Modeling Network Behaviour By Full-System Simulation

Andres Ortiz
University of Malaga/Departamento de Ingenieria de Comunicaciones, Malaga, Spain
Email: aortiz@ic.uma.es

Julio Ortega, Antonio F. Diaz, Alberto Prieto
University of Granada/Departamento de Arquitectura y Tecnología de Computadores, Granada, Spain
Email: {julio, afdiaz, aprieto}@atc.ugr.es

Abstract— **Due to the network technology advances, an order-of-magnitude jump has been produced in the network bandwidth. This fact has returned to wake up the interest on protocol offloading, since network communication is a key factor for system performance. Thus, much research work on offloading is been done, particularly on TCP offloading as it has been the main transport protocol for many years, and in fact, it is the most used protocol in the networks. Nevertheless, some recent studies on protocol offloