Protocol offload analysis by simulation

Andrés Ortiz, Julio Ortega, Antonio F. Díaz, Pablo Cascón, Alberto Prieto

Department of Communications Engineering, University of Malaga, Spain
Department of Computer Architecture and Technology, University of Granada, Spain

A R T I C L E   I N F O

Article history:
Received 27 May 2007
Accepted 17 July 2008
Available online 7 August 2008

Keywords:
Full-system simulation
HDL simulation
LAWs model
Protocol offloading
Network interfaces
Simics

A B S T R A C T

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

The contributions of this paper deal with the implementation and evaluation of offloading strategies and with the need of accurate tools for researching the computer system issues that, as networking, require the analysis of interactions among applications, operating system, and hardware. Thus, in this paper, a way to include timing models in a full-system simulator (Simics) to provide a suitable tool for network subsystem simulation is proposed.

Moreover, we compare two kinds of simulators, a hardware description language level simulator and a full-system simulator (including our proposed timing models), in the analysis of protocol offloading at different levels. We also explain the results obtained from the perspective of the previously described LAWs model and propose some changes in this model to get a more accurate approach to the experimental results. From these results, it is possible to conclude that offloading allows a relevant throughput improvement in some circumstances that can be qualitatively predicted by the LAWs model.
processing. Moreover, the use of dedicated processors for network processing either in an SMP [36,47] or in a multi-core microprocessor [40,51] has been also proposed. This last technique, also called onloading, is one of the features of the Intel I/O acceleration technology [24].

The network interface card (NIC) is the hardware that provides the physical access to the network by usually including a low-level addressing system (the MAC addresses) and the functions of physical and data link layers. In the past 10 years, some network acceleration features have been included in the NICs (mainly in Ethernet NICs) [17,25]. Thus, almost all NICs for Gigabit/s and 10 Gigabit/s Ethereans can determine and check the TCP/IP checksums. However, they also implement strategies to reduce the interrupt frequency by generating one interrupt request for multiple packets sent or received instead of one request per packet (interrupt coalescing) [4]. Other common features of NICs are the use of header splitting [37,50], that place protocol headers and payloads in separate buffers, and Jumbo frames [2], which are larger frames than the Ethernet maximum frame size of 1500 bytes (up to 9000 bytes), used to reduce the per-frame processing overhead. Besides these features, currently, many NICs include programmable processors. These Intelligent NICs (INICs) are frequent in the interconnection networks in current cluster-based computing systems, and much research has been done towards the use of these processors to offload network processing from the host CPU [27]. This way, the CPU is free from communication overhead and a faster implementation of more flexible communication systems is possible. The TCP offload engines (TOEs) are examples of NICs following this alternative [6,8,9,13,18,32]. Moreover, inside this trend, it is possible to include one of the primary features that prevent performance to scale with processor speeds [15,22]. The CPU speed is usually higher than only shifting them to the NIC [42].

Thus, by offloading, a distribution of the communication tasks among the different elements of the host, particularly between the host CPU and the processor in the NIC, is provided. The communication tasks that imply interactions with the network can be implemented in the NIC in order to leave more CPU cycles for the computation work required by the applications. When the CPU needs to send or receive data through the network, it can write or read them to/from the main memory where the NIC would read or have written them. This way, protocol offloading should be seen as a technique that enables both the parallelization of the network communication work and the direct data placement on the main memory, thus avoiding some communication overheads rather than only shifting them to the NIC [42].

However, some works [10,12,31,37] criticize protocol offloading and provide experimental results to argue that TCP offloading does not clearly benefit the communication performance. Nevertheless, there are other works that demonstrate the benefits of TCP offloading. For example, in [50] an experimental study is carried out based on the emulation of a NIC connected to the I/O bus and controlled by one of the CPUs in the SMP. The results show improvements from 600% to 900% in the TCP-emuleted offload. Moreover, in [16], counterarguments to the TCP offloading criticism of [33] are provided.

On the one hand, the reasons for the scepticism on offloading benefits are the difficulties in the implementation, debugging, quality assurance, and management of the offloaded protocols [33]. The communication between the NIC (with the offloaded protocol) and the CPU and the API could be as complex as the protocol to be offloaded [13] (cited in [33]). Protocol offloading requires the coordination between the NIC and the OS for a correct management of resources such as buffers, port numbers, etc. In case of protocols such as TCP, the control of the buffers is complicated and could hamper the offloading benefits (for example, the TCP buffers must be held until acknowledged or pending reassembly). Moreover, the inefficiency of short TCP connections is due to the overhead of processing the events that are visible to the application and cannot be avoided by protocol offloading [33]. These are not definitive arguments with respect to the offloading usefulness but they counterbalance the possible benefits. In any case, this means that an efficient host/NIC interface for offloading is one of the main issues to take advantage of this technique [16].

On the other hand, there are fundamental reasons that affect the possible offloading advantages. One of them is the ratio of host CPU speed to NIC processing speed. The CPU speed is usually higher than the speed of the processors in the NIC and, moreover, the increment in the CPU speeds according to Moore’s law tends to maintain or even to increase this ratio in the case of the specific purpose processors in the NIC. Thus, part of the protocol that is offloaded would require more execution time in the NIC than in the CPU, and the NIC could become the communication bottleneck. The use of general-purpose processors in the NIC (with speeds similar to the CPU) could represent a bad compromise between performance and cost [11]. Moreover, the limitations in the resources (memory) which are available in the NIC could imply restrictions in the system scalability (for example, limitations in the size of the IP routing table). According to these arguments, it is clear that NIC processing and memory capabilities are important issues. Nevertheless, faster CPUs are not enough to avoid the effect of the operations that prevent performance to scale with processor speeds [15,22].

The problems of offloading are clearly apparent in the use of TCP protocol either in WAN applications (such as FTP and e-mail) or in LAN applications that require low bandwidth (such as Telnet). In these cases, the overheads of the connection management are the most important and the more difficult to avoid by protocol offloading. In this way, [33] concludes that offloading is more
adequate in applications requiring high bandwidths, low latencies, and long-term connections. RDMA (remote direct access memory) is an example where protocol offloading can be efficient. RDMA is a protocol that allows packet transfer to the right memory buffer, thus giving an adequate procedure for 0-copy. As the RDMA component DDP (direct data placement) requires an early de-multiplexing of the input packets, its implementation (and the TCP protocols below it) in the NIC could be advantageous.

Thus, it should be understood why the benefits of protocol offload are so elusive and difficult to predict [42]. It is clear that the system communication performance depends on many factors, from the application computation/communication profile, to the interactions between operating system, application, and hardware. In particular, the detailed profile of memory accesses for a given network application is difficult to evaluate and take into account to predict performance. The goal of this paper, which is an extension of our conference papers [33,34], is to use simulation at different levels to get insight into the offloading effects.

Section 2 describes the LAWS model [42], which has been recently proposed to predict the offloading effects. Then, Section 3 uses an HDL (hardware description language) simulator to analyze the behavior of the different elements in the communication path in order to understand their role in the communication performance either without offloading or with different offloading alternatives. In this section, together with the CPU, the NIC, and the network links, we also model the effect of the buses, the bridge, and the main memory. Section 4 considers the use of Simics, a full-system simulator that allows a detailed simulation of hardware, application software, and operating system, and Section 5 compares the experimental results obtained by Simics with those predicted by LAWS and proposes some modifications to this model in order to improve its accuracy. Finally, Section 6 provides the conclusions of the paper and states the questions that remain to be considered in future works.

2. A model to estimate offloading performance

Some papers [17,39,44,49] have recently appeared to study the offloading fundamental principles under the experimental results. The paper [42] introduces the LAWS model to characterize the protocol offloading benefits in Internet services and streaming data applications. In [19], the EMO (extensible message-oriented offload) model is proposed to analyze the performance of various offload strategies for message oriented protocols. In this paper, we have used the LAWS model.

LAWS model [42] gives an estimation of the peak throughput of the pipelined communication path according to the throughput provided by the corresponding bottleneck (the link, the NIC, or the host CPU). The model only includes applications that are throughput limited (such as Internet servers) and fully pipelined [23,48], when the parameters used by the model (CPU occupancy for communication overhead and for application processing, occupancy scale factors for host, and NIC processing, etc.) can be accurately known. The analysis provided in [42] considers that the performance is CPU limited before applying the protocol offload, as this technique always yields no improvement otherwise.

Fig. 1 explains how the LAWS model views the system before and after offloading. The notation which is used is similar to that of [42]. Before offloading (Fig. 1a), the system is considered as a pipeline with two stages, the host and the network. In the host, to transfer m bits, the application processing causes a CPU work equal to \( aXM \) and the communication processing produces a CPU work \( oXM \). In these processing delays, \( a \) and \( o \) are the amount of CPU work per data unit, and \( X \) is a scaling parameter used to take into account variations in processing power with respect to a reference host. Moreover, \( m/B \) is the latency to provide the \( m \) bits by a network link with a bandwidth, \( B \).

Thus, as the peak throughput provided before offloading is determined by the bottleneck stage, we have \( B_{before} = \min(B, 1/(aX + oX)) \). After offloading, we have a pipeline with three stages (Fig. 1b), and a portion \( p \) of the communication overhead has been transferred to the NIC. In this way, the latencies in the stages for transferring \( m \) bits are \( m/B \) for the network link, \( aXM + (1 - p)aXM \) for the CPU stage, and \( pOY/m \) for the NIC stage. In the expression for the NIC latency, \( Y \) is a scaling parameter to take into account the difference in processing power with respect to a reference, and \( \beta \) is a parameter that quantifies the improvement in the communication overhead that could be reached with offloading, i.e., \( \beta \) is the normalized overhead that remains in the system after offloading when \( p = 1 \) (full offloading) [42].

In this way, after offloading, the peak throughput is \( B_{after} = \min(B, 1/(aX + (1 - p)aX), 1/pOY) \) and the relative improvement in peak throughput is defined as \( \delta = (B_{after} - B_{before})/B_{before} \). The LAWS acronym comes from the parameters used to characterize the offloading benefits. Besides the parameter \( \beta \) (structural ratio), we have the parameters \( \chi = Y/X \) (lag ratio), that considers the ratio between the CPU speed to NIC computing speed; \( \gamma = a/o \) (application ratio) that, for the given application, measures the ratio of computation cost to communication cost; and \( \sigma = 1/\alpha XB \) (wire ratio), that corresponds to the portion of the network bandwidth that the host can provide before offloading. In terms of the parameters...
\(\alpha, \beta, \gamma,\) and \(\sigma,\) the relative peak throughput improvement can be expressed as:

\[
\delta b = \min\left(\frac{1}{\gamma + 1}; \frac{1}{\gamma + \sigma + 1}\right) - \min\left(\frac{1}{\gamma}; \frac{1}{\gamma + \sigma}\right)
\]

From LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from LAWS model some conclusions can be derived in terms of simple relationships among the LAWS ratios (see Fig. 1c, obtained from

- Protocol offloading provides an improvement that grows linearly in applications with low computation/communication rate (low \(\gamma\)). This profile corresponds to streaming data processing applications, network storage servers with large number of disks, etc. In case of CPU intensive applications, the throughput improvement reached by offloading is bounded by \(1/\gamma\) and goes to zero as the computation cost increases (i.e. \(\gamma\) grows). The best improvement is obtained for \(\gamma = \max(\alpha/\beta, \sigma)\). Moreover, as the slope of the improvement function \((\gamma + 1)/\gamma - 1\) is \(1/c\) and \(c = \max(\alpha/\beta, \sigma)\), the throughput improvement grows faster as \(\alpha/\beta\) or \(\sigma\) decrease.

- Protocol offloading may reduce the communication throughput (negative improvement, \(\delta b < 0\)) if the function \((\gamma + 1)/c - 1\) takes negative values. This means that \(\gamma < (c - 1)\) and, as \(\gamma > 0\) and \(\sigma < 1\), it should verify that \(c = \alpha/\beta\) and \(\alpha/\beta > 1\). Thus, if the NIC speed is lower than the CPU speed \((\alpha > 1)\) offloading may reduce performance if the NIC saturates before the network link, i.e. \(\alpha/\beta > \sigma\), as the improvement is bounded by \(1/\alpha\) (whenever \(\beta = 1\)). Nevertheless, if an efficient offload implementation (for example, by using direct data placement techniques) allows structural improvements (such that \(\beta < 1\)) that make \(\alpha/\beta < 1\), it is possible to maintain the offloading usefulness even for \(\alpha > 1\).

- There is no improvement in slow networks \((\sigma \gg 1)\) where the host is able to assume the communication overhead without aid. The offloading technique is useful whenever the host is not able to communicate at link speed \((\sigma \ll 1)\), but in these circumstances, \(\gamma\) has to be low, as it has been previously said. As there is a trend to faster networks \((\sigma\) decreases), offloading can be seen as a very useful technique. When \(\sigma\) is near one, the best improvement corresponds to cases where there is a balance between computation and communication costs before offloading \((\gamma = \sigma = 1)\).

Nevertheless, as LAWS estimates peak throughput improvements in fully pipelined communication paths, it could only represent a first approach, that is perhaps far from the performance observed in real communication systems. In this paper, we have used two kinds of simulators to get experimental evidences about the LAWS model usefulness.

### 3. A first analysis of offloading through HDL simulation

LAWS and other models, such as EMO, provide good starting points to get a first insight about the conditions that make offloading useful, and contribute to guide the experimental work in the design space of the offloading. Nevertheless, a detailed experimental validation with wider sets of application domains and benchmarks is required. Simulation is a good way to get this. It can be considered the most frequent technique to evaluate computer architecture proposals. To be a useful tool, a simulator needs to simulate the target machine (and to drive it by using realistic workloads) with the details required by the questions to be answered. Many realistic workloads, such as web servers, databases, and other network-based applications use operating system services and simulators that run only user-mode, single-threaded workloads are not adequate because a network-oriented simulation requires timing models of the network DMA activity, coherent and accurate models of the system memory, and full-system simulation including the OS.

Although the main part of our analysis of offloading has been implemented by using the full-system simulator Simics [29], in this section, we have firstly used an HDL (hardware description language) model of the communication path. This model allows us the simulation of the hardware characteristics at an adequate level of detail to introduce and understand the influence of the different implementation characteristics: delays produced by the software and buses, contention to access memory and other shared hardware elements, register level transactions, etc. Moreover, for real applications, it is difficult to reproduce experimentally the LAWS curves corresponding to the peak throughput improvement against the application ratio, \(\gamma\), when one of the others parameters, \(\alpha/\beta\) or \(\sigma\), changes while the other is kept constant (see the figures of [42] and Fig. 1b). For example, it is difficult to keep unchanged the application CPU work per data unit, \(a\), before and after offloading, and it is also difficult to modify an application to get the values of \(a\) and \(o\), that allow a (more or less) complete set of values for \(\gamma\). In this way, an HDL model makes these situations easier to reach, as the delays used to represent the different procedures (application or communication tasks) can be directly changed.

#### 3.1. An HDL model for offloading

Fig. 2 gives the modules included in the HDL model which we have written by using the Verilog hardware description language. Besides the NIC, CPU and cache, chipset and memory, the HDL model includes the delays and contention problems in the I/O bus and memory bus, and makes possible to inject packets at different speeds through the network module.

As an example of the level of detail used in the different modules of our HDL model, Fig. 3 illustrates the elements included in the HDL description of the NIC. It includes a buffer that the data coming from the network link are stored. There are two modules that control the way the data are, respectively, written to the queue, or read from it. The other element in the NIC implements the communication protocol if it is offloaded, initializes the DMA, generates the interrupt requests to the CPU, etc. We have considered a programmable NIC with enough local memory to store data and code. It also includes faster memories to implement

---

**Fig. 2.** Modules of the host HDL model.
the queues of buffers (in Fig. 3, we represent only one queue) and other control registers. The CPU and the NIC processor interact by reading from or writing to some shared registers. The role of the NIC processor in the communication depends on the protocol implementation to be simulated. Whenever offloading is considered, the behavior of the NIC is controlled by a program stored in the NIC memory that runs in the NIC processor. The NIC is modeled as a cut-through device rather than a store-forward one. It overlaps the transfers across the I/O bus and the network links.

Fig. 4 compares non-offloading (Fig. 4a) with two offloading alternatives (Figs. 4b and c) at the receive-side. Whenever the reception is done without offloading, the NIC, after getting some information from the received packet, interrupts the CPU. It executes the driver to initialize the DMA operation between the NIC and the main memory (for example, the memory address in which to store the packet data has to be communicated to the NIC). The NIC stores the received data in main memory through a DMA operation and informs the CPU at the end of this operation. Then, the CPU starts the protocol processing of the packet in main memory.

In the alternative for offloading shown in Fig. 4b, the NIC is able to start the process of the protocol once the whole packet is received (or even after receiving only part of it). Then, the NIC interrupts the CPU and executes the driver to initialize the DMA operation, as in the non-offloading alternative. After that, the NIC starts the DMA operation to transfer the packet data to main memory and informs the CPU when data are available in its main memory at the end of the DMA transfer. The alternative shown in Fig. 4c frees the CPU from almost all the communication overhead, as the NIC is able to process the protocol and initializes the DMA to send the payload of the received packet to memory. When these data are stored in the corresponding addresses, the NIC informs the CPU that the application can use them.

Fig. 5 shows some of the signals exchanged among the different modules in the HDL simulations. They correspond to packet reception with and without offloading. To get shorter timing diagrams, the size of the injected packet is small and only eight words have to be transferred to memory by a DMA operation. The timing diagrams of Figs. 5a and b correspond to the same reception without offloading, although they show different time scales. Fig. 5c corresponds to the offloaded reception.

The link signal corresponds to the data entering the NIC from the network link. The arrival of data to the main memory is indicated by mem signal. The intr and intrack signals are the signals exchanged between the CPU and the NIC for, respectively, interrupt

---

**Fig. 3.** Main elements of the NIC module.

**Fig. 4.** Reception with offloading (a), and with two offloading alternatives (b) and (c).
The protocol is processed by the CPU and means a finished DMA transfer. The transfer in progress from the NIC to the CPU, whereas processing is done in the NIC. In the three figures (Figs. 5a–c), the loading. As it can be seen from illustrating the differences between reception without and with off-

- Fig. 5b and c correspond, respectively, to Figs. 4a and b, and illustrate the differences between reception without and with offloading. As it can be seen from link signal in both figures (Figs. 5b and c), the link bandwidth saturates the node: firstly the packets enter the NIC and are stored in the NIC queue (Fig. 3) at the link speed, but as the packets cannot be processed at the link speed, the queue gets full and the speed of packet arrival decreases. In case of reception without offloading (Figs. 5a and b), the CPU is interrupted (intr and intrack signals), it initializes de DMA operation of the incoming data (dmainitCPU) and, after the DMA transfer to memory (see mem signal), the CPU processes the packet according to the communication protocol (protCPU signal). In case of an offloaded reception (Fig. 5c), the packet is firstly processed in the NIC (protNIC) and the CPU is interrupted to start the transfer of the payload (intr and intrack). Then, the CPU initializes the DMA operation (dmainitCPU) and the DMA transfer is done (mem). As has been said, in Fig. 5, eight words have been transferred for either offloading or non-offloading simulations. Nevertheless, in case of offloading, the number of words to be transferred for a given packet is lower as the packet is processed in the NIC and the header bits of the packet can be removed.

3.2. Offloading performance evaluation by HDL simulation

Our HDL model allows the simulation of the CPU, the NIC, and the network link, besides the effects of the buses, the bridge, and the main memory. Recently, Intel has launched its I/O accelerating technology (I/OAT) to keep pace with the emergence of multi-Gigabit/s links by allowing the servers to take advantage of that high bandwidth in order to increment its throughput and quality of service [18]. Among the main bottlenecks referred in the whitepapers by Intel describing the main I/OAT characteristics are the system overheads and the memory accesses [24]. The profile of memory accesses for the application executed in the CPU, the protocol processing in the CPU or NIC, and the DMA accesses to read/store the packets or payloads in the main memory have a significant influence in the communication performance and thus, determine the improvement that can be obtained from offloading.

Fig. 6 shows the improvement in throughput against the memory accesses generated by the CPU running either the application or the communication protocol. The experiments have been done considering negligible delays in the I/O and memory buses, and in the chipset circuits. Small packets with eight words have been simulated. The 100% columns mean that all the words of a packet are transferred to the main memory by DMA, whilst the 75%

![Fig. 5. Timing diagrams for packet reception: (a and b) no-offload; (c) offload.](image-url)
columns correspond to a reduction in the number of words to be transferred (a 75% of the packet size is transferred to system memory), as the protocol has been processed in the NIC when offloading is applied and it is not necessary to store the packet head in the main memory. From Fig. 6a, it can be seen that as the application generates more memory accesses, the improvement obtained by offloading grows as offloading allows concurrency between the NIC protocol processing and the execution of the application in the CPU. Nevertheless, the increment in the throughput performance goes down above certain amount of accesses. This situation can be explained by taking into account that, as the number of memory accesses increases, the rate of communication overhead with respect to the application processing decreases (the parameter $\gamma$ in the LAWS model grows).

Figs. 6b and c illustrate the performance improvement obtained by protocol offloading as the memory accesses generated by the communication protocol grow. This improvement comes from both, the reduction in the main memory accesses obtained when the protocol is offloaded (thus the LAWS parameter $\beta$ in less than one, $\beta < 1$, and decreases as the amount of memory accesses increases), and the increase of protocol overhead with respect to the application processing requirements (the parameter $\gamma$ in the LAWS model decreases).

Fig. 7 shows the percentage of throughput improvement against the application ratio parameter $\gamma$ (the computation/communication rate) in the LAWS model for two values of the product $\alpha \beta$, where $\alpha$ is the ratio between the CPU and the NIC processor computing speeds (lag ratio in LAWS), and $\beta$ quantifies the improvement in communication overhead after offloading (structural ratio in LAWS). In the experiments corresponding to Fig. 7a, no memory accesses are generated by the application, while in the case of Fig. 7b, a given maximum of memory accesses are generated by the application (in this case, approximately a 5% of the application time corresponds to memory accesses). In our simulations, the system is host limited before offloading, since offload does not yield any benefit otherwise [42]. The packet size used in the experiments of Fig. 7 corresponds to DMA transfers of eight words between the NIC and the main memory. The characteristics and the behavior shown in the curves of Fig. 7 agree with the conclusions extracted from the LAWS model [42] in Section 1. Thus, in Fig. 7a, no improvement is obtained whenever $\alpha \beta = 1$ (it is not low enough). As $\alpha \beta$ is decreased, a throughput improvement is obtained. This improvement decreases as the application ratio grows: the amount of communication overhead decreases with respect to the computation needs of the application (as the LAWS model predicts).
The behavior shown in the curves of Fig. 7b is also similar to that predicted by Eq. (1) (with $p = 1$) and shown in Fig. 1b. The throughput improvement grows with $\gamma$ for $\gamma < 1$, and decreases as $\gamma$ grows for $\gamma > 1$. Moreover, as the product $\alpha/\beta$ decreases, the maximum achieved for the throughput improvement grows. Nevertheless, although the qualitative behavior of the curves agrees with the LAWS models, there are important quantitative differences in the experimental results. In the case of $\alpha/\beta = 0.8$, the differences lie between 47% and 68%, and between 62% and 82% for $\alpha/\beta = 1.0$.

These differences between the simulation results and the LAWS model predictions are not difficult to justify. First of all, LAWS model predicts the improvement in peak throughputs. Moreover, LAWS takes into account the distribution of the application and communication works before and after offloading, and introduces the possible effects of the implementation in the change of the total amount of work to do through the parameter $\beta$. Nevertheless, there are also effects on the CPU time which is consumed by the applications due to the specific profiles of the use of the memory hierarchy and other elements of the I/O subsystem [6]. These effects could decrease the throughputs with respect to their peak values. In the following section, a full-system simulation of protocol offloading is considered to reach experiments with realistic interactions among operating system, hardware, and applications, as they can be difficult to reconstruct by HDL simulations.

4. Offloading analysis through full-system simulation

The research in the computer system design issues dealing with high-bandwidth networking requires an adequate simulation tool providing a computer model that makes possible to run commercial OS kernels (as the most part of the network code runs at the system level), and other features for network-oriented simulation, such as a timing model of the network DMA activity and a coherent and accurate model of the system memory [5]. There are not many simulators with these characteristics. Some examples are M5 [28], SimOS [41], and some other simulators based on Simics [27,45], such as GEMS [31] and TFSim [32].

Simics is a commercial full system simulator that allows engineers to have accurate hardware models in such a way that software cannot detect the difference between real hardware and the provided virtual environment. This way, Simics allows the simulation of application code, operating system, device drivers and protocol stacks running on the hardware modeled. Moreover, Simics is a fast functional simulator that makes possible to simulate complex applications, operating systems, network stack protocols, and other real workloads.

Nevertheless, Simics is a functional simulator in itself and does not provide an accurate timing model. In [47], some limitations are reported with respect to the capabilities of Simics in the model of x86 processors (out-of-order microarchitectural issues such as branch prediction, reorder buffer, functional units, etc. are not properly modeled) and in the simulation of cc-NUMA computers with accurate cache miss models. In these cases, the functionality of Simics should be extended to allow accurate evaluations of some commercial workloads. This way, to use Simics for protocol offloading evaluation, we have required to develop a network interface model that processes the protocol instead of the main CPU of the system, and to overcome some Simics limitations in the simulation of protocol offloading or networks in general:

(a) Networks are simulated at packet level. The transactions are performed as one event. Thus, the details of a network packet transaction (by sending individual bytes) are not simulated. Instead, the complete transaction is simulated as one action. In this way, the network and I/O devices are simulated in a transaction-based style. This constitutes an important drawback in the network-oriented system simulation, where detailed timing models of network DMA events are required [5].

(b) Simulated link bandwidth could be potentially infinite, but in practice, a very high bandwidth (i.e.: 10 Gbps), requires a high simulation time and the results are not as expected (although Simics is able to handle high bandwidths, the NIC model does not).

(c) Packets are delivered to the network with a configurable latency that depends on the length of a time slice. A time slice in Simics is the minimum time that we could measure. It can be modified, but the lower bound is determined by the CPU speed. So, it is necessary to ensure that the minimum latency that we are simulating is enough to allow the maximum bandwidth needed for our purposes.

(d) Using shorter time slices (lower latencies) in multi-machine configurations slows down the simulation speed. So, this latency could not be as low as one would like.

(e) In order to build simulation models at hardware level, Simics provides the stall execution mode that allows us to simulate latencies or time accesses, but only between the CPU and the memory, and not for the buses.

Despite these limitations, we have preferred Simics instead of simulators such as M5 [28] or SimOS [41] due to the ability to change the simulation parameters and to create hardware models, as well as to simulate a lot of different CPU models with Simics. In fact, it provides the DML (device modeling language) language. This is not only a configuration language but also a hardware description language for device modeling that can be connected to our simulated architecture through a Simics connector. Furthermore, because of its C++ features, the debugging process in simulators such as M5 and SimOS is harder compared with that in Simics, which also provides effective tools for debugging and profiling. There are other simulators based on Simics, such as GEMS [31] and TFSim [32], that provide accurate timing models but they are focused to specific systems. For instance, GEMS is a Simics based simulator for Sparc-based computers.

Taking into account the above issues, to overcome the drawbacks related with the modeling of the time behavior of the simulated system that prevents the use of Simics for network system simulation, we propose extending the functionality of Simics including more detailed timing models as we describe in what follows.

By default, Simics relates the execution time with the number of executed instructions. In this way, whenever the instructions are executed in order, each instruction corresponds to a (configurable) number of clock cycles. In a multiprocessor system, although there is a time interval assigned to simulate the execution of instructions corresponding to each processor, all the processors in the system execute the same number of instructions after a given amount of simulated time. In case of an out-of-order execution, there is not any correspondence between the number of cycles and the number of executed instructions. In Simics, the memory accesses can be generated not only from the processors, but also from other devices. Given a memory physical address, to identify the object to which that address belongs, Simics uses the memory-space concept. This way, a memory-space maps a physical address to any object that would be able to participate in a transfer (a RAM, a flash memory, or a device). A memory-space can also include other memory-spaces, thus building a hierarchy. The possibility to define memory-space hierarchies has allowed us to model latencies in the transfers as can be explained from Fig. 8. Simics internally uses the so called simulator translation cache (STC) to speed up the
simulations by using a table for translated addresses that avoids having to go through all the memory hierarchy. Nevertheless, this strategy implies that sometimes (whenever the memory address to be accessed is included in the STC) the timing models are not applied. Thus, to have more accurate timing models, the STC should be disabled as is shown in Fig. 8, although at the cost of slower simulations. Thus, the timing model can be defined through the `timing_model_operate()` function of the Simics API by adequately setting its parameters. The timing model we have defined is applied to all the devices to which it is connected. Whenever one of these devices tries to generate a memory access, the system checks if no other device is using the bus. If the bus is busy, the contention is simulated by adding a given (and configurable) number of cycles to the memory access latency.

In [1] Simics, it is also extended with two timing models, a memory simulator that implements a cache hierarchy with cache controllers and an interconnection network for multicomputer systems, and a processor timing simulator for Sparc V9 instruction set. In the approach here proposed, all the timing models present the same interface to set their parameters, they are defined by taking advantage of the resources provided by the Simics environment, and thus, it is possible to build timing models not only for the processors, the memory hierarchy, and the connection between the processors and the memory, but also for the NIC and the I/O buses.

We have built a Simics simulation model by defining two customized machines and a standard Ethernet network connecting them in the same way as we could have in the real world. Simics even allows us the connection between the simulated machine and a real network, using the `Simics Central module`. Nevertheless, we have avoided the use of this `Simics Central module` in order to reduce the simulation time and to increase the attainable maximum bandwidth: since `Simics Central` acts as a router, it limits the simulated effective bandwidth. This way, we have connected our two machines directly, something similar to using a crossover Ethernet cable in the real world.

We have used two models in our simulations. The first one corresponds to a non-offloaded system, in which we have a Pentium 4 based machine running at 400 MHz (enough to have up to 1 Gbps at network level and not to slow down the simulation speed). We have also used Simics NIC gigabit models in the BCM5703 PCI based Ethernet card included in our system. The model is shown in Fig. 9.

With this model, we have determined the maximum performance we can achieve using a simple machine with one processor, and no-offloading effects. This way, the CPU of the system executes the application and processes the communication protocols. The maximum throughputs and the CPU loads for this model are shown in Section 5.
In order to offload the protocols, and so remove the protocol processing work from the CPU, we have used the model shown in Fig. 10, corresponding to a system where one of the processors has been isolated from the other and the NIC is directly connected to this CPU in order to improve the parallelism between application and network processes. In Simics, by default, the bridges merely act as connectors and, in this case, no contention is modeled at simulation time. The way to simulate contention effects is through the use of the timing models we have previously described. Thus, a timing model is connected to each entry of the bridge where access contention is possible. This is not an exact way to model contention, but it provides an adequate simulation of the contention behavior. Thus, in our model (Fig. 10), the north bridge and the buses use timing models and do not only act as connectors. In the ideal case, where no timing models are used, transfers between CPUs and memory would not hold any other transfer.

On the other hand, in Simics, PCI I/O and memory spaces are mapped in the main memory (MEM0). So, at hardware level, transfers between these memory spaces would not necessarily require a bridge because Simics allows us the definition of full-custom hardware architectures. We add a north bridge in our architecture in order to simulate a real and standard machine in which we can install a standard operating system (i.e.: Linux).

The computer of Fig. 10 includes two Pentium 4 CPUs, a DRAM module of 128 MB, an APIC bus, a PCI bus with a Tigon-3 (BCM5703C) gigabit Ethernet card attached, and a text serial console. The use of a text serial console is due to a limitation in Simics that at the moment is not able to have more than one machine running over a single Simics instance with graphical consoles. It only
can simulate and communicate several Simics instances through the Simics Central module. Furthermore, using text serial consoles (thus avoiding the use of graphical consoles), we have reached a faster simulation [48].

Once we have two machines defined and networked, Simics allows an operating system to be installed over them. For our purposes, we have used Debian Linux with a 2.6 kernel [26]. It allows us the necessary support for our system architecture and the implementation of the required changes. In the following section, we provide the experimental results obtained and analyze them according to the LAWS model.

5. Experimental results

In order to evaluate protocol offloading, we have used several Simics and operating system features. Using the kernel 2.6, it is possible to assign a CPU to the communication subsystem, isolating it from any other workload. This could be done with Linux cpuset [35], which avoids attaching processes to that isolated CPU. The cpusets are Linux lightweight objects that allow us the machine partition. This partition makes possible to assign memory nodes to each created cpuset object. Moreover, the memory assigned to a particular cpuset can be restricted to be exclusively used by this cpuset.

In this way, we have a system with a CPU, CPU0, for running applications and the operating system processes and another processor, CPU1, for running the communication subsystem.

In order to test our model and evaluate offloading, we have used netpipe [45], which is a protocol-independent tool that measures the network performance in terms of the available throughput between two hosts. It consists of two parts: a protocol independent driver, and a protocol specific communication section. The communication section depends on the specific protocol used, since it implements the connection and transferring functions, whereas the driver remains the same. For each measurement, netpipe increments the block size following its own algorithm.

In our experiments, we have used optimized network parameters in order to achieve the maximum throughput in every test, with or without offloading. For instance, we have used 9000 bytes MTUs (jumbo frames). We have also applied standard TCP windows, which sometimes produce oscillations in the throughput. This could be avoided by using oversized TCP windows (i.e.: 256 kbytes), but the maximum attainable throughput should not be affected.

In the following graphs (Figs. 11 and 12), we illustrate some experimental results using the TCP stack as transport-layer protocol [43] and a Gigabit Ethernet network. In Fig. 11, the loads of CPU0 in the no-offloaded and offloaded cases are compared.

When the protocol is offloaded, the load of CPU0 is lower, as it only executes the application that generates data and the NIC driver. When CPU0 has to generate data and process the protocol, its load grows up to 90% of the maximum: CPU0 is busy and there are not many cycles available for other tasks.

The curves of Fig. 12 provide the percentages of the interrupts requested to CPU0 in the non-offloading case that are avoided when the communication protocol is offloaded. Thus, a 0% in the figure means that with and without offloading, CPU0 receives the same number of interrupts, and a 50%, that when the protocol is offloaded, CPU0 receives half of the interrupts received in the non-offloading case.

