancho de banda crece más a medida que *αβ* y *σ* disminuyen.

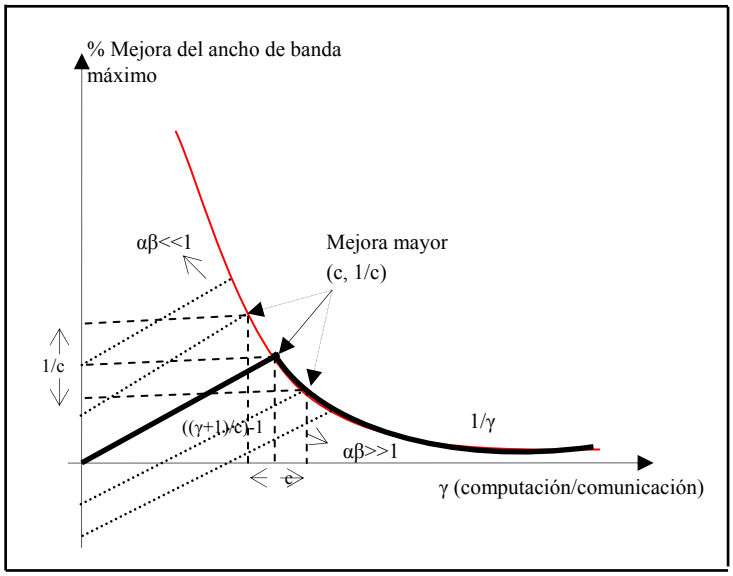

Figura 1.14 Porcentaje de mejora de ancho de banda máximo según el modelo LAWS

- La *externalización* de protocolos puede reducir el ancho de banda máximo alcanzable (es decir, se tendría $\delta b<0$) si la función (γ+1)/c-1 pudiese tomar valores negativos. Esto significaría que γ<(c-1) y, puesto que γ>0 y σ<1, debería verificarse que c=$\alpha\beta$ y $\alpha\beta$>1. Así, si la velocidad del procesador de la NIC es menor que la de la CPU, la *externalización* puede empeorar las prestaciones si la NIC se satura antes que el enlace de red ($\alpha\beta>\sigma$) dado que la mejora estaría limitada por $1/\alpha$ (suponiendo β=1). Sin embargo, si mediante una implementación eficiente de la *externalización* se producen mejoras estructurales suficientes (que dieran lugar a la reducción necesaria de β) sería posible sacar beneficios de la técnica de *externalización* aunque se tuvieran valores α mayores que 1

- En el caso de redes lentas ($\sigma>>1$) no se puede esperar beneficio alguno de la *externalización* dado que el computador es capaz de asumir la sobrecarga de comunicación sin ninguna ayuda. La *externalización* es útil siempre que el computador no sea capaz de comunicarse a la velocidad del enlace ($\sigma<<1$). No obstante, en estas circunstancias, γ debe ser bajo, tal y como se ha dicho anteriormente. Dado que existe una clara tendencia hacia redes rápidas (hacia una reducción en σ), la externalización puede resultar bastante útil. Cuando σ está próximo a 1, la mayor mejora corresponde a aquellos casos en los que existe cierto equilibrio entre computación y comunicación antes de la *externalización* ($\gamma=\sigma=1$).

En la Figura 1.15 se muestra que la ventaja marginal máxima de la técnica de *offloading* para una variedad de aplicaciones (correspondiente a distintas valores de γ), y con los valores de los otras tres parámetros (α, β, σ) iguales a 1, el beneficio máximo se obtiene para γ=1, cuando la CPU del *nodo* esta exactamente equilibrada entre el procesamiento de aplicación y el *overhead* de comunicación sin *offloading*.

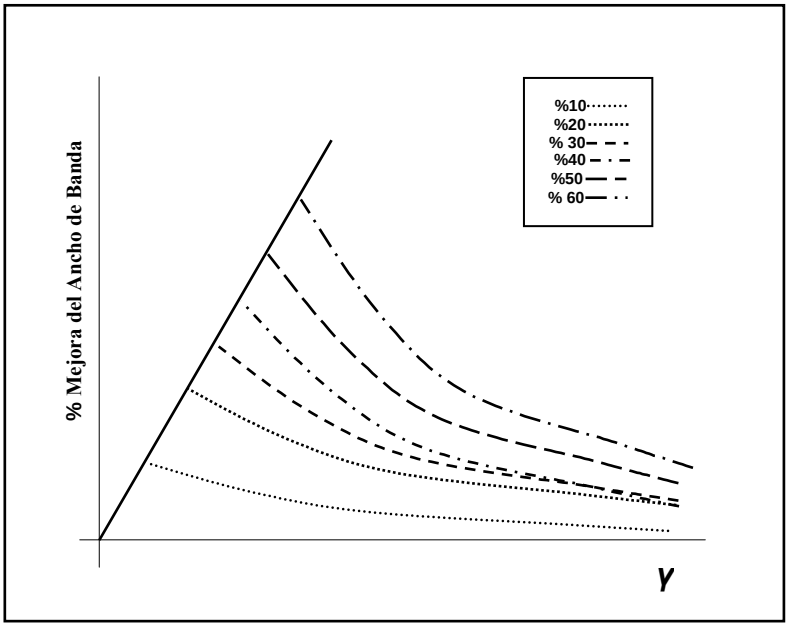

Figura 1.15 Beneficio de *offloading* en función del coeficiente de aplicación (γ) y para varias valores de p, y con α, β, σ iguales a 1.

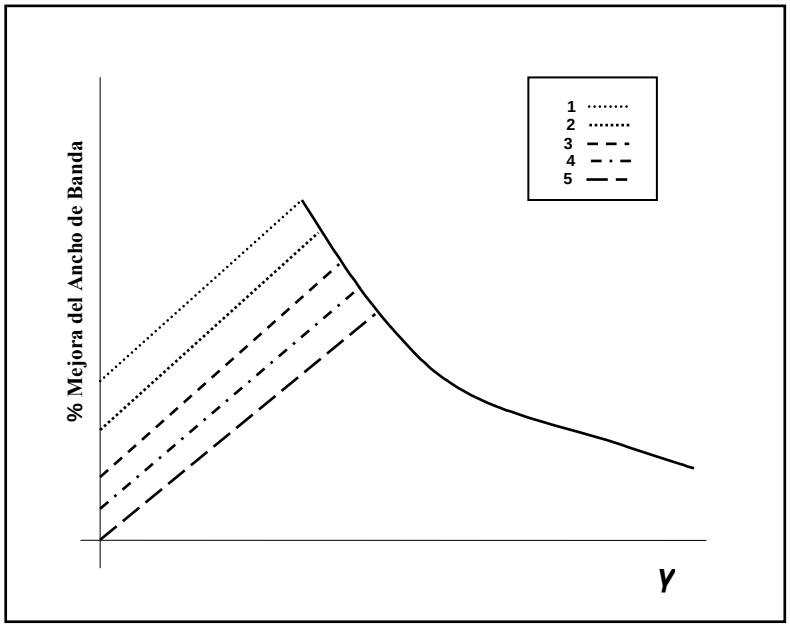

Figura 1.16 Beneficio de la técnica de *offloading* completo en función de coeficiente de aplicaciones y para varios valores de β, con α=2.

La Figura 1.16 muestra el efecto de varios valores de β. El beneficio máximo de *offloading* está limitado por $1/\alpha\beta$, y esto ocurre cuando $\gamma = \alpha\beta$.

1.6.2 Evaluación de las prestaciones de comunicación mediante el modelo EMO

El modelo LAWS nos ayuda a evaluar diferentes estrategias para el o*ffloading* aunque es un modelo que puede mejorarse para analizar comunicación con mensajes donde haya que evaluar fundamentalmente la latencia. Un modelo que se ha propuesto para evaluar la comunicación en computación de alto rendimiento es el modelo EMO [BUC01]. La Figura 1.17 muestra la arquitectura de comunicación en la que se basa el modelo EMO.

Las variables de este modelo según la terminología utilizada en [BUC01] son las siguientes:

C_n= Número de ciclos de procesamiento de protocolo en la NIC.

R_n= Velocidad del procesador de la NIC.

L_{nh}=Tiempo para mover datos y señales de control entre la NIC y el SO del host.

C_h= Número de ciclos de procesamiento del protocolo en la CPU del host.

R_h= Velocidad de la CPU del Host.

L_{ha}= Tiempo para mover datos y control entre el host y la aplicación.

L_{na}= Tiempo para mover datos y control entre la NIC y la aplicación.

C_a= Número de ciclos de procesamiento del protocolo por parte de la aplicación.

O_{nh}= Número de ciclos del procesador del host para mover datos y control entre el SO y la NIC.

O_{ha}= Número de ciclos del host para mover datos y control desde el SO a la aplicación.

O_{na}= Número de ciclos del host necesarios para comunicar y mover datos desde la NIC a la aplicación.

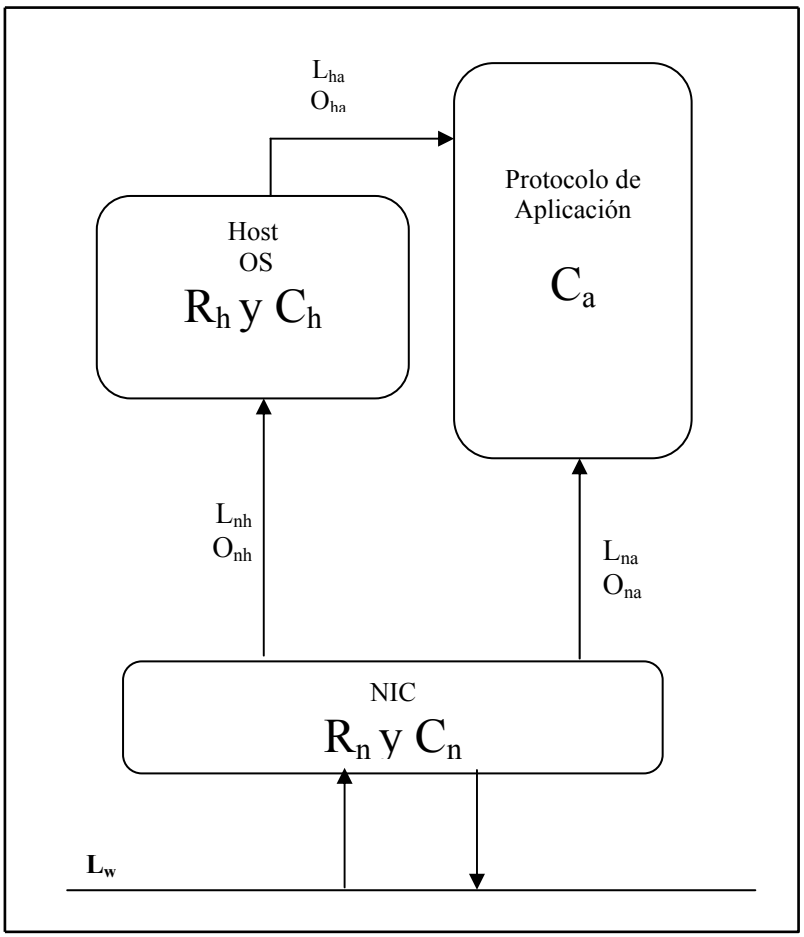

Figura 1.17 Modelo EMO para el o*ffloading*

Modelado de la sobrecarga de comunicación en EMO. El modelo EMO permite analizar el coste de la implementación de un protocolo. La sobrecarga (*overhead*) se puede evaluar por mensaje y por byte utilizando la siguiente expresión:

$$Overhead\ offloading = O_{nh} + C_h + O_{ha} + C_a + O_{na} \qquad (1.8)$$

La sobrecarga (o*verhead*) cucando no se emplea el *offloading* tradicional, no usará el camino de comunicación entre la NIC y la aplicación, y no implicará ningún procesamiento en la aplicación. Por lo tanto quedaría:

$$Overhead = O_{nh} + C_h + O_{ha} \qquad (1.9)$$

Modelado de la latencia en EMO. La latencia se puede evaluar mediante la expresión:

$$Latencia\ offloading = \frac{C_n}{R_n} + L_{nh} + \frac{C_h}{R_h} + L_{ha} + L_{na} + \frac{C_a}{R_h} + L_w \qquad (1.10)$$

Los protocolos de red tradicional no usan el camino de comunicación entre la NIC y la aplicación y no implica ningún procesamiento en la aplicación. Por lo tanto se tendría que:

$$Latencia = \frac{C_n}{R_n} + L_{nh} + \frac{C_h}{R_h} + L_{\_ha} \qquad (1.11)$$

En [PAT02] se describe la técnica de división (*Splintering TCP*) y se utiliza el modelo EMO para evaluarlo. Se describe aquí como ejemplo de uso del modelo EMO. La filosofía de *splintering* no es la de evitar el uso del sistema operativo en la comunicación, sino la da que debería usarse con eficacia. En el caso de usar o*ffloading*, habría que minimizar la sobrecarga asociada con las llamadas al SO, mientras se permite que el SO mantenga el control de la comunicación. La Figura 1.18 ilustra el modelo de *splintering* para el procesamiento del protocolo TCP en la recepción de los datos [PAT02].

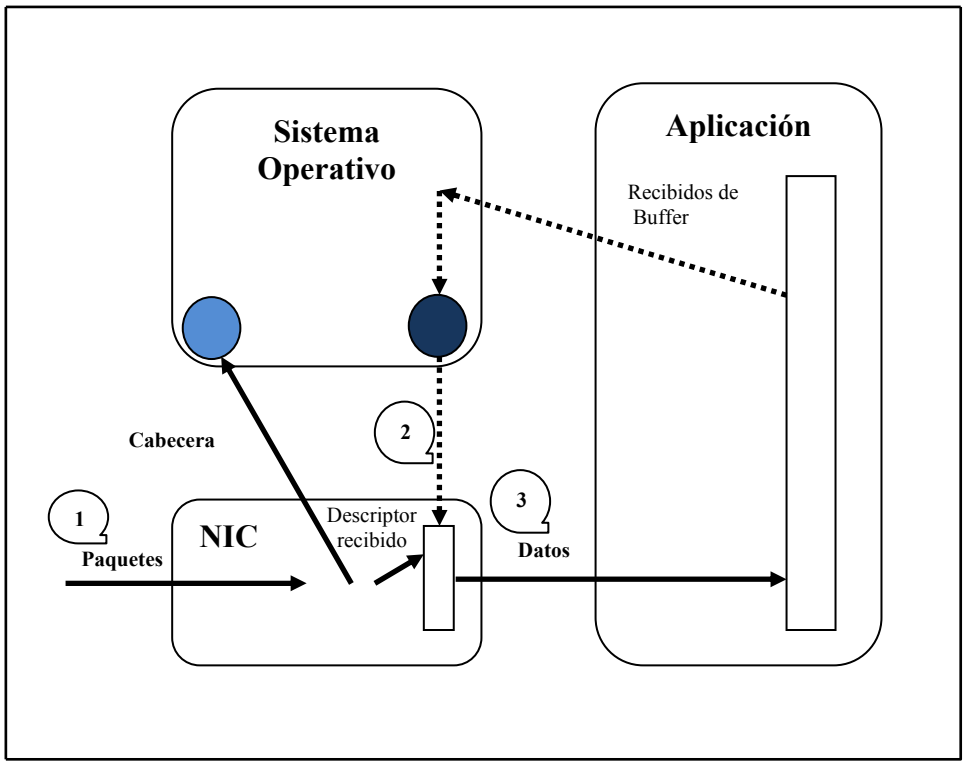

Figura 1.18 Esquema de operación de la técnica de *Splintering* de TCP

Cuando llega un paquete llega, la NIC primero comprueba si el paquete corresponde a un descriptor utilizado antes por el sistema operativo. Sí se encuentra este descriptor, la NIC va a transferir la parte de datos del paquete directamente al buffer de la aplicación, proporcionando un *copia-0* real, y deja la cabecera disponible para el sistema operativo. Si la NIC no encuentra el descriptor, simplemente pone los datos al disposición del SO para que éste se encargue del procesamiento normal. Además, el *splintering* permite que el sistema operativo mantenga el control de los recursos de dirección del procesador del host y la NIC [PAT02].

En cualquier caso, la fusión de interrupción amortiza el coste de la sobrecarga O_{nh} sobre muchos mensajes. La fusión de interrupciones todavía necesita la copia entre el sistema operativo y la aplicación, por lo tanto, la sobrecarga sigue siendo lineal con el tamaño del mensaje. En la técnica de *copia-0*, si no se considera el coste de los mecanismos de protección de memoria, la sobrecarga elimina la dependencia lineal con el tamaño de mensaje en O_{ha}. La sobrecarga O_{ha} se acerca a cero porque la copia de memoria es el factor el más restrictivo en O_{ha}.

El *splintering* TCP usa los mismos métodos de protección de memoria de la c*opia-0* en la pila TCP. La diferencia entre el *splintering* TCP y la c*opia-0* está en los métodos de procesamiento del protocolo de comunicación TCP.

La notificación de eventos entre la NIC y el host se llevan a cabo entre la NIC y la aplicación en lugar de que entre la NIC y el sistema operativo. Sin embargo, la sobrecarga es la misma (O_{nh}=O_{na}), y por lo tanto, la sobrecarga total de la división (*splintering*) TCP es la misma que con la técnica de c*opia-0*.

La Figura 1.19 muestra el efecto de la sobrecarga de los tres métodos que se han descrito.

La Figura 1.20 representa la composición de la latencia y su relación con alguno método para reducir la latencia de un mensaje cuando se usa el s*plintering* TCP. En el caso de mensajes pequeños, la mejora se consigue al transferirlos mediante el DMA a la cola de eventos de la aplicación.

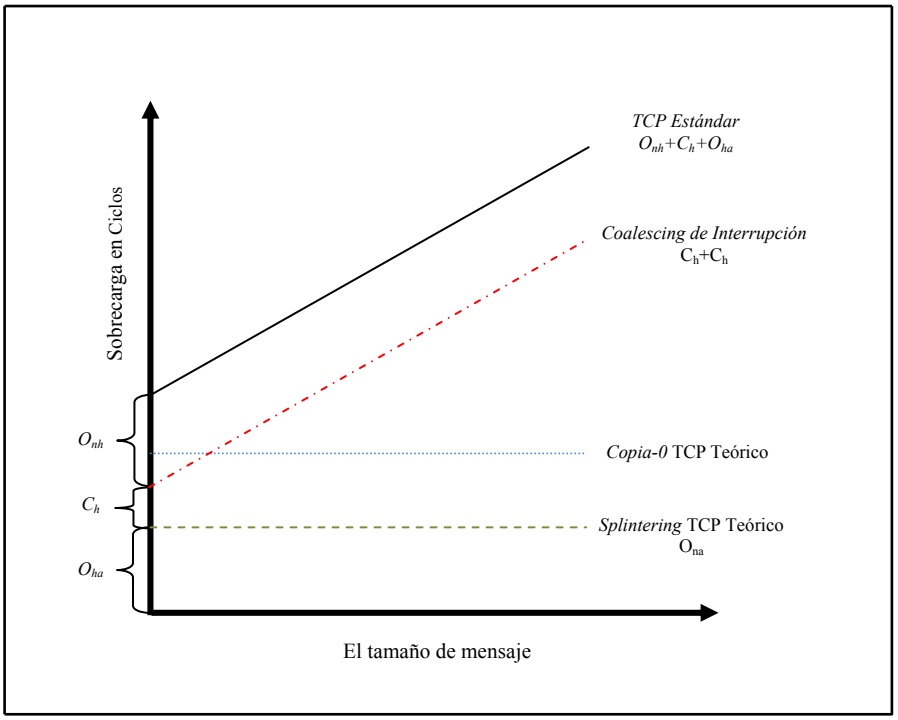

Figura 1.19 Composición de la sobrecarga según el modelo EMO

El modelo EMO proporciona una representación gráfica de la reducción de latencia que se consigue mediante el uso de o*ffloding*. Además, EMO proporciona una forma de comparar las latencias de distintas aproximaciones para el desarrollo de los protocolos de comunicación.

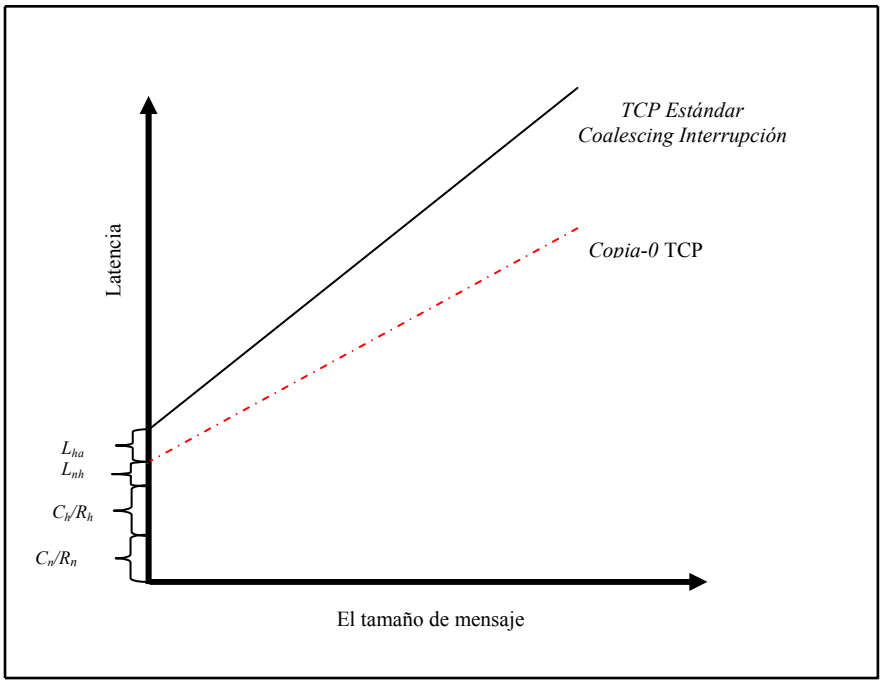

Figura 1.20 Composición de la latencia según el modelo EMO

1.6.3 Relación entre los modelos EMO y LAWS

Se pueden relacionar el modelo EMO y el modelo LAWS para analizar el *offloading*. Como el modelo LAWS considera un número arbitrario de bytes y una cantidad prefijada de tiempo, mientras que el modelo EMO utiliza una cantidad arbitraria de tiempo para una cantidad especificada de bytes, se tendrán que hacer algunas suposiciones. Dado que el modelo LAWS asume una cantidad fija de tiempo, ese tiempo puede ser igual al necesario para recibir un mensaje de longitud N en el *host*. Este tiempo, T, corresponde a un número de ciclos del procesador del host [BUC01]. Sí la sobrecarga (*overhead*) total para el modelo EMO viene dada por:

$$T = \textit{Overhead offloading} = C_n + O_{nh} + C_h + O_{ha} + C_a + O_{na} \qquad (1.12)$$

Dado que tenemos un tiempo fijo, T, un número fijo de bytes, N, y un número total de ciclos del procesador del host ($C_t = R_h * T$), se puede expresar el modelo EMO en términos de los parámetros del modelo LAWS. La parte más difícil de la correspondencia entre el modelo EMO y LAWS viene del hecho de que la sobrecarga (*overhead*) de comunicación, o, es constante mientras que el porcentaje de *offloading*, p, es variable. Por tanto:

$$o = O_{nh} + C_h + O_{ha} + C_a + O_n \qquad (1.13)$$

$$a = C_t - o \qquad (1.14)$$

$$X = R_h \qquad (1.15)$$

$$Y = R_n \qquad (1.16)$$

$$P = C_h - C_{h1} / o \qquad (1.17)$$

$$B = N/T \qquad (1.18)$$

Como se puede ver, a excepción de β, se pueden obtener todos los parámetros del modelo LAWS. No obstante, es posible cuantificar este parámetro directamente del modelo EMO:

$$\beta = \frac{C_{n1} + O_{nh1} + C_{h1} + O_{ha1} + C_{a1} + O_{na1}}{C_n + O_{nh} + C_h + O_{ha} + C_a + O_{na}} \qquad (1.19)$$

De esta manera, ya tenemos todos los elementos que necesitamos para representar el modelo EMO en términos del modelo LAWS.

1.7 Análisis y optimización el sistema de comunicación mediante la simulación

La simulación es la técnica más frecuente para evaluar las propuestas en arquitectura de computadores. El simulador utilizado para este fin, debe ser capaz de modelar el sistema con un nivel de detalle suficiente, y permitir la ejecución de aplicaciones reales, tal como servidores web, bases de datos, etc., aparte de los correspondientes *benchmarks* estándar que pueden ser más o menos complejos. Para esto necesitamos un simulador de sistema completo en el que se pueda reproducir el funcionamiento real del sistema de lo más fielmente posible [ENG03]. En el mercado existen simuladores que permiten, en principio, realizar las simulaciones que necesitamos, como son M5 [M5S07], y SIMICS [VIR08a]. En este trabajo partimos del presupuesto de que un simulador besado en el uso de lenguaje de descripción de hardware (HDL) cumple los requisitos necesarios para disponer de un simulador de nivel bajo, ya que permite realizar una configuración a medida de la máquina, y modelar el hardware con el detalle funcional y estructural necesario. No obstante, aunque el simulador basado en el lenguaje HDL que utilizamos (*Verilog*) es un simulador que se ajusta a los requisitos relacionados con el análisis de los elementos más próximos al hardware, presenta algunos problemas y limitaciones que hay que resolver para poder llevar a cabo simulaciones realistas de las aplicaciones de comunicación. En el capítulo 2 de esta tesis se detallen las características del los modelos desarrollados responder a todas estas cuestiones.

Como se ha dicho, la simulación es el método más usado para el análisis de la arquitectura de computadores, y también de los sistemas de comunicación. Un simulador adecuado debería ser capaz de reproducir las características del sistema real con la fidelidad necesaria. Obviamente, es necesario llegar a un compromiso entre precisión/detalle de la simulación y tiempo/coste en cuanto a recursos del sistema utilizados por el simulador [YI05].

La simulación del funcionamiento de una arquitectura de computador es una técnica que se utiliza desde hace mucho tiempo y de forma habitual para caracterizar el comportamiento de los diferentes componentes del computador (procesador, caches, bus de E/S, memoria…). Por lo tanto, para reproducir el comportamiento de todos los elementos que integran un sistema real, se necesita un simulador de sistema completo [ENG03]. Es decir, un simulador donde se simula los partes de hardware y software de la arquitectura del sistema real.

Un simulador del sistema completo de un sistema real debe tener las siguientes características: El simulador tiene que permitir la construcción de una computadora con elementos especificados como procesadores, caches, buses de E/S, chipset, memoria, NIC y otros elementos de hardware. Además, el simulador debe poder simular redes de computadores en el caso de la comunicación entre computadores interconectados en la misma subred, o en una subred diferente. El simulador debe disponer de modelos de simulación de todas las partes hardware incluidas en las arquitecturas, por ejemplo, debe disponer de modelos de simulación de buses estándar como PCI y controladores de interrupción. La ejecución de las instrucciones no tendría que tardar mucho más tiempo del necesario normalmente en un sistema real para que el simulador del sistema de comunicación sea útil. Además, es conveniente poseer alguna herramienta para modelar todos los elementos hardware que se deben simular, a través de algún lenguaje de descripción hardware como VHDL [BRO05, ASH08], *Verilog* [VIG05, THO02, LEE03, MIC05], o lenguaje C/C++ [DER08, PET08]. Finalmente, para simular la arquitectura de un computador se necesita un simulador de sistema completo con gran flexibilidad, que cumpla todas las características anteriores y permite evaluar las prestaciones del sistema de una forma precisa. A continuación, se describen algunos simuladores de sistema completo que pueden usar para evaluar las prestaciones de comunicación.

1.7.1 El simulador M5

El simulador M5 es un simulador del sistema completo desarrollado por la universidad de Michigan. M5 proporciona dos modelos de CPU (SimpleCPU y O3CPU). El modelo SimpleCPU se utiliza para una ejecución rápida para las instrucciones. El O3CPU se incluye segmentación de cauce y la ejecución detallada de las instrucciones maquina, así como la ejecución de multihebra simultánea.

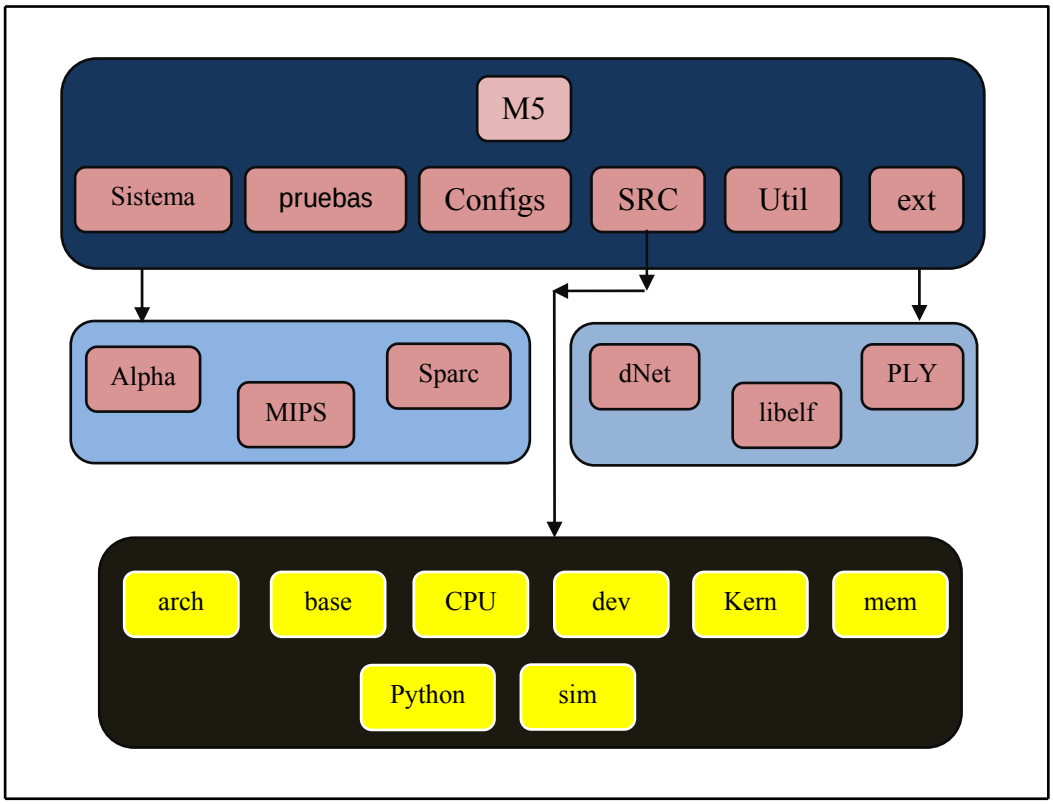

Figura 1.21 Los módulos del simulador M5

Además, el simulador contiene modelos de cache, buses y dispositivos de E/S, y todo lo necesario para modelos de sistema completo. Por otro lado, entre estos modelos se dispone de un sistema de interfaz para transferir los datos entre ellos. La Figura 1.21 muestra el árbol de módulos del simulador M5 [M5S07].

1.7.2 El simulador SimOS

El simulador SimOS es otro tipo de simulador de sistema completa que incluye módulos de los procesadores MIPS R4000 y R10000, y sólo permite la simulación de arquitectura MIPS y Alpha [ROS97]. Por otro lado, no tiene interfaces detallados para las redes. En Figura 1.22 se muestran módulos de simulación de todos los componentes de un sistema actual, como procesadores, MMU (*Memory Managment Units*), caches, memoria de sistemas, buses de E/S, redes Ethernet, relojes y repertorios de instrucción.

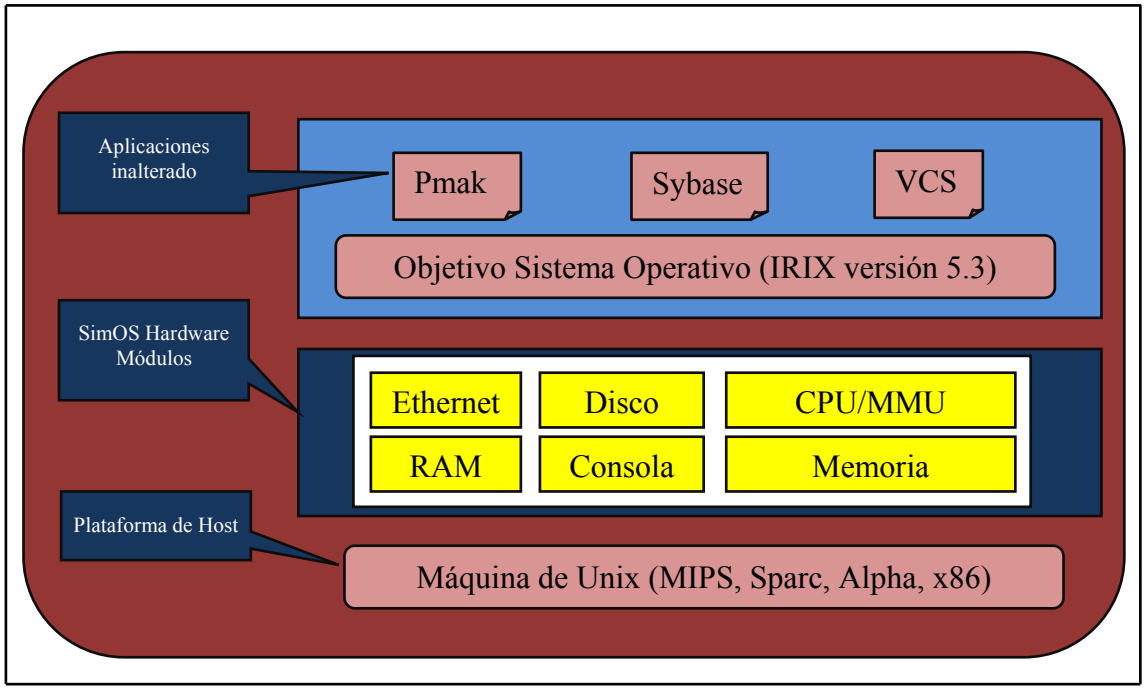

Figura 1.22 El envaramiento del simulador SimOS

1.7.3 El simulador SIMICS

El simulador SIMICS es un simulador del sistema completo desarrollado por la empresa *Virtutech*. El simulador SIMICS es una herramienta muy común en sectores como el de los servidores para computaciones de altas prestaciones, de diseño del hardware de red, de vehículos militares, de automóviles, dado que ayuda mucho a los desarrolladores de hardware y software antes a evaluar el sistema antes de su fabricación [VIR10].

El simulador SIMICS contiene las herramientas necesarias para hacer la simulación de sistema completo. Además, se puede ampliar a nuevas arquitecturas,

tanto de un procesador, como de multiprocesadores. El simulador SIMICS dispone de las prestaciones y las características funcionales para ejecutar aplicaciones reales y para simular modelos hardware detallados. Por lo tanto, mediante el simulador SIMICS se puede simular la ejecución de aplicaciones, el sistema operativo, los controladores y los protocolos de comunicación. Sin embargo, los modelos de temporización del simulador SIMICS pueden resultar insuficientes para algunas propuestas.

Se puede modificar el comportamiento del simulador SIMICS agregando modelos de temporización. En estos modelos, las instrucciones se ejecutan secuencialmente. Por lo tanto, se puede conseguir una simulación funcional rápida aunque, al mismo tiempo no se modela bien el comportamiento temporal que se tiene en los sistemas reales. Se puede evitar esta situación mediante los modelos de temporización que controlan el tiempo de las transacciones. El simulador SIMICS tiene otras características que lo distinguen de otros simuladores como la posibilidad de agregar y programar hardware mediante el lenguaje de descripción de hardware DMI (*Device Modeling Languege*).

Además, el simulador SIMICS permite conectar un simulador VHDL. La Figura 1.23 muestra la arquitectura completa del simulador SIMICS con todos sus módulos [MAG02].

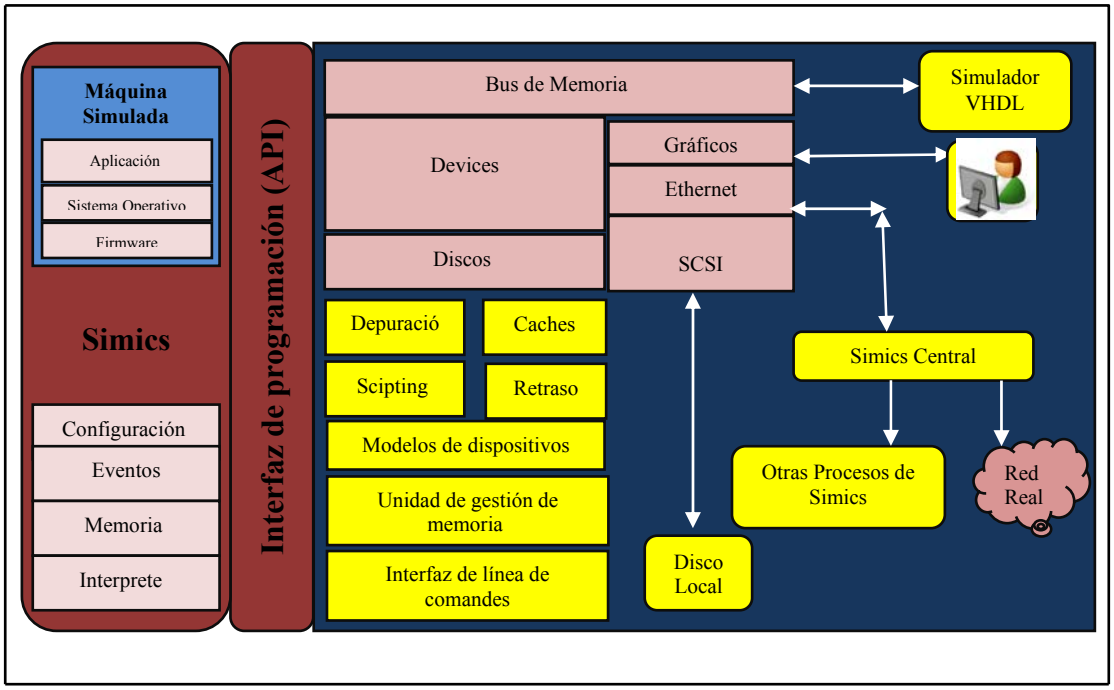

Figura 1.23 La arquitectura del simulador SIMICS

El simulador GEMS también es un simulador de sistema completo que contiene módulos específicos para ser utilizados con el simulador SIMICS y evaluar arquitecturas de multiprocesador y aplicaciones reales, superando algunas de las limitaciones y defectos del simulador SIMICS. El corazón del simulador GEMS (*General Execution-driven Multiprocessor Simulator*) es el simulador del sistema de memoria Ruby. En la Figura 1.24 se muestra la arquitectura de GEMS [MAR05, GEM08].

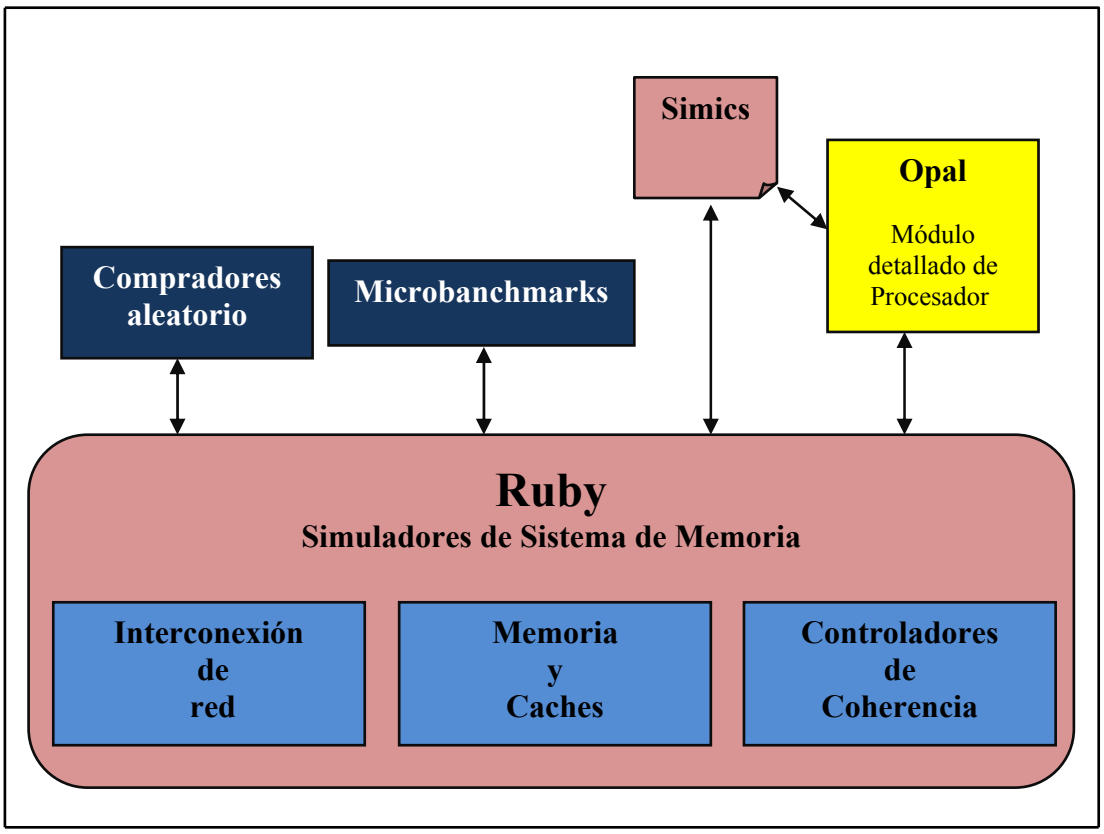

Figura 1.24 La arquitectura del simulador GEMS

El simulador TFsim (*Timung First Simulation*) es un simulador de tiempo para ejecutar instrucciones máquina junto con un simulador funcional. Este simulador permite modelar detalladamente el tiempo y la microarquitectura. Además, la simulación funcional con TFsim es correcta y sólo produce a un error 4.8% en las aplicaciones [MAU02].

1.8 Conclusiones

Este capítulo ha introducido las cuestiones previas que ponen de manifiesto los problemas que plantea la arquitectura de la interfaz de red y justifica los objetivos planteados en este trabajo. Así, se han descrito las tareas de la tarjeta de red y como se transfieren los datos a través de ella. También se ha considerado la estructura de los TOE (*TCP Offload engines*) y algunas alternativas para reducir la sobrecarga en el nodo, y evitar el efecto de los posibles cuellos de botella en la interfaz de red.

También, se han presentado dos modelos cuantitativos aproximados para evaluar las prestaciones de comunicación, LAWS y EMO. Estos modelos ayudan a analizar el espacio de parámetros a considerar en el análisis de la *externalización*. Además, se han amanerado y descrito brevemente varios simuladores del sistema completo (M5, SimOS, SIMICS, GEMS, TFsim).

En el capítulo 2 se introducirán los modelos HDL de simulación que hemos desarrollado para el sistema de comunicación que utilizaremos en este trabajo. Los resultados experimentales obtenidos para distintas alternativas de mejora de la interfaz de red con distintos valores de los parámetros del sistema de comunicaciones se describen en el capítulo 3.

Capítulo 2

MODELOS EN LENGUAJE DE DESCRIPCIÓN DE HARDWARE (HDL) Y OPTIMIZACIÓN DE LA INTERFAZ DE RED

Se puede afirmar que la simulación es la técnica más frecuentemente usada para evaluar las propuestas en arquitecturas de computadores. En este capítulo se describen los modelos de simulación HDL que se han desarrollado para estudiar las condiciones en las que las opciones de *externalización* (*offloading* y *onloading*) pueden aportar mejoras y comparar sus prestaciones con las predicciones del modelo LAWS. La simulación HDL permite analizar con mayor detalle las características de la implementación hardware de nuevas propuestas, complementar los resultados obtenidos a través de simuladores funcionales, y generar estimaciones de la velocidad de reloj que puede alcanzar el hardware (si el modelo con el que se trabaja es un modelo HDL sintetizable). El uso de simulaciones HDL también hace posible la identificación de caminos críticos y de los aspectos del diseño que pueden mejorarse [PIR00].

El capítulo se ha estructurado tal y como se indica a continuación. La sección 2.1 se centra en los aspectos generales de la simulación HDL y las distintas alternativas de mejora de la interfaz de red. Posteriormente, en la sección 2.2 se proporcionan los detalles más relevantes de los distintos modelos HDL que se han elaborado para realizar el estudio que se presenta en esta memoria, incluyendo una propuesta de mejora de la interfaz de red a la que hemos llegado a partir de los resultados de nuestro trabajo de investigación que se describen en el capítulo 3.

2.1 La simulación HDL del sistema de comunicación

Usualmente, los simuladores son las herramientas más utilizadas para desarrollar nuevas arquitecturas. Gracias a la simulación se puede analizar el comportamiento de propuestas para combinaciones de parámetros del espacio de diseño que no se han implementado en máquinas reales o que, aún siendo disponibles comercialmente, no son accesibles al investigador.

No obstante, hay que tener en cuenta que el simulador que se utilice debe incluir o permitir la implementación de modelos del sistema a simular con el nivel de precisión necesario en cada caso. Aunque un modelo de simulación HDL permite estudiar, a un nivel próximo al hardware, el efecto de los retardos en las distintas etapas del camino de comunicación, su utilidad es menos evidente cuando es necesario simular la ejecución códigos realistas y su interacción tanto con el sistema operativo, como con el hardware del computador. Como veremos en esta memoria para el caso de la interfaz de red, es posible elaborar modelos HDL que tengan en cuenta los tiempos de ejecución del software de sistema y de usuario, pero se trata de modelos de simulación que suelen resultar mucho más pesados para los computadores en los que se suelen ejecutar los simuladores HDL. Dado que el objetivo de nuestro trabajo de investigación se dirige al estudio y mejora de propuestas para el diseño eficiente de la interfaz de red, resaltaremos algunos aspectos de las principales propuestas para la mejora de dicha interfaz, antes de pasar a describir los modelos HDL que hemos elaborado.

En la Figura 2.1 se puede observar el tiempo de procesador consumido por los procesos de usuario, el sistema operativo, y los procesos asociados a la gestión de red y demás tareas de comunicación en la ejecución de un servidor web Apache para el

sistema operativo Linux [RAN02]. Como se puede observar, la mayoría porcentaje del tiempo corresponde a los procesos de red por (1) la sobrecarga (*overhead*) de comunicación debida al procesamiento de las interrupciones y el intercambio de datos con el sistema operativo; (2) el procesamiento de la pila de protocolos de comunicación; y (3) los accesos a memoria debido a las transferencias de datos entre el buffer de la NIC y la memoria principal.

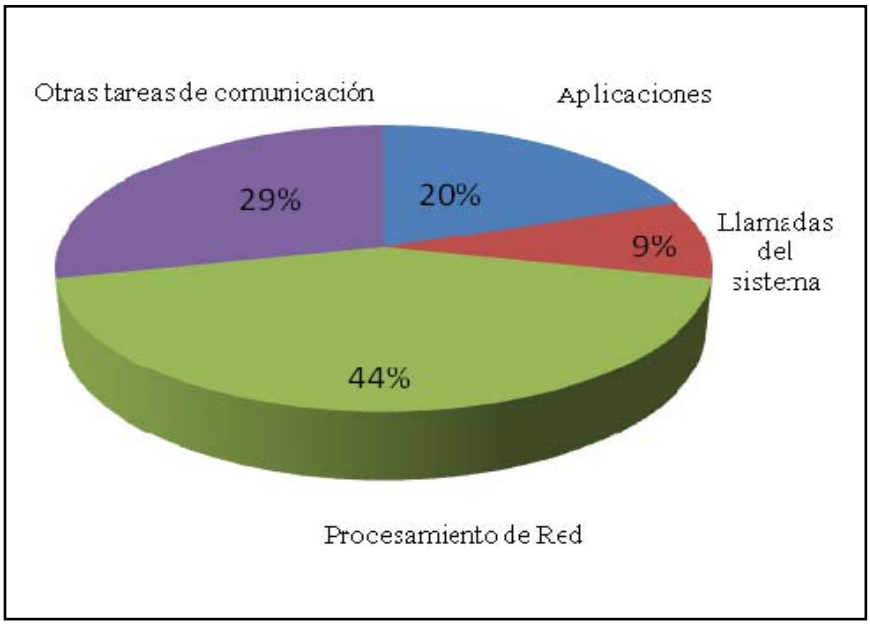

Figura 2.1 Distribución el tiempo de procesador de un servidor apache [RAN02]

Como se ha comentado en el primer capítulo, en las alternativas basadas en la *externalización* mediante *offloading* [WES04] los protocolos de comunicación se implementan en la tarjeta de red, que incorpora elementos hardware entre los que pueden encontrarse procesadores con arquitecturas más o menos especificas para el procesamiento de tareas de comunicación. La otra alternativa de *externalización* se basa en la propuesta de *onloading*, que considera la implementación de los protocolos de comunicación en alguno de los procesadores presente en el sistema, ya sea uno de los núcleos de un microprocesador multi-núcleo o una de las CPU disponibles en un nodo multiprocesador. Para esta alternativa de *onloading* se han propuesto implementaciones comerciales como el I/OAT de Intel, tal y como se ha comentado en el primer capítulo [IOA06].

Para concretar más lo que suponen las distintas propuestas de mejora de la interfaz de red, en la Figura 2.2 se ilustran las cuatro principales alternativas para situar la tarjeta de red (NIC) en el sistema de comunicación [BIN05]:

1. La tarjeta de red está conectada a un bus de E/S (PCI) a través del puente sur.
2. La tarjeta de red está conectada al bus del sistema FSB (*Front Side Bus* o *Hypertransport* [HYP06]) o se encuentra integrada en el *chipset*.
3. La tarjeta de red está conectada directamente al puente norte a través de un bus de alta velocidad (PCI-X de alta velocidad).
4. La tarjeta de red está incluida en el mismo circuito integrado que incluye una o varias CPU (o núcleos).

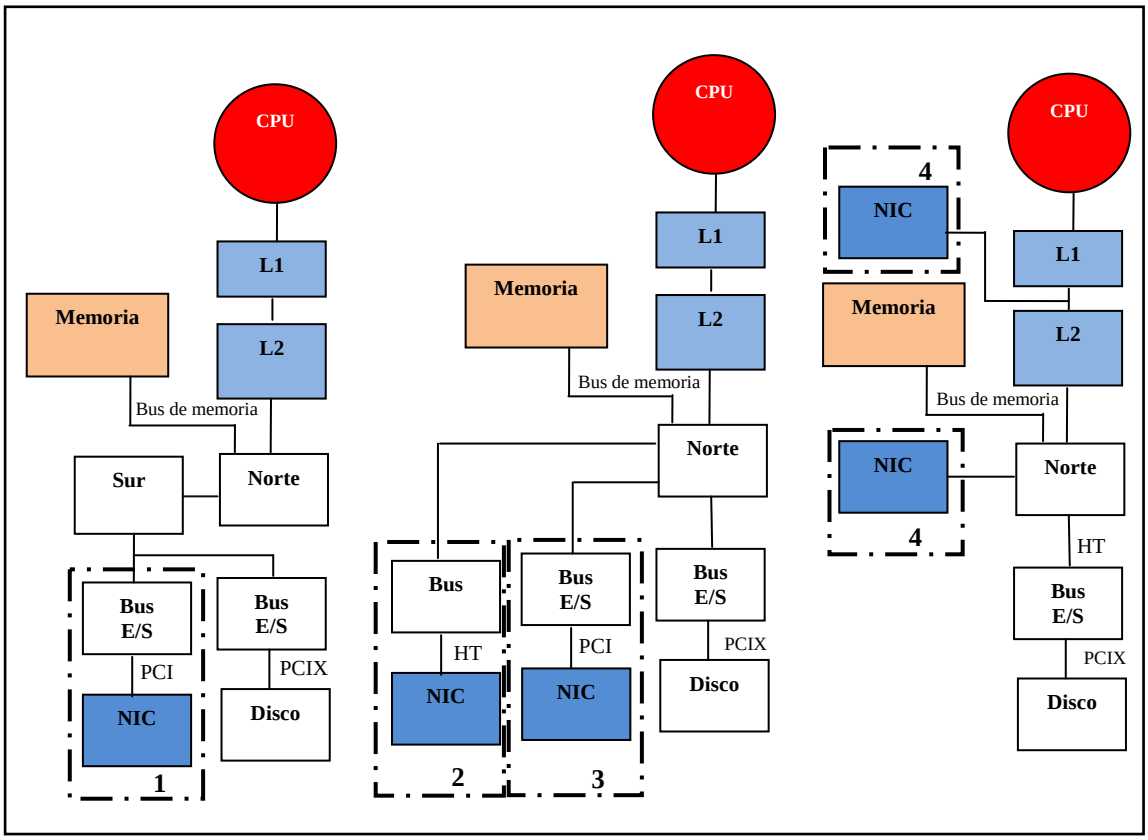

Figura 2.2 La ubicación de la tarjeta de red [BIN05]

La más habitual de las anteriores alternativas es la primera. Sin embargo, en ese caso, cada acceso a memoria debe pasar 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 el controlador de memoria. Cuando la NIC está conectada al puente sur a través un bus de E/S, se incrementa la latencia en los accesos de memoria desde la tarjeta de red dado que hay que sumar las latencias introducidas por los puentes norte y sur. Las cuatro

posibilidades de conexión de la NIC pueden implementarse a través de elementos disponibles comercialmente, por ejemplo utilizando FSB [SUR10] o *Hypertransport* [HYP06] (alternativa 4).

En nuestra propuesta del modelo de simulación, la tarjeta de red (NIC) se conecta al puente norte a través de bus PCI de alta velocidad. Esta ubicación es actualmente la más frecuente para interfaces de red de altas prestaciones ya que, como se ha comentado, los buses de E/S pueden constituir un cuello de botella importante para las prestaciones del sistema de comunicación. Cuando la NIC está conectada directamente al puente norte, la latencia se reduce considerablemente. Además, la posibilidad de acceder directamente a la memoria cache local del procesador puede suponer una mejora en las prestaciones [MAT05]. No obstante, hay que tener en cuenta la latencia introducida por el puente norte, así como las colisiones que pueden producirse en el mismo.

En el caso de que se utilicen microprocesadores con más de un núcleo, uno de estos núcleos podría dedicarse a procesar las funciones de red. Se comentará con detalle el uso de doble núcleo más adelante (Sección 2.3.3).

Se han elaborado distintos modelos HDL para analizar las diferencias entre las técnicas de *offloading* y *onloading*, y comparar sus prestaciones, entre si, y con la configuración de base inicial. Estos cambios incluyen la utilización del puente norte como único conector.

Para aprovechar las ventajas que puede proporcionar la *externalización* del sistema de comunicación en la tarjeta de red, hay que tener en cuenta la implementación de las siguientes funciones importantes: (1) el procesamiento de las interrupciones, para evitar que las interrupciones generadas por la llegada de la paquetes colapsen el procesador del nodo; (2) las copias de datos en memoria, dado que para transferir un dato desde la aplicaciones a la red son necesarios dos transacciones (desde la memoria de usuario a los buffers del núcleo, y desde aquí al interfaz de red) esto genera un elevado tráfico en los buses de memoria y de E/S del sistema [PLA00]; (3) el procesamiento de la pila de protocolos aunque, como se ha indicado, no es la principal fuente de sobrecarga del sistema de comunicaciones [SEN04]. En general, y teniendo en cuenta la estimación que se ha realizado, el procesamiento de la interfaz de red para enlaces de 10 Gbps puede saturar la capacidad de los procesadores actuales, a no ser que se realice una implementación eficiente de dicha interfaz de red.

La *externalización* mediante la técnica *onloading* consiste en transferir el procesamiento de los protocolos de comunicación a uno de los procesadores centrales del nodo, de forma que los paquetes TCP/IP sean procesados por un núcleo de propósito general distinto del núcleo donde se ejecuta la aplicación. Es decir, que el *onloading* se trata de una *externalizacion* de los procesos de comunicación a otro procesador disponible en el nodo que no está ubicado en la NIC, como en el caso del *offloading*, y que tiene acceso directo a la jerarquía de memoria del sistema como los otros núcleos del nodo en donde se ejecutan las aplicaciones y el sistema operativo. Como ejemplo de implementación de la alternativa de *onloading* está la propuesta por Intel [REG04a], a la que nos referimos en el primer capítulo, y entre cuyas características específicas se pueden especificar las siguientes:

1. Para disminuir las latencias asociadas a las transferencias de datos se utiliza un sistema de DMA mejorada.
2. Para reducir la sobrecarga al procesamiento de TCP/IP protocolos se optimiza la pila de dichos protocolos.
3. Para paralelizar el procesamiento de los paquetes que llegan a la tarjeta de red, dichos paquetes se segmentan.

De todo lo que se ha comentado antes se puede concluir que la alternativa de de *onloading* trata de reducir la sobrecarga asociada del procesamiento de las comunicaciones, incluyendo la gestión de los datos correspondientes, sin que ello suponga cambios en el hardware del sistema y de forma de transparente a las aplicaciones.

Normalmente, cuando se recibe un paquete, el sistema inicia una serie de tareas que implican la interacción de la tarjeta de red (NIC) con el procesador del sistema, generándose usualmente una interrupción por parte de la NIC. Una vez que el procesador realiza ciertas comprobaciones con los datos de paquetes recibidos y se ha verificado que no contienen errores, se transfieren a la memoria de usuario. Teniendo en cuenta la cantidad de parámetros a considerar, y la casuística tan variada que pueden presentar las aplicaciones de comunicación, resulta complicado realizar una exploración significativa del espacio de parámetros mediante experimentos con sistemas reales. Más aún cuando no se conocen con precisión los detalles de implementación de las alternativas. Por lo tanto, es muy conveniente disponer de una herramienta que permita

modificar los parámetros que definen el espacio de diseño de una interfaz de red en un sistema simulado. Esto es lo que se ha pretendido y se muestra en el resto del capítulo.

2.2 Modelos HDL para simulación del sistema de comunicación

Como se ha comentado anteriormente, para analizar y evaluar las mejoras que supone la *externalizacion* y las propuestas que planteamos en esta memoria se han implementado dos modelos de simulación. Uno de ellos permite la simulación del sistema sin *externalización* y el otro la simulación del sistema con *externalización*, mediante *offloading* y mediante *onloading*.

La Figura 2.3 compara lo que podría ser una recepción en un camino de E/S sin *externalización*, 2.3(a), con dos alternativas para la *externalización* con *offloading*, 2.3(b) y 2.3(c). En el caso de recepción sin *externalización*, el NIC, después de extraer la información correspondiente del paquete recibido, interrumpe a la CPU. Después, se ejecuta el driver para iniciar la operación de DMA entre el NIC y la memoria principal (a la dirección donde se almacenan), e informa a la CPU al final de esta operación. Entonces, la CPU se encarga del procesamiento del protocolo del paquete almacenado en memoria principal.

En la alternativa de *offloading* que se muestra en la Figura 2.3(b), el NIC es capaz de comenzar el procesamiento del protocolo de comunicación después de recibir el paquete entero (o hasta después de recibir parte de él). Entonces, el NIC interrumpe a la CPU, y ejecuta al driver para inicializar la operación de DMA como en la alternativa de que no utiliza *externalización*. Después de esto, el NIC comienza la operación de DMA para transferir los datos del paquete a la memoria principal e informa a la CPU de que dichos datos están disponibles en su memoria principal al final del DMA. La alternativa que se muestra en la Figura 2.7(c) libera a la CPU de casi toda la sobrecarga de la comunicación cuando el NIC es capaz de procesar el protocolo, e inicializa el DMA para enviar los datos del paquete a memoria. Cuando éstos se han almacenado, la NIC informa a la CPU de que la aplicación puede usarlos.

En la Figura 2.4, se muestra el modelo de simulación, que incluye un procesador, el puente norte del chipset, la memoria, el bus PCI y la tarjeta de red. En los módulos que hemos realizado, el software se simula mediante retardos de diferente magnitud según se tenga una mayor o menor sobrecarga asociada al procesamiento de

los protocolos, los *drivers*, etc. Así, se puede explorar más fácilmente y de forma más completa el espacio de los parámetros de diseño para comparar los resultados de simulación con lo previsto según modelos como el LAWS. En cualquier caso, para obtener resultados más próximos a los que se observan en aplicaciones reales, hay que generar las correspondientes trazas [NAY79, THO91].

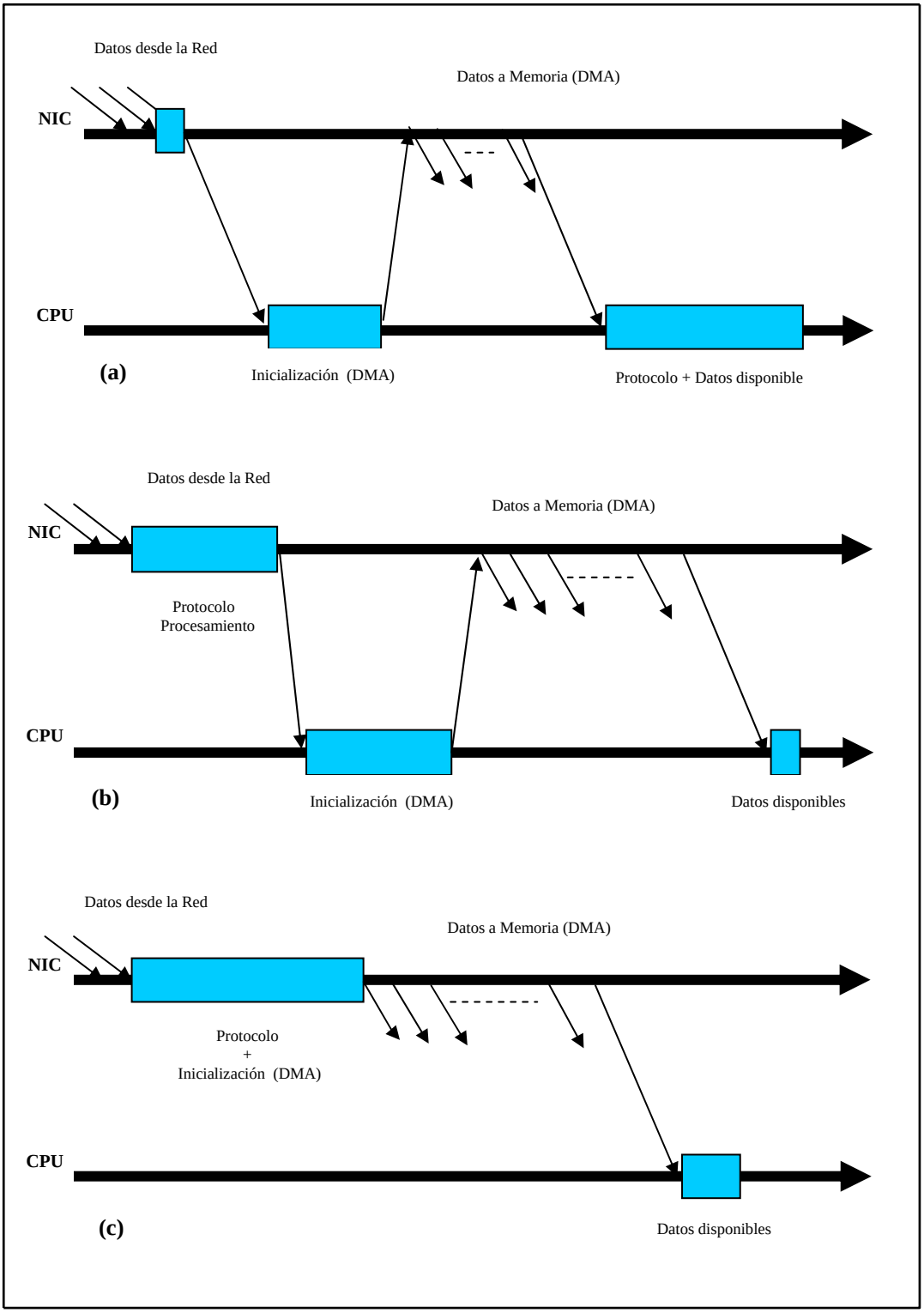

Figura 2.3 Recepción sin *externalización* (a); y dos opciones de *offloading* (b) y (c).

En el capítulo 3 se muestran los resultados del modelo de simulación HDL cuando se usa un fichero de trazas generadas por una aplicación real (*Ethernet at the Bellcore Morristown Research*). El fichero de trazas contiene paquetes Ethernet de distintos tamaños (entre 64 y 1518 bytes), y el tiempo de llegada de los paquetes [INT10].

Las alternativas de *externalización* mediante *offloading* o mediante *onloading* tratan de reducir la sobrecarga (*overhead*) asociada al procesamiento de las comunicaciones.

Como se ha indicado antes, cuando la interfaz de red recibe un paquete se inicia una serie de de operaciones asociadas a la interacción entre la NIC y el procesador del nodo. Por lo tanto, todas estas operaciones se inician con una interrupción generada por la NIC. Una vez que el procesador accede a los datos del paquete y verifica que son correctos, los transfiere a la memoria de usuario, donde la aplicación podrá usarlos.

El uso del *offloading* proporciona mayores beneficios en el caso los paquetes de tipo *Jumbo* (tamaño grande), porque el uso de paquetes pequeños generaría una latencia elevada debido al gran número de accesos, a memoria y también por la interfaz entre la NIC y el sistema operativo.

2.2.1 El modelo de simulación HDL sin *externalización*

En la Figura 2.4 se muestra un esquema del camino de comunicación que se va a simular para recepción y emisión de paquetes sin *externalización*. En dicha figura, se muestran dos computadores conectados a través de la red, cuyos retardos se modelan mediante el correspondiente módulo HDL (módulo *Network*). Supongamos que dos procesos, uno en cada nodo, se comunican transmitiéndose datos desde la memoria de usuario del emisor a la memoria de usuario del receptor. En primer lugar, los datos se pasan desde el espacio de memoria de usuario modelado en el módulo *Memoria* a través del módulo *Chipset* y el bus de *E/S* modelado en el módulo *E/S* hasta el módulo de la tarjeta de red (*NIC*). El tiempo que se consume en estas transferencias, incluyendo el necesario para la transmisión a través de los buses del nodo, el tiempo para controlar la transferencia y para dar a los datos el formato adecuado para la transmisión, constituye la sobrecarga (*overhead*) de emisión. Después de esta operación, el tiempo para transferir los paquetes a través de la red, desde la *NIC* del emisor a la *NIC* del receptor, se agrega como una parte más de la sobrecarga de comunicación. Finalmente tenemos la

sobrecarga de comunicación en el receptor, que incluye el tiempo necesario para pasar los datos desde la NIC al módulo de memoria (módulo *Memoria*) del proceso receptor.

En el esquema de la Figura 2.4 se muestran también los módulos en que se ha organizado el modelo de simulación HDL para la recepción. En las simulaciones realizadas, una vez que se recibe un paquete de la red (módulo *Network*) en la tarjeta de red (*NIC*) del receptor, éste se almacenan en una cola de *buffers* y se transfieren por DMA a la memoria del computador a través del bus de E/S. Cuando se llena el buffer de la tarjeta de red el módulo *NIC* genera una interrupción hardware que alcanza al procesador, cuyo modelo HDL se encuentra en el módulo *CPU*.

Dicha interrupción hace que se ejecute el *driver* de la NIC, que copiará los paquetes desde el buffer de la tarjeta de red (NIC) al módulo de memoria, quedando disponibles para la aplicación. En el siguiente capítulo se mostrarán los resultados de las simulaciones que hemos realizado para analizar las prestaciones de este sistema. Los módulos *NIC* y *CPU* se implementan de forma diferente según se modele algún tipo de *externalización* o no.

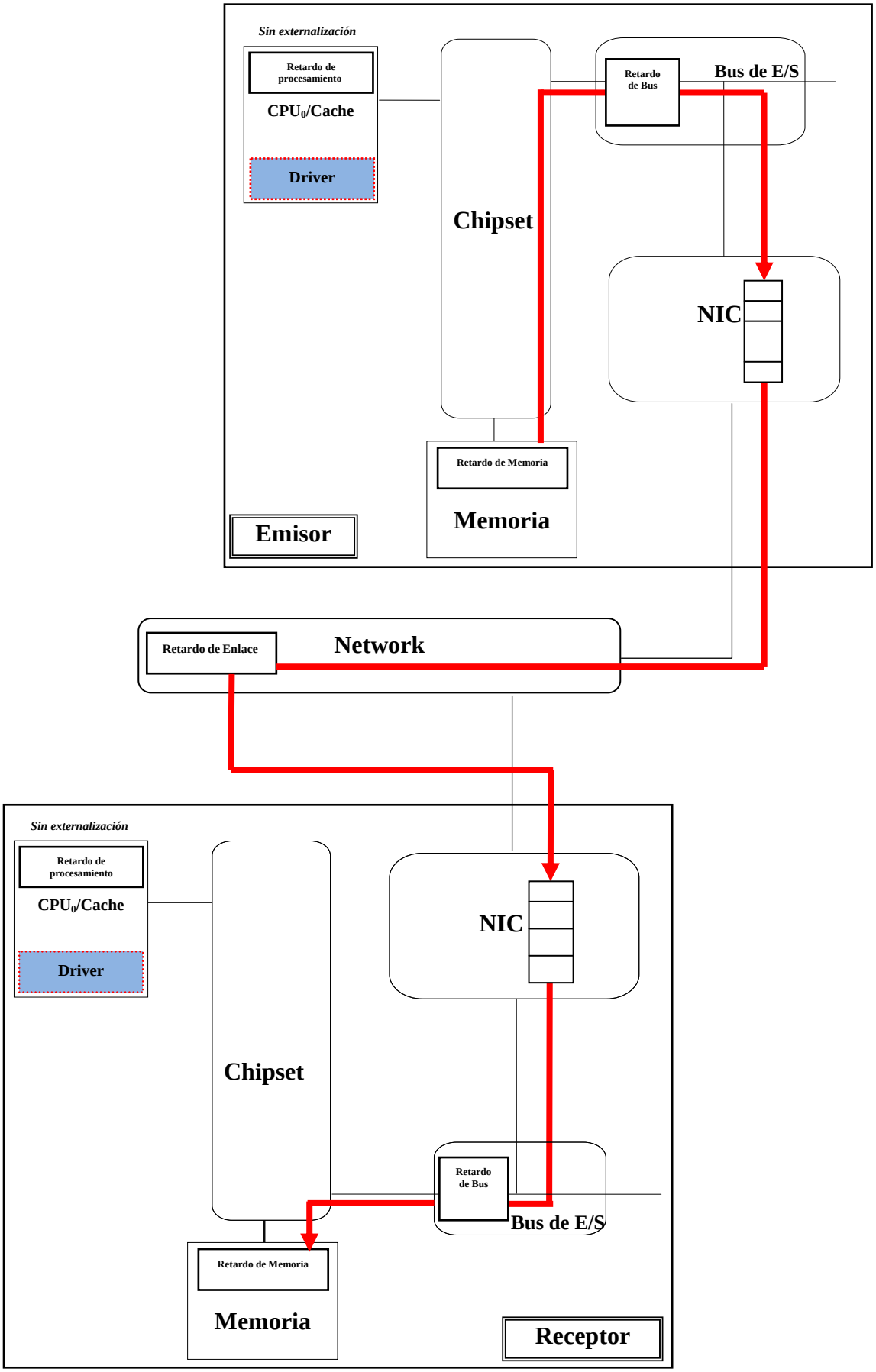

Figura 2.4 El modelo de simulación HDL de todo el camino de comunicación (Sin *externalización*)

2.2.2 El modelo de simulación HDL para la *externalización* mediante *offloading*

En el caso de utilizar *externalización* mediante *offloading*, la carga de la CPU en el receptor se reduce considerablemente dado que el *driver* se ejecuta en orto procesador, ubicado en la tarjeta de red, y posiblemente implementado con una tecnología que ofrece menos prestaciones que la CPU. Como se comenta en [WES04], es necesario tener en cuenta algunas cuestiones. La primera es que el sistema operativo considera que solo hay un procesador. Esto puede suponer un problema debido a que los procesadores de un SMP comparten recursos y es necesario asegurar que en el procesador de la tarjeta de red no van a ejecutarse hebras del sistema operativo ni de las aplicaciones. La segunda característica es que la NIC dispone de su propia memoria y buses internos. Por tanto, la NIC simulada no debe compartir recursos del sistema que pueden ocasionar colisiones. Así, desde el punto de vista hardware, esta característica supone un aislamiento del procesador de la tarjeta de red del resto del sistema operativo. La última cuestión es que los ciclos de CPU utilizados para realizar el procesamiento del los protocolos en el caso no externalizado debe ser similar al empleado por la alternativa de *offloading*. Así, para simular la alternativa de *offloading*, se ha utilizado otro modelo de simulación que se muestra en la Figura 2.5, y que corresponde a un sistema configurado de forma que libere el procesador central de ejecutar los ciclos de los protocolos de la comunicación. Las transacciones entre la interfaz de red y el procesador de la tarjeta de red no se ven limitados por la latencia impuesta por el bus de E/S. En este caso, el procesador central se queda más libre porque todo el procesamiento de la pila de TCP/IP se realizará en la tarjeta de red.

Cuando se externaliza el sistema mediante la técnica de *offloading*, todo el procesamiento del protocolo de comunicación se hace en la NIC y el procesador central sigue ejecutado la aplicación y el resto del sistema operativo.

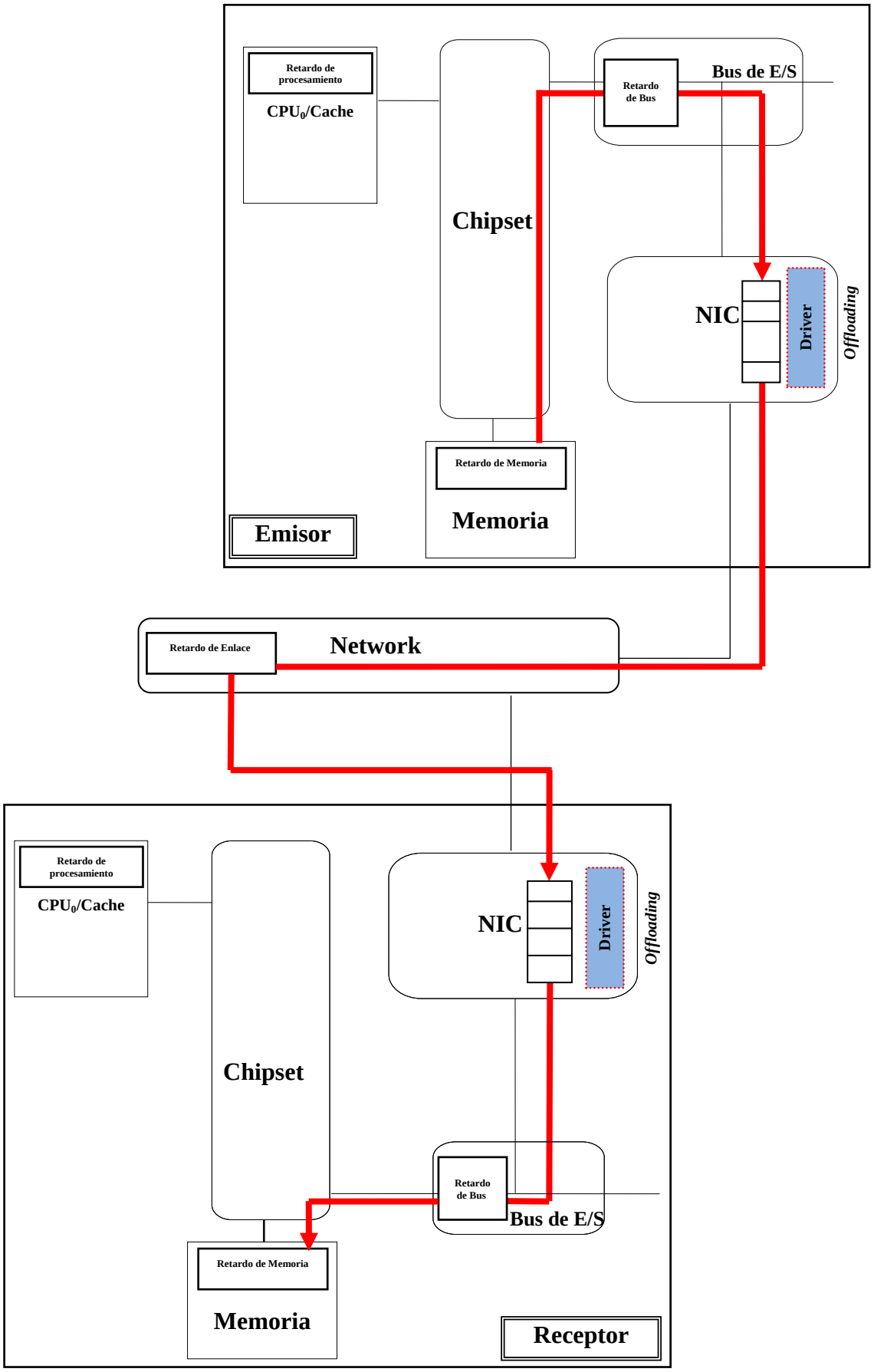

Figura 2.5 El modelo de simulación HDL de todo el camino de comunicación para la externalización (*offloading*)

Como se he indicado, el modelo que hemos diseñado tiene como objetivo cumplir con las características mencionadas en la sección anterior, y así poder simular un procesador incluido en la tarjeta de red.

En la Figura 2.5 se muestra el camino de comunicación para la alternativa de *offloading* en la emisión y recepción de paquetes. En la parte de recepción, los paquetes van llegando a la tarjeta de red y se almacenen en el correspondiente buffer. Cuando el buffer se llena, el procesador de la tarjeta de red se encarga del procesamiento de los protocolos y posteriormente se ejecuta el *driver* para transferir los datos a la memoria usuario. Por tanto, según este esquema, se genera una interrupción cuando el buffer de tarjeta de red se llena, no se genera una interrupción por cada paquete.

2.2.3 El modelo de simulación HDL para la *externalización* mediante *onloading*

Para el *onloading* se utilizan dos módulos de CPU [REG04a, REG04b]. En uno de ellos se ejecutan todas las tareas relacionadas con el procesamiento de comunicaciones y en el otro las aplicaciones y el resto de tareas del sistema operativo relacionadas con la comunicación. Para analizar los beneficios que aporta esta alternativa y compararlos con los obtenidos a través del *offloading*, se ha desarrollado otro modelo de simulación

La Figura 2.6 muestra el camino de comunicación para la simulación de la técnica de *onloading* incluyendo dos procesadores por nodo, que pueden corresponder a un CMP o a un SMP. Como en el caso del modelo de simulación de la *externalización* mediante *offloading*, se han utilizado diferentes valores de retardos para simular el efecto de las aplicaciones, las tareas del sistema operativo, los retardos de los buses del sistema, etc. Para evaluar las mejoras introducidas por la *externalización* mediante esta técnica se compararán los resultados con los obtenidos por el modelo de simulación sin *externalización* (Figura 2.4).

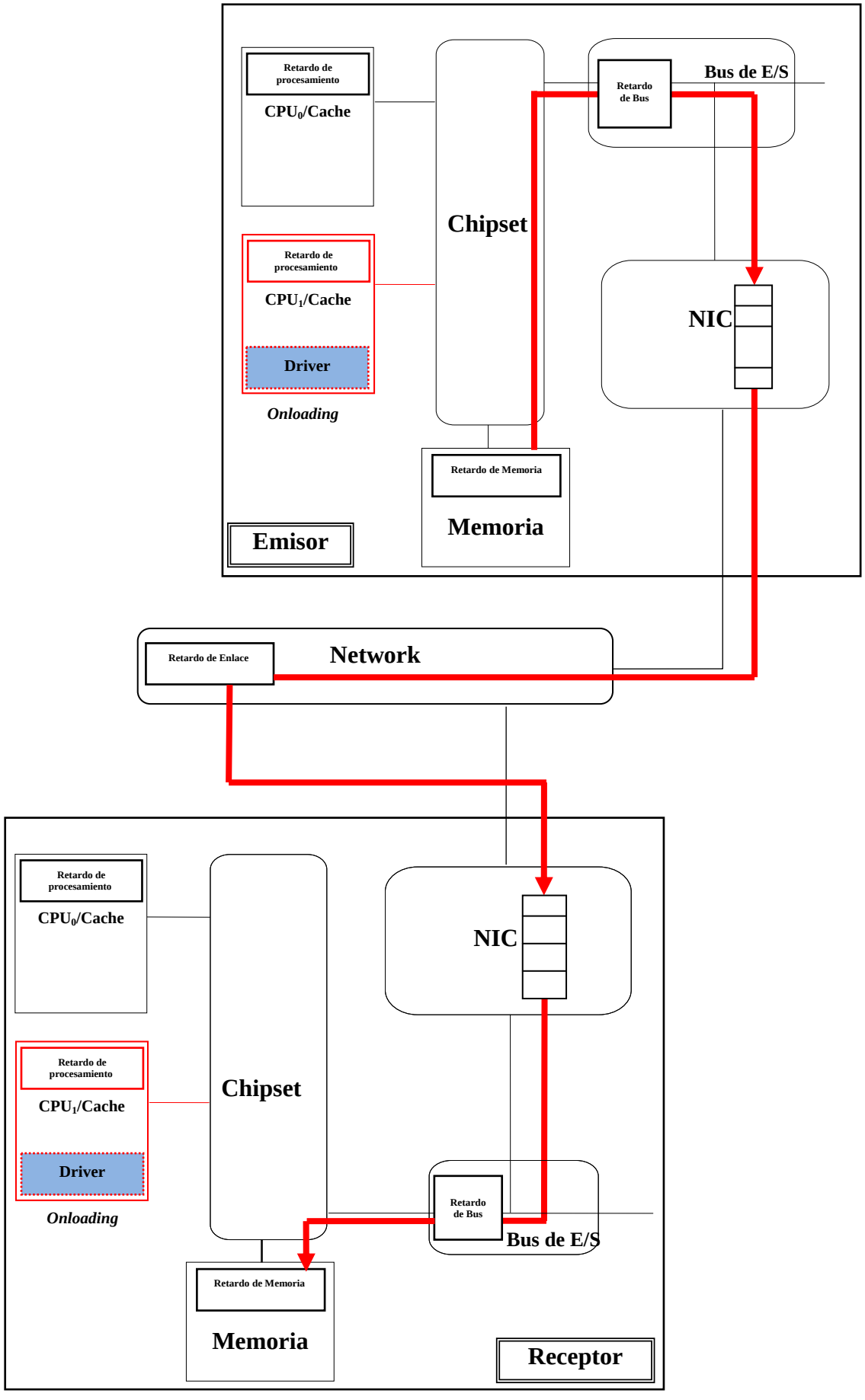

Figura 2.6 El modelo de simulación HDL de todo el camino de comunicación para la externalización (*onloading*)

Cuando la NIC recibe paquetes, estos se almacenan en el correspondiente buffer hasta que se llena. Entonces, se genera una interrupción a la CPU_1 en el caso de un SMP o a un núcleo en el caso de CMP que, para la *externalización* mediante *onloading*, es el procesador que se encarga de todas las tareas de comunicación, incluyendo la ejecución del *driver* de la tarjeta.

Como se ha indicado, en la Figura 2.6 se muestra el camino que siguen los paquetes en el sistema de comunicación para la alternativa de *onloading*. Las principales diferencias entre los modelos de simulación de la *externalización* mediante *onloading* u *offloading* son las siguientes:

- En el modelo de *externalización* mediante *offloading*, el *driver* de la NIC se ejecuta en la tarjeta de red, mientras que en el modelo de *externalización* con *onloading* dicho driver se ejecuta en la CPU_1 que procesa todos los protocolos de comunicación.
- En las dos alternativas, las hebras de la familia de protocolos TCP/IP se ejecutan en otro procesador distinto a la CPU_0 donde se ejecuta la aplicación. No obstante, en la alternativa de *offloading* el procesador ubicado en la tarjeta de red y en la de *onloading* se usa otro procesador SMP, u otro núcleo del mismo procesador CMP.
- En el caso del *offloading*, se eliminan ciertas transferencias a través del bus de E/S reduciendo las colisiones que se producen en el mismo.

De esta forma, en el *onloading*, cada vez que se llena el buffer de la tarjeta de red con paquetes recibidos, la NIC genera una interrupción que llega a la CPU_1, y el sistema operativo inicia la ejecución del *driver* en la CPU_1. Cuando termina la ejecución de la interrupción, el sistema operativo realiza la copia de los datos a la zona de memoria de sistema.

En las Figuras 2.7 (a, b y c) se muestran diagramas de tiempo para el intercambio de algunas de las señales de los distintos modelos HDL de la simulación, dependiendo de la técnica de *externalización* utilizada para procesar el protocolo de comunicación (*offloading*, *onloading*, y *sin externalización*).

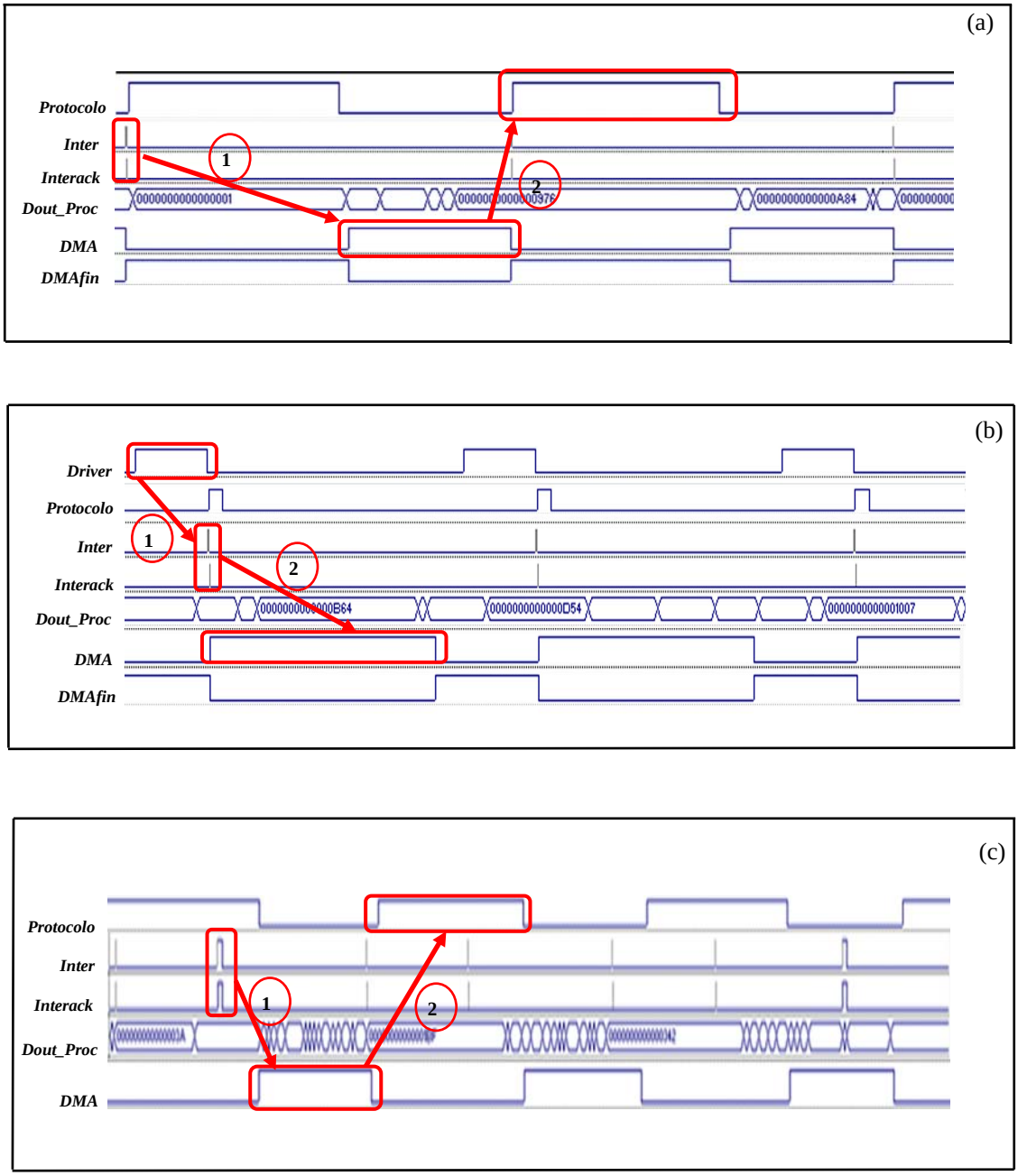

Figura 2.7 Diagrama de tiempo para las tres alternativas (a) *sin-externalización* , (b) *offloading*, y (c) *onloading*.

La señal *Protocolo* en la Figura 2.7(a) indica que el protocolo de comunicación se está procesando en el módulo de CPU (*sin externalización*), mientras la señal *Driver* en la Figura 2.7(b) indica que el protocolo de comunicación se procesa en la NIC (*offloading*). Por último, en el caso del *onloading* (Figura 2.7(c)), la CPU_1 se encarga de hacer el procesamiento del protocolo de comunicación, y se indica con la señal *Protocolo*.

Los paquetes entran a la NIC y se guardan en el *buffer* de la tarjeta de red a la velocidad del enlace. En el caso de no usar *externalización*, como se muestra en la Figura 2.7(a), (1), la CPU se interrumpe intercambiando las señales *inter* y *interack*, luego la CPU inicializa el DMA mediante la señal *DMA* para transferir los paquetes que están almacenados en la tarjeta de red al módulo *Memoria*. La operación finaliza al activarse la señal *DMAfin*. Entonces, (2), la CPU procesa los paquetes mediante la señal *protocolo*. En el caso de *externalización* mediante *offloading*, como se muestra en la Figura 2.7(b), (1) los paquetes se procesan en la tarjeta de red tal y como se indica al estar activada la señal *driver*, y (2) la transferencia de los paquetes se inicializa mediante la CPU. Por último, en el caso de *externalización* mediante *onloading*, como se muestra en la Figura 2.7(c), el módulo CPU_1 se encarga de procesar el protocolo tras la interrupción generada por la NIC (1), mientras la señal *protocolo* esta activada (2).

En la Tabla 2.1 se muestra la distribución de las tareas de comunicación entre los distintos elementos, según los modelos de simulación de *externalización*, mediante la técnica *offloading* y *onloading*, y cuando no hay *externalización*.

Tabla 2.1 Distribución de las tares de comunicación

Tarea	***Sin externalización***	***Offloading***	***Onloading***
Driver	CPU_0	NIC	CPU_1
Interrupción	CPU_0	CPU_0	CPU_1
Protocolo de comunicación	CPU_0	NIC	CPU_1

En la Tabla 2.1 se puede ver que, en el caso en que no se utiliza *externalización*, el procesador CPU_0 se encarga de todo el procesamiento del protocolo de comunicación y de las aplicaciones. En cambio, en el caso de *externalización* mediante *offloading*, se utiliza la NIC para el procesamiento del protocolo de comunicación y para ejecutar el driver, mientras que el procesamiento de las interrupciones lo hace el procesador central. En el caso de la *externalización* mediante *onloding* se utiliza el núcleo CPU_1 para procesar el protocolo de comunicación, ejecutar el driver, y manejar las interrupciones, liberándose ciclos de la CPU_0, que se encargaría de procesar el sistema operativo y las tareas asociadas a la aplicación.

2.3 Los módulos de la simulación HDL del sistema de comunicación

Como se ha comentado anteriormente en este capítulo, el modelo utilizado para simular cada alternativa (*sin externalización*, o con *externalización*) contiene varios módulos HDL que dependen de las características de la interface de red en cada caso. A continuación, se describen cada uno de estos módulos con sus señales de control, elementos de almacenamiento, etc. El código HDL para cada módulo se muestra con más detalles en el Apéndice I. En la sección 2.3.1, se explican las características del módulo HDL CPU y de la cache del mismo, mientras en las secciones 2.3.2 y 2.3.3 se muestran los módulos HDL del *Chipset* y de la memoria respectivamente. En la sección 2.3.4, se muestra el diseño del módulo HDL del bus de E/S con todas sus señales de control. Además, en la sección 2.3.5 se explican las características del módulo NIC y el movimiento de las señales de control en el mismo. Finalmente, en la sección 2.3.6, se muestra el funcionalmente del módulo de red (módulo *Network*) y las señales de entrada y salida del dicho módulo.

2.3.1 El módulo CPU

Este módulo ejecuta las instrucciones correspondientes a la aplicación de usuario al ritmo de una instrucción por ciclo, como se muestra en la Figura 2.8. Es decir, se considera un procesador segmentado superescalar que podría terminar más de una instrucción por ciclo pero termina una instrucción por ciclo en media.

En este caso se considera que pueden ocasionarse colisiones que ralentizan al procesador si éste genera accesos a memoria que no están en cache y el número de ciclos necesarios para completar una instrucción se incrementa. En el caso de que el procesador tenga que ejecutar código correspondiente al *driver* o al protocolo de comunicación, deja de procesar instrucciones de la aplicación de usuario durante el tiempo correspondiente.

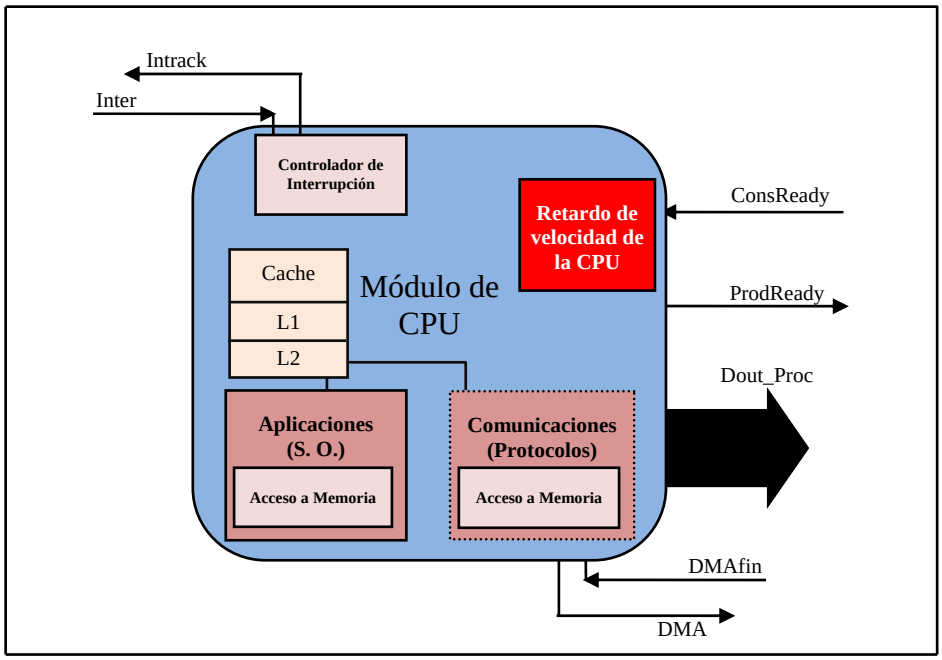

Figura 2.8 Módulos y señales de control del Módulo CPU

En la Figura 2.8 se muestran los módulos que simulan la aplicación y el sistema operativo, que se modelan a través de un bucle que permite controlar el tiempo consumido por el sistema operativo cada vez que intervine como se muestra en la Figura 2.13 y en el Apéndice I (en la descripción del módulo del procesador principal).

En la Figura 2.9 se muestran las señales de solicitud y reconocimiento de interrupción se intercambien entre el módulo de CPU el módulo de NIC, donde la señal *Inter* se utiliza para solicitar una interrupción y la señal *Interack* se usa para aceptar la interrupción. Las señales *DMA* y *DMAfin* se utilizan para marcar el acceso directo a memoria entre la NIC y la CPU. Además, hay otras señales de control de *handshaking* (*ConsReady*, *ProdReady*), que se utilizan para controlar la transferencia de datos (leer/escribir datos). Como se muestra en la Figura 2.9, la CPU se interrumpe mediante la señales *Inter* y *Interack* (1), luego la CPU comprueba el estado del módulo *Chipset* mediante la señal *ConsReady* si está listo para recibir datos. Entonces, la CPU inicializa el DMA para transferir los datos y se finaliza la operación al activarse la señal *DMAfin* (2).

Se ha modelado una memoria cache de dos niveles por cada CPU [REA10, ZA07] para las simulaciones de los distintos modelos. El funcionamiento de la memoria cache se puede resumir en el diagrama de flujo de la Figura 2.10.

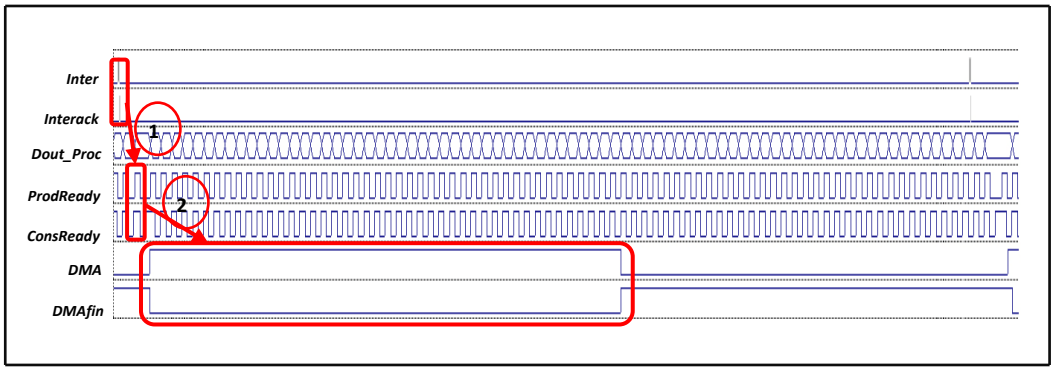

Figura 2.9 Diagrama de tiempo para las señales del módulo de CPU

En la figura se describe el proceso de traducción de la dirección física procedente de la CPU al dato ubicado a la posición de memoria cache determinada por dicha dirección.

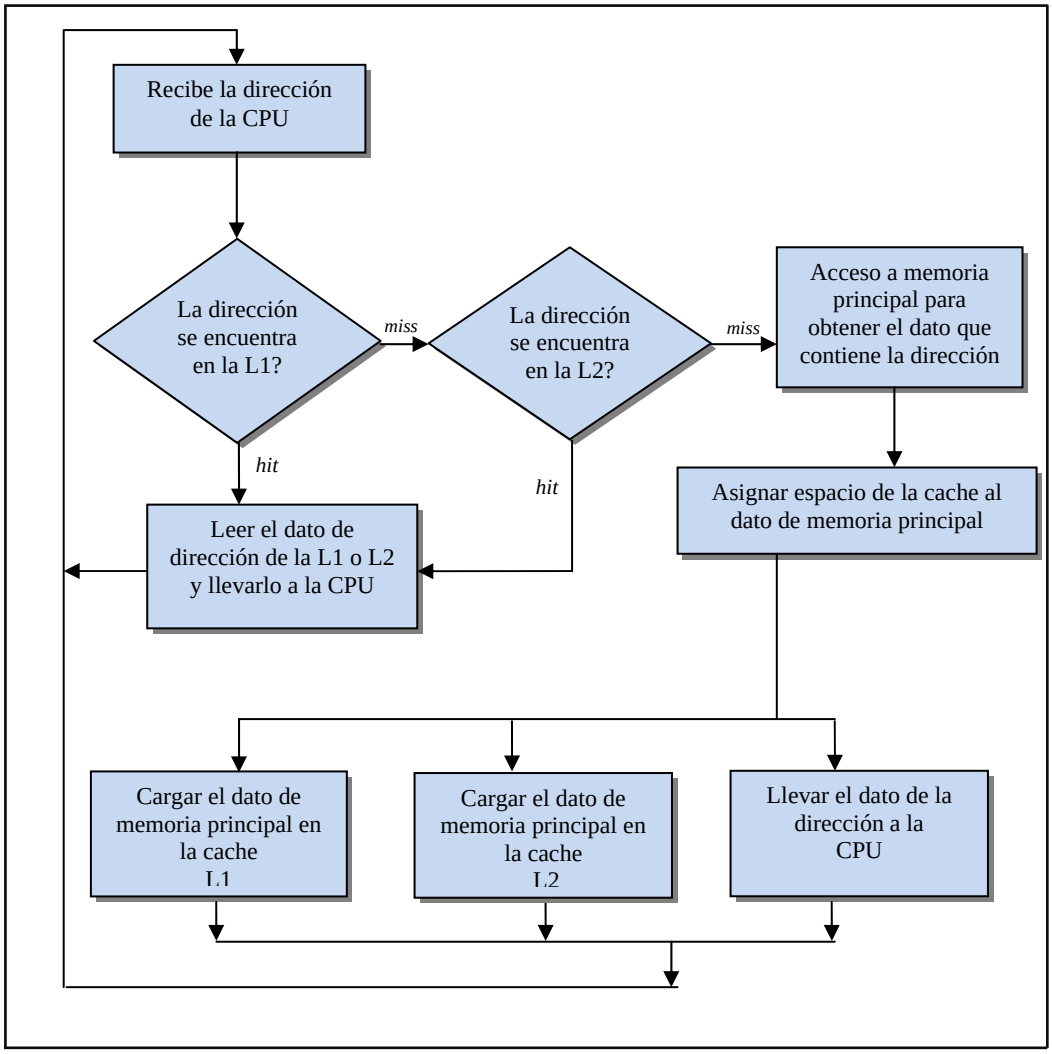

Figura 2.10 Diagrama de flujo del funcionamiento de cache

Como se muestra en la Figura 2.10, cuando una dirección se presenta en la cache (L1 ó L2) pueden ocurrir dos cosas: el contenido de la dirección se encuentra en una línea de la cache (*hit*), se lleva el contenido a la CPU. Cuando no se encuentre el dato en ninguno de los dos niveles de la cache (*miss*) se hace acceso a la memoria principal para encontrar el dato y asignarlo en la memoria cache L1 y L2, y luego enviarlo al procesador CPU.

2.3.2 El módulo *Chipset*

La Figura 2.11 muestra el diseño del módulo *Chipset* que se usa como interfaz para pasar datos entre los correspondientes módulos HDL que conecta, equivale al puente norte y al puente sur. Este módulo se encarga de controlar las interrupciones, el DMA, y el movimiento de los datos entre los distintos espacios de almacenamiento.

En la Figura 2.12 se observan las señales de control y del bus de E/S que entran y salen del módulo *Chipset*. Las señales de control *cReadyIn, cReadyOut, pReadyIn* y *pReadyOut* se utilizan para controlar mediante *handshaking* las transferencias de datos entre el *chipset* y los demás módulos, mientras que las señales de control entre el mismo *Chipset* y el procesador son *pReeadyInP* y *cReadyInP*. Las señales *dmaIn, dmafinIn, dmaInP* y *dmafinInp* se utilizan para el control del acceso directo a memoria.

Además, el módulo C*hipset* tiene una señal de control muy importante, la señal *Busrequest* que se usa para comprobar el estado del bus al trasferir los datos entre los módulos del sistema de comunicación. Se puede utilizar el retado de la señal *Busrequest* para controlar la velocidad del módulo *Chipset*.

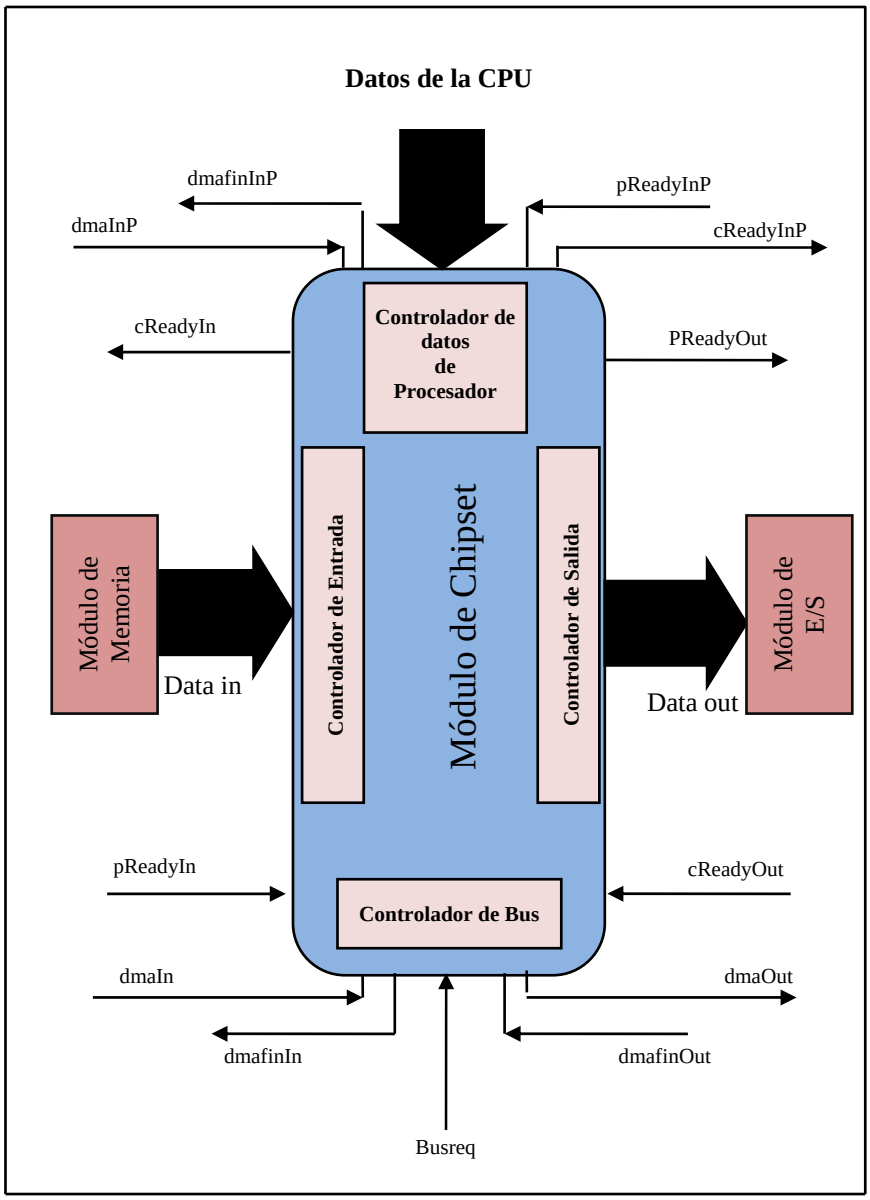

Figura 2.11 Módulos y señales de controles del módulo *Chipset* en la emisión

En la Figura 2.12 se muestra el movimiento de los datos del módulo de *Chipset* entre distintos elementos dependiendo del tipo de transferencia de datos entre el módulo *Chipset* y los demás módulos (leer datos del procesador central, leer datos del módulo de memoria, y escribir datos en el módulo de E/S). En el caso de recibir datos del módulo de procesador central, (1) Figura 2.12(a), cuando la señal *Busrequest* activa, eso significa que el bus está listo para transferir los datos, y entonces el módulo de chipset empieza a comprobar el estado del módulo del procesador mediante la señal *pReadyInP* (el módulo de procesador está listo para enviar datos al módulo de *Chipset*), mientras

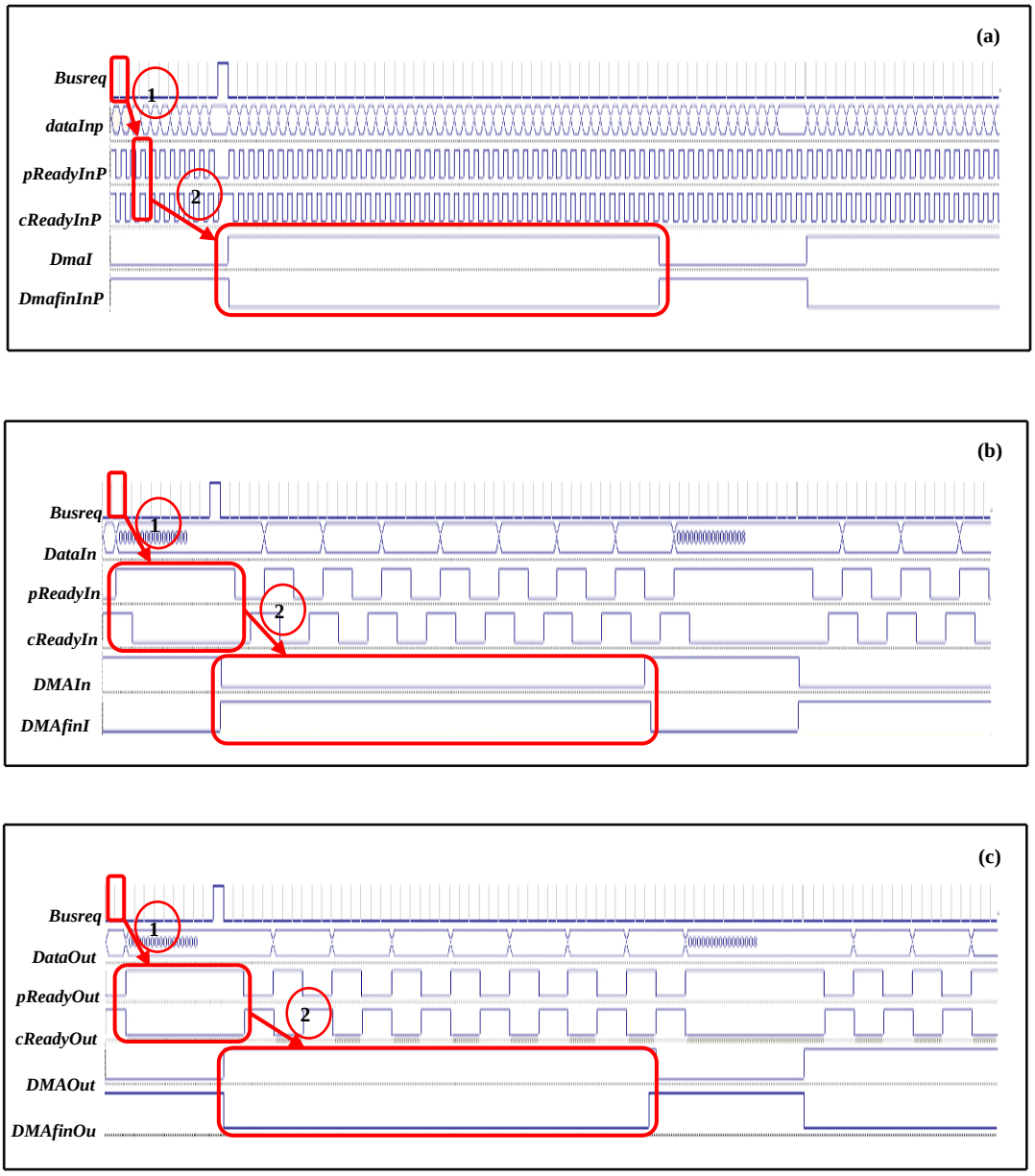

Figura 2.12 Diagrama de tiempo para las señales del módulo de *Chipset* (a) leer datos del procesador, (b) leer datos, y (c) escribir datos.

que el módulo de *Chipset* envía la señal *cReadyInP* al módulo de procesador para avisarle de que el módulo de *Chipset* está listo para recibir los datos. El módulo de *Chipset* sigue haciendo el movimiento de los datos y al mismo tiempo se comprueba el estado de la señal *DMAInP*, (2) en Figura 2.12(a) donde se indica el fin del movimiento de los datos entre el módulo de CPU y el módulo *Chipset*. En el caso de recibir datos del módulo de memoria, (1) en Figura 2.12(b), cuando el bus está listo para transferir datos, el módulo *Chipset* comprueba el estado del módulo de memoria mediante la señal *pReadyIn*, cuando el módulo de memoria está listo se activa la señal *cReadyIn* para

indicar que el módulo de *Chipset* está listo recibir datos hasta que desactiva la señal *DMAIn*, (2) en la Figura 2.12(b), donde este señal se indiaca el fin del movimiento de datos entre el módulo de *Chipset* y el módulo de memoria. Finalmente, en el caso de enviar datos al módulo de bus E/S, se comprueba el estado del bus mediante la señal *Busrequest* (1) en la Figura 2.12(c), si el módulo de bus E/S está listo para recibir datos se activa la señal *cReadyOut*, entonces el módulo *Chipset* activa la señal *pReadyOut* para indicar que el módulo el módulo de *Chipset* está listo para enviar datos al módulo de bus E/S, (2) en la Figura 2.12(c), al mismo tiempo se comprueba el estado de la señal *DMAfinout* para terminar la transferencia de los datos entre el módulo *Chipset* y el módulo de bus de E/S. Mediante retardos que se pueden fijar a través de las señales de control mediante *handshaking* se pueden controlar las transferencias de datos entre el *chipset* y los demás módulos (módulo de bus de E/S, módulo de memoria y módulo de CPU).

2.3.3 El módulo de memoria y el bus de memoria

En la Figura 2.13 (a) se muestra el módulo de bus de memoria que se usa para transferir los datos entre el módulo de memoria y el controlador de memoria en el módulo *Chipset*. En este módulo las señales *dmaIn* y *damfinIn* se usan para trasferir los datos, mientras que las señales de control *cReadyIn*, *cReadyOut*, *pReadyIn* y *pReadyOut* se utilizan para controlar las transferencias de datos entre la memoria y los demás módulos. Además, se observa en la misma figura un módulo que introduce un retardo de tiempo para controlar la velocidad del módulo de bus de memoria. En las Figuras 2.13 (b) y (c) se muestra el módulo de memoria para el emisor y el receptor, respectivamente. El módulo de memoria también puede incluir en el emisor un generador de paquetes sintético (un generador de paquetes de distintos tamaños se genera los paquetes) o bien utilizar un fichero de trazas reales. En el caso del receptor el módulo de memoria incluye un consumidor de datos como se muestra en las Figuras 2.13 (b) y (c). La señal *DMAon* se utiliza para controlar las transferencias de datos entre el módulo de memoria y los demás. Mediante el estado de la señal *DMAon* se pueden sincronizar las señales *DMAfin* y *DMA* para hacer la transferencia entre el módulo de memoria y otros módulos.

El módulo de memoria se comunica con el procesador central a través del módulo de bus de memoria. El módulo de retardo que se incluye en el módulo del bus de memoria permite cambiar la velocidad del dicho bus y sincronizar las transferencias de los datos entre los módulos interconectados, como se muestra en la Figura 2.13(a).

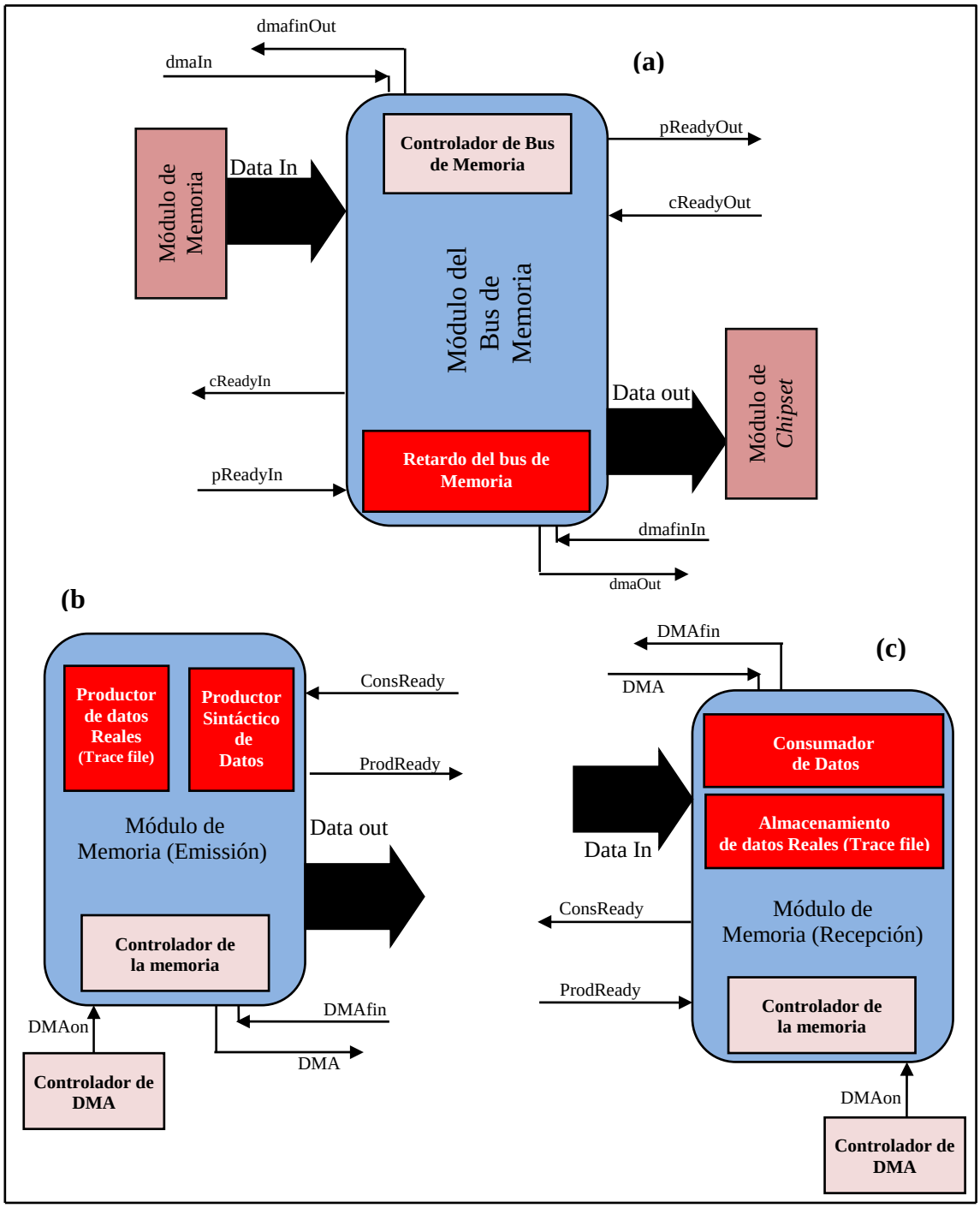

Figura 2.13 Módulos de memoria y bus de memoria (a) Bus de memoria (b) Memoria del emisor (c) Memoria del receptor

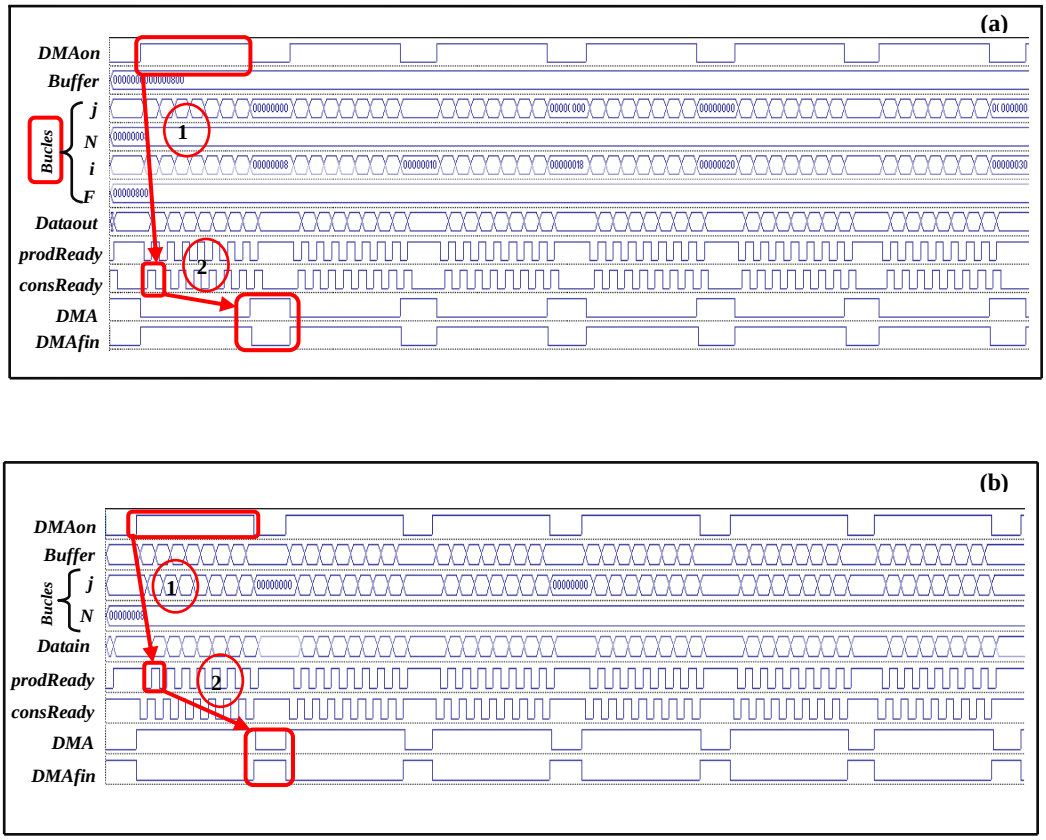

Figura 2.14 Diagrama de tiempo para las señales del módulo de memoria en el caso de (a) emisión, y (b) recepción.