The decrease in the interrupts per second obtained with offloading is about 60% for TCP, and about 50% for UDP. So, with regards to the offloading effects in the overall performance, the more cycles are required for protocol processing, the higher is the improvement in the time spent in interrupt servicing (less interrupts and less CPU time spent in servicing them). Thus, as TCP requires more CPU cycles to be processed than UDP, the benefits are more apparent in the case of TCP. In our simulations, we have not used techniques, such as interrupt coalescing, that are common in present NICs, as they cannot be supported by our Simic model for the NIC. If they would be simulated, a reduction in the interrupts with non-offloading should be observed, and probably the difference in the number of interrupts between offloading and non-offloading would be lower.

The results obtained with netpipe are shown in Figs. 13 and 14. These graphs provide the throughput for each transferred block size and the maximum attainable throughput. Fig. 13 shows the improvement that could be reached in the ideal case in which the NIC could communicate with the bus without latency: the
NIC that processes the protocols is connected to a bus without latency and it could access the onboard memory also without any latency. As the figure shows, the throughput obtained in this case can be almost the bandwidth of the network.

To obtain the results shown in Fig. 14, we have modeled the effect of having a non-ideal connection between CPU0 and the processor of the NIC, CPU1. In order to simulate this, we have introduced the corresponding timing models in the NIC bus and in the memory accesses from the NIC processor.

In Fig. 14a, the throughput for different latency values in the NIC accesses is shown against the message sizes. In the legend, Offload $x$, $x$ is the number of delays in memory accesses from CPU1, with respect to a reference value. So, Offload 2 means a double memory access delay from CPU1 as compared to Offload 1. We can see that the memory latency is decisive in the performance, as is mentioned in many previous papers (see for example [23]). The lower throughputs obtained in the case of small block sizes are due to the ACKs required by TCP protocol to transfer a block.

**Fig. 14.** Throughput of offloading vs. non-offloading: (a) for different memory latencies from the NIC; (b) for limited host (30% of the link bandwidth).
The curves in Fig. 14b correspond to a limited host (in this case, it can only deliver about 30% of the link bandwidth) without any other overhead source (i.e. without any application overhead) and a NIC with the same memory latency than the host CPU. The LAWS model will be considered below, to analyze these results in detail.
Fig. 15 shows the effect of the technology used to implement the NIC processor in the performance of protocol offloading. As this is one of the arguments to question the protocol offloading benefits, this analysis is important. In order to run the corresponding simulations, we have modified the step rate of the NIC processor. The curves in Fig. 15 correspond to the communication performance for a NIC processor running at 75%, 50%, and 25% of the host CPU speed.

### Table 1

<table>
<thead>
<tr>
<th>Offloading</th>
<th>Latency (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/1</td>
<td>24.4</td>
</tr>
<tr>
<td>10/2</td>
<td>31.68</td>
</tr>
<tr>
<td>10/4</td>
<td>42.53</td>
</tr>
<tr>
<td>10/6</td>
<td>46.61</td>
</tr>
<tr>
<td>No-offloading</td>
<td>66.9</td>
</tr>
</tbody>
</table>

Fig. 17. Saturation points for different offloading alternatives using different scales.
As we can see from Fig. 15, the speed of the NIC processor (CPU1) affects in a decisive way the throughputs. The performance gets worse as the processor speed decreases. Moreover, in the case of a very slow NIC processor, the performance for protocol offloading is even worse than the performance without offloading. So, it is clear that offloading improves the communication performance only if the processor included in the NIC is sufficiently fast compared with the host CPU (CPU0). Otherwise, offloading could even diminish the performance.

Fig. 16 shows the Round Trip Time vs. Block Size, and Table 1 provides the latency, both measured with netpipe. The meaning of Offload x in Fig. 16 is similar to that of Fig. 14a. It is clear that high throughput does not mean low latency. In our experiments with TCP, we have seen how latency depends a lot on TCP configuration factors. As the parameters we have used in TCP simulations have been optimized, the improvement obtained is less apparent than other effects such as throughput improvement, as is shown in Fig. 16. Table 1 provides the latency improvement for different offloading conditions.

Fig. 17a shows the saturation points for both offloaded and non-offloaded TCP experiments. The dependence between the saturation points and the offloading capabilities is clear. In Fig. 17b, the latency improvement is also shown. Nevertheless, the latency and the location of the saturation point have a heavy dependence on the TCP configuration, such as TCP buffers, the use of Nagle algorithm, etc.

5.1. The LAWS model and the simulation results

To conclude this section, we compare the results obtained from our simulations with the predictions provided by the LAWS model. Thus, Fig. 18 shows two curves corresponding to the peak throughput improvement against the application ratio (γ), respectively, predicted by LAWS, and the curve obtained experimentally with Simics. These specific curves correspond to the following values in the other parameters of LAWS: p = 0.75, α = 1, β = 0.40, and B = 1 Gbps. As can be seen, there are important differences between both curves, not only in the amount of throughput improvement achieved, but also in the location of the maximum and the rate of decreasing throughput improvement with the application ratio. To get a more accurate fit between the experimental and the theoretical curves and a more deep insight about the causes of the differences between the LAWS predictions and the experimental results, we have added three new parameters, δα, δω, and τ, to the expression of the peak throughput:

\[ \delta b = \frac{\min \left[ \frac{B}{\frac{1}{oX} + \frac{1}{aX} + \frac{1}{bX} + \frac{1}{cX} + \frac{1}{dX} + \frac{1}{oY} + \frac{1}{aY} + \frac{1}{bY} + \frac{1}{cY} + \frac{1}{dY}} \right] - 1}{\min \left[ \frac{B}{\frac{1}{oX} + \frac{1}{aX} + \frac{1}{bX} + \frac{1}{cX} + \frac{1}{dX} + \frac{1}{oY} + \frac{1}{aY} + \frac{1}{bY} + \frac{1}{cY} + \frac{1}{dY}} \right]} \]  

(2)

The effects which we try to model through the parameters δα, δω, and τ, can be understood from expression (2). The parameters δα and δω represent rates of change in the work per data after offloading, whereas the parameter τ is a rate of change in the CPU workload after offloading, due to the overheads of the communication between the CPU and the NIC through the I/O subsystem. Thus, parameter α changes to α + αδα in expression (2); α changes to α + αδω; and the CPU workload after offloading, W, changes to W + τW.