En la Figura 2.14(a) se observa el intercambio de señales en una emisión y en una recepción. En la emisión se utilizan las señales *consReady* y *DMAon* para comprobar el estado del consumidor y empezar a generar datos (1) en la Figura 2.14(a). Entonces, se inicializa el DMA mediante la señal *DMA* para transferir los datos que están almacenados en el módulo *Memoria* al módulo *Chipset* (2) en la Figura 2.14(a). Mientras en el caso de la recepción (Figura 2.14 (b)), se utilizan las señales *prodReady* y *DMAon* para comprobar el estado de la memoria si puede recibir datos y guardarlos en el buffer (1), y luego mediante la señal *DMA* se hace la transferencia de los datos al módulo de memoria (2). Se puede utilizar el retardo que está ubicado en el módulo de memoria en el caso de emisión (*Bucle de retardo*) para generar datos con distintos anchas de banda como se muestra en la Figura 2.14(a).

2.3.4 El módulo de bus de E/S

Las Figuras 2.15(a) y 2.15(b) muestran los módulos del bus de E/S en el receptor y en el emisor de datos. El módulo de bus de E/S introduce un retardo de tiempo que permite cambiar la velocidad de transferencia de los paquetes a través de dicho bus.

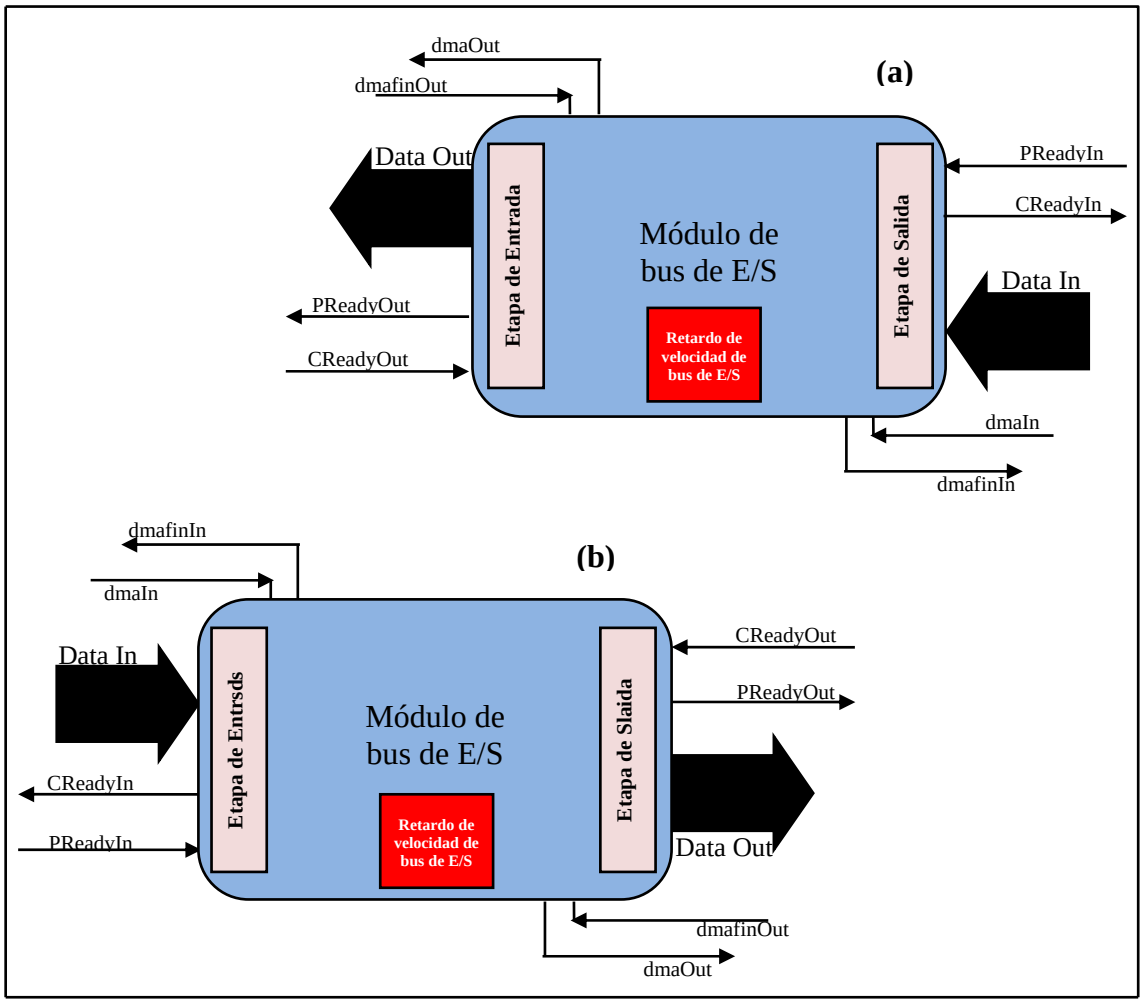

Figura 2.15 Módulos y señales de controles del módulo de bus de E/S (a) recepción (b) emisión

Las señales *dmaIn*, *damfinIn*, *dmaOut y dmafinOut* se usan para trasferir los datos, mientras que las señales de control *CReadyIn*, *CReadyOut*, *PReadyIn* y *PReadyOut* se utilizan para controlar las transferencias de datos entre el módulo del bus E/S y los demás módulos, como se muestra en la Figura 2.15. El Intercambio de señales en una recepción y en una emisión para transferir datos entre los módulos se utiliza las señales de control igual como se ha mostrado en las secciones anteriores.

2.3.5 El módulo NIC

El módulo NIC se implementa de dos formas según corresponda al emisor o al receptor, como se muestra en la Figura 2.16(a) y 2.16(b). En el primer caso, cuando la NIC está en el receptor, el módulo de la NIC es un consumidor de los datos de red y por lo tanto produce los datos que pasan a través del módulo al bus de E/S.

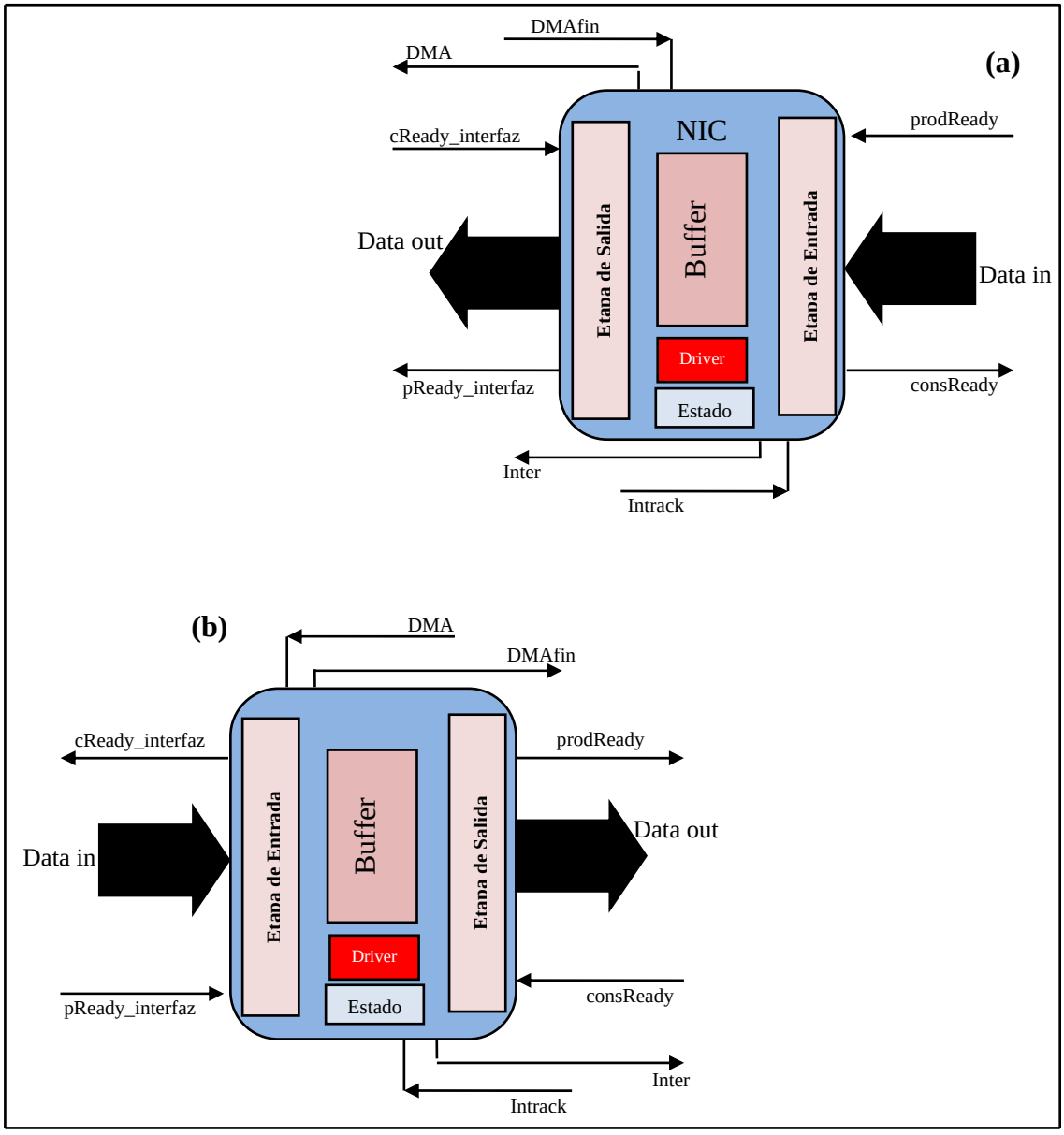

Figura 2.16 Módulos y señales de controles de NIC (a) recepción (b) emisión

La Figura 2.17 muestra los detalles del movimiento de las señales de control del módulo NIC para la emisión y la recepción de datos. En la parte de recepción, se muestra que

cuando llegan los paquetes a la NIC se guardan en su buffer hasta que se llena. Después se envía una señal de interrupción *Inter* al módulo de procesador y se espera hasta que llega la señal *Interack*. Entonces, la NIC empieza el procesamiento de los paquetes durante un tiempo igual al retardo del *driver*. Cuando el módulo NIC termina el procesamiento de los paquetes, se empieza a hacer la transferencia de los paquetes procesados por DMA, desde el buffer de la NIC hasta la memoria de usuario. Las señales de control *consReady*, *podReady*, c*Ready_Interfaz* y *pReady_Interfaz* se utilizan para controlar las transferencias de datos entre el módulo de la NIC y el módulo de la red (*Network*) por un lado, y el módulo del bus de E/S por otro, tal y como se muestra en la Figura 2.17. Se utiliza la señal *pReady_Interfaz* para indicar que el módulo de NIC está listo para producir los datos, mientras que la señal c*Ready_Interfaz* se usa para comprobar si el consumidor (bus de E/S) está listo para recibir los datos. El módulo NIC sigue con el movimiento de los datos, mientras al mismo tiempo se comprueba el estado de la señal *DMA* para terminar la transferencia de los datos al módulo de bus de E/S.

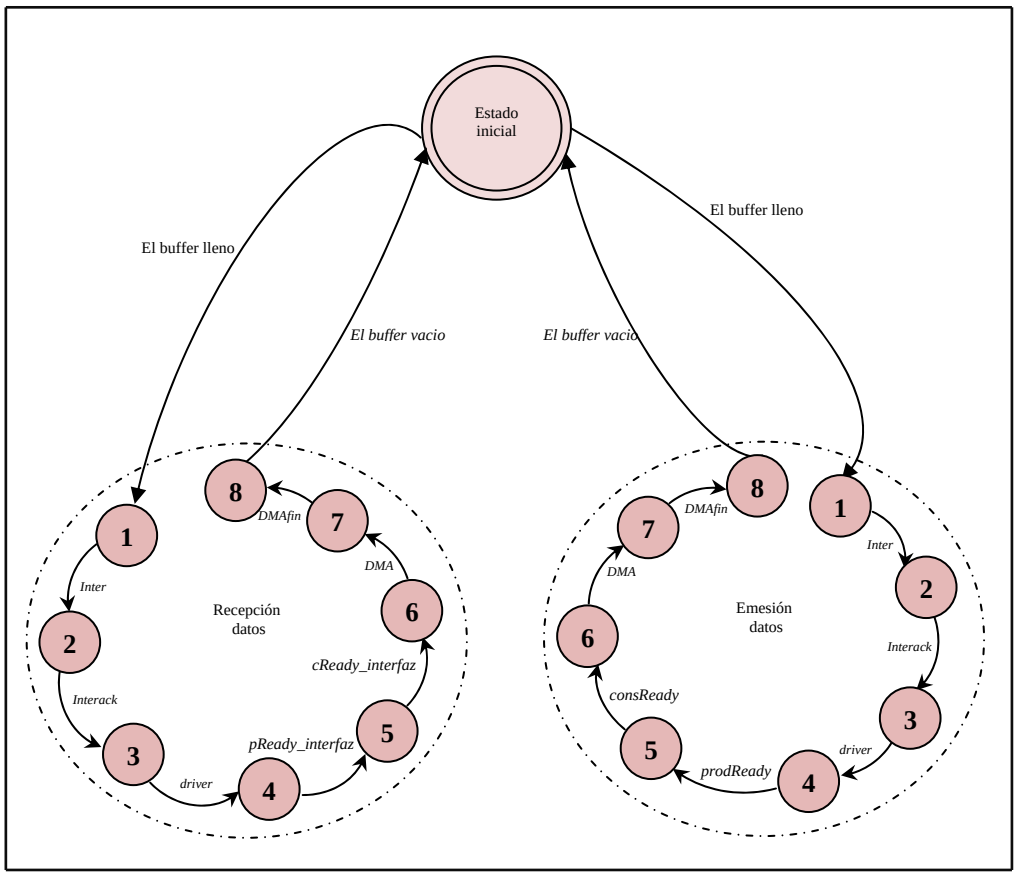

Figura 2.17 Diagrama de estado del movimiento de datos en el módulo *NIC*

En el caso de la emisión, el módulo de la NIC produce los datos que pasan a la red y consume los datos que provienen del bus de E/S. Cuando se llena el buffer de la NIC empieza el procesamiento de los paquetes mediante las señales de interrupción *Inter* y *Interack*. Se activa la señal *podReady* para indicar que el módulo de la NIC está listo para pasar los datos al módulo de red *Network*, y se comprueba el estado de la señal *consReady* para indicar que el bus de E/S está listo para recibir los datos.

El módulo NIC sigue transfiriendo los datos hasta que se activa la señal *DMAfin*, que indica el final del movimiento de los datos.

Los módulos NIC y CPU se implementan de forma diferente según se modele la *externalización* o no. En el caso de la *externalización* mediante la técnica *offloading* se utiliza un único módulo CPU en el camino de comunicación y el caso del *onloading* se utilizan dos módulos CPU, como se ha indicado en las secciones 2.2.2 y 2.2.3. En uno de ellos se ejecutan todas las tareas relacionadas con el procesamiento de comunicaciones y en el otro las aplicaciones y el resto de tareas del sistema operativo.

2.3.6 El módulo de red (módulo *Network*)

El módulo *Network* implementa el enlace y actúa como un productor de datos hacia el receptor, y como un consumidor hacia el emisor, tal y como se muestra en la Figura 2.18.

Cuando se simula todo el camino de comunicación se utiliza un retardo para modificar la velocidad del enlace de la red. El modelo HDL realizado permite controlar la velocidad a la que la red genera los paquetes, y variar los retardos del bus de E/S, de los accesos de memoria, y de la transferencia entre los distintos elementos de los módulos del simulador HDL.

Como se ha explicado en las secciones anteriores, las señales de control c*ReadyIn*, c*ReadyOut*, *pReadyIn* y *pReadyOut* se utilizan para controlar las transferencias de datos entre el módulo de la red y los demás módulos, como se muestra en la Figura 2.19. Por lo tanto, se puede cambiar el valor del retardo en el módulo de red fijando el número de iteración de bucle de retardo.

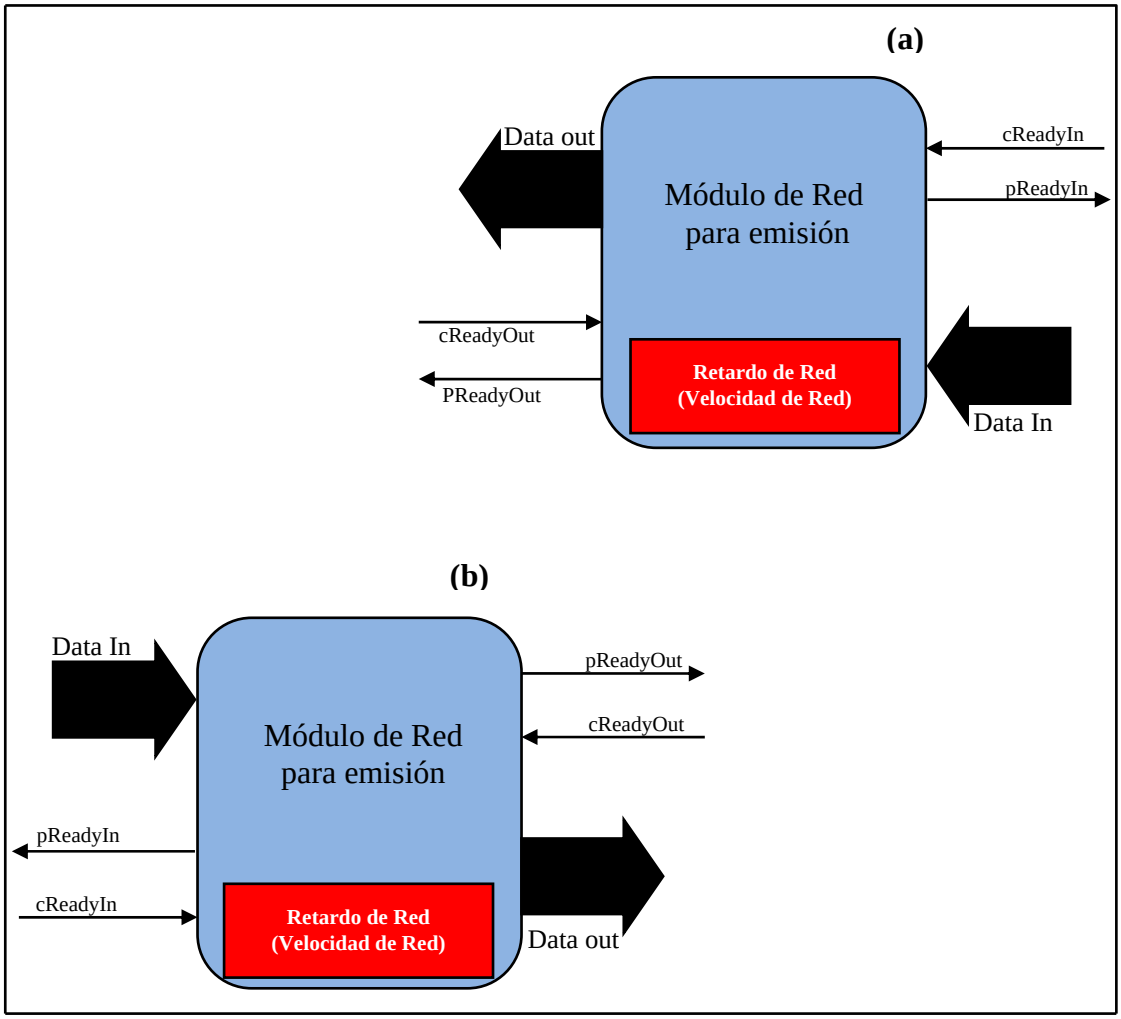

Figura 2.18 Módulos de red de (a) productor de datos (b) consumidor de datos

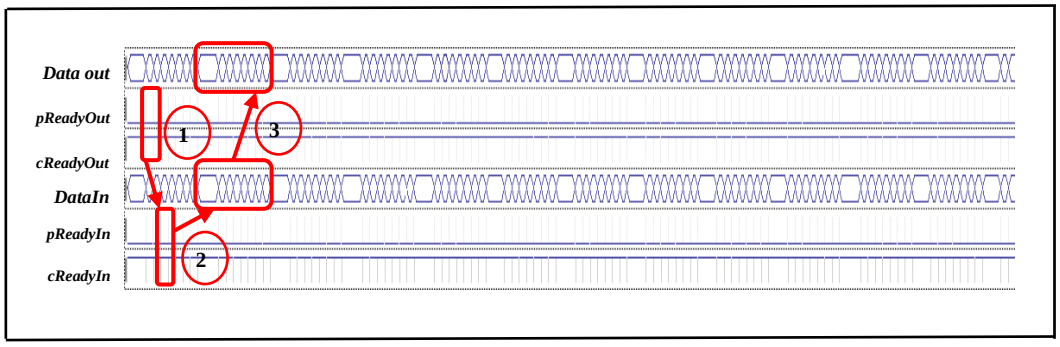

Figura 2.19 Diagrama de tiempo para los módulos de red

En la emisión se utilizan las señales *cReadyOut* y *pReadyOut* para comprobar el estado del consumidor (el módulo de la NIC). Si está listo, (1) el módulo de red compruebe el estado del productor mediante las señales *cReadyIn* y *pReadyIn* (2) en la Figura 2.19. Entonces, se inicia la transferencia de los datos que están almacenados en el buffer del módulo de la NIC del emisor al buffer del módulo de la NIC del receptor (3) en la Figura 2.19.

2.4 Conclusiones

En este capítulo se han descrito las características que deben tener los simuladores para constituir herramientas útiles en la evaluación de las arquitecturas de comunicación. Concretamente, se ha puesto de manifiesto que la simulación del sistema de comunicación nos permite no sólo simular el comportamiento del hardware del nodo computador, sino también incluir los efectos de sistema operativo como de las aplicaciones comerciales.

Se ha utilizado la simulación HDL porque permite descubrir con detalle las características del hardware del camino de comunicación y tomar decisiones en las primeras fases de diseño, con un esfuerzo y coste mucho menor que si se realizase en fases posteriores.

A partir de nuestros modelos de simulación HDL, podremos evaluar experimentalmente las distintas alternativas de diseño para la interfaz de red. Precisamente, entre las aportaciones del trabajo que se presenta en esta memoria se encuentra la construcción de los modelos de simulación adecuados que pongan de manifiesto el efecto de distintos elementos del camino de comunicación. Además, también podremos validar y justificar las características de modelos teóricos de prestaciones, como es el caso del modelo LAWS que consideramos aquí.

En el siguiente capítulo se evalúan los resultados obtenidos de las simulaciones con los modelos que se han elaborado y que se descrito en este capítulo.

Capítulo 3

EVALUACIÓN DE LOS RESULTADOS EXPERIMENTALES E IDENTIFICACIÓN DE OPORTUNIDADES DE MEJORA

En los capítulos anteriores se ha descrito detalladamente la *externalización* de protocolos de comunicación con las técnicas *offloading* [WES04, MOG03] y *onloading* [REG04a]. En este capítulo utilizamos los modelos HDL que se han desarrollado y que se han descrito en el capítulo anterior para evaluar las prestaciones del uso de la *externalización* en la interfaz de red. En las simulaciones realizadas se ha supuesto la *externalización* de todas las tareas relacionadas con el procesamiento de los protocolos. En el caso de utilizar *offloding*, el procesador central se encarga de transferir los datos desde la tarjeta de red hasta la memoria principal y viceversa. En el caso de *onloding*, se utilizan dos procesadores con acceso a memoria principal, uno para ejecutar las aplicaciones, y otro para procesar las tareas de comunicación. Las simulaciones se han realizado con dos tipos de trazas de comunicación, el primer tipo se obtiene mediante un generador de paquetes modelado en la memoria de emisor. El segundo tipo usa un fichero de trazas reales para verificar el modelo de simulación con un perfil de comunicación de una aplicación real

(*Ethernet at the Bellcore Morristown Research*) [INT10]. De los resultados experimentales obtenidos de las simulaciones, se pueden extraer conclusiones y propuestas de mejora para las arquitecturas de comunicación de los nodos.

En la sección 3.1 se validan los modelos HDL comparando los resultados con lo obtenido por un simulador de sistema completo como el SIMICS. Después, en la sección 3.2 se presentan los resultados de la *externalización* mediante *offloading*. En la sección 3.3 se muestra la evaluación del sistema de comunicación en el que la interfaz de red implemente la técnica de *onloading*. Tras analizar los resultados experimentales del *offloading* y del *onloading*, se observa que las mejoras alcanzadas por cada alternativa varían dependiendo del ancho de banda de la red si no es el cuello de botella y de las características del nodo, como se muestra en las secciones 3.2 y 3.3. En la sección 3.4 se proporciona la comparación entre los resultados de las dos alternativas. Posteriormente, la sección 3.4 se presenta el efecto de la cache en las dos técnicas. Además, en la sección 3.5 se propone una simulación HDL de una propuesta de sistema de comunicación que hibrida las técnicas de *offloading* y *onloading*. El modelo teórico LAWS [SHI03] se usa como referencia para analizar el comportamiento de las alternativas (*offloading* y *onloading*) según los parámetros del sistema de comunicación, como se ha explicado en los capítulos anteriores.

3.1 Validación del modelo HDL utilizado

Antes de utilizar la simulación de nuestros modelos HDL para evaluar las mejoras que las distintas técnicas de *externalización* pueden aportar, es necesario validar los resultados que se pueden obtener con esta metodología basada en modelos HDL. Para llevar a cabo esta validación hemos utilizado los resultados que se obtienen con el simulador de sistema completo SIMICS en el caso del sistema de base que no implementa ninguna de las técnicas de mejora consideradas. Mediante el uso de un simulador de sistema completo como SIMICS se consigue una descripción bastante realista del sistema que se desea evaluar dado que en este tipo de simuladores se tienen en cuenta la ejecución de las aplicaciones y del sistema operativo. Además, los resultados de SIMICS que se han utilizado para la validación de nuestros modelos HDL incluyen modelos temporales que permiten tener en cuenta los retardos de los buses de E/S, el sistema de memoria, y demás elementos del computador simulado.

Dado que los parámetros del modelo simulado se pueden controlar de forma relativamente sencilla, es posible asegurar que los modelos de simulación HDL y SIMICS comparados corresponden a máquinas similares, situación que es mucho más difícil de garantizar si se intentan reproducir todas las características de una máquina real, dado que estas características pueden no conocerse con total detalle en todos los casos .

En la Figura 3.1 se muestra la comparación entre los resultados obtenidos cuando no hay *externalización* mediante el simulador de sistema completo SIMICS [ORT08] y los resultados obtenidos cuando no hay *externalización* mediante la simulación HDL.

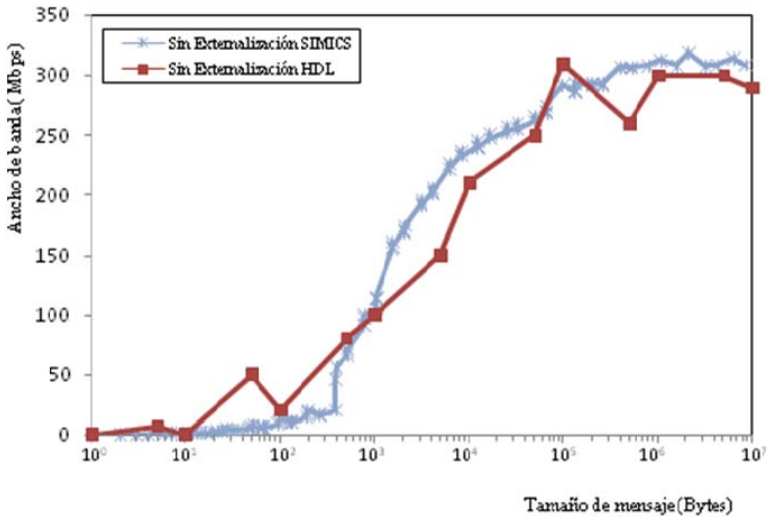

Figura 3.1 Comparación entre el ancho de banda obtenido cuando no hay *externalización* mediante el simulador SIMICS y el HDL para varios tamaños de mensajes.

Como se ha dicho, en el simulador SIMICS se tiene en cuenta la ejecución de las aplicaciones y del sistema operativo, por eso se utiliza como una referencia real para validar los resultados de nuestras modelos de simulación HDL, en las que la ejecución del sistema operativo y las aplicaciones se consideran a través de retardos en distintos puntos del camino de comunicación. La Figura 3.1 pone de manifiesto que los resultados de ancho de banda en función del tamaño de los mensajes obtenidos cuando no hay *externalización*, son relativamente similares para la simulación HDL y la simulación SIMICS.

En la Figura 3.2 se pone de manifiesto que los resultados obtenidos del ancho de banda con el *offloading* mediante la simulación HDL son algo mayores que los resultados obtenidos con el simulador SIMICS [ORT08]. Las causas de esta

diferencia se pueden explicar analizando el cambio en las curvas obtenidas para la simulación HDL al cambiar algunos parámetros del sistema de comunicación (los retardos de acceso de memoria, el valor del parámetro β, etc.).

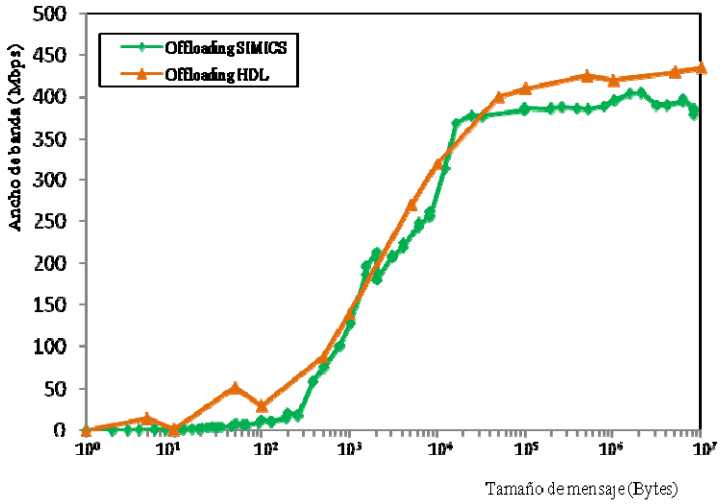

Figura 3.2 Comparación entre el ancho de banda obtenido con el *offloading* mediante el simulador SIMICS y el HDL para varios tamaños de mensajes.

En el caso de la comparación entre las simulaciones HDL y SIMICS para los anchos de banda con *offloading*, y utilizando α=1, es posible un mejor ajuste de los resultados de la simulación HDL con los de SIMICS si se aumenta el parámetro β cambiando el tiempo de acceso a memoria del procesador. Como se ha indicado en la Sección 1.6.1, este parámetro β está en gran parte relacionado con el acceso de memoria, y cuanto más se aumente dicho parámetro β más ajustarán los resultados que hemos obtenido con los del simulador SIMICS. Por tanto, la razón de la discrepancia cuantitativa observada en este caso, puede deberse al modelado que se hace del acceso a memoria.

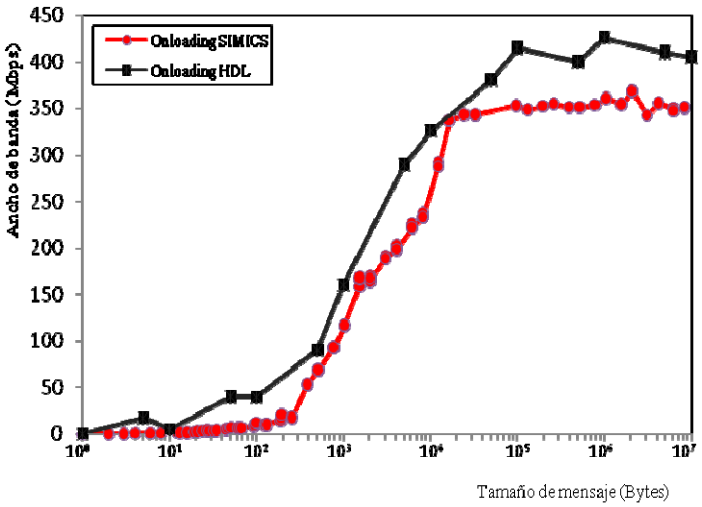

Figura 3.3 Comparación entre el ancho de banda con *onloading* mediante el simulador SIMICS y HDL para varios tamaños de mensajes.

La Figura 3.3 proporciona la comparación entre los anchos de banda obtenidos con la técnica de *onloading* mediante el simulador de sistema completo SIMICS [ORT08] y los obtenidos con nuestros modelos y la simulación HDL.

El comportamiento obtenido para el ancho de banda con *onloading* mediante nuestras simulaciones HDL es cualitativamente similar a los resultados generados por SIMICS. Los anchos de banda con *onloading* que se han medido con SIMICS para mensajes de pequeño tamaño son casi iguales que los obtenidos con las simulaciones HDL. No obstante, como pone de manifiesto la Figura 3.3, cuando se aumenta el tamaño de los mensajes se observan diferencias entre los anchos de banda medidos por cada uno de los simuladores. El sentido de estas diferencias es similar al que se ha experimentado para el caso del *offloading*, como muestra la Figura 3.2, y sus causas son básicamente las mismas.

Se puede concluir que, teniendo en cuenta el comportamiento similar de las curvas de ancho de banda en función del tamaño de los paquetes para todos los casos y los valores cuantitativamente similares que se han obtenido para la alternativa sin *externalización*, y para mensajes de pequeño tamaño en todos los casos, se puede considerar que el modelo HDL nos permitirá extraer consecuencias válidas de los resultados de simulación para las distintas alternativas de mejora de la interfaz de red. En las secciones que siguen se describen y analizan dichos resultados.

3.2 Evaluación y análisis de las prestaciones mediante *offloading*

En esta sección se van a mostrar los resultados experimentales obtenidos para la técnica de *offloading* a partir de la simulación de los modelos HDL. Además, se evalúan las prestaciones del camino de comunicación mediante el uso del modelo teórico LAWS con distintas condiciones de carga del sistema. Se utiliza el modelo de simulación HDL para *offloading* descrito en la sección 2.3.2, utilizando un generador de paquetes de 64 bits para evaluar las prestaciones del camino de comunicación. Como se ha indicado, los distintos resultados obtenidos no sólo permiten estimar las prestaciones del camino de comunicación, sino también analizar los parámetros que influyen en el rendimiento de dicha técnica.

Después de evaluar las prestaciones, y según los parámetros del sistema, se lleva a cabo un análisis de los resultados de simulación comparándolos con los

estimados por el modelo LAWS. Más concretamente, se realizan distintas simulaciones para determinar el efecto en las prestaciones del camino de comunicación cuando varían los parámetros del modelo LAWS. En las primeras simulaciones se ha modelado el efecto de la memoria cache a través del tiempo de latencia para el acceso a memoria, es decir, utilizando un retardo promedio para modelar el efecto del acceso a los datos. Todas estas simulaciones se corresponden, por tanto, con modelos HDL que no incluyen una descripción detallada de la cache. Posteriormente, se ha modelado el funcionamiento de una cache de dos niveles, L1 y L2, y se analiza el efecto de la misma en las prestaciones del camino de comunicación completo, al estilo de lo que se hace en [NAH97].

3.2.1 El efecto de los parámetros *αβ* en la mejora del ancho de banda

Uno de los principales objetivos del *offloading* en el procesamiento de los protocolos de comunicación es reducir la sobrecarga en la CPU principal, que se encargaría solo de ejecutar las aplicaciones y parte del sistema operativo. Cuando no se implementa *externalización* (*Sin-offloading* en Figura 3.4), la CPU central se encarga de todas las tareas, por lo que su carga de trabajo aumenta considerablemente. Por lo tanto, la disminución de carga permite a la CPU ejecutar otras tareas de la aplicación y esto puede traducirse en una mejora del ancho de banda aprovechado por la aplicación, tal y como se ilustra en la Figura 3.4.

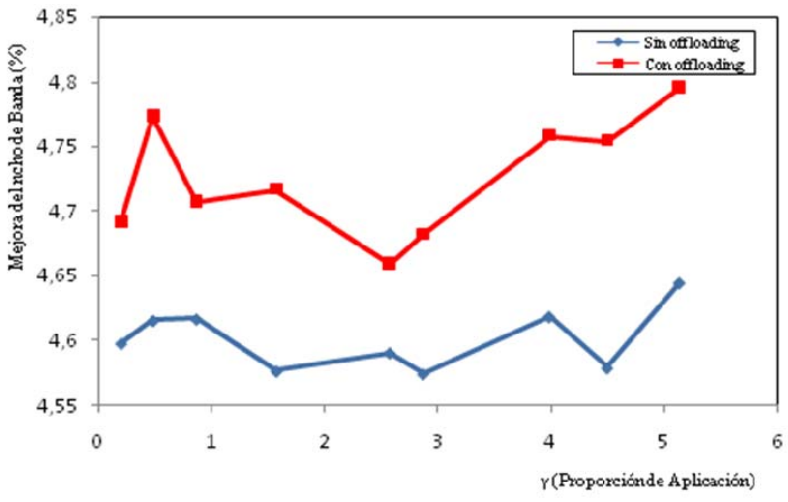

Figura 3.4 Comparación entre el ancho de banda cuando hay *offloading* y no hay (*αβ*=1.01, 64 bits).

En la Figura 3.4 pone de manifiesto el efecto del *offloading* en las prestaciones a medida que la CPU central consume más ciclos para procesar el protocolo de comunicación. Es decir, a medida que disminuye el parámetro γ, que, como se ha visto, refleja la proporción entre la carga de la aplicación y la de comunicación.

En la Figura 3.5 se muestra el efecto del *offloading* cuando se cambia el valor de $\alpha\beta$ (la velocidad del procesador de la NIC respecto a la velocidad del procesador central multiplicada por el factor estructural β). Como se ha visto en el Capítulo 1, estos dos parámetros son importantes para caracterizar el comportamiento frente al *offloading* de la interfaz de red que se está analizando. Concretamente, la Figura 3.5 muestra la mejora del ancho de banda obtenida cuando se aumenta el valor de $\alpha\beta$ para los valores 0.3, 0.45 y 1.01 (incremento en $\alpha\beta$). Como se puede ver, se produce una reducción de la mejora máxima que se alcanza y ésta prácticamente se mantiene a medida que crece γ. Es decir, que la técnica de *offloading* mejoraría las prestaciones del sistema de comunicación cuando la velocidad del procesador de la NIC aumenta en relación con la de la CPU central (α disminuye). Esto se puede afirmar siempre y cuando la implementación del *offloading* se haga de forma eficiente en todos los casos, de manera que el valor de β se mantenga constante para las distintas configuraciones de las NICs.

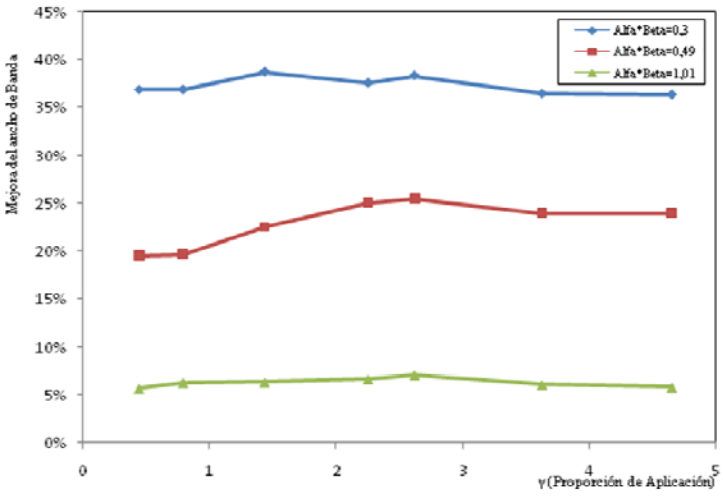

Figura 3.5 Mejora del ancho de banda con *offloading* para varios valores de $\alpha\beta$ y enlace de 1Gbps

En la Figura 3.6 se muestra la mejora del ancho de banda cuando se cambia la velocidad de la red de 1Gbps a 10Gbps con las mismas condiciones de la simulación

cuyos resultados se muestran en la Figura 3.5. En este caso se produce más ganancia en la mejora del ancho de banda cuando disminuye el valor de $\alpha\beta$.

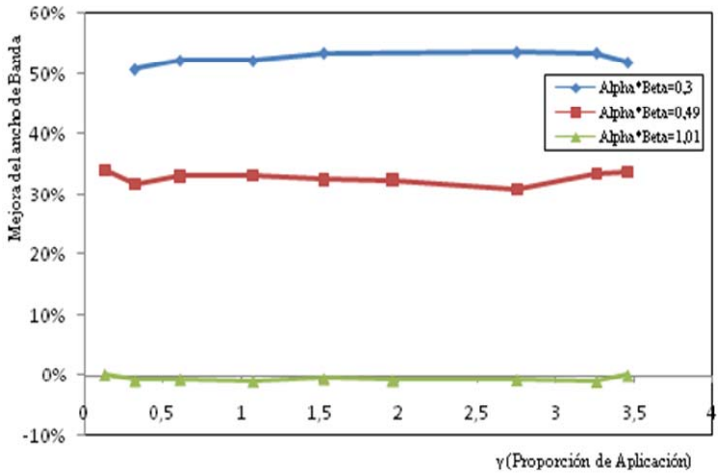

Figura 3.6 Mejora del ancho de banda con *offloading* para varios valores de $\alpha\beta$ y 10Gbps

La Figura 3.7 se muestra la comparación de la mejora del ancho de banda de la Figura 2.2 (1Gbps) y la Figura 2.3 (10Gbps) con las mismas condiciones de la simulación.

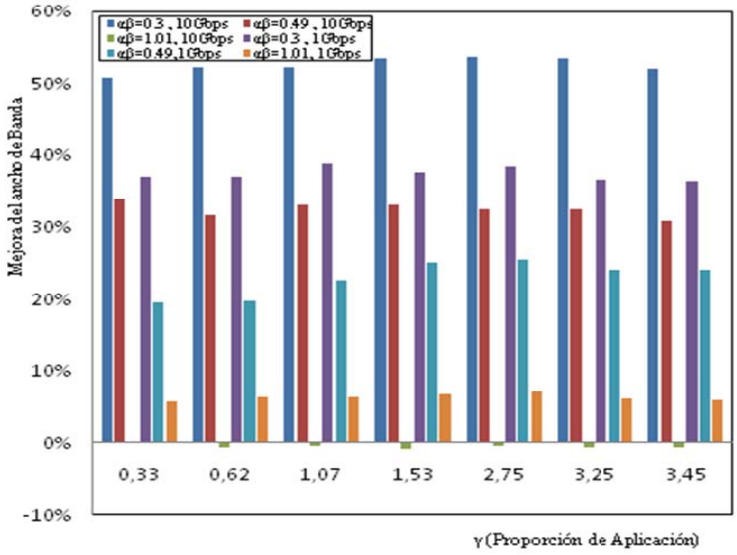

Figura 3.7 Comparación de la mejora del ancho de banda para 1Gbps y 10Gbps y para varios valores de $\alpha\beta$

No obstante, en el caso de simulación de trazas correspondiente a aplicaciones reales (*Ethernet at the Bellcore Morristown Research*) [INT10], el fichero de trazas que se

ha usado contiene paquetes Ethernet de distintos tamaños (entre 64 y 1,518 bytes) como se muestra en la Figura 3.8.

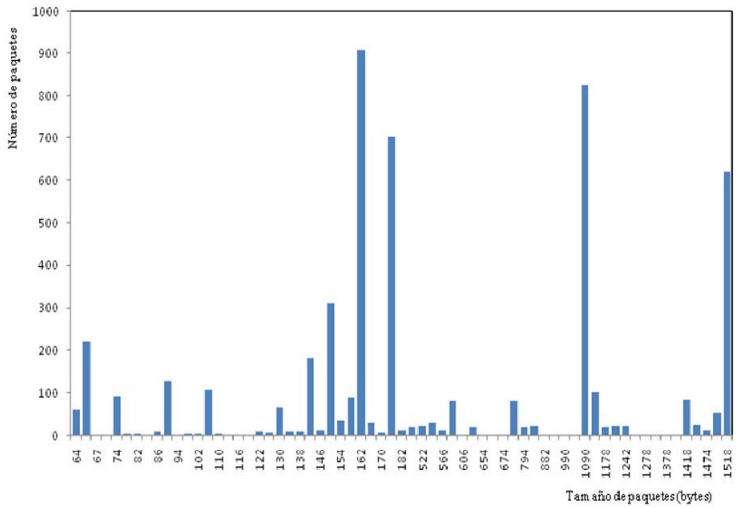

Figura 3.8 El número de los paquetes en el fichero de trazas (*Ethernet at the Bellcore Morristown Research*) **[INT10].**

La Figura 3.9 muestra la mejora de ancho de banda cuando se usa el fichero de trazas [INT10] con varios valores de $\alpha\beta$.

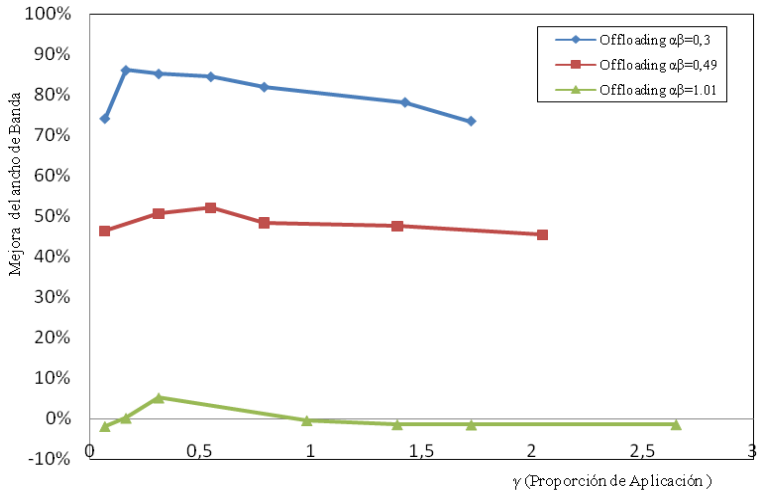

Figura 3.9 Comparación de la mejora del ancho de banda para un fichero de trazas [INT10] y para varios valores de $\alpha\beta$

La información que proporciona el modelo LAWS corresponde a una mejora de los valores de los anchos de banda pico que se pueden alcanzar con y sin *offloading*. El modelo considera que el camino de comunicación está segmentado entre el procesador, el NIC, y la red, como se ha mostrado en el capítulo 2. Obviamente, estos elementos del camino de comunicación comparten recursos. En concreto, el procesador genera accesos a memoria, y se pueden producir colisiones con las

transferencias de DMA desde el NIC. Es por ello que, si bien se obtienen conclusiones importantes desde el punto de vista cualitativo, las diferencias cuantitativas pueden ser importantes en relación a los que predice LAWS.

La Figura 3.10 muestra el ancho de banda para diferentes tamaños de mensaje cuando no hay *externalización* en el sistema de comunicación y cuando se externaliza mediante la técnica *offloading*.

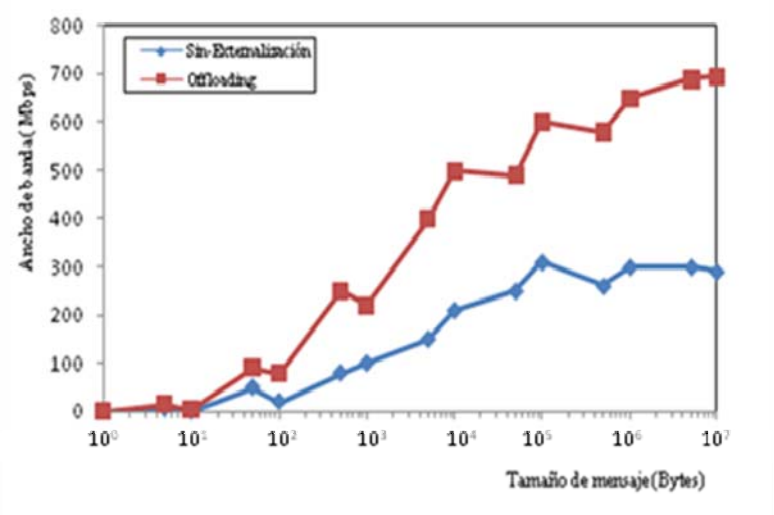

Figura 3.10 El ancho de banda con *offloading* y sin *externalización* para varios tamaños de mensajes ($\alpha\beta$=1.01).

A partir de la Figura 3.10, se puede observar que la técnica de *offloading* afecta considerablemente al ancho de banda obtenido. Como hemos explicado antes, cuando el procesador de la NIC es lento, la *externalización* del protocolo de comunicación incluso puede empeorar las prestaciones del sistema de comunicación.

3.2.2 Efecto del parámetro $\alpha\beta$ en la latencia de comunicación

En la Figura 3.11se muestra la latencia medida para diferentes valores de γ con varios valores de $\alpha\beta$, en el caso del sistema sin *externalización* (*No-offloading* en la Figura 3.10), y para el sistema externalizado con *offloading*. Las latencias son menores para el *offloading* que cuando no se utiliza *externalización*.

De la Figura 3.11 se puede concluir que el efecto de la *externalización* mediante *offloading* disminuye la latencia porque todo el procesamiento del protocolo de comunicación se realiza en otro procesador diferente a la CPU, ubicado en la tarjeta de red y por lo tanto próximo a la red. De esta manera, se pueden evitar

transferencias a través del bus de E/S del sistema de comunicación de ciertos paquetes de control del enlace.

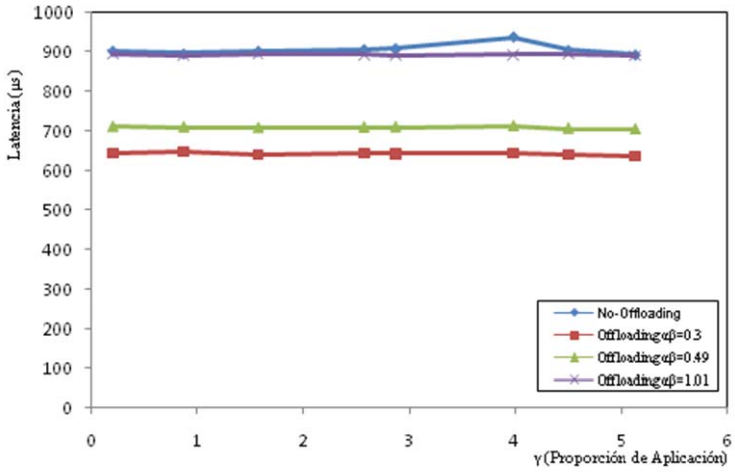

Figura 3.11 Latencias para *offloading* y sin *externalización* con varios valores de *αβ*.

La Figura 3.12 muestra la latencia del sistema de comunicación para varios tamaños de mensaje. Como se puede ver, las latencias en el caso de la *externalización* mediante *offloading* son menores que las latencias sin *externalización*. Lógicamente, tanto en el caso de *offloading* como en el de no usar *externalización*, el incremento en el tamaño del mensaje aumenta las latencias, entre otros sitios, en el bus de E/S, por lo tanto, aumenta la latencia del sistema de comunicación.

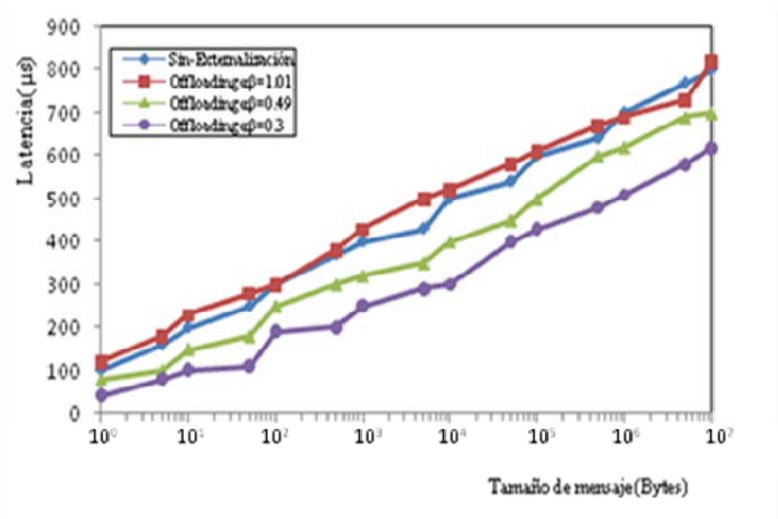

Figura 3.12 Comparación de latencias para varios tamaños de mensaje con y sin *externalización* (mediante *offloading*).

Además, en la Figura 3.13 se puede ver el efecto de la latencia de memoria en la mejora del ancho de banda. La mejora en las prestaciones del camino de comunicación es mayor cuando la latencia de memoria principal es de 500 ciclos. Cuando la latencia de memoria principal es menor (5 ciclos en nuestros experimentos) se reduce la mejora de prestaciones que aporta el *offloading*, como se muestra en la Figura 3.13. Es decir, la mejora es mayor con la técnica *offloading* para memorias más lentas.

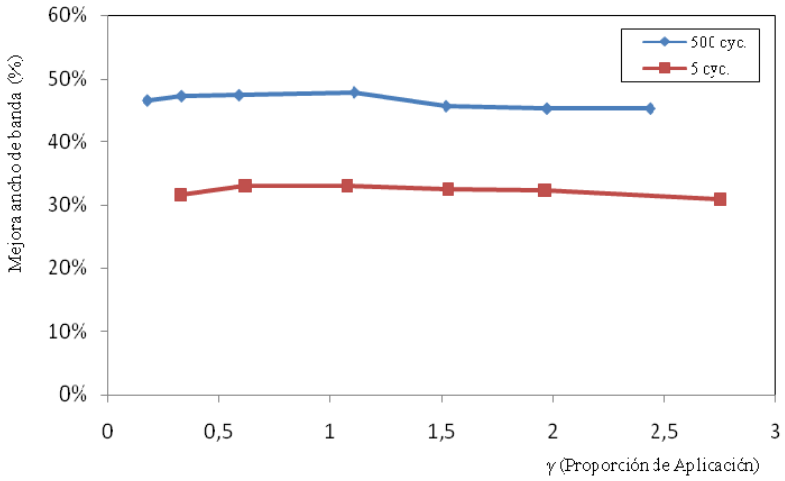

Figura 3.13 Comparación entre la mejora del ancho de banda para varios velocidades de la memoria principal.

3.3 Evaluación y análisis de las prestaciones mediante *onloading*

En esta sección se muestran y comentan los resultados de las simulaciones correspondientes al uso de *externalización* mediante *onloading*. A través de estos resultados se analizan las prestaciones del camino de comunicación mediante simulaciones en las que se usa un procesador central *multinúcleo* entre cuyos núcleos se distribuye el procesamiento de la carga de trabajo incluyendo el software de comunicación.

3.3.1 El efecto en la mejora del ancho de banda

La Figura 3.14 muestra el caso en el que un núcleo de la CPU se encarga de procesar la carga de comunicación. Se trata de la alternativa de *externalización* mediante *onloading*. En la Figura 3.14 el tamaño de los paquetes es de 64 bits.

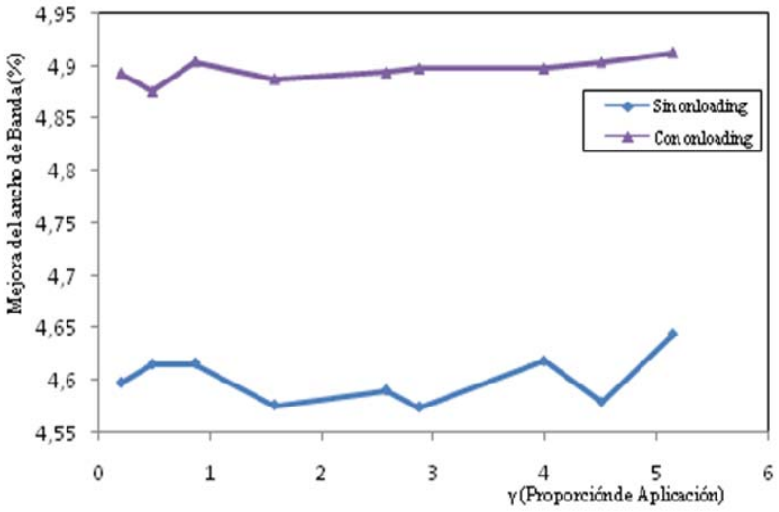

Figura 3.14 Comparación entre el ancho de banda cuando hay *onloading* y no hay ($\alpha\beta$=1.01).

Cuando se externalizan los protocolos de comunicación mediante *onloading*, el núcleo que ejecuta la aplicación dispone de más ciclos y se reduce el tiempo dedicado al procesamiento de interrupciones por parte del núcleo dedicado a la aplicación.

Como se ha dicho, la Figura 3.14 muestra la mejora en las prestaciones del camino de comunicación que se obtiene con la *externalización* mediante *onloading*. En la *externalización* mediante *onloading* no se puede cambiar el valor de $\alpha\beta$ por dos razones: (1) el parámetro α depende de la velocidad de los procesadores y, dado que hemos considerado que todos los núcleos del procesador tienen la misma velocidad, el parámetro α siempre se ha considerado igual a 1; (2) el parámetro β está relacionado fundamentalmente con el retardo de la tarjeta de red, en este caso, el *driver* se ejecuta en un núcleo del procesador central, y por lo tanto, el parámetro β tendrá el mismo en todos los experimentos.

La Figura 3.15 muestra el ancho de banda que se puede alcanzar mediante la *externalización* con *onloading* cuando se cambia el tamaño del mensaje.

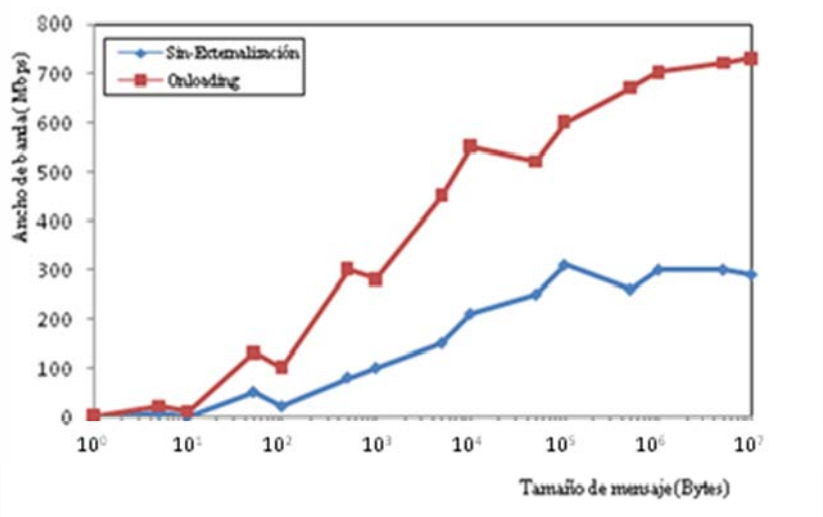

Figura 3.15 Ancho de banda con *onloading* y sin él para varios tamaños de mensajes.

3.3.2 El efecto de la latencia de comunicación

En la Figura 3.16 se muestra el efecto de la latencia en el acceso a memoria cuando se externalizan los protocolos de comunicación mediante *onloading*. Se trata de un experimento similar al que se ha realizado para la *externalización* mediante *offloading*.

En la Figura 3.16 se pone de manifiesto que la latencia de la *externalización* mediante *onloading* es menor que cuando no hay *externalización*. La reducción en la latencia se puede atribuir a que todas las interrupciones generadas por la tarjeta de la red se atiendan por un núcleo de la CPU central dedicado específicamente al procesamiento de los protocolos de comunicación.

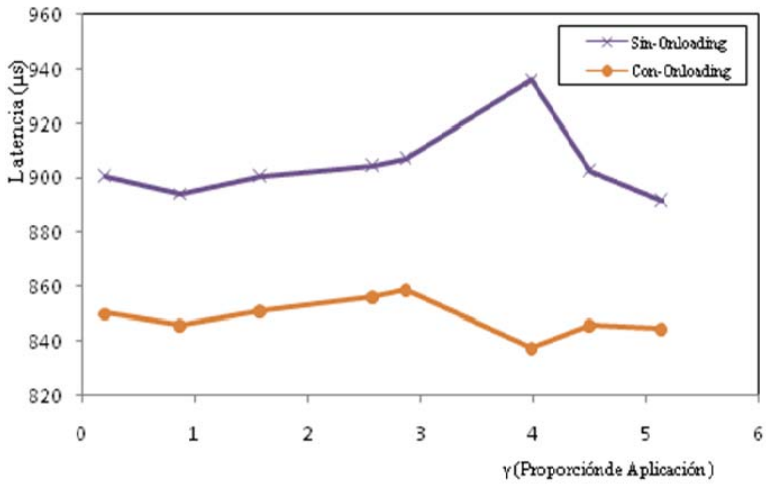

Figura 3.16 Comparación entre la latencia cuando hay *onloading* y no hay.

Además, hay que recordar que en el caso del *onloading*, los núcleos de la CPU que ejecutan las aplicaciones y procesan los protocolos de comunicación comparten recursos, como por ejemplo, los buses. La sobrecarga asociada a los paquetes pequeños y las interrupciones generadas por la llegada de dichos paquetes a la tarjeta de red disminuyen la mejora en la latencia.

En la Figura 3.17 se puede ver la latencia de la técnica de *onloading* en relación con el caso en el que no hay *externalización*, cuando se cambia el tamaño del mensaje. La menor latencia de la *externalización* mediante *onloading* se debe a que las interrupciones generadas por la llegada de los paquetes a la tarjeta de red se atienden en el segundo núcleo (CPU_1) de la CPU. En el caso de los paquetes pequeños, las latencias no disminuyen tanto porque la sobrecarga del sistema de comunicación es mayor.

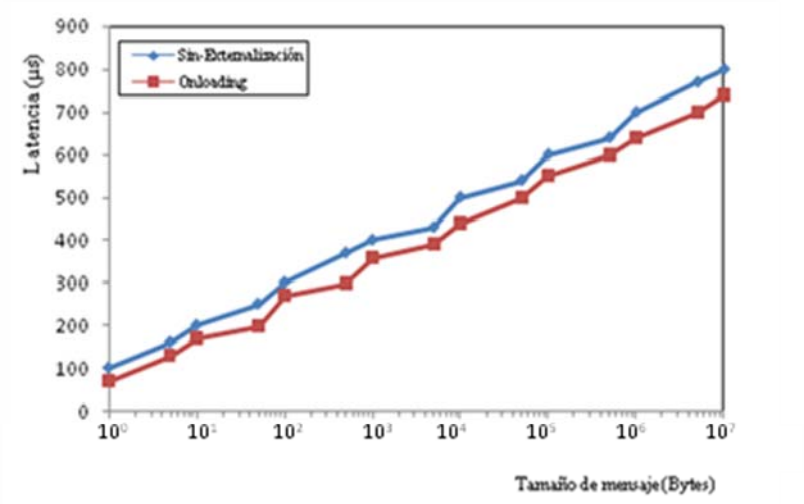

Figura 3.17 Latencias para *onloading* ($\alpha\beta$=1.01) y sin él con varios tamaños de mensaje.

Como se ha mostrado anteriormente en la sección 3.2.2, la latencia en el caso de la *externalización* mediante *offloading* es menor que en el *onloading*. En la siguiente sección se muestra la comparación detallada entre las latencias de comunicación observada en cada técnica.

3.4 Comparación entre la mejora del *offloading* y el *onloading*

En esta sección se comparan las técnicas de *externalización* mediante *offloading* y *onloding*, usando los resultados de los experimentos realizados mediante los modelos HDL que implementen cada técnica.

3.4.1 Comparación de las mejoras en el ancho de banda

Como se ha mostrado en las secciones anteriores, la Figura 3.18 muestra la mejora de prestación del camino de comunicación cuando se utilizan las técnicas de *externalización* mediante *offloading*, *onloading,*y sin *externalización* (con un tamaño de los paquetes de 64 bits). El ancho de banda alcanzado mediante la técnica de *externalización* con *onloading* es mayor que el obtenido con *offloading*, y sin *externalización*.

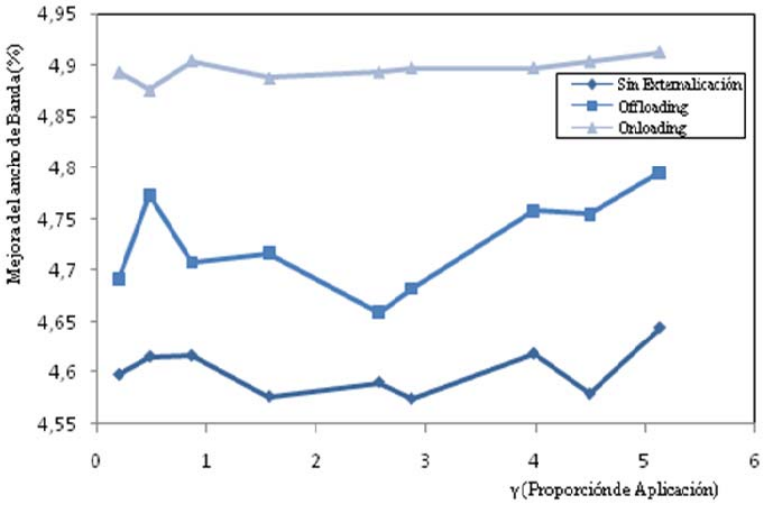

Figura 3.18 Comparación entre el ancho de banda cuando hay *offloading*, *onloading* y sin *externalización* ($\alpha\beta$=1.01).

En el caso de usar *onloading* (sección 2.3.3), el primer núcleo (CPU_0) del procesador multinúcleo se dedica a las aplicaciones y el sistema operativo, mientras el otro núcleo (CPU_1) se encarga de ejecutar el driver de la NIC e implementar el procesamiento del protocolo de comunicación. Por lo tanto, se reducirá el uso del primer núcleo liberándose más ciclos del mismo. En el caso de *offloading*, el *driver* se ejecuta en la misma CPU (en el mismo núcleo), por lo que aumenta la sobrecarga y se

utilizarán más ciclos de la CPU dedicas a las aplicaciones que con respecto al *onloading*. Obviamente, las interrupciones que llegan a la CPU_0 en el caso del *onloading* son pocas porque quien se encarga de hacer el procesamiento de los protocolos de comunicación es la CPU_1. Por lo tanto, son más los ciclos de CPU_0 que se pueden aprovechar para otras aplicaciones. Tanto en el caso de utilizar *offloading* y como en el de no usar ningún tipo de *externalización*, llegan muchas interrupciones a la CPU_0. Esta cantidad de interrupciones disminuye cuando aumenta el tamaño del mensaje.

La Figura 3.19 se muestra la comparación entre la mejora de ancho de banda mediante el *offloading* y el *onloading* cuando se usa el fichero de trazas (Figura 3.8), en el caso de simulación de trazas correspondiente a aplicaciones real (*Ethernet at the Bellcore Morristown Research*) [INT10].

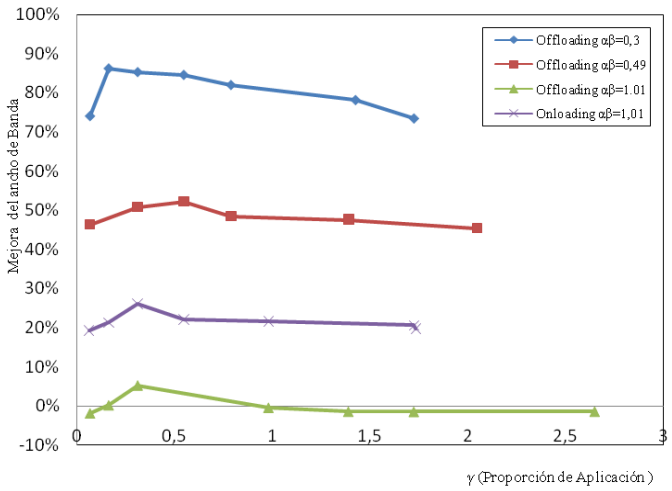

Figura 3.19 Comparación de la mejora del ancho de banda del *offloading* y *onloading* para un fichero de trazas [INT10] y para varios valores de *αβ*

La Figura 3.20 muestra el ancho de banda alcanzado mediante las técnicas de *offloading*, *onloading* y en el caso de no utilizar *externalización*, cuando cambia el tamaño del mensaje. Se puede ver que hay diferencias relevantes en el ancho de banda para mensajes grandes e intermedios. Estas diferencias se deben al efecto asociado a la distinta forma de gestionar las interrupciones que tiene el sistema de comunicación en cada alternativa.

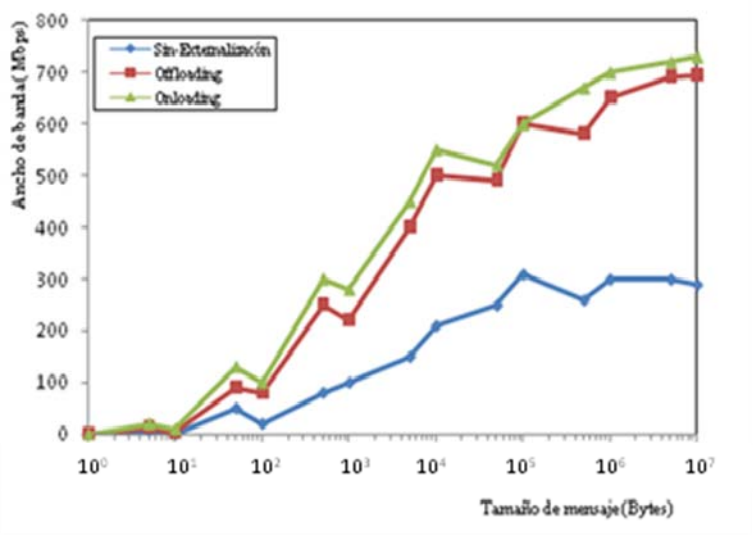

Figura 3.20 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* con varios tamaños de mensajes y $\alpha\beta$=1.01.

3.4.2 Comparación entre la mejora de la latencia de comunicación

En la Figura 3.21 se muestra la latencia para varios valores de la proporción de carga entre aplicación y comunicación (γ), en el caso del *offloading*, *onloading*, y cuando no hay *externalización*. La Figura 3.21 muestra que la latencia de la *externalización* mediante *offloading* es menor que la *externalización* mediante *onloading* y cuando no hay *externalización*. La menor latencia alcanzada con la técnica del *offloading* se debe a la *externalización* del protocolo de comunicación a la tarjeta de red (NIC), que evita transferencias a través del bus de E/S.

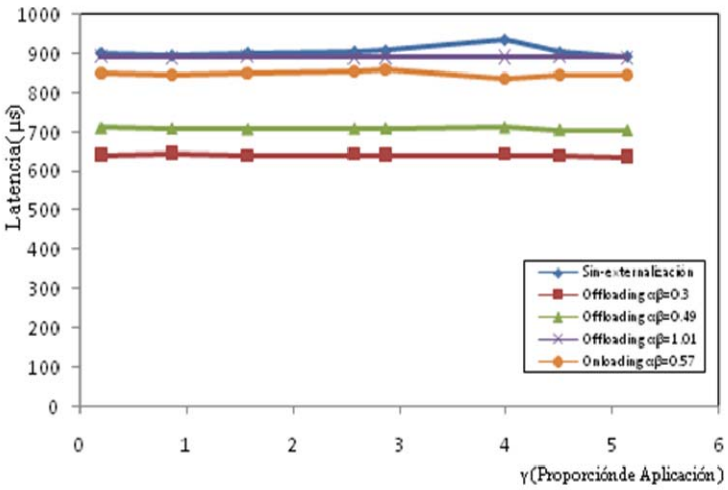

Figura 3.21 Comparación entre la latencia del *offloading*, *onloading* y sin *externalización*.

Las latencias observadas son muy similares para el caso de no utilizar *externalización* y cuando se utiliza la técnica de *onloading*. De los resultados experimentales obtenidos se puede concluir que la técnica de *offloading* proporciona mejores valores para la latencia que la técnica de *onloading*.

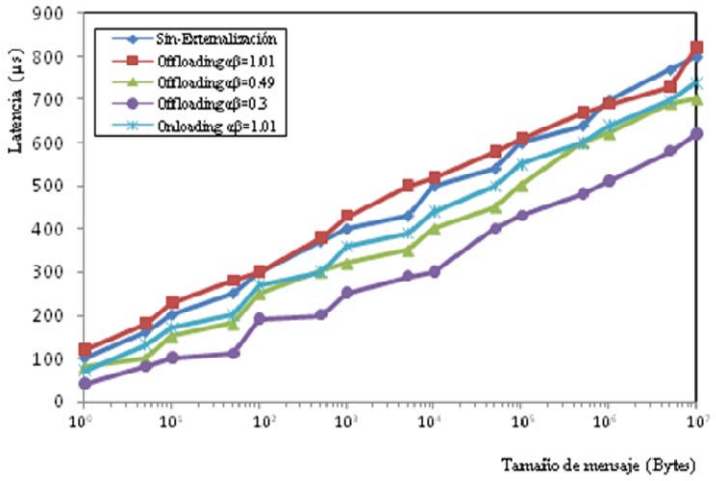

Figura 3.22 Comparación entre la latencia observada con *offloading*, *onloading* y sin *externalización* para varios tamaños de mensajes ($\alpha\beta$=1.01).

La Figura 3.22 muestra la latencia para diferentes tamaños de mensajes cuando se externaliza el sistema mediante la técnica de *offloading*, *onloading*, y para el sistema sin *externalización*. Se puede ver en la figura que las latencias observadas con la técnica de *onloading* son menores que las obtenidas con *offloading* para un valor de $\alpha\beta$ igual a 1. En caso de que el valor de $\alpha\beta$ sea menor que 1 (como corresponde a algunas casos que se muestran en la Figura 3.21), se tiene que las latencias son menores para el *offloading* que para el *onloading*, porque la *externalización* del protocolo de comunicación en la tarjeta de red puede reducir el uso del bus de la E/S, como se ha explicado antes.

3.5 El efecto del bus de E/S en las técnicas de *offloading* y *onloading*

Si se considera que la tarjeta de red está conectada directamente al *Chipset* a través de un bus de E/S sin retardo y sin colisiones (bus de E/S ideal), como se mostrado en la Sección 2.1. En la Figura 3.23 se muestra la diferencia entre el ancho de banda obtenida mediante la simulaciones del *offloading*, *onloading* y sin *externalización*

cuando no hay bus de E/S (el caso de las tarjetas de red de altas prestaciones) y cuando hay bus de E/S (el caso más común).

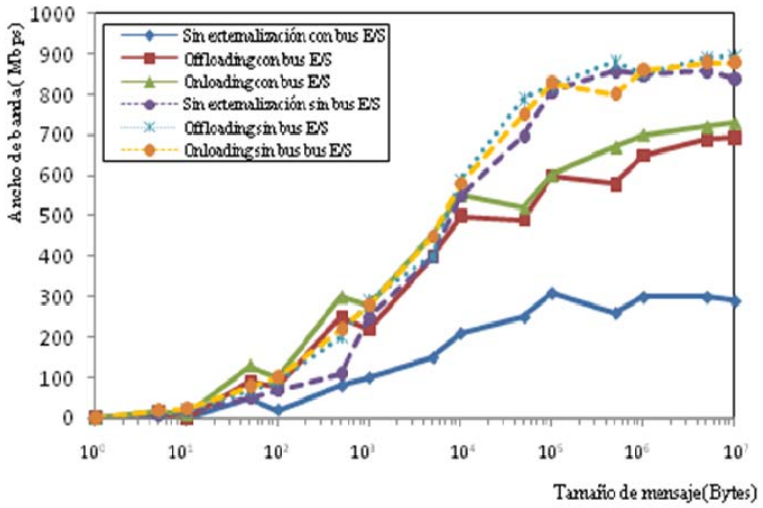

Figura 3.23 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* cuando hay bus de E/S y no, para varios tamaños de mensajes y $\alpha\beta$=1.01.

Se puede ver en la Figura 3.23, que cuando no hay bus de E/S, hay diferencia entre los resultados obtenidos mediante el *offloading*, *onlonding* y sin *externalización* para mensajes pequeños e intermedios. Cuando el tamaño de los mensajes es grande no se nota mucha deferencia entre las técnicas de *externalización* y sin *externalizacón* porque la sobrecarga de la aplicación es baja y la carga de la comunicación es igual, por lo tanto, se puede alcanzar el ancho de banda de la red sin usar la *externalización*.

Para obtener resultados reales, se debe considerar un sistema con buses de E/S (retardo del bus E/S) como se muestra en los resultados obtenidos de la Figura 3.23. Como se ha explicado en la Sección 2.3.4, se utiliza un bus PCI de 64 bits y de velocidad 33 MHz. En la Figura 3.23, se puede ver que los resultados de la simulación sin *externalización* y la simulación con *externalización* para mensajes de tamaño pequeño son casi iguales. Esto se debe al efecto de la sobrecarga del procesador donde se procesa el protocolo de comunicación, y al efecto de los buses de E/S. Cuando se aumenta el tamaño del mensaje, las técnicas de *externalización* mediante *offloading* y *onloading* empiezan mejorar el ancho de banda comparado con el sistema sin *externalización*.

3.6 El efecto de la cache en las técnicas de *offloading* y *onloading*

Es importante analizar y estudiar el efecto de la cache para obtener unos resultados de simulación más próximos a los reales. En nuestros experimentos hemos modelado una memoria cache de dos niveles por cada procesador, como se ha indicado en el capitulo anterior. En el primer nivel se usa una cache separada para los datos y para las instrucciones, mientras que en el segundo nivel se usa una cache unificada para instrucciones y datos. En el caso de la *externalización* mediante *offloading* se incluye una cache en la CPU del sistema, y se considera al mismo tiempo que el procesador de la tarjeta de la red no tiene cache. En el caso de la *externalización* mediante *onloading* se agrega una memoria cache de dos niveles a los dos procesadores (CPU_0 y CPU_1). Al incluir las memorias cache en el modelado del sistema de comunicación, se reduce el número de los ciclos que se necesitan para a acceder a los datos que están almacenados en la memoria cache. En general, la función T(mp) representa el tiempo de acceso a un dato en la memoria principal, y la función T(mc) representa el tiempo de acceso a un dato en una cache. La utilización de cache permite que T(mc) < T(mp), y eso dependerá a la tasa de aciertos de la cache, es decir, el porcentaje de veces que un dato se encuentra en la cache cuando éste es referenciado. Donde la tasa de aciertos, T_a es igual a (*aciertos* / (*aciertos*+*fallos*)), y la latencia de acceso medio, $T_{\text{acceso medio}}$ es igual a (T_a + (1- T_a)*la latencia de acceso a mp).

La mejora en el ancho de banda que se puede conseguir cuando se agrega una memoria cache depende de la tasa de acierto de cache para los datos o las instrucciones. En la Figura 3.24 se muestra el ancho de banda con *externalización* mediante *offloading*, con y sin memoria cache, y también se muestra la *externalizacion* mediante la técnica de *onloading* con y sin cache. De la figura se puede concluir que, con paquetes pequeños, los anchos de banda con *offloading* y con *onloading* son iguales. Esto se puede explicar por el efecto de la interconexión en el bus de la E/S y el bus de la memoria, que también puede afectar a la sobrecarga del sistema de comunicación.

Además, al utilizar una cache en el procesador donde se ejecuta el protocolo de comunicación se mejora el ancho de banda porque los mensajes de tamaño pequeño se guardan en la cache de nivel 1 (L1), y podrá evitarse ir a la memoria principal para acceder a los datos, y también el uso de los bues de E/S. Cuando se aumenta el tamaño de los mensajes, los de mayor tamaño no se pueden guardar

completamente en la cache L1, ni en la cache L2, y por tanto se pierden más ciclos por fallos en la memoria cache.