Fig. 19 shows that it is possible to get better approaches than the experimental results by using expression (2) and adequate values for δα, δω, and τ. In the figure, the curve LAWSmod(1) corresponds to values δα = 5 \times 10^{-2}, δω = 0 and τ = 5 \times 10^{-2}, and the curve LAWSmod(2) to δα = 5 \times 10^{-5}, δω = -26 \times 10^{-6} and τ = 0.147. It is clear that our modified LAWS model makes possible to approach the performance predictions to the experimental results. Particularly, it is possible to obtain very accurate information about the value of γ where experimental improvements higher than zero start, and the value of γ with the highest experimental improvement. In this way, according to the values of the parameters in the modified LAWS model that allows the best approach to the experimental curve, it can be concluded that after offloading: (1) the CPU workload requires more execution time (τ > 0); (2) the application workload increases as compared to that used
in the LAWS model ($\delta_a > 0$); (3) the communication overhead decreases ($\delta_b < 0$).

It can be also seen that in experimental results, the decrease of the throughput improvement as $\gamma$ increases is similar to the $1/\gamma$ rate predicted by LAWS model. This means that the LAWS model (with the modifications we have included) provides enough flexibility to get an accurate explanation of the offload behavior. Nevertheless, other models should be also considered to allow accurate predictions in circumstances different to those considered by the LAWS model (throughput-limited applications rather than message-oriented ones).

6. Conclusions

The scientific and engineering progress requires adequate tools to obtain experimental data. As has been previously claimed [31], to build a timing simulator for evaluating systems with workloads requiring operating systems support is difficult. This is the case of network-oriented simulations, as most part of the network code runs at the operating system level. In this paper, we leverage the full-system simulator Simics as the basis for modeling the timing of the memory system, the CPU, and the I/O buses to provide a suitable tool for researching in the computer system issues dealing with networking.

Moreover, in this paper, we have compared HDL simulation and full-system simulation to analyze the protocol offloading technique. First of all, an HDL model has allowed us the study of the offloading performance with an easy control of the parameters that determine the behavior of the different hardware elements. The experimental results obtained by the HDL simulations are qualitatively similar to that predicted by the LAWS model, although there are important quantitative differences among the obtained and the predicted improvements achieved by offloading. Moreover, the need for analyzing the system behavior under realistic workloads and traffic profiles (taking into account the interaction among operating system, hardware, and applications) requires a full-system simulation. To do that, we have used Simics.

Although Simics presents some limitations and it is possible to use other simulators for our purposes, the resources provided by Simics for device modeling and its debugging facilities make Simics an appropriate tool. Moreover, it allows a relative fast simulation of the different models. Thus, we have developed timing models that have been included in Simics to overcome the referred limitations of this full-system simulator (that does not provide either timing models or TOE’s models by itself). Thanks to the Simics models we have developed, it is possible to analyze the most important parameters and the conditions in which offloading determines greater improvements in the overall communication performance. The obtained simulation results show the improvement provided by offloading heavy protocols like TCP, not only in the ideal case, in which we use ideal buses, but also in more realistic situations, in which memory latencies and non-ideal buses are modeled.

The results obtained in our experiments show that offloading allows throughput improvements in all the cases where the host and the NIC processors have similar speed. Moreover, it is shown that offloading releases the 40% of the system CPU cycles in applications with intensive processor utilization. On the other side, we also present results that show how the technology of the processor included in the NIC affects the overall communication performance. The behavior which we have observed in our experiments coincides with the analyses and conclusions reached from the LAWS model. This situation constitutes an evidence of the correctness of our Simic model for protocol offloading. Nevertheless, as there are important quantitative differences among the LAWS predictions and the results of the Simics simulations, we have included some parameters in the LAWS model to take into account the effect of the memory accesses contention and the communication between the NIC and the CPU through the I/O subsystem.

In any case, the LAWS model can be only applied to environments in which throughput is limited either by network bandwidth or processing overhead rather than latency. However, other performance models can be analyzed with our simulation methodology to offer a wider knowledge about the offloading behavior in other scenarios, with the corresponding benchmarks.
and real applications. One of these performance models is the EMO model [19], applicable to message-oriented environments.

Acknowledgements

This work has been funded by projects TIN2007-60587 (Ministerio de Ciencia y Tecnología, Spain) and TIC-1395 (Junta de Andalucía, Spain). The authors also thank the reviewers for their suggestions.

References

Antonio F. Díaz received the M.Sc. degree in electronic physics in 1991, and the Ph.D. degree in 2001, all from the University of Granada, Spain. He is currently an Associate Professor in the Department of Computer Architecture and Computer Technology. His research interests are in the areas of network protocols, distributed systems and network area storage.

Pablo Cascón is a researcher and Ph.D. Student at the University of Granada. He received the M.Sc. degree in 2004 in Computer Science from the University of Granada. His research interest are in the fields of protocol offloading, network processors and high performance communications.

Alberto Prieto earned his B.Sc. in Physics (Electronics) in 1968 from the Complutense University in Madrid. In 1976, he completed a Ph.D. at the University of Granada. From 1971 to 1984 he was founder and Head of the Computing Centre, and he headed Computer Science and Technology Studies at the University of Granada from 1985 to 1990. He is currently a fulltime Professor and Head of the Department of Computer Architecture and Technology. Is the co-author of four text-books published by McGraw-Hill and Thomson editorials, has co-edited five volumes of the LNCS, and is co-author of more than 250 articles. His area of research primarily focuses on intelligence systems.