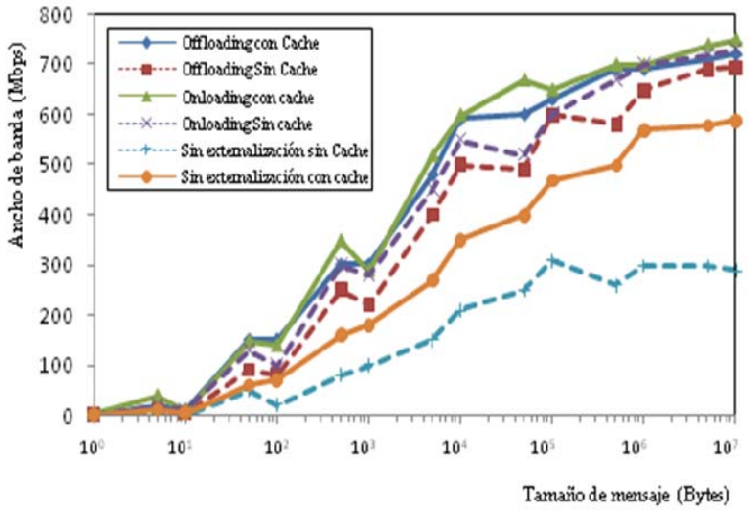

Figura 3.24 Comparación entre el ancho de banda para *offloading*, *onloading* y sin *externalización* con cache (L1=32KB, L2=1024KB) y sin cache.

Así, se puede observar en la misma Figura 3.24, que al incrementar el tamaño del mensaje hasta que llega un tamaño similar al de la cache de segundo nivel (L2), pueden empeorar las prestaciones del sistema de comunicación.

La Figura 3.25 muestra el efecto de la memoria cache en la latencia. La *externalización* mediante *onloading* proporciona una latencia menor que cuando se externaliza el sistema mediante *offloading*.

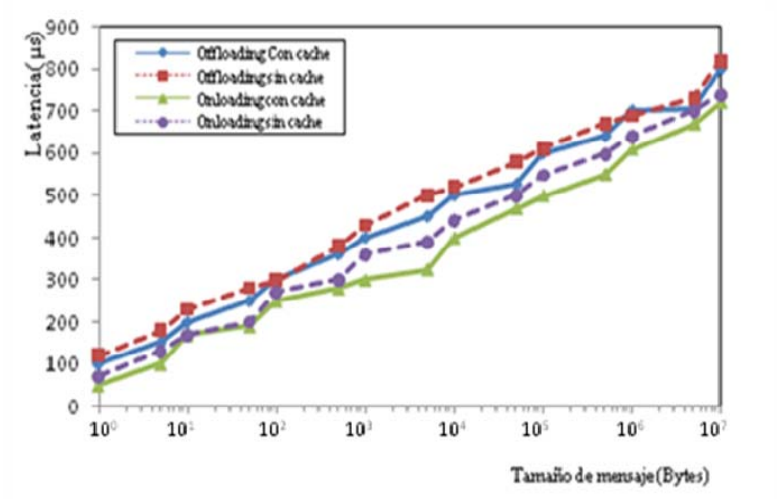

Figura 3.25 Comparación entre la latencia de *offloading* y *onloading* con cache (L1=32kbs, L2=1024kbs) y sin cache

La menor latencia observada en la Figura 3.25 para la técnica de *onloading* se explica porque al acceder a la memoria cache, el procesador que procesa el protocolo de

comunicación necesita menos ciclos de los que se requieren cuando el procesador está ubicado en la tarjeta de red, en el caso de *externalización* mediante *offloading*. Para explicar el comportamiento en cuanto a prestaciones del sistema de comunicación sin *externalización*, y con *externalización* mediante la técnica *offloading* y *onloading*, se puede recurrir al modelo LAWS [SHI03], que pone de manifiesto la mejora del ancho de banda en términos de la relación entre las cargas de computación y comunicación en el procesador central (expresada a través del parámetro γ).

3.6.1 Efecto del tamaño de las caches L1 y L2 en la mejora del ancho de banda

En esta sección se analiza el efecto de las memorias cache L1 y L2 en la mejora del ancho de banda del sistema de comunicación con *offloading*, *onloading* y sin *externalización*. Las simulaciones usadas en esta parte consideran varios tamaños de la memoria cache en los niveles L1 y L2 para analizar el efecto de la memoria cache en las prestaciones del sistema de comunicación.

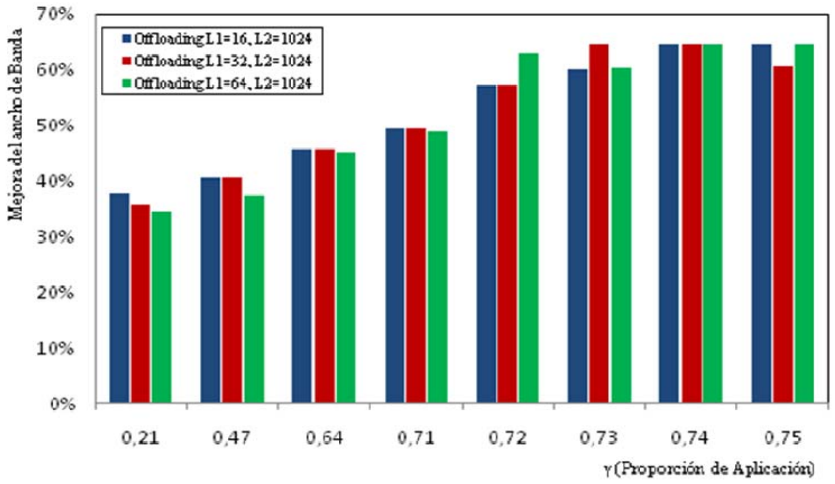

Figura 3.26 Comparación entre la mejora del ancho de banda del *offloading* (*αβ*=0.3), con varios tamaños de L, y el tamaño de los paquetes 64 Kbyte.

La Figura 3.26 muestra el efecto del tamaño de la memoria cache del primer nivel L1 en las prestaciones de la *externalizacion* mediante *offloading*. Se puede ver que el incremento en la memoria cache L1 (16KB, 32KB y 64KB) mejora el ancho de banda cuando se aumenta el parámetro de la aplicación *γ*. Es decir, que la mejora en las

prestaciones del sistema de comunicación estaría relacionada con la reducción en el número de ciclos perdidos por fallos de la cache L1. Cuando el tamaño de la memoria cache L1 es de 64KB, no se observa tanta mejora en el ancho de banda por las características propias del protocolo de comunicación y las características de los buses de E/S como se ha explicado en la sección 3.5.

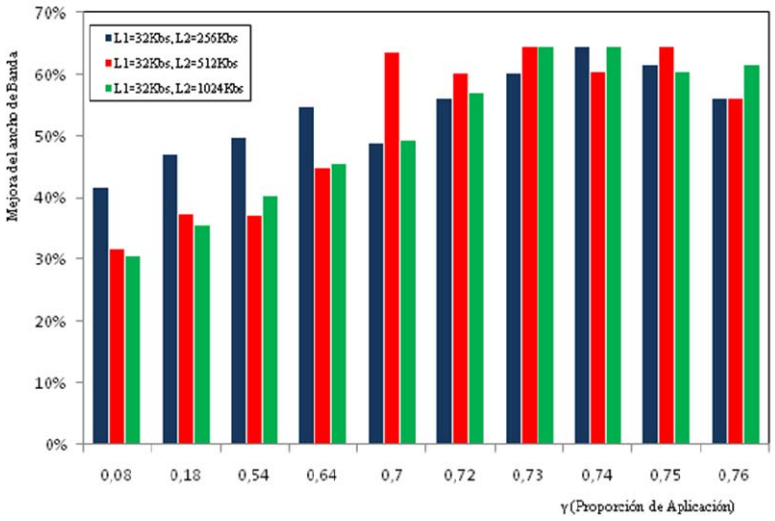

Figura 3.27 Comparación entre la mejora del ancho de banda del *offloading* ($\alpha\beta$=0.3), con varios tamaños de L2, y el tamaño de los paquetes 64 Kbyte.

Por otro lado, la Figura 3.27 muestra el efecto del cambio de tamaño en la memoria cache del nivel L2 en las prestaciones de *externalizacion* mediante *offloading*. En la misma figura se puede ver que, cuando aumenta el tamaño de la memoria cache L2 (256KB, 512KB, 1024KB) se mejora el ancho de banda. Un mayor tamaño en la cache L2 incrementa la tasa acierto de la jerarquía de cache, y mejoran las prestaciones del sistema de comunicación.

De los experimentos que hemos realizado se puede concluir que, usar una memoria cache en la CPU cuando se externaliza el protocolo de comunicación mediante *offloading*, no mejora demasiado las prestaciones del sistema de comunicación. Esto se debe a que dado que la mayor parte del procesamiento del protocolo de comunicación se realiza en la tarjeta de red, no afecta mucho agregar una memoria cache en la CPU del nodo.

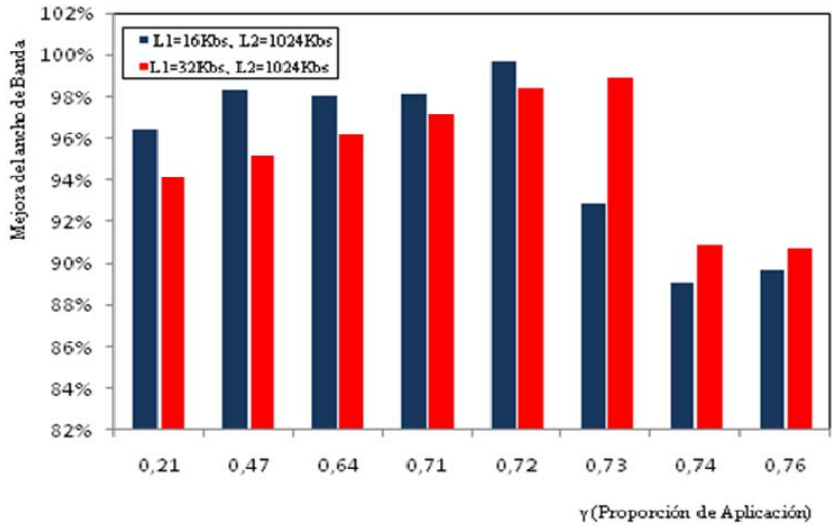

Figura 3.28 Comparación entre la mejora del ancho de banda del *onloading*, con varios tamaños de L1, y el tamaño de los paquetes 64 Kbyte.

En la Figura 3.28, se puede ver el efecto del cambio del tamaño de la memoria cache de primer nivel L1 en las prestaciones de *externalizacion* mediante *onloading*. Cuando se incrementa el tamaño de la memoria cache L1 (16KB, 32KB) se obtiene un aumento en la mejora del ancho de banda dependiendo al parámetro γ.

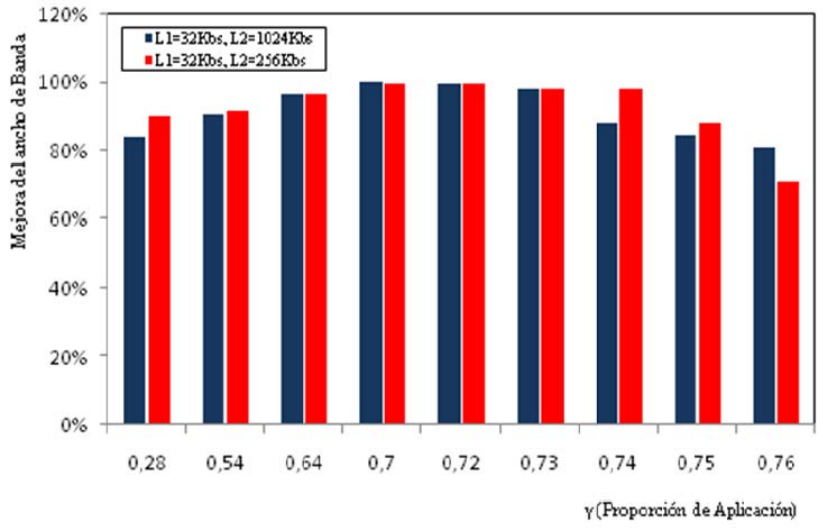

Figura 3.29 Comparación entre la mejora del ancho de banda del *onloading*, con varios tamaños de L2 y cuando el tamaño de paquetes 64 Kbyte

La Figura 3.29 muestra el efecto del cambio en el tamaño de la memoria cache del nivel L2 en las prestaciones de la *externalizacion* mediante la técnica *onloading*. Se puede ver que, cuando aumenta el tamaño de la memoria cache del nivel L2 (256KB,

1024KB), aumenta la mejora el ancho de banda dependiendo al parámetro γ. La mejora obtenida en las prestaciones del sistema de comunicación pone de manifiesto la ventaja de usar una CPU *multinúcleo* y con memoria cache de dos niveles (L1y L2) para datos e instrucciones, en el procesamiento del protocolo de comunicación.

3.6.2 El efecto del tamaño de las caches L1 y L2 en la latencia

En la Figura 3.30 muestra el efecto de la memoria cache en la latencia del sistema de comunicación. Cuando se externaliza el protocolo de comunicación mediante *offloading*, no se mejora mucho la latencia al agregar una memoria cache porque la memoria cache está ubicada en el procesador central no en el procesador de la tarjeta de red (NIC). En las simulaciones de la técnica de *offloading*, se modifica el modulo de CPU (el *driver* se ejecuta en un procesador ubicado en la tarjeta de red) como se ha mostrado en la sección 2.2.2 para evaluar estos efectos. En la misma figura se muestra el efecto de la memoria cache cuando se externaliza mediante *onloading*. Se observa que, al añadir una cache al procesador que va a hacer el procesamiento del protocolo de comunicación, disminuye la latencia, mejorándose las prestaciones del sistema de comunicación. En todos los casos, como muestra la Figura 3.30, la latencia de la *externalización* mediante *onloading* es menor que la *externalización* mediante *offloading*, y cuando el protocolo de comunicación no está externalizado.

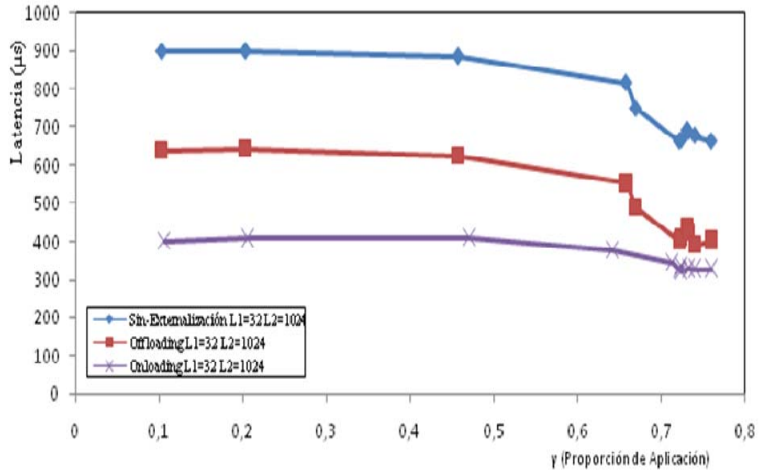

Figura 3.30 Comparación entre la latencia del *sin-externalización*, *onloading* y *offloading* con memoria cache de dos niveles.

Por lo tanto, en los resultados experimentales de todas las simulaciones con memoria cache de dos niveles, se pone de manifiesto el efecto beneficioso de la memoria cache en la mejora de las prestaciones del camino de comunicación como se ha mostrado en la Figura 3.24.

3.7 Modelo de simulación HDL para interfaz de red que combina las técnicas de *offloading* y *onloading*.

En la Figura 3.31 se muestra un esquema de los módulos del modelo HDL para el camino de comunicación que hibrida las técnicas de *offloading* y *onloading*. En este modelo combinamos las dos técnicas de *externalización* que se han analizado en el capítulo 2.

En esta propuesta de *externalización* que realizamos se aprovecha el procesamiento paralelo que permite el disponer de un procesador *multinúcleo* y un procesador en la NIC, distribuyendo las tares de comunicación entre ellos. Se puede aprovechar el núcleo CPU_1 (Figura 3.31) para hacer otras tareas del sistema operativo y liberar más ciclos del primer núcleo, CPU_0 pero, al mismo tiempo, el núcleo CPU_1 tiene preferencia para el procesamiento de las tareas de comunicación.

En este modelo se han utilizado distintos valores para los retardos que permiten simular los buses, los accesos de memoria, las tareas de la aplicación, etc. Cuando se llena el buffer de paquetes de la NIC, se genera una interrupción a la CPU_1, que se encarga de procesar las interrupciones y copiar los datos a los buffers de aplicación. En la NIC se procesan los protocolos de comunicación y después se ejecuta el *driver* para transferir los datos a la memoria principal. Por lo tanto, la CPU_0 no recibe interrupciones y se encarga solo del procesamiento de la aplicación. Usando el modelo de simulación de la Figura 3.31, se realizan los experimentos para evaluar las prestaciones de comunicación de nuestra propuesta, igual que hemos hecho con la *externalización* mediante el *offloading* (Sección 3.2) y mediante *onloading* (Sección 3.3).

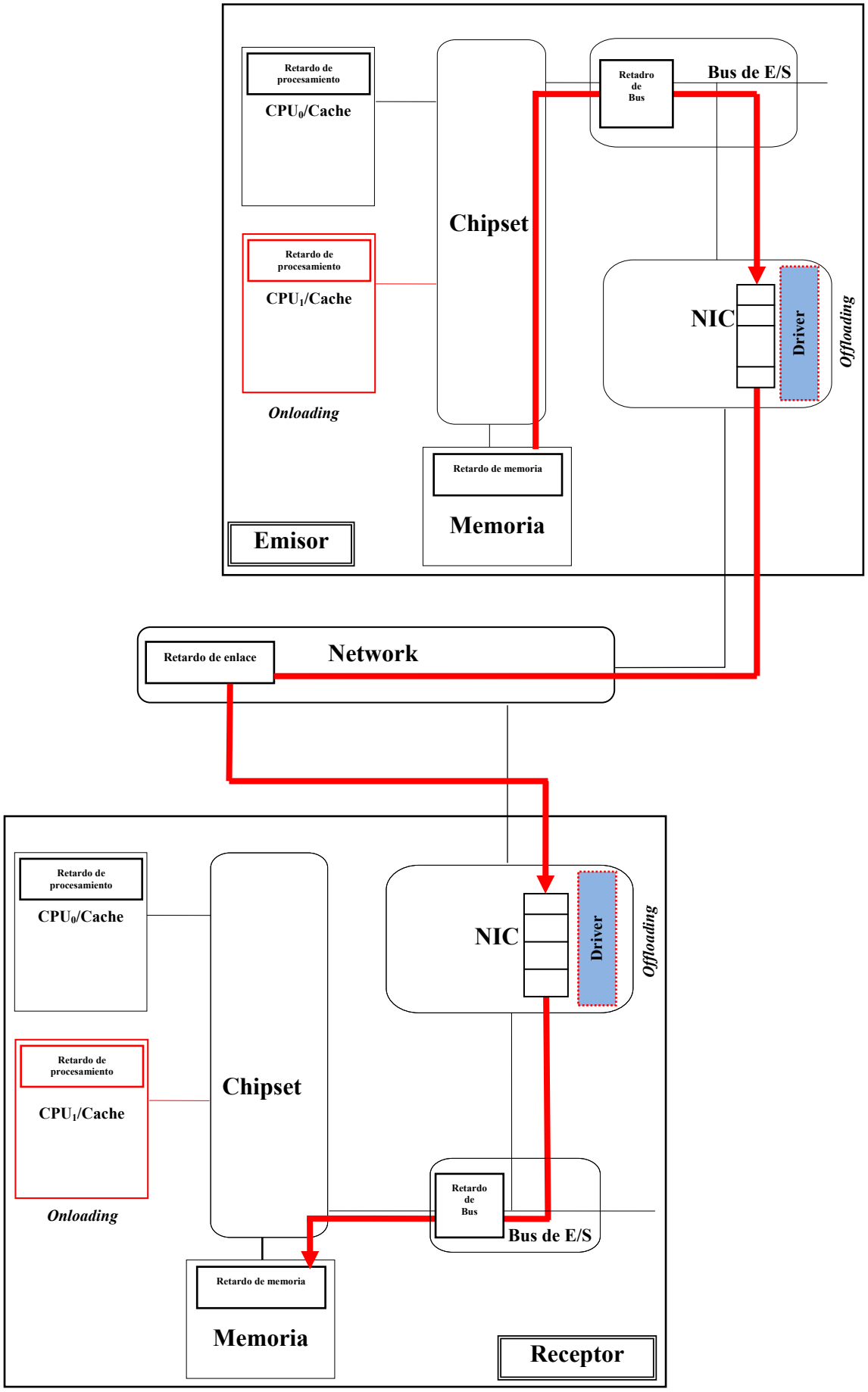

Figura 3.31 El modelo de simulación HDL del camino de comunicación completo para la propuesta híbrida *offloading*/*onloading*

En la Tabla 3.1 se muestran las diferentes distribuciones de las tareas de comunicación en los distintos elementos de los modelos de simulación HDL de *externalización* mediante la técnica *sin-externalización, offloading*, *onloading*, y nuestra propuesta híbrida (*offloading*/*onloading*).

Tabla 3.1 La distribución de las tares de comunicación entre los elementos de la simulación con la propuesta híbrida

Tarea	***Sin externalización***	***Offloading***	***Onloading***	***Híbrida***
Driver	CPU_0	NIC	CPU_1	NIC
Interrupción	CPU_0	CPU_0	CPU_1	CPU_1
Protocolo de comunicación	CPU_0	NIC	CPU_1	NIC

La Tabla 2.1 del capítulo anterior muestra las diferencias entre la ubicación del procesamiento para la *externalización* con *offloading*, con *onloading*, y sin *externalización*, en la Tabla 3.1 se muestran las diferencias entre nuestra propuesta hibrida y las demás alternativas. En el caso de *externalización* híbrida (*offloading/onloading*) se utiliza la NIC para hacer el procesamiento del protocolo de comunicación y ejecutar el driver, el procesamiento de las interrupciones se encarga de hacerlo la CPU_1. Por lo tanto, se liberan muchos ciclos de la CPU_0, que quedan libres para procesar el sistema operativo y otras tareas.

El modelo híbrido que se ha propuesto en [ORT08], utiliza dos procesadores para hacer el procesamiento del sistema de comunicación, uno de ellos está ubicado en la tarjeta de red (CPU_{NIC}) y se encarga de hacer todo el procesamiento del protocolo de comunicación, y el otro (CPU_1) se encarga de la ejecución del driver de la tarjeta de red y hacer el procesamiento de las interrupciones. Mientras tanto, nuestra propuesta híbrida considera que la tarjeta de red se encarga de hacer todo el procesamiento del protocolo de comunicación, y además se ejecuta el driver de la tarjeta de red. Esto puede liberar más ciclos de la CPU_1 para hacer otras tareas de aplicación como se mostrará en los resultados experimentos de nuestro modelo híbrido.

3.7.1 Análisis de prestaciones de la interfaz híbrida *offloading/onloading.*

A continuación se describen y analizan los resultados obtenidos con nuestra propuesta de interfaz de red híbrida *offloading/onloading*. Como se ha explicado anteriormente, en esta nueva alternativa se combinan los dos modelos de simulación, el *offloading* (Sección 2.2.2) y el *onloading* (Sección 2.2.3), para aprovechar las ventajas de las dos técnicas de *externalización* en la mejora de las prestaciones del camino de comunicación.

3.7.1.1 Mejora del ancho de banda

La Figura 2.32 muestra la comparación del ancho de banda cuando no hay *externalización* y cuando se utilizan técnicas de *externalización* mediante *offloading*, *onloading* y con nuestra prepuesta de hibridación *offloading*/*onloading* para diferentes valores del parámetro γ, que indica la distribución de carga de computación y comunicación. El ancho de banda obtenido mediante la nueva técnica de *externalización* híbrida (mediante *offloading* y *onloading*) es mayor que la de las demás alternativas.

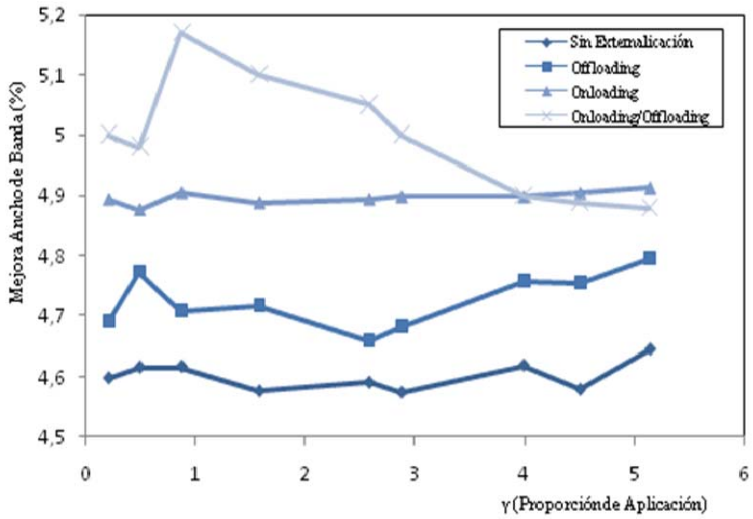

Figura 3.32 Comparación entre el ancho de banda para *offloading, onloading, offloading/onloading* y sin *externalización* ($\alpha\beta$=1.01).

Cuando se externaliza el sistema mediante la hibridación de *offloading* y *onloading*, el núcleo CPU_1 (Figura 3.31) del nodo se encarga del procesamiento de las interrupciones y de la copia de los datos a la aplicación, mientras el *driver* y los protocolos de comunicación se procesan en la tarjeta de red. Por lo tanto, esta nueva propuesta de *externalización* híbrida reduce el uso del primer núcleo, CPU_0, y libera ciclos del mismo para hacer otras tareas.

En la Figura 3.33 se muestra la comparación entre el ancho de banda de las alternativas de la *externalización* mediante *offloading*, *onloading* y nuestra prepuesta que híbrida *offloading/onloading*, para varios tamaños de mensajes. El ancho de banda obtenido con nuestra propuesta híbrida es mayor que el de las otras alternativas de *externalización*, como se muestra en la Figura 3.33. Todo el procesamiento del protocolo de comunicación se realiza un núcleo específico del procesador *multinúcleo*, distinto del utilizado para la aplicación. Además, el procesamiento de las interrupciones y la ejecución del *driver* también se realizan en otro procesador distinto del que se encarga de las aplicaciones. En los resultados de la Figura 3.33 se ha considerado que $\alpha\beta$=1. Esto supone que, si α>1 dado que el procesador de la red es más lento, la asignación de tareas al procesador de la NIC permite ahorrar cierto coste de transferencias y se podría conseguir un valor β<1.

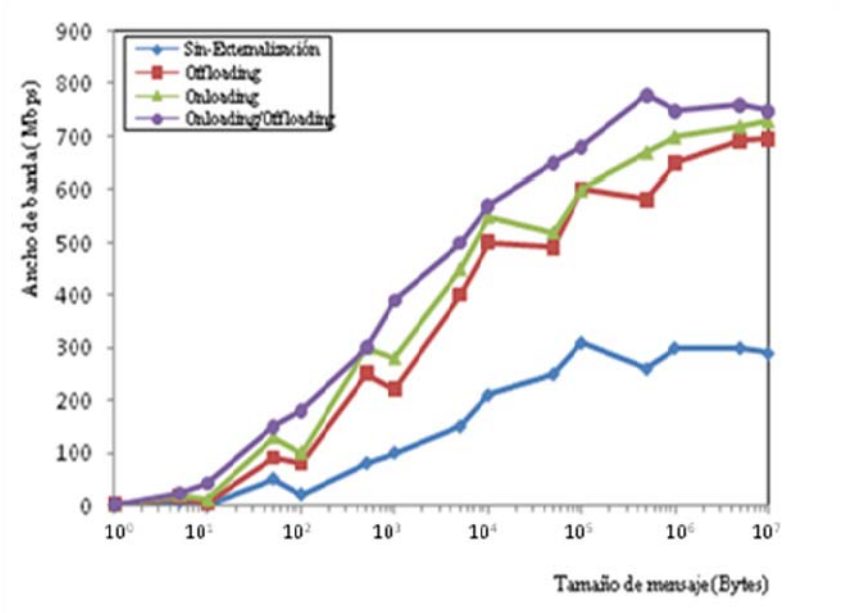

Figura 3.33 Comparación entre el ancho de banda para *offloading*, *onloading*, *offloading/onloading* y sin *externalización* con varios tamaños de mensajes y αβ=1.01.

3.7.1.2 Mejora en la latencia

En la Figura 3.34 se muestra la latencia para varios tamaños de mensaje y cuando se externaliza mediante nuestra propuesta de técnica híbrida *offloading/onloading*. En la misma gráfica se comparan con latencias obtenidas con otras técnicas de *externalización* (*offloading* y *onloading*), y la opción de partida sin *esternalización*.

Se puede ver en la misma Figura 3.34 que la latencia obtenida con la propuesta híbrida *offloading/onloading* es menor que las latencias por la *externalización* mediante el *onloading*, y un poco mayor que el mejor valor de latencia mediante *offloading*.

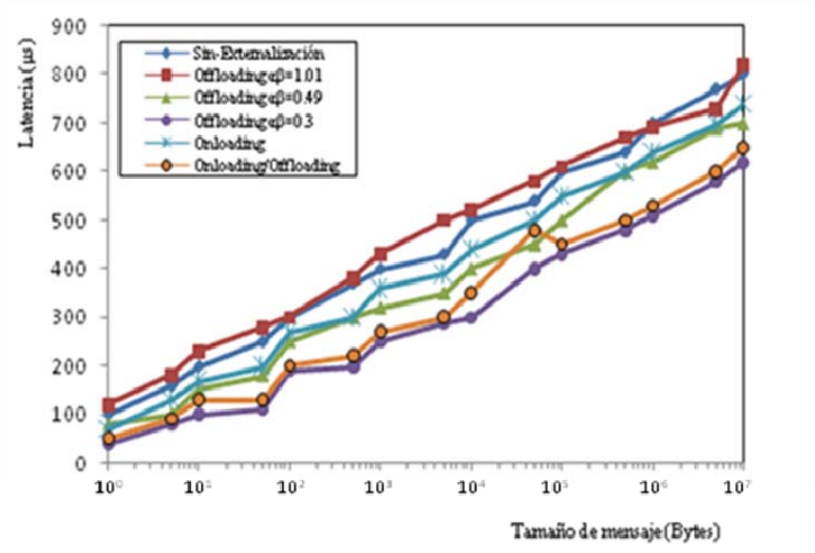

Figura 3.34 Comparación entre la latencia observada con offloading, *onloading*, *offloading/onloading* y sin *externalización* para varios tamaños de mensajes.

Del análisis realizado se puede concluir que la nueva propuesta híbrida de *externalización* mejora las prestaciones del camino de comunicación reduciendo el uso del procesador central y una gran proporción de transferencias a través del bus de E/S.

3.8 Conclusiones

Este capítulo ha presentado los resultados experimentales obtenidos a partir del modelo de simulación HDL del sistema de comunicación. En la elaboración de este modelo se ha utilizado el lenguaje *Verilog*, y además se ha considerado el modelo

teórico LAWS para validar y entender el comportamiento de las prestaciones del camino de comunicación mediante los parámetros del mismo modelo.

A partir de los resultados de este capítulo se pude concluir que, el uso de la *externalización* para librar ciclos de la CPU es una técnica adecuada. En general, las prestaciones obtenidas con *onloading* son mejores que los proporcionados mediante *offloading*.

Para alcanzar resultados próximos a los obtenidos en sistemas reales, se ha incluido en el modelo de simulación una memoria cache de dos niveles L1 y L2 y para analizar sus efectos en la *externalización*. En nuestros experimentos se ha puesto de manifiesto que la memoria cache mejora las prestaciones del sistema de comunicación tanto con *offloading* como con *onloading*. En el *offloading* no mejora tanto las prestaciones del sistema de comunicación cuando se agrega una memoria cache en la CPU del nodo. Esto se debe a que dado que la mayor parte del procesamiento del protocolo de comunicación se realiza en la tarjeta de red.

De los resultados anteriores se puede concluir que, tanto con *offloading* como con *onloading*, se producen mejoras para valores pequeños del factor de aplicación del modelo LAWS. Además, el efecto de la memoria cache en la mejora del ancho de banda mediante *onloading* es mayor que con *offloading*.

Se han comparado los resultados que se han obtenido con los modelos HDL y los obtenidos a partir de un simulador de sistema completo como SIMICS [ORT08]. Las diferencias entre los resultados experimentales con el simulador HDL y con SIMICS son relativamente pequeñas para la configuración de base sin *externalización*. Por lo tanto, dado el nivel de detalle y realismo que proporciona el simulador de sistema completo se puede considerar que la simulación de nuestros modelos HDL permite extraer conclusiones válidas acerca del comportamiento de las interfaces de red a distintas circunstancias.

Además, se ha presentado una propuesta de la interfaz de red híbrida de *externalización*, que combina las alternativas de *offloading* y *onloading*. Se ha mostrado un nuevo modelo de HDL dependiendo al los modelos de simulación presentados en la sección 2.2.2 y 2.2.3. Además, se realiza un análisis de las prestaciones de la propuesta híbrida, y se comparan los resultados obtenidos por nuestra propuesta con las alternativas de *externalización*.

En el próximo capítulo se mostrarán los diseños factoriales que incluyen los experimentos óptimos para estudiar qué variables del modelo LAWS influyen de forma más relevante en el sistema de comunicación.

Capítulo 4

DISEÑO DE EXPERIMENTOS EN EL ANÁLISIS DE LOS PARÁMETROS DEL CAMINO DE COMUNICACIÓN

En este capítulo se describen las técnicas del diseño experimental para evaluar las arquitecturas del camino de comunicación y para entender el compartimiento funcional de los modelos incorporados y su relación con las prestaciones del interfaz de la red. Los modelos para el diseño de experimentos [JAI91] son modelos de estadísticos clásicos cuyo objetivo es verificar si determinados factores influyen en la variable experimental de interés. El diseño de experimentos utiliza una metodología besada, lógicamente, en la experimentación, y es importante porque permite al investigador mejorar un proceso o sistema por medio de la observación de cambios en el mismo a través de los resultados de experimentos. Se necesita una herramienta estadística (por ejemplo *Excel*, *Matlab Statistical Toolbox*, etc.) para poder facilitar el análisis de los experimentos y obtener resultados que significan una mejora, una corrección, o una estabilización en ese proceso o sistema.

El primer objetivo del diseño de experimentos es controlar la variabilidad en un proceso estocástico que puede tener distintos orígenes. Los resultados de cualquier

experimento están sometidos a tres tipos de variabilidad: variabilidad sistemática y planificada, no planificada, y variabilidad típica de la naturaleza del problema y del experimento.

Un diseño de experimentos se realiza por alguno de los siguientes motivos: (1) determinar los principales cauces de variación en la respuesta; (2) encontrar las condiciones experimentales con las que se consigue un valor extremo en la variable de interés; (3) comparar las respuestas para diferentes niveles de observación de los parámetros controlados; y (4) obtener un modelo estadístico que permita hacer predicciones de respuesta futuras.

El capítulo se ha organizado de forma que en primer lugar se presenten las principales estrategias del diseño experimental y las características de cada una (Sección 4.1 y Sección 4.2). Después, se describe con detalle cómo se usan estas estrategias en nuestras simulaciones (Secciones 4.3, 4.4 y 4.5). En los capítulos anteriores, se presentaron los modelos que se han usado para contrastar las alternativas de *offloading* y *onloading* para mejorar las prestaciones del camino de comunicación. Aquí se comparan los resultados extraídos del diseño de experimentos con las predicciones de esos modelos.

4.1 El diseño de experimentos

Para optimizar los procesos de fabricación, las condiciones de reacción y los métodos de análisis de resultados experimentales, entre otros, es necesario conocer qué variables influyen significativamente en el sistema y cómo afectan. A menudo esta información no está disponible y se genera experimentando. En nuestro caso, se trata de determinar las características más importantes en el comportamiento del camino de comunicación, y ayudar al diseño para mejorar en las interfaces de red.

En primer lugar, se recogen en una lista todas las variables que podrían influir en la respuesta que interesa analizar. Después, se realizan una serie de experimentos en los que se mantienen constantes los parámetros que no interesa modificar, se anota el valor de los que no se pueden controlar, y se varían los restantes. Finalmente, se obtiene la información buscada comparando la variación de la respuesta entre experimentos. El elevado coste que puede suponer la realización de un experimento y las limitaciones de tiempo obligan a disponer de una estrategia para ejecutar sólo los

experimentos imprescindibles. El método tradicional de variar un parámetro cada vez, no suele ser la mejor opción. Puede implicar más experimentos de los necesarios y, a pesar de ello, proporcionar sólo información parcial. Por ejemplo, puede no mostrar si existe interacción entre distintos factores. Las interacciones suelen ser frecuentes y a veces son los efectos más importantes, por lo que conocerlas es imprescindible para comprender el comportamiento de muchos sistemas.

El diseño estadístico de experimentos contempla una amplia variedad de estrategias experimentales que son eficientes para generar la información que se busca. En este capítulo se consideren dos estrategias: el *diseño factorial completo* 2^k y el *diseño factorial fraccional*.

4.1.1 El diseño factorial completo 2^k

Este diseño describe los experimentos más adecuados para conocer simultáneamente qué efecto tienen k factores sobre una respuesta y descubrir si interaccionan entre ellos. Estos experimentos están planeados de forma que se varían simultáneamente varios factores pero se evita que se cambien siempre en la misma dirección. Al no haber factores correlacionados se ahorran bastantes experimentos. Además, los experimentos se complementan de tal modo que la información buscada se obtiene combinando las respuestas de todos ellos. Esto permite obtener la información buscada con el mínimo número de experimentos (por tanto, con el menor coste experimental), y con la menor incertidumbre posible (porque los errores aleatorios de las respuestas se promedian).

Una matriz de experimentos factorial completa 2^k no requiere un código especializado para construirla ni para analizar sus resultados. En estos diseños, cada factor se estudia en sólo dos niveles (+1 y -1) y sus experimentos contemplan todas las combinaciones de cada nivel de un factor con todos los niveles de los otros factores. Las Tablas 4.1 (a, b y c) muestran las matrices 2^2, 2^3 y 2^4, para el estudio de 2, 3 y 4 parámetros respectivamente. La matriz comprende 2^k filas y k columnas, que corresponden a los k parámetros en estudio. Si se construye en el *orden estándar*, cada columna empieza por el valor –1, y se alternan los valores –1 y +1 con frecuencia 2^0 para Z_1, 2^1 para Z_2, 2^2 para Z_3, y así sucesivamente hasta Z_k, donde los signos se

alternan con una frecuencia 2^{k-1}, representando Z_k el número de los experimentos en el diseño.

Tabla 4.1 Matriz de experimentos para los diseños factoriales completos (a) 2^2, (b) 2^3, y (c) 2^4.

(a)

Experimento	Z_1	Z_2
1	-1	-1
2	+1	-1
3	-1	+1
4	+1	+1

(b)

Experimento	Z_1	Z_2	Z_3
1	-1	-1	-1
2	+1	-1	-1
3	-1	+1	-1
4	+1	+1	-1
5	-1	-1	+1
6	+1	-1	+1
7	-1	+1	+1
8	+1	+1	+1

(c)

Experimento	Z_1	Z_2	Z_3	Z_4
1	-1	-1	-1	-1
2	+1	-1	-1	-1
3	-1	+1	-1	-1
4	+1	+1	-1	-1
5	-1	-1	+1	-1
6	+1	-1	+1	-1
7	-1	+1	+1	-1
8	+1	+1	+1	-1
9	-1	-1	-1	+1
10	+1	-1	-1	+1
11	-1	+1	-1	+1
12	+1	+1	-1	+1
13	-1	-1	+1	+1
14	+1	-1	+1	+1
15	-1	+1	+1	+1
16	+1	+1	+1	+1

El diseño de los experimentos posee numerosas aplicaciones para la determinación de mejoras del sistema de comunicación. Tenido en cuenta que este se centra en el estudio de la interfaces de red, en lo que sigue, utilizarnos las alternativas de mejora de las prestaciones de comunicación para ilustrar el uso de un diseño factorial completo 2^k, el cálculo de los efectos y el concepto de interacción entre factores.

4.1.2 El diseño factorial fraccional

El número de factores afecta al número de los experimentos. Así, el número de condiciones experimentales crece exponencialmente en un experimento factorial. Además, el número de efectos a evaluar (interacciones) también crece exponencialmente y varía con el número de factores. Así, por ejemplo, cuando se tienen siete factores, existen 128 posibles condiciones experimentales, lo que implica que hacer una repetición por celda de todo el experimento requiere hasta un total de 128 observaciones.

Considérese el caso en el que se estudian tres factores con dos niveles cada uno, pero en el que los experimentadores no pueden realizar las 2^3= 8 combinaciones de condiciones experimentales, aunque, sí se pueden llevar a cabo 4 observaciones. Esto sugiere una fracción igual a la mitad de un diseño 2^3. Este planteamiento se conoce también como diseño 2^{k-1} (en nuestro ejemplo 2^{3-1} = 4 combinaciones). En la Tabla 4.2 aparecen signos positivos y negativos del diseño 2^k por k=3. Para componer la mitad de experimentos, se seleccionan las combinaciones usando indistintamente la notación convencional (Z_1, Z_2, Z_3,...) y la de signos positivos y negativos.

Tabla 4.2. Matriz completo del diseño fraccional 2^K por K=3.

Experimento	**I**	Z_1	Z_2	Z_3	$Z_1 Z_2$	$Z_1 Z_3$	$Z_2 Z_3$	$Z_1 Z_2 Z_3$
1	+1	-1	-1	-1	1	1	1	-1
2	+1	1	-1	-1	-1	-1	1	1
3	+1	-1	1	-1	-1	1	-1	1
4	+1	1	1	-1	1	-1	-1	-1
5	+1	-1	-1	1	1	-1	-1	1
6	+1	1	-1	1	-1	1	-1	-1
7	+1	-1	1	1	-1	-1	1	-1
8	+1	1	1	1	1	1	1	1

A partir de la Tabla 4.2, el diseño 2^{3-1} se construye al seleccionar solo las combinaciones que producen un signo negativo sobre la columna Z_3. Por esto Z_3 no se denomina generador de una fracción particular. En general, la relación que define un factorial fraccionario siempre es el conjunto de todas las columnas que son iguales a la columna identidad I. En la sección 4.4.2 se aplicará esta estrategia a los experimentos que se han hecho sobre las distintas alternativas de interfaces de red.

4.2 El análisis de la varianza ANOVA

El análisis de la varianza (ANOVA) constituye una potente herramienta estadística, de gran utilidad para el control de procesos y el análisis experimental. Las técnicas iniciales del análisis de la varianza fueron desarrolladas por estadísticos y se conocen frecuentemente como ANOVA de Fisher o análisis de varianza de Fisher, debido al uso de la distribución F de Fisher [FIS71] como parte del contraste de hipótesis. El análisis realizado mediante ANOVA es una colección de modelos estadísticos y sus procedimientos asociados, en el cual la varianza se distribuye entre ciertos componentes correspondientes a diferentes variables explicativas. En esta sección se introduce la técnica ANOVA de un factor y la forma según la que se aplicará a nuestro objetivo.

4.2.1 ANOVA de un factor

El análisis de la varianza ANOVA de un factor sirve para comparar varios grupos (resultados de experimentos) en una variable cuantitativa. El objetivo del ANOVA aquí es comparar los errores sistemáticos con los errores aleatorios obtenidos al realizar diversos experimentos en cada parámetro. Las condiciones importantes para cada parámetro se analizan considerando sus muestras de manera independiente, y con precisiones parecidas a las del resto de parámetros.

El objetivo principal del ANOVA es comparar los diferentes valores medios para determinar si alguno de ellos difiere significativamente del resto. Para ello se considera que, si los resultados proporcionados por los diferentes experimentos no contienen errores sistemáticos, los valores medios respectivos no diferirán mucho los unos de los otros y su dispersión, debida a los errores aleatorios, será comparable a la dispersión presente individualmente en cada parámetro. Matemáticamente, la suma de cuadrados total, *SS*T, puede descomponerse como una suma de dos sumas de cuadrados:

$$SST = SSR + SSP \tag{4.1}$$

Donde *SS*T es la suma de las diferencias al cuadrado de cada resultado individual respecto a la media de todos los resultados y por tanto, representa la variación total de los datos; SSR mide las desviaciones entre los resultados individuales X_{kj}, de cada parámetro k y repetición j. La media del parámetro se represente como X_k y, por lo tanto, es una medida de la dispersión para cada parámetro. Cuando se divide *SS*R por los correspondientes grados de libertad del experimento, *N*-*k*, se obtiene la medida, *MS*R. La suma de los cuadrados SSP en (4.1) mide las desviaciones entre los resultados medios de los parámetros y el resultado medio global y, dividida por sus grados de libertad, *k*-1, constituye su valor medio, *MSP*. La Tabla 4.3 muestra las diferentes expresiones para calcular las sumas de cuadrados y las correspondientes varianzas.

Tabla 4.3 Expresiones para el cálculo del ANOVA de un factor.

Origen de las variaciones	Suma de cuadrados	Grados de libertad	Varianza	$F_{calculada}$	F_{tabla}
Entre Parámetros	$SSP = \sum_{K=1}^{K} n_k(\overline{x_k} - \bar{\bar{x}})^2$	*k - 1*	$MSP = \frac{SSP}{K-1}$	-	-
Dentro Parámetros	$SSD = \sum_{K=1}^{K}\sum_{j=1}^{n_k}(x_{kj} - \overline{x_k})^2$	*N – k*	$MSD = \frac{SSD}{N-K}$	$\frac{MSP}{MSE}$	*[1-α;k-1,N(k-1)*
Error	*SSE=SST-SSP*	*N(k -1)*	$MSE = \frac{SSE}{N(K-1)}$	-	-
Total	$SST = \sum_{K=1}^{K}\sum_{j=1}^{n_k}(x_{kj} - \bar{\bar{x}})^2$	*N - 1*	$MST = \frac{SST}{N-1}$	-	-

Por tanto, *MS*P y *MS*D se calculan como una medida de las dispersiones comentadas y se comparan mediante una prueba de hipótesis, *F*. Sí no existe diferencia estadísticamente significativa entre $F_{calculada}$ y F_{tabla}, la presencia de errores aleatorios será la causa predominante de la discrepancia entre los valores medios. Sí, por el contrario, existe algún error sistemático, *MS*P será mucho mayor que *MS*D, con lo cual el valor calculado de $F_{calculada}$ será mayor que el valor tabulado F_{tabla} para el nivel de significación a escogido y los grados de libertad mencionados [JAI91].

Tenido esto en cuenta, ANOVA puede utilizarse para comparar entre sí las medias de los resultados obtenidos por diferentes análisis, métodos de análisis, etc. Además, ANOVA se utiliza para descomponer la variación total de un proceso entre las fuentes de variación parciales, lo cual nos puede resultar muy útil para determinar qué los factores afectan más a un determinado resultado.

4.2.2 ANOVA factorial

Cuando hay más de un factor, los modelos factoriales de análisis de varianza permiten evaluar el efecto individual de dos o más factores sobre una variable cuantitativa dependiente. Utilizar más de un factor en un diseño experimental permite estudiar el

efecto de la interacción entre los distintos factores. En un modelo de dos factores, el interés se centra en los efectos principales por cada factor y en el efecto de interacción entre ambos factores. Así, los efectos principales en un modelo de tres factores son los tres factores principales, los tres efectos de las interacciones por cada interacción entre cada dos factores, y el efecto interacción triple.

En este capítulo, se va a utilizar ANOVA en los experimentos realizados con las interfaces de red para analizar los resultados experimentales estadísticamente y comprobar el efecto de los factores considerados en el modelo LAWS sobre la interfaz de red.

4.3 Diseño de 2 parámetros para el análisis de la interfaz de red

Para ilustrar el uso de un diseño factorial 2^2 en el análisis de la interfaz de red, hemos considerado el efecto de los dos factores $\alpha\beta$ y γ en la mejora del ancho de banda del sistema de comunicación. Se quiere comprobar en qué medida estos dos factores influyen en el comportamiento del rendimiento y los valores que permitirían alcanzar un comportamiento óptimo. Desde el punto de vista del modelo LAWS, hemos elegido los parámetros $\alpha\beta$ y γ porque son más efectivos para el análisis de la interfaz de red. Por el comportamiento de la mejora del ancho de banda según el modelo LAWS (Figura 1.14), que crece para valores bajos de γ y se reduce para valores grandes del mismo, deberían escoger tres valores para el parámetro γ para hacer dos diseños experimentales para utilizar el diseño factorial 2^2.

Los factores escogidos para los dos diseños experimentos y su dominio experimental se muestran en la tabla 4.4 (a) y (b).

Tabla 4.4 Factores y dominio experimental (a) y (b).

(a)

Parámetros	**Mínimo (-1)**	**Máximo (+1)**
Z_1: $\alpha\beta$	0.3	1.01
Z_2: γ	0.326	1.53

(b)

Parámetros	Mínimo (-1)	Máximo (+1)
Z_1: αβ	0.3	1.01
Z_2: γ	1.53	3.45

Como ambos factores son continuos, su dominio experimental se expresa mediante los valores máximo y mínimo o dos valores significativos que puedan tomar en los experimentos. Las Tablas 4.4 (a) y (b) también muestran la notación codificada más habitual para factores continuos: se asigna el valor -1 al extremo inferior del dominio experimental y el valor +1 al extremo superior. Esta codificación también es válida para valores intermedios dentro del domino experimental. Es necesario definir la correspondencia entre variables reales y codificadas porque el diseño de experimentos describe la experimentación óptima empleando variables codificadas sin dimensión (Z_1, Z_2,…).

Un vez presentado el domino experimental combinado para los dos factores αβ y γ expresado en unidades codificadas y particularizado para los parámetros de la respuesta, los experimentos permitirían descubrir cómo influyen los dos factores en el rendimiento y si existe interacción entre ellos. La experimentación más económica se tiene cuando se toma el mínimo número de experimentos y cada factor toma sólo dos valores. Las Tablas 4.5 (a) y (b) muestra la matriz de experimentos que se obtiene combinado los dos niveles de los dos factores. Cada fila es un experimento y cada columna es un factor estudiado. Como se ha dicho antes, este diseño se denomina diseño factorial completo 2^2. Para cada uno de estos cuatro experimentos se indican los valores minamos o máximos de cada parámetro. En esta Tabla 4.5 se proporciona toda la información del diseño de experimentos.

Las cuatro respuestas para la mejora del ancho de banda se pueden combinar para obtener cuatro efectos de cada nivel de cada factor. El efecto principal del parámetro $\alpha\beta$, es igual a $(-Z_1+Z_1-Z_1+Z_1)/2^2$, el efecto principal del parámetro γ, es igual a $(-Z_2.Z_2-Z_2+Z_2)/2^2$ y el efecto de interacción entre los dos parámetros, es igual a $(+Z_1Z_2-Z_1Z_2-Z_1Z_2+Z_1Z_2)$. Por lo tanto, todos los efectos son importantes y cada uno de ellos tiene sus características y su efecto.

El valor promedio indica alrededor de qué valor están distribuidas las respuestas. En caso de que ningún factor tenga un efecto significativo, la distribución estaría determinada por la incertidumbre experimental.

Tabla 4.5 Diseño factorial completo 2^2 para los factores $\alpha\beta$ y γ, (a) para la tabla 4.4 (a) y (b) para la tabla 4.4 (b).

(a)

Matriz de Experimento			**Plan de Experimento**		**Mejora de ancho de Banda**
Z_1	Z_2	$Z_1 Z_2$	$\alpha\beta$	γ	**Rendimiento**
-1	-1	1	0.3	0.326	0.50730
1	-1	-1	1.01	0.326	-0.0079
-1	1	-1	0.3	1.53	0.5349
1	1	1	1.01	1.53	-0.0054
-1.0512	0.0301	-0.0251		**Total**	
-0.2628	0.007527	-0.006275		**Total/2^2**	
95.6%	3.2%	1.2%		**Efecto**	

(b)

Matriz de Experimento			**Plan de Experimento**		**Mejora de ancho de Banda**
Z_1	Z_2	$Z_1 Z_2$	$\alpha\beta$	γ	**Rendimiento**
-1	-1	1	0.3	1.53	0.5349
1	-1	-1	1.01	1.53	-0.0054
-1	1	-1	0.3	3.45	0.51909
1	1	1	1.01	3.45	-0.00009
-1.05948	-0.0105	0.02112		**Total**	
-0.26487	-0.00263	0.00528		**Total/2^2**	
97.1%	1.8%	1.1%		**Efecto**	

El efecto principal de cada factor indica la variación promedio de la respuesta cuando cambia este factor. El primer paso para interpretar los efectos principales es comprobar que la variación observada en la respuesta se debería a una influencia real del factor y no al error experimental. Para comprobar los efectos de cada factor con una estimación del error experimental se utilizan las pruebas estadísticas. Así, determinaremos si los dos efectos son significativos y sus efectos no son fruto de la imprecisión en la experimentación. En cualquier caso, podemos interpretar sus valores, dado que cuanto más varía la respuesta, mayor sería el efecto.

El efecto de interacción entre los dos factores es la cuarta información que se puede obtener del diseño factorial 2^2. No existe mucha interacción en la prueba que se ha mostrado anteriormente, porque con el modelo LAWS se sabe que al cambiar γ cambia de forma no constante la mejora (Sección 1.6.1) y no es muy adecuado tomar sólo dos niveles que den casi iguales valores de salida.

La experimentación del diseño factorial completo 2^2 ha permitido descubrir que el nivel de la mejora del ancho de banda baja al aumentar la factor de aplicación y aumenta al aumentar el factor $\alpha\beta$. Puesto que el mayor efecto lo tiene el primer factor $\alpha\beta$, este factor es el que se debe controlar más cuidadosamente, y es el primer que hay que considerar para optimizar el rendimiento.

4.4 Diseño de tres parámetros para el análisis de la interfaz de red

En este caso se tienen en cuenta tres factores. Concretamente, consideramos el efecto de los factores αβ, γ y el ancho de banda de la red en la mejora del ancho de banda aprovechado por el sistema de comunicación. Cualquiera de los tres factores puede influir en el resultado, y posiblemente de modo distinto según qué conjunto de parámetros se considere.

Los factores escogidos y su domino experimental se muestran en la tabla 4.6.

Tabla 4.6. Factores y dominio experimental.

Parámetros	**Mínimo (-1)**	**Máximo (+1)**
Z_1: αβ	0.3	1.01
Z_2: γ	0.326	3.45
Z_3: Ancho de banda de red BW_{red}	1Gbps	10Gbps

Como se ha hecho en el primer experimento en la sección 4.3, el dominio experimental de un factor continuo se expresará con los valores mínimo y máximo que puede tomar de los tres factores ($\alpha\beta$, γ, y BW_{red}) como se muestra en la Tabla 4.6, y se asigna la notación -1 y +1 al nivel inferior y superior respectivamente. Esta codificación permitirá obtener una estimación numérica de su efecto incluso a pesar de que el factor no tome valores numéricos.

Las Tablas 4.7 y 4.8 muestran el plan del experimento y el diseño factorial de 3 factores. Para estudiar el efecto de un factor es suficiente con hacerlo variar entre dos valores. Lo más adecuados son los extremos de su dominio experimental entre los niveles inferior y superior. Además, se debe experimentar con variaciones en cada posible combinación de los valores de los demás factores. Esto permitirá descubrir si el efecto de un factor depende del valor que tomen los otros factores. Todas estas combinaciones se contemplan en el diseño factorial completo 2^3.

Tabla 4.7 El plan experimento del diseño de 3 factores.

Parámetros		**Z_3: BW_{red} (-1) 1Gbps**		**Z_3: BW_{red} (+1) 10Gbps**	
		Z_1: αβ (-1)	**Z_1: αβ (+1)**	**Z_1: αβ (-1)**	**Z_2: αβ (+1)**
Z_2: γ (-1)	0.4409	0.3687	0.0567	0.5073	-0.0079
Z_2: γ (+1)	4.6527	0.3637	0.05785	0.51909	-0.00009

En total se tienen ocho experimentos correspondientes a 2 niveles del ancho de banda del enlace BW_{red} (Z_3), dos niveles de factor αβ (Z_1), dos niveles del factor γ (Z_2). De la Tabla 4.8 se obtiene como conclusión que las columnas no están correlacionadas sino que son ortogonales, con tantos valores +1 como valores -1. Esto permite estimar un efecto independientemente de los otros. En la misma tabla se muestran los resultados que se obtiene al reemplazar los valores -1 y +1 con los valores de las variables reales. Los ocho experimentos se pueden combinar para obtener ocho datos: el valor medio, los tres efectos principales, los tres efectos de interacción de dos factores, y un efecto de interacción de tres factores.

El orden en el que se sumen y resten las respuestas viene dado por la matriz de los efectos. La matriz codificada de la Tabla 4.8 tiene tantas filas como experimentos, y tantas columnas como efectos se estimarán. Cada efecto se calcula sumando o restando las respuestas de acuerdo con el orden de signos de su columna. Esta forma rápida y sistemática de calcular los efectos será muy útil cuando tratemos los diseños factoriales fraccionados en la próxima sección.

Tabla 4.8 Matriz de experimentos de un diseño factorial completo 2^3 y respuestas medidas.

Experimento	Z_1	Z_2	Z_3	$Z_1 Z_2$	$Z_1 Z_3$	$Z_2 Z_3$	$Z_1 Z_2 Z_3$	Throughput
1	-1	-1	-1	1	1	1	-1	0,3687
2	1	-1	-1	-1	-1	1	1	0.05673
3	-1	1	-1	-1	1	-1	1	0.3637
4	1	1	-1	1	-1	-1	-1	0.05785
5	-1	-1	1	1	-1	-1	1	0.5073
6	1	-1	1	-1	1	-1	-1	-0.0079
7	-1	1	1	-1	-1	1	-1	0.51909
8	1	1	1	1	1	1	1	-0.00009
Total	-1.652	0.00195	0.17142	0.00214	-0.41656	-0.05017	0.07133	
(Medio) Total/8	-0.2065	0.00195	0.02142	0.00026	-0.05207	-0.0062	0.00891	
Efecto	92.8%	0%	1%	0%	5.9%	0%	0.1%	

Los resultados del diseño factorial completo en orden creciente de complejidad se interpretan a continuación. El valor medio indica alrededor de qué valor han variado las respuestas. Generalmente también corresponde al valor predicho en el centro de dominio. Los efectos principales de los tres factores miden cómo afecta cada factor a la respuesta. El mayor cambio de rendimiento en la mejora del ancho de banda se produce al variar αβ (Z_1) de 0.3 a 1.01. El efecto de variar los otros factores y las interacciones entre ellos mismos se muestra en la fila *Efecto* de la Tabla 4.8.

Es muy interesante interpretar estos efectos a partir de la Tabla 4.8. Observando sólo los efectos principales, las mejores prestaciones parecen obtenerse con valores altos de $\alpha\beta$ y γ, y el ancho de banda de la red BW_{red}. Sin embargo, esta interpretación se deberá matizar al considerar las interacciones.

Los efectos de interacción de dos factores Z_1Z_2, Z_1Z_3 y Z_2Z_3 miden la influencia que tiene una combinación de factores en la respuesta. Existe interacción cuando el efecto de un factor es diferente a los distintos niveles de otros factores. Se puede comprender fácilmente si evaluamos el efecto de cada factor por pares de experimentos. Por ejemplo, el efecto del ancho de banda depende de αβ, o el efecto de αβ depende de la γ.

En conclusión, antes de interpretar los efectos principales, hay que considerar si existen efectos de interacción significativos. Si es así, no se puede interpretar individualmente el valor de cada efecto principal. Respecto a los resultados lo que hemos obtenidos, la interacción más efectiva se encuentra entre el factor αβ y el ancho de banda del enlace.

Para terminar el análisis del diseño factorial completo 2^3 tenemos el efecto de interacción de tres factores. El valor $Z_1Z_2Z_3$ indica en qué grado el efecto de un factor depende del valor combinado de los otros dos factores. En nuestro experimento este efecto es muy pequeño comparando con los demás. Es habitual que los efectos de interacción sean cada vez menos importantes cuantos más factores se consideran en la interacción.

Finalmente, los experimentos del diseño factorial 2^3 nos han permitido descubrir que el rendimiento mejora al aumentar el valor de $\alpha\beta$ y el ancho de banda de la enlace, y que el sistema proporciona mayores rendimientos.

4.5 Diseño experimental de más de tres parámetros

En el caso considerar más de 3 factores, un diseño completo exigiría la realización de 2^6 =64 experimentos, y tal volumen de experimentación resulta, en la mayoría de ocasiones, prohibitivo. Los diseños factoriales fraccionales permiten estudiar un elevado número de factores en un número de experimentos mucho menor de lo que requería un factorial completo. En la práctica resulta extremadamente raro que aparezcan interacciones de tres o más factores que resulten ser significativos. En general, se obtienen modelos suficientemente aproximados considerando sólo los efectos principales y las interacciones de dos o tres factores.

Para el diseño fraccional se utiliza 2^{k-p} como se ha explicado antes, donde 2 sigue siendo el número de niveles, k el número de factores con los que se experimentará y la letra p indica el grado de fraccionamiento, de tal manera que el resultado de elevar 2 a k-p indica el número de experimentos que se van a realizar.

El diseño 2^{6-3} permite estudiar seis parámetros en solamente ocho experimentos. Suponiendo que todas las interacciones sean cero, permitiría estimar los efectos principales de las seis variables. En la Tabla 4.9 se muestra los factores escogidos y su domino experimental.

Tabla 4.9. Los seis factores y el dominio experimental.

Parámetros	Minimum (-1)	Maximum (+1)
Z_1: $\alpha\beta$	0.3	1.01
Z_2: γ	0.326	3.45
Z_3: Ancho de banda de la red BW_{red}	1Gbps	10Gbps
Z_4: Cache L1	16Kbs	64Kbs
Z_5: Cache L2	256Kbs	1024Kbs
Z_6: Velocidad de Memoria	5 Cyc.	500 Cyc.

Como se ha mostrado en los dos experimentos anteriores, el dominio experimental de un factor continuo se expresa con los valores mínimo y máximo que puede tomar, y se asigna la notación -1 y +1.

Por lo general se deben usar diseños fraccionarios con la mayor resolución posible, congruentes con el fraccionamiento requerido. A mayor resolución, las suposiciones relativas a las interacciones que deben ignorarse con el propósito de hacer una interpretación única de los datos son menos restrictivas.

Es posible construir un medio de mayor resolución, de diseño 2^k, escribiendo primero las combinaciones de tratamiento del diseño 2^{k-p} completo y agregando después el factor k, identificando sus niveles positivos y negativos mediante los signos positivos y negativos de la interacción de mayor orden. Por lo tanto, el diseño factorial fraccionario 2^{6-3} se obtiene escribiendo el diseño 2^3 completo e igualando después los factores Z_3, Z_4, Z_5 y Z_6 con la interacción de ellos mismos como se muestra en la Tabla 4.10.

Tabla 4.10 Matriz de experimento de un diseño factorial completo 2^{6-3} y respuesta medida.

Experimento	Z_1	Z_2	Z_3	$Z_1 Z_3$	Z_4	Z_5	Z_6	La mejora del ancho de banda
1	-1	-1	-1	1	1	1	-1	0,374
2	1	-1	-1	-1	-1	1	1	0.0129
3	-1	1	-1	-1	1	-1	1	0.622
4	1	1	-1	1	-1	-1	-1	0.0089
5	-1	-1	1	1	-1	-1	1	0.8137
6	1	-1	1	-1	1	-1	-1	0,2819
7	-1	1	1	-1	-1	1	-1	1,7613
8	1	1	1	1	1	1	1	0.0094
Total	-2.5099	0.9191	1.8485	-1.4721	-1.3095	0.4311	-0.9681	
(Medio) Total/8	-0.31373	0.11488	0.23106	-0.18401	-0.16368	0.05388	-0.12101	
Efecto	40.46%	5.42%	21,94%	13.91%	11.01%	1.10%	6.02%	

En general, cualquier efecto de interacción puede usarse para generar la columna del factor k. Sin embargo, si no utiliza el efecto de interacción no se produce

el diseño de mayor o más alta resolución. La Tabla 4.9 muestra el efecto de cada factor separado y sustituye el efecto del cuarto factor por el efecto de la interacción del factor Z_1Z_3.

Cualquier diseño factorial fraccionario de resolución k contiene diseños factoriales completos de cualquier subconjunto de k-p factores. Este concepto es muy importante y útil. Sí el diseño tiene varios factores de posible interés, pero considera que sólo k-p de ellos tienen consecuencia importantes. La elección del diseño factorial fraccionario de resolución k resulta apropiada. Este diseño se proyectará en un diseño factorial completo de los k-p efectos significativos.

Por lo tanto, el efecto de cada factor y las interacciones entre ellos se nota por el cambio de la mejora del ancho de banda (el interfaz de red). Además, se puede usar el diseño experimental para elegir en qué condiciones se encuentra la mejora máxima del ancho de banda y como se pueden variar para controlar el rendimiento del interfaz de la red.

4.6 El diseño ANOVA

En la Tabla 4.11 se muestran los resultados experimentales para la mejora del ancho de banda para diferentes niveles de la aplicación, γ, y tres valores distintos de $\alpha\beta$. En la misma tabla se mide el efecto de cada factor, γ y $\alpha\beta$, usando el análisis de varianza.

Tabla 4.11 Los resultados experimentales de la mejora del ancho de banda.

Aplicación (γ)	$\alpha\beta_{1(0.3)}$	$\alpha\beta_{2(049)}$	$\alpha\beta_{3(1.01)}$
0,06737	0,8411	0,4933	-0,01897
0,16261	0,86166	0,5075	-0,00156
0,3108	0,8525	0,5012	0,0001197
0,5466	0,84529	0,4841	-0,00455
0,7878	0,8198	0,47586	-0,0145
1,4246	0,8117	0,46429	-0,01498
1,7232	0,8135	0,44	-0,01445

Es importante comprobar que las hipótesis se verifican para poder sacar conclusiones fiables a partir de un análisis de la varianza. A partir de entonces, el análisis de la varianza nos ayudará a encontrar que nivel de aplicación se va influido cuando se cambian los valores de $\alpha\beta$, es decir, con el cambio del valor de la relación entre la velocidad del nodo y la velocidad del procesador de la NIC se cambia la mejora de las prestaciones de comunicación.

Tabla 4.12 Análisis de varianza de dos factores.

RESUMEN	Cuenta	Suma	Promedio	Varianza
Fila 1	3	1,3154	0,4384	0,1871
Fila 2	3	1,3676	0,4558	0,1882
Fila 3	3	1,3538	0,4512	0,1835
Fila 4	3	1,3248	0,4416	0,1819
Fila 5	3	1,2811	0,4270	0,1758
Fila 6	3	1,2610	0,4203	0,1722
Fila 7	3	1,2390	0,4130	0,1719
Columna 1	7	5,8455	0,8350	0,000399373
Columna 2	7	3,3662	0,4808	0,000542959
Columna 3	7	-0,06889	-0,0098	5,80804E-05

En la Tabla 4.12 se muestran los análisis de varianza de dos parámetros con el promedio y la varianza de cada grupo (fila o columna). Después, se usa la Tabla 4.12 para calcular el grado de libertad de cada factor, y el valor F. A partir de estos datos se pude comprobar sí el valor de F es significativo o no, como se muestra en la Tabla 4.13.

Tabla 4.13Análisis de varianza

Origen de las variaciones	Suma de cuadrados	Grados de libertad	Promedio de los cuadrados	F	Probabilidad
Aplicación	0,004549488	6	0,000758248	6,262268305	0,003543059
Alfa*Beta	2,520367687	2	1,260183844	10407,68882	3,65831E-20
Error	0,001452984	12	0,000121082		
Total	2,526370159	20			

En caso de que existan más de dos grupos, como estamos trabajando bajo la hipótesis de que en todos los grupos la variabilidad es la misma (es decir, se supone que tienen la misma varianza) habría que utilizar los datos contenidos en todos los grupos para estimar esa varianza, en vez de usar simplemente los resultados de los grupos. A partir de la Tabla 4.12, se tiene que los valores de F son más grandes que la probabilidad correspondiente, por lo tanto, el análisis de los resultados de la Tabla 4.10 son significativos. Es decir, respeto al diseño de la interfaz de red, cuando se aumenta el valor de $\alpha\beta$ se disminuye la mejora en las prestaciones de comunicación por el efecto del factor β (reducción del coste de la comunicación debido del uso de *offloading* y *onloading*) y cuando α=1.

4.7 Conclusión

Los diseños factoriales comprenden los experimentos óptimos para estudiar qué variables influyen en el sistema. Estos diseños definen la estrategia experimental para estudiar simultáneamente el efecto de varios factores en el comportamiento del sistema y sus interacciones. Para estudiar un número elevado de factores, es más eficaz utilizar sólo una fracción de un diseño factorial completo. En este capítulo se ha mostrado cómo escoger los factores en el caso de los diseños factoriales completos y fraccionarios. Además, se ha mostrado el uso del análisis de la varianza ANOVA para comprobar la influencia de los factores del modelo LAWS.

En nuestros experimentos, con sólo 4 y 8 experimentos hemos determinado los efectos principales de 2, 3 o más factores, y sus interacciones. Estos efectos al principio pueden no ser obvios para el experimentador y no se habrían descubierto variando un factor cada vez. Las experimentaciones posteriores han permitido estudiar el efecto del factor αβ (coeficiente de retardo y coeficiente estructural) y γ (la relación entre el tiempo de procesamiento de la aplicación y el de procesamiento de comunicación) y los demás factores en relación a la mejora del ancho de banda.

De los análisis de los experimentos se puede concluir que al cambiar el valor de la relación entre la velocidad del nodo y la velocidad del procesador de la NIC se afecta de forma muy clara la mejora de las prestaciones de comunicación alcanzables con *offloading*. Además, también cambia la mejora de las prestaciones cuando cambia

el valor de la relación entre el tiempo de procesamiento de la aplicación y el de la sobrecarga de comunicación.

En el próximo capítulo se presentarán las conclusiones y principales aportaciones del trabajo realizado y las líneas de futuras de trabajo que quedan abiertas para proseguir nuestra investigación

Capítulo 5

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se recogen las conclusiones y principales aportaciones de este trabajo de investigación en el que se ha abordado el estudio experimental, a través de la simulación de modelos basados en lenguaje de descripción de hardware (HDL) que hemos elaborado para las distintas alternativas de *externalización* (*offloading* y *onloading*) de la interfaz de red, con el objetivo mejorar las prestaciones del sistema de comunicación evitando cuellos de botella en el nodo. Esta memoria corresponde a la cuarta tesis dentro de la línea de trabajo en arquitecturas de comunicación de altas prestaciones del grupo CASIP (TIC117). En la tesis de Antonio F. Díaz (*Comunicación Eficiente en Redes de Área Local para el Procesamiento Paralelo en Clusters*, 2001) se propuso una capa de mensajes, denominada CLIC, para reducir la sobrecarga de comunicación en clusters interconectados con redes Ethernet [DIA01]. A diferencia de la tendencia que marcan las interfaces de red a nivel de usuario, tratando de evitar la intervención del sistema operativo en la comunicación, CLIC optimizaba el soporte del sistema operativo

Linux para la interfaz de red facilitando la portabilidad de la capa de mensajes a sistemas con distintas tarjetas de red. Posteriormente, las tesis de Andrés Ortiz (*Alternatives for Network Interface Offloading. Analysis and Optimization by Full System Simulation*, 2008) [ORT08] y de Pablo Cascón (*Improving Communications by using Network Processors*, 2010) [CAS10] persiguieron el diseño de interfaces de red eficientes en las que el procesamiento de las tareas de comunicación se realiza, en mayor o menor medida, en otros procesadores disponibles en el nodo, distintos de la CPU en la que se ejecutan las aplicaciones y el resto de tareas del sistema operativo. Mientras que en la tesis de Pablo Cascón se investigó el uso de procesadores de red en la mejora de las prestaciones de las aplicaciones de comunicación, en la tesis de Andrés Ortiz se analizaron las alternativas de *externalización* mediante *offloading* y *onloading* utilizando el simulador de sistema completo SIMICS. La investigación que se presenta aquí sigue esta línea de trabajo, aunque en este caso se aborda el análisis de las distintas alternativas para la mejora de la interfaz de red mediante la simulación de modelos HDL del camino de comunicación. Con esto se pretende un acercamiento más directo a los elementos hardware que afectan a las prestaciones de la interfaz y analizar la forma en que los retardos y la estructura de colas de buffers y memorias del nodo pueden organizarse para mejorar el rendimiento de las comunicaciones.

5.1 Aportaciones

La simulación HDL de las alternativas de *externalización* y la comparación de los resultados obtenidos con los proporcionados por otro tipo de simuladores, como es el caso del simulador de sistema completo SIMICS, y con las predicciones que se pueden extraer del modelo LAWS, nos han permitido poner de manifiesto los aspectos a tener en cuenta para mejorar las prestaciones de las interfaces de red y, en general, las arquitecturas de comunicación de los nodos.

Por un lado, es evidente que tanto la alternativa de *externalización* basada en *offloading*, como la que utiliza la técnica de *onloading*, proporcionan mejoras en el ancho de banda de comunicación que aprovecha el nodo del ancho de banda máximo que proporciona el enlace de red. Para las condiciones en que se han realizado los

experimentos, las prestaciones en cuanto al ancho de banda de comunicación son más elevadas con *onloading* que con *offloading*.

En cuanto a la validación de los resultados, se han realizado comparaciones con los resultados obtenidos en trabajos previos del grupo mediante el simulador de sistema completo SIMICS. Este simulador, al tener en cuenta los detalles de la ejecución de las aplicaciones y del sistema operativo, constituye un punto de referencia adecuado y suficientemente realista para evaluar la validez de los resultados de nuestras simulaciones HDL, en las que la ejecución del sistema operativo y las aplicaciones se consideran a través de retardos en distintos puntos del camino de comunicación. Por otro parte, también se ha utilizado el modelo teórico LAWS para analizar la plausibilidad de nuestros resultados. Por otra parte, el modelo LAWS bastante útil para organizar la exploración del espacio de diseño de la *externalización*, orientando las condiciones y los valores de los parámetros que deben utilizarse para realizar las simulaciones que pueden poner de manifiesto los aspectos relevantes del comportamiento de la interfaz de red. En relación a las comparaciones de nuestros resultados con el modelo LAWS, si bien existen diferencias cuantitativas importantes entre los valores obtenidos experimentalmente y los predichos por dicho modelo, LAWS nos proporciona una justificación cualitativa adecuada de los resultados experimentales obtenidos. Evidentemente, existen aspectos importantes del espacio de diseño, como la influencia de los tamaños de los paquetes, y la simulación de trazas correspondientes tanto a aplicaciones reales, que aunque quedan fuera de los parámetros que se tienen en cuanta en el modelo LAWS. No obstante, también se han realizado simulaciones en las que se analiza el efecto de dichos elementos.

El análisis de los resultados nos ha llevado a proponer una mejora en la interfaz de red que permite hibridar las características de las *externalizaciones* mediante *offloading* y *onloading*, para intentar aprovechar sus aspectos beneficiosos, evitando alguno de los problemas que plantean.

Además, en esta memoria se han utilizado los diseños factoriales para definir una estrategia experimental que permita reducir el número de simulaciones a realizar para alcanzar resultados suficientemente significativos de lo que ocurre en el espacio de diseño, y para estudiar simultáneamente el efecto de varios factores o parámetros y sus interacciones en el comportamiento del sistema de comunicación. En este ámbito, también se ha aplicado ANOVA para comprobar el nivel de significación de la

influencia en las prestaciones de los factores que se proponen en el modelo LAWS. Así, los experimentos realizados tras el análisis estadístico nos han permitido estudiar el efecto de los factores $\alpha\beta$ y γ, en relación con la mejora del ancho de banda.

A continuación se detallan las aportaciones del trabajo realizado, incluyendo las conclusiones más relevantes que se pueden extraer de los resultados que hemos obtenido a partir de los experimentos realizados con los modelos de simulación HDL que se han elaborado:

- Se ha utilizado el lenguaje de programación (HDL) para describir todos los módulos que intervienen en el camino de comunicación. Los modelos realizados nos ha permitido utilizar un simulador HDL para analizar y evaluar, mediante los resultados de las simulaciones realizadas, los efectos de la *externalización* de los protocolos de comunicación en el comportamiento de la interfaz de red. En cada uno de los modelo de simulación que se han elaborado se incluyen los módulos que intervienen en las arquitecturas reales de los computadores. Así, se han elaborado módulos descriptivos del procesador (*module Procesador_CPU*), la memoria principal (*module Main_Memory*), los buses y los circuitos de interfaz entre ellos (*module Bridge_Chipset, module Memory_Bus,* y *module E/S_Bus*), y la tarjeta de red (*module Network_Interface_Card*). Además, también se ha incluido un módulo para generador los paquetes de comunicación, bien con perfiles sintéticos, bien con datos extraídos de aplicaciones reales que pueden encontrarse en Internet o pueden generarse específicamente (*module Red_Crossover*).

- En cuanto al módulo HDL que describe las características de la tarjeta de red, se han escrito distintas versiones según las características de la alternativa de *externalización* (*offloading* u *onloading*) a considerar. Lo mismo se ha hecho para los módulos que describen a los procesadores.

- Mediante la simulación de los distintos modelos HDL, se han estudiado las prestaciones del sistema de comunicación completo implementando las distintas técnicas d*e offloading* y *onloading*, que se han comparado con la alternativa de base, sin *externalización*. Los módulos de simulación que se han

desarrollado permiten analizar la interacción entre las aplicaciones, los elementos del sistema de comunicación y los distintos módulos de NIC, ya se traten de elementos software o hardware. En las simulaciones, se ha usado un generador de paquetes que se ha incluido en el módulo de memoria *(module Main_Memory*) de la parte del emisor. De esta manera, se pueden realizar simulaciones con diferentes perfiles de comunicación. Así, se pueden generar paquetes de varios tamaños, frecuencias, etc., y también se pueden utilizar trazas de aplicaciones reales de las que se pueda disponer [INT10].

- Se han interpretado y analizado todos los resultados obtenidos de las simulaciones HDL de los modelos para *offloading* y *onloading.* Los resultados muestran mejoras tanto en el ancho de banda como en la latencia en las dos alternativas de la *externalización*. Más concretamente, la *externalización* proporciona una mejora que crece linealmente para valores bajos de la relación entre coste de computación y coste de comunicación ($\gamma >> 1$), pero cuando la aplicación es menos intensiva en comunicación, la mejora relativa del ancho de banda se ve limitada (según el modelo LAWS, ese límite es $1/\gamma$). Además, se ha mostrado que la *externalización* no es útil cuando el computador puede gestionar todo el ancho de banda de la enlace ($\sigma >> 1$, con el parámetro σ definido en el modelo LAWS), ya que en ese caso el cuello de botella del camino de comunicación no está en los nodos sino, precisamente, en el enlace. Por tanto, la *externalización* es útil solo cuando el computador no sea capaz de comunicarse a la velocidad del enlace ($\sigma << 1$). Además, se ha mostrado que la memoria cache tiene un efecto considerable en la mejora de las prestaciones del sistema de comunicación. Esta mejora se pone de manifiesto en tanto en el ancho de banda, como en la latencia, y en la sobrecarga (*overhead*) de comunicación. Además, en este caso, la *externalización* mediante *offloading* proporciona menor ancho de banda que mediante *onloading.*

- Se ha presentado una propuesta de la interfaz de red híbrida de *externalización*, que se combina las alternativas de *offloading* y *onloading*. Se ha elaborado el correspondiente modelo de HDL a partir de los módulos desarrollados para los caminos de comunicación con *offloading* y *onloading*, y

se ha realizado un análisis de las prestaciones de esta propuesta híbrida, comparándola con las alternativas de *externalización* analizadas previamente. Como conclusión del análisis comparativo realizado se puede indicar que la nueva propuesta mejora los anchos de banda obtenidos por las opciones de *externalización*, con unos resultados de latencia prácticamente iguales al mejor caso de la *externalización*. Por lo tanto, se trata de una propuesta a tener en cuenta para el desarrollo de interfaces de red en nodos con varios procesadores, dado que permite distribuir eficiente la carga de comunicación entre ellos.

- Se ha usado el modelo teórico LAWS para validar el comportamiento de los modelos de simulación. Para definir las ecuaciones que permiten estimar la mejora máxima en cuanto a ganancia de ancho de banda, el modelo LAWS considera que antes de la *externalización*, el sistema está constituido por un cauce de dos etapas, el computador y la red, el máximo ancho de banda antes está determinado por el cuello de botella del cauce, y vendrá dado por $B_{before}=min(B,1/(aX+oX))$. Después de la *externalización*, se tendría un cauce de tres etapas (el procesador central, el procesador de la NIC, y la red) y el ancho de banda máximo vendrá dado por $B_{after}=min(B,1/(aX+(1-p)oX),1/poY\beta)$. Por tanto, la mejora relativa máxima en el ancho de banda vendrá dada por la expresión $\delta b=(B_{after}-B_{before})/B_{before}$.

- La comparación de los resultados que se han obtenido con los modelos HDL elaborados y los obtenidos a partir de un simulador de sistema completo como SIMICS, que permite una simulación detallada de la ejecución de las aplicaciones y del sistema operativo, pone de manifiesto coincidencias cualitativas en los resultados obtenidos. En cuanto a la comparación cuantitativa, las diferencias entre los resultados experimentales con el simulador HDL y con SIMICS son relativamente pequeñas. Por lo tanto, dado el nivel de detalle y realismo que proporciona el simulador de sistema completo, se puede concluir que los modelos HDL constituyen una herramienta adecuada para estudiar el comportamiento de la interfaz de red y las posibles mejoras que se pueden implementar en la misma. Se trata, por

tanto de un recurso útil tanto para la investigación como para la docencia de los conceptos relacionados con el camino de comunicación.

- Tras interpretar e identificar las prestaciones de la *externalización* mediante *offloading* y *onloading*, se ha usado el diseño de experimentos para estudiar y analizar qué variables influyen de forma más significativa en el sistema de comunicación. Además, se ha usado ANOVA para comprobar la influencia de los factores del modelo LAWS. Estos diseños experimentales han permitido analizar el efecto de los factores $\alpha\beta$ y γ. Como resultado de ese análisis se puede concluir que al cambiar el valor de la relación entre la velocidad del nodo y la velocidad del procesador de la NIC se afecta de forma muy clara la mejora de las prestaciones de comunicación alcanzables con *offloading*. Además, también cambia la mejora de las prestaciones cuando cambia el valor de la relación entre el tiempo de procesamiento de la aplicación y el de la sobrecarga de comunicación.

- Los resultados experimentales se han obtenido a través de un generador de paquetes que permite definir secuencias sintéticas de paquetes de varios tamaños y anchos de banda entre 1 y 10Gbps, y también leerlos de ficheros de trazas de aplicaciones reales. Los experimentos ponen de manifiesto que el tipo de tráfico de paquetes utilizado en las simulaciones da lugar a mejoras de prestaciones cuantitativamente distintas, dependiendo al tamaño y la velocidad de los paquetes, aunque las conclusiones respecto a las características cualitativas de las mejoras son similares, y corresponden a las que se han indicado anteriormente.

Los detalles del modelo de HDL que hemos desarrollado y los resultados experimentales que hemos obtenido han sido publicados en los siguientes artículos:

- Waseem M. Haider; Andrés Ortiz; Pablo Cascón; Julio Ortega; Antonio F. Díaz: “El modelo LAWS y la simulación HDL de la *externalización* de las comunicaciones".CEDI, Zaragoza-España, pp. 319-326, 11-14 Septiembre, 2007.

- Waseem M. Haider; Julio Ortega; Antonio Francisco Díaz: "Analyzing approaches for network interface design by HDL simulation". The IASTED International Conference on Parallel and Distributed Computing and Networks, PDCN, Insbruk-Austria, pp. 226-233, 16-18 February, 2009.

- Pablo Cascón; Julio Ortega; Waseem M. Haider; Antonio F. Diaz; Ignacio Rojas: "A Multi-threaded network interface using network processors ". Parallel, Distributed and Network-based Processing, PDP, Weimar-Alemania, pp.196-200, 18-20 February, 2009.

- Waseem M. Haider; Julio Ortega; Antonio F. Díaz; Andrés Ortiz: "Análisis de opciones de *externalización* de Interfaces de Red con modelos HDL". XX Jornadas de paralelismo, A Coruña-España, pp. 475-480,16-18 Septiembre, 2009.

- Waseem M. Haider; Julio Ortega; Antonio F. Díaz; Andrés Ortiz: "Performance Analysis by HDL Simulation of Network Interface Design Alternatives". IAENG, The International MultiConference of Engineers and Computer Scientists 2010, Hong Kong, pp. 410-419, 17-19 March, 2010.

5.2 Trabajo Futuro

Como resultado de la investigación que hemos descrito en esta memoria han surgido distintos temas que se van a explorar en nuestro trabajo futuro, tal y como se indican a continuación:

- Una de las líneas de trabajo consiste en la elaboración de un modelo HDL suficientemente detallado de un procesador de red. De esta forma se podría estudiar el efecto de las arquitecturas heterogéneas de varios núcleos multihebra, y el aprovechamiento del paralelismo que proporcionan en la mejora de las prestaciones de comunicación.

- Otros aspectos a considerar a partir de los modelos y las simulaciones HDL realizadas son, tanto la propuesta de mejoras en las arquitecturas de comunicación, como la búsqueda de modelos teóricos de prestaciones más precisos que el modelo LAWS, que se ha considerado como referencia en este trabajo.

- También pretendemos estudiar el efecto de otras alternativas de paralelización para mejorar la interfaz de red. Específicamente, se considerará la arquitectura de los procesadores *multinúcleo* para implementar otras posibles optimizaciones de la interfaz de red, teniendo en cuenta la afinidad entre el procesador que ejecuta la correspondiente tarea de comunicación, el que recibe la interrupción asociada a la comunicación por parte de la tarjeta de red, y la ubicación de los datos que se utilizan.

Para terminar, se puede afirmar que se han obtenido algunas conclusiones significativas sobre el aprovechamiento de la presencia de varios procesadores en el nodo para desarrollar distintas alternativas de mejora de las prestaciones del camino de comunicación. Además, podemos concluir que el uso de las técnicas de *externalización* (*offloading* u *onloading*) tiene un efecto relevante en las prestaciones de comunicación, aunque dependen del perfil de comunicación de la aplicación considerada.

Por otro parte, los modelos HDL ofrecen una descripción del sistema a evaluar con un nivel de precisión adecuado para interpretar las transferencias de datos y control en el nodo, y la interacción de los distintos elementos del camino de comunicación, permitiendo extraer conclusiones útiles acerca de las consecuencias de la situación de los cuellos de botella y el diseño de propuestas de mejora.

Apéndice I

Módulos *Verilog* para la simulación del sistema de comunicación

En este apéndice se resumen las configuraciones y las características detalladas de algunos de los módulos para la simulación HDL del sistema de comunicación como se muestra en la Figura 1, y que se han presentado en el segundo capítulo. Listado 1 se muestra el código de interconexión entre todos los módulos de la simulación del camino de comunicación.

1. Módulo de la tarjeta de red NIC

Este módulo permite el movimiento de los datos entre la red y los demás módulos, como se muestra en el código de la Listado 2. En el caso de la *externalización* mediante *offloading* el procesamiento de los datos se lleva a cabo en este mismo módulo y luego se transfieren los datos al módulo de la memoria principal.

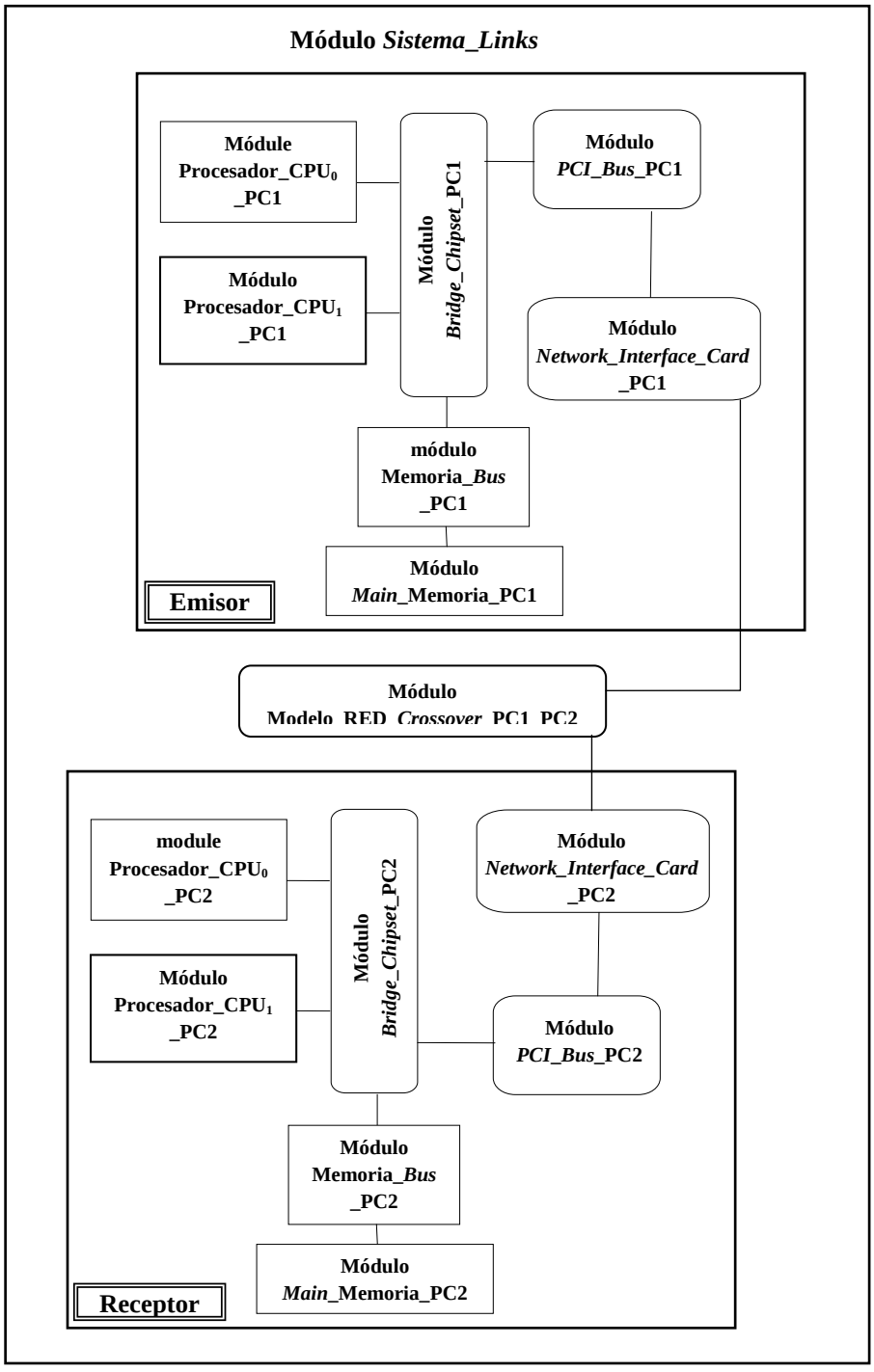

Figura I.1. Los módulos *Verilog* para la simulación HDL de todo el camino de comunicación.

Listado 1. El módulo de la interconexión entre los módulos.

```
///module Sistema_Links;
        wire    [63:0]   d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12;
        wire             pR1,cR1;
        wire             pR2,cR2;
        wire             pR3,cR3;
        wire             pR4,cR4;
        wire             pR5,cR5;
        wire             pR6,cR6;
        wire             pR7,cR7;
        wire             pR8,cR8;
        wire             pR9,cR9;
        wire             pR10,cR10;
        wire             pR11,cR11;
        wire             pR12,cR12;
        wire             intr,intrack;
        wire             dma2,dmafin2;
        wire             dma3,dmafin3;
        wire             dma4,dmafin4;
        wire             dma5,dmafin5;
        wire             dma8,dmafin8;
        wire             dma9,dmafin9;
        wire             dma10,dmafin10;
        wire             dma11,dmafin11;
        wire             dma12,dmafin12;
// PC para enviar Datos a la Modelo de Red
Main_Memoria_PC1
   memoriaPC1(d1,pR1,cR1,dma1,dmafin1);
Bridge_Chipset_PC1
   bridgePC1(d2,pR2,cR2,dma2,dmafin2,d3,pR3,cR3,dma3,dmafin3,d1,pR1,cR1,dma1,dmafin1);
Memoria_Bus_PC1
   busPC1(d4,pR4,cR4,dma4,dmafin4,d3,pR3,cR3,dma3,dmafin3);
Procesador_CPU_PC1
   procesadorPC1(intr,intrack,d4,pR4,cR4,dma4,dmafin4);
PCI_Bus_PC1
        pciPC1(d5,pR5,cR5,dma5,dmafin5,d2,pR2,cR2,dma2,dmafin2);
Network_Interface_Card_PC1
        nicPC1(d6,pR6,cR6,d5,pR5,cR5,dma5,dmafin5,intr,intrack);

////////////////////////////////////////////////////////////
          Modelo_RED_Crossover_PC1_PC2
               crossover(d7,pR7,cR7,d6,pR6,cR6);
////////////////////////////////////////////////////////////

// PC para recevier los Datos de la Modelo Red
Network_Interface_Card_PC2
        nicPC2(d7,pR7,cR7,d8,pR8,cR8,dma8,dmafin8,intr,intrack);
PCI_Bus_PC2
        pciPC2(d8,pR8,cR8,dma8,dmafin8,d9,pR9,cR9,dma9,dmafin9);
Procesador_CPU_PC2
   procesadorPC2(intr,intrack,d11,pR11,cR11,dma11,dmafin11);
Memoria_Bus_PC2
   busPC2(d11,pR11,cR11,dma11,dmafin11,d10,pR10,cR10,dma10,dmafin10);
Bridge_Chipset_PC2
bridgePC2(d9,pR9,cR9,dma9,dmafin9,d10,pR10,cR10,dma10,dmafin10,d12,pR12,cR12,dma12,dmafin1
2);
Main_Memoria_PC2
   memoriaPC2(d12,pR12,cR12,dma12,dmafin12);

endmodule
```

Listado 2. El módulo de la interfaz de red NIC de la parte del emisor.

```
////////////// Modulo de Interface de red (Tarjeta de Red)

module Network_Interface_Card_PC1(dataOut,prodReady,consReady,dIn_interfaz,
pReady_interfaz,cReady_interfaz,dma,dmafin,intr,intrack);
output [63:0]  dataOut;
input  [63:0] dIn_interfaz;
output       prodReady,cReady_interfaz,intr;
output                    dmafin;
input       consReady,pReady_interfaz,intrack;
input                     dma;
reg                       dmafin;
reg         prodReady,cReady_interfaz,intr,escrito,leido,escribir,leer;
reg         estado,driver,interrupcion;
reg   [63:0]  dataInCopy;
reg   [63:0]  dataOut;
reg    [63:0]  buffer [0:256];
reg    [63:0]  buffer_id [0:256];
reg   [16:0]  cabecera,cola;
integer          N,k,M,tiempoNIC,tiempoTotalNIC,tiempoLatencyNIC1;
time             t0,t1,t2;
initial
                          begin
                          N=64;
                          M=8;
                          k=0;
                          dmafin=1;
                          tiempoNIC=0;
                          tiempoTotalNIC=0;
                          tiempoLatencyNIC1=0;
                          cabecera=0;
                          cola=0;
                          intr=0;
                          escribir=1;     /////////// parámetro importante
                          estado=0;
                          leer=0;          /////////// parámetro importante
                          escrito=0;
                          leido=0;
                          driver=0;
                          prodReady=0;////////// parámetro importante
                          cReady_interfaz=1;////////// parámetro importante
                          interrupcion=1;
                          end
always
                          begin
                          fork


//Guardar los datos en el buffer de la tarjeta de red


          forever
                          begin
                                    wait (escrito||leido)
                                    begin
                                                        if (escrito==1)
                                                        begin
                                                        cabecera=(cabecera+1)%N;
                                                        escrito=0;
                          end
```

```
                                        if (leido==1)
                                        begin
                    cola=(cola+1)%N;
                    leido=0;
                    end
                    if (((cabecera+1)%N)!=cola) escribir=1;
                    if (cola!=cabecera) leer=1;
                    end
                    end

                                        ////////////////////////////////
                    forever
                    begin
                    wait  (pReady_interfaz&&escribir)
                    #1000 buffer[cabecera]=dIn_interfaz;
                    buffer_id[cabecera]=k;
                    k=(k+1)%M;
                    escrito=1;
                    escribir=0;
                    cReady_interfaz=0;
                    wait(!pReady_interfaz)
                    #1000 cReady_interfaz=1;
                    end

                    forever
                    begin
                    wait(leer&interrupcion)

                    // Sí  hay externalizacion mediante la técnica offloading  con el
                    // activo de la señal driver para hacer el procesamiento del
                    //protocolo de comunicación

                    driver=1;
                    t0=$time;

                    // Aquí se cambia el afecto del tiempo de driver

                    #170000; driver=0; // tiempo del procesamiento de protocolo
                    de comunicación  (ALfa*Beta=0.49)
                    tiempoNIC=tiempoNIC+170;

                    // Manda el solicitud de la  interrupción

                    intr=1;
                    wait(intrack)

                    //retardo de protocolo & tiempo de interapcion
                    //inicializacion de DMA
                    #1000 estado=1;


                    wait(!intrack)
          //retardo de protocolo & tiempo de interrupción

          #1000 estado=1;
          interrupcion=0;
          t1=$time;
          tiempoNIC=tiempoNIC+2;
          tiempoTotalNIC=tiempoTotalNIC+((t1-t0)/1000);
          end
```

```
                        ////////////////////////////  Enviar los datos al otro ordenador

                        forever
                        begin
                        wait(leer)
                        if (((buffer_id[cola])%M)==0)
                        begin
                        interrupcion=1;
                        estado=0;
                        end
                        t2=$time;

                        wait (consReady)
                        t0=$time;
                        #1000 dataOut=buffer[cola];///Aqui se puede cambiar el retado para
                        cambiar Alfa///////
                        leido=1;
                        leer=0;

                        prodReady=1;        /////////c
                        wait(!consReady)
                        #1000 prodReady=0; ///////////c

                        t1=$time;
                        tiempoTotalNIC=tiempoTotalNIC+((t1-t0)/1000); // el tiempo total
                        tiempoLatencyNIC1=tiempoLatencyNIC1+(t2/1000);// el tiempo de la
                        //Latencia en la tarjeta de red
                        end


                        join
                        end
endmodule
```

En el caso del receptor, el módulo de la interfaz de red se cambia en la sección donde se guarden los paquetes y luego en el procesamiento de los mismos como muestra en el Listado 2.

2. Módulo de la memoria principal

Este módulo se permite el almacenamiento de los datos que han llegado de la tarjeta de red. En el caso del emisor (Listado 3), el generador de los paquetes se incluye en este módulo de memoria principal, mientras que en el caso de receptor (Listado 4) el módulo se usa para almacenar los paquetes procesados en la tarjeta de red.

Listado 3. El módulo de la memoria de la parte del emisor.

```
// Módulo del Main_Memoria del emisor

 module Main_Memoria_PC1(dataOut,prodReady,consReady,dma,dmafin);

          output [63:0]        dataOut;
          output                        dma;
          output       prodReady;
          input        consReady;
          input                         dmafin;
          reg   [63:0]   dataOut;
          reg         prodReady;
          reg                                   dma,dmaon;
          reg          [63:0]    buffer;
          integer                       j,N;
          integer       i,F;

          initial
                    begin
                       dma=0;
                              dma=1;
                              dmaon=0;
                              i=0;
                              buffer=2048;
                              F=buffer;
                              prodReady=0;
                              j=0;
                              N=8; // N es el número de transferencias en el DMA
                    end

          always
                    begin
                     fork
                     forever
                        begin
                                  while (i<F)
                                            begin
                                            wait(consReady)
                                            #31000 dataOut=i; // Donde se genera los datos
                                                      prodReady=1;
                                                      wait(dmaon&!consReady)
                                            #31000 prodReady=0;
                                                      i=i+1;
                                                      j=(j+1)%N;
                                                      if (j==0)
                                                                begin
                                                                          dma=1;
                                                                          dmaon=0;
                                                      end
                                            end
                                 end

                    forever
                                            begin
                                                      @(posedge dmafin)
                                                      #1000 dmaon=1;
                                                      dma=0;
                                            end

                    join
                    end

 endmodule
```

Listado 4. El módulo de la memoria de la parte del receptor.

```
// Módulo del Main_Memoria del receptor

module Main_Memoria_PC2(dataIn,prodReady,consReady,dma,dmafin);

        input [63:0]    dataIn;
        input           dma;
        input           prodReady;
        output          consReady;
        output          dmafin;
        reg     consReady;
        reg             dmafin,dmaon;
        reg      [63:0] buffer;
        integer         j,N;

        initial
                begin
                  dmafin=0;
                        #1000 dmafin=1;
                        dmaon=0;
                        consReady=1;
                        j=0;
                        N=8; // N es el número de transferencias en el DMA
                end
                always
                begin
                 fork
                        forever
                          begin
                                wait(dmaon&prodReady)
                                        #31000 buffer=dataIn;
                                        consReady=0;
                                        wait(!prodReady)
                                        #31000 consReady=1;
                                j=(j+1)%N; // N es el valor con que se ha inicializado el DMA
                                        if (j==0)
                                                begin
                                                        dmafin=1;
                                                        dmaon=0;
                                                end
                                        end
                        forever
                                begin
                                        @(posedge dma)
                                        #1000 dmaon=1;
                                        dmafin=0;
                                end

                 join
                end
endmodule
```

3. Módulo del procesador principal

Este módulo se usa para el procesamiento de los datos en el caso de no utilizar *externalización*. El Listado 5 muestra el código correspondiente al procesador central.

Listado 5. El módulo del procesador CPU.

```
// Modelo del Procesador del computadores

module Procesador_CPU_PC1(intr,intrack,dOut_proc,pReady_proc,cReady_proc,dma,dmafin);

        input           cReady_proc;
        input                   intr;
        input                   dmafin;
        output  [63:0]  dOut_proc;
        output                  pReady_proc;
        output          intrack;
        output                  dma;
        reg             [63:0]  dOut_proc;
        reg                     pReady_proc;
        reg     intrack;
        reg                             dma,gestint,protocolo;
        integer                 i,tiempoCPU,tiempoAplic,tiempoTotal,tiempoOverhead;
        integer                 ii,jj,semilla;
        time                    t0,t1,t2,t3,t4;

        initial
                begin
                        dma=0;
                        pReady_proc=0;
                        dOut_proc=0;
                        protocolo=0;
                        gestint=0;
                        intrack=0;
                        semilla=$random(100);
                        i=0;
                        ii=0;
                        tiempoCPU=0;
                        tiempoOverhead=0;
                        tiempoAplic=0;
                        tiempoTotal=0;
                end

        always
                begin
                 fork
    forever
                        begin
                          t0=$time;

                          #1000
                          i=(i+1); //Los ciclos  de procesador de  parte de la Aplicación

                          tiempoAplic=tiempoAplic+1;


        // Parte de acceso a memoria con el procesador.
                            if ((($random(semilla)%1)==0)&&(ii<1024))
                                begin
                                                wait(cReady_proc)
                                                #1000 dOut_proc=i;
```

```
                                    pReady_proc=1;
wait(!cReady_proc)
                        #1000 pReady_proc=0;
                        ii=ii+1;
                        tiempoAplic=tiempoAplic+2;
                                end

                                t1=$time;
                                tiempoTotal=tiempoTotal+((t1-t0)/1000);

                         if (protocolo==1)  // ////la interrupción
                                begin
                                  for (jj=0;jj<32;jj=jj+1)
                                        begin
        #1000 tiempoCPU=tiempoCPU+1; // se tiene en cuenta el coste de entrar al SO
         // el procesamiento del protocolo de comunicación

                                if ((($random(semilla)%30)==0)&&(jj<64))

                                        begin
                                                wait(cReady_proc)
                                                #1000 dOut_proc=0;
                                                pReady_proc=1;
                                                wait(!cReady_proc)
                                                #1000 pReady_proc=0;

        tiempoCPU=tiempoCPU+2;//////////////////////
                                        end

                                   end
        $display("tiempoCPU%d",tiempoCPU);
                         protocolo=0;// El coste del procesamiento del  SO
                         intrack=0;
                         dma=1;
                         end
                        t2=$time;
                                tiempoOverhead=tiempoOverhead+((t2-t1)/1000);

                        end

                        $display("timpoOverhead%d",tiempoOverhead);


                        forever
                                begin
                                @(posedge dmafin)
                                        dma=0;
                                        t3=$time; // El efecto del modelo EMO en el OVerhead


                                        wait(intr);
                                wait((intr)&(dmafin))   //Usamos cuando inicialización DMA en CPU
                                  //dma=0;
                                        #1000 intrack=1;
                                wait (!intr)
                                        #1000

                           // Cuando cambiamos la interapcion de esta caso se puede
                                          // reducir el tiempo de CPU

                                        intrack=0;
                                        protocolo=1;
                                        t4=$time;
```

```
            tiempoCPU=tiempoCPU+2;  // El effecto del modelo EMO en el Protocolo
            tiempoOverhead=tiempoOverhead+((t4-t3)/1000); // EL efecto del Modelo EMO en el SO
                                                gestint=1;
                                                inicializacion de DMA en CPU
                                                #300000
                                                dma=1;


                                        end
                        join
                        end
endmodule
```

En el caso de *externalización* mediante *offloding* se cambia la parte donde se procesan los datos, y la tarjeta de red se encarga de hacer el procesamiento de protocolo de comunicación en vez de que lo haga el procesador central. En el caso de *externalización* mediante *onloading* se agrega otro módulo de (CPU_1) que se encarga de hacer el procesamiento del protocolo de comunicación como se muestra en la Figura 1, mientras el otro procesador (CPU_0) se encarga procesar las aplicaciones y el sistema operativo. Por lo tanto, el módulo del procesador CPU_1 no difiere de menor significativo del otro módulo CPU_0.

REFERENCIAS

[ATM59] "Asynchronous Transfer Mode (ATM) Technical Overview". IBM International Technical Support Organization, October 1995.

[ASH08] Ashenden, P.J.; Lewis, J.“VHDL-2008 Just the New Stuff ”, Elsevier Inc., 2008.

[BHO98] Bhoedjang, R.A.F; Rühl, T.; Bal, H.E.; “User-Level Network Interface Protocolos”. IEEE computer,pp.53-60.noviember, 1998.

[BRO05] Brown, S.; Vranesica Z.; “Fundementales of the Digital logic with VHDL design”,McGraw-Hill, 2005.

[BIN05] 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”, In 8th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW). Febrero, 2005.

[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). Febrero, 2003.

[BUC01] Buchanan, W.; Wilson, A.:" Advance PC Architecture". William and Austin, 2001.

[BUR10] Burger, D.; Emer, J.; Hoe, J.C.; Chiou, D.; Sendag, R.; Yi, J.J.:"The future of architectural simulation". IEEE Micro, pp.8-18. May/June, 2010.

[CAS10] Cascón, P.; "Optimizing Communications by Using Network Processors". Departamento de Arquitectura y Tecnología de Computadores, Universidad de Granada, Febrero, 2010.

[CLA89] Clark, D.D.; Jakobson, V.; Romkey, J.; Salwen, H.:" An Analysis Of TCP Processing Overhead ". IEEE Communication Magazine, June 1989.

[CIS00] "Internetworking Technology Handbook, Third Edition". Asynchronous transfer Mode Switching, Chapter 27, Cisco System, Inc. 2000.

[CHU80] Chun, B.N.; Mainwaming, A.M.; Culler, D.A.: "Virtual Network Transport Protocols for Myrint ". IEEE,Micro, P.P. 53, Enero 1980.

[CHE05] Cherkasova, L.; Gardner, R.; "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor". USENIX Annual Technical Conference, 2005.

[DOL10] Pagina web http://www.dolphinics.no/.

[DAL04] Dally, W.J.; Towles, B.; "Principles and practices of interconnection networks". Morgan Kaufmann, 2004.

[DIA01] Diaz, A.F.; "Comunicación eficiente en redes de área local para procesamiento paralelo en clusters". Departamento de Arquitectura y Tecnología de Computadores, Universidad de Granada. Septiembre, 2001.

[DIA05] Diaz, A.F.; Ortega, J.; Prieto, A.; Ortiz, A.:" Análisis de la Externalización de protocolo Mediante de Simulación HDL". Actas de las XVI Jornadas de Paralelismo, Septiembre 2005.

[DER08] Derek, M. J.; "The New C Standard", 2008.

[ENG03] Engblom, J. " Full system simulator", European summer school on mebedded system, 2003.

[FOO03] Foong, A.P.; Hufff, T.R.; Hum, H.H.; Patwardhan, J.P.; Regnier, G.J.:" TCP Performance Re-Visited". IEEE, 2003.

[FIS71] Fisher, R.A.; Sc.D.; F.R.S ;" *The Design of Experiments*".New York: Hafner Press,1971.

[GAD07] GadelRab, S.;"10-Gigabit Ethernet Connectivity for computer server". IEEE computer society, 2007.

[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.

[GEM08] Multifacet GEMS Development Team web page.

[GRO05] Crowely, P.; Frankline, M.A.; Hadimioglu, H.; Onufryk, P.Z.:"Network Processor Design: Issues and Practices Volume 3". Eleservir,Inc., 2005.

[HYD00] Hyde, D.C.: "Teaching Design in a Computer Architecture Course". Bucknell University, IEEE, 2000.

[HEL03] Held, G.: "Ethernet Network, Design, Implementation operation and management". Johan Wiley and Sons, Furth Edition, 2003.

[HEN95] Hennessy, J.L; Patterson, D. A.;“A Quantitativa Approche”. Computer Architecture. Morgan Kaufman Publishers Inc., San Francisco, 1995.

[HAL05] Halsall F.; “Computer Networking and the Internet”. 5 Edition, Pearson Education Limited, 2005.

[HYP06] “ HyperTransport I/O Link specification” , Hyper transport Technology consortium, 2008.

[ICA05] Intel Corporation: "Acceleration high-speed Networking with I/O Acceleration Technology". Intel, May 2005.

[IXP05] Intel Corporation:" Intel IXP 2805 Network processor: High-Performance Solution for applications from multi-Gbps to 10 Gbps ". Printed in USA, 2005.

[IOA06] Intel® I/O Acceleration Technology, 2006.

[INT10] The Internet Traffic Archive, http: //ita.ee.lbl.gov/ html/contrib/BC.html

[JOH03] Johanson, E.J.; Kunze, A.R.:" IXP 2400/2800 Programming, the complete Microengine Coding Guide". Intel press, April 2003.

[JIN05] Jin, H.W.; Balaji, P.; Yoo, C.; Choi, J.Y.; Panda, D.K.:"Exploiting NIC Architectural Support for Enhancing IP based Protocols on High Performance Networks". Journal of Parallel and Distributed Computing archive Volume 65, Issue 11, November, 2005.

[JAI91] Jain, R.; “ The atr of computer systems performance analysis: techniques for experimental design, Mesasurment, simulation and modeling”. John wiley and sons, Inc., 1991.

[KEE95] Keeton, K.K.; Anderson, T.E.; Patterson, D.A.: " LogP Quintified: the Case for Low-Overhead Local Area Networks". Hot Interconnect III, Agosto1995.

[LEO02] León-Garcia, A.; Widjaja, I.; "Redes de Comunicación: Conceptos Fundamentales y Arquitecturas Básicas". McGraw-Hill, 2002.

[LEE03] LEE, W.F.; "Verilog Coding for Logic Synthesis", John Wiley & Sons, 2003.

[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.

[MOS02] Mosberger, D.; Eranian, S.; "IA-64 Linux Kernel: Design and Implementation". Prentice Hall, 2002.

[MIC05] Michael, D.C.; "Advanced Digital Design with the Verilog HDL", Prentice Hall, 2005.

[M5S10] "The M5 Simulator System", http://www.m5sim.org/wiki/index.php/Main_Page 2010.

[MAG02] Magnusson, P.S.; Christensson, M.; Eskilson, J., Forsgren, D.; Hållberg, G.; Högberg, J.; Larsson, F.; Moestedt, A.; Werner, B.; "Simics: A Full System Simulation Platform" IEEE, 2002.

[MAR05] Martin, M.M.K.; Sorin, D.J.; Beckmann, B.M.; Marty, M.R.; Xu, M., Alameldeen, A.R.; Moore, K.E., Hill, M.D.; Wood, D.A.; "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset "Appears in Computer Architecture News (CAN), September 2005.

[MAU02] Mauer, C.J.; Hill, M.D.; Wood, D.A.;"Full-System Timing-First Simulation", ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, 2002.

[MAT05] Matus, J. R. : "Algoritmo de Reemplazamiento en memoria cache", 2005.

[MEL03] Mellanox Technologies:"Introduction to InfiniBand". Mellanox Technologies Inc., 2003.

[NAY79] Naylor, D.; Jones, S.: " VHDL: A Logic Synthesis Approach". Champan & Hall, 1979.

[NAH97] Nahum, E.; Yates, D.; Kurose, J.; Towsley, D.: "Cache Behavior of Network Protocols". Sigmetrics conference, 1997.

[ORT05] Ortega, J.; Anguita, M.; Prieto, A.:" Arquietectura de Computadores". Intrnational Thomson Editor, Spain, 2005.

[ORT07] Orteiz, A.; Ortega, J.; Díaz, A. F.; Prieto, A.;" Analyzing the Benefits of protocol offload by full-system Simulation". 15th Euromicro International Conference on Parallel, Distributed and Network-based Processaing (PDP 2007). Nápoles, 2007.

[ORT08] Orteiz, A.;" Alternativas de Externalización para la Interfaz de Red. Análysis y optimización mediante simulación sistema completo". Departamento de Arquitectura y Tecnología de Computadores, Universidad de Granada, Agosto, 2008.

[ORT09] Orteiz, A.; Ortega, J.; Díaz, A. F.; Prieto, A.;" A new offloaded/onloaded network interface for high performance communication". 17th Euromicro International Conference on Parallel, Distributed and Network-based Processaing (PDP 2009). Germany, 2009.

[OLI00] Olivera, C.; "Redes de Área Local". Escuela de Ingeniería, Pontificia Universidad Católica de Chile,2000.

[OME06] "Un paso por Fiber Channel". Suporte Técnico OEM, Fujitsu España, Marzo 2006.

[PAT00] Patterson, D.A.; Hennessy, J.L.:" Estructura y diseño de Computadores Interface Circuitería / Programación". Editorial Reverte, S. A. 2000.

[PIR00] Pirva, M.; Bhuyan, L.; Mahapatra, R.:" Hierarchical Simulation of Multiprocessor Arquitecture" .IEEE International Conference on Computer design, 2000.

[PAP04] Papaefstathious, L.; Nikolaou, N.A.; Doshi, B.; Grosse, E.: "Network Processor for future High-End Systems and Applicationes ".IEEE, 2004.

[PAT02] Patricia, G.; Arthur B. M.: “Splintering TCP”. Proceeedings of the 17th International Symposium on Computer and Information Sciences, 2002.

[PET03] Peterson, L.L.; Davie, B.S.; “Computer Networks A Systems Approach”. Morgan Kaufmann Publishers,2003.

[PET08] Pete, B.; “Working Draft, Standard for Programming Language C++”, Roundhouse Consulting, Ltd., 2008.

[PIR20] Pirvu, M.; Bhuyan, L.; Mahapatra, R.:”Hierarchical simulation of a multiprocessor architecture”. Proc. Of the 2000 IEEE Int. Conference on Computer Design: VLSI in Computers and Processors, 2000.

[QUA10] Pagina web //www.quadrics.com.

[RUS00] Russell R.; Javier, R.; Medina, C.:"Linux Networking-concepts HOWTO:Qué Aspectos tienen los Paquetes ". Mayo 2000.

[RAN02] Rangarajan, M.; Bohra, A.; Banerjee, K.; Carrera, E.V., Bianchini, R.; "TCP Servers: Offloading TCP Processing in Internet Servers. Design, Implementation, and Performance”, 2002.

[ROS97] Rosenblum, M.; Edousrd, B.; Scott D.; Stephen, A.H.; “Using the SimOS Machine Simulator to Study Complex Computer Systems”, ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 1, January 1997.

[REG04a] Regnier, G.; Makineni S.; Illikkal R.; Iyer R.; Minturn D.; Huggahalli R.; Newell D.; Cline L.; Foong A.; “TCP Onloading for Data Center Servers” , IEEE Computer Society, 2004.

[REG04b] Regnier, G.; Minturn, D.; McAlpine, G.; Saletore, V.; Foong, A.; “ETA: Experience with an Intel® XeonTM Processor as a Packet Processing Engine “,Intel Corporation, 2004.

[REA10]Reas, M. R.; Alvarez, A. B.; Reyes, J. A. P.; “Simulation of Standard Benchmarks in Hardware Implementations of L2 Cache Models in Verilog HDL”. UKSIM 10 Proceedings of the 2010 12th International Conference on Computer Modelling and Simulation, Washington, 2010.

[ROM05] Romanow, A.; Mogul, J.; Talpey, T.; Bailey, S.; “Remote Direct Memory Access (RDMA) over IP Problem Statement”. The Internet Society, December 2005

[SHI03] Shivam, P.: Chase, J.S.:"On the elusive benefits of protocol offloads". SIGCOMM’03 *Workshop on Network-I/O convergence: Experience, Lesons, Implications (NICELI)*. August, 2003.

[SEN04] Senapathis, S.; Hernadez, R.: "Introduction to TCP Offload Engines". Power Solutions, Marzo 2004.

[SKE01] Skevik, K.A.; Plagemann, T.; Goebel, V.; Halvorsen, P.; “Evaluation of a Zero-Copy Protocol Implementation”. In 27th Euromicro Conference, 2001.

[SHI09] Shinde,S.S.; “Computer Network”. New age international limited, 2009.

[SAR06] Sarkar, N.; “Tools for teaching computer networking and hardware concepts”. Information science publishing, 2006.

[SCH07] Schuff, D., L.; Pai, V. S.; Willmann, P.; Rixner, S.:”Parallel Programmable Ethernet Controllers: Performance and Security”. IEEE Network, pp.22-28. July/August, 2007.

[SUR10] Surhone, L. M., Tennoe, M.T., Henssonow, S.F. “Front Side bus “.VDM Verlag Dr. Mueller AG & Co. Kg, 2010

[THO91] Thomas, D.E.; Moorby, P.: "The Verilog Hardware Description Languege". Kluwer Acadmic Publisher, 1991.

[TCP10] Página web http://www.tcpdump.org

[THO02] Thomas, D.E.; Moorby, P.R.;“The Verilog® Hardware Description Language”, Kluwer Academic Publishers, 2002.

[VID10] Página web http://www.vidf.org.

[VIJ05] Vijayaraghavan, S.; Ramanathan, M.;“A Practical Guide for System Verilog Assertions”, Springer Science-i-Business Media, Inc., 2005.

[VIR10] Virtutech web page: http:// www.virtutech.com/, 2010.

[WES04] Westrelin, R.; Fugier, N.; Nordmark, E.; Kunze, K.; Lemoine, E.: "Studying Network Protocol Offload with Emulation: Approach and preliminary Results ". Sun Microsystems Inc., 2004.

[WIL05] Willman, P.; Kim, H.Y.; Rixner, S.; Pai, V.S.:" An Efficient Programmable 10 Gigabit Ethernet Network Interface Card". IEEE proceedings of the 11th int'l Symposium on High-Performance Computer Architecture,2005.

[WAN98] Wang, R.Y.; Krishnamurthy, A.; Martin, R.P.; Anderson, T.E.; Culler, D.E.:"Towards a Theory of Optimal Communication Pipelines". Technical Report, 1998.

[WUN06] Wun, B.; Crowley, P.; "Network I/O Acceleration in Heterogeneous Multicore Processors". 14th IEEE Symposium on High-Performance Interconnects (HOTI'06), 2006.

[YI05] Yi, J. J.; Lilja, D. J.; Hawkins, D. M. ;"Improving Computer Architecture Simulation Methodology by Adding Statistical Rigor". IEEE Transactions on Computers, Vol. 54, November 2005.

[ZAH07] Zahran, M.; Albayraktaroglu, K.; Franklin, M.; "Non-Inclusion Property in Multi-level Caches Revisited". Int'l J. Computers and their Applications, June 2007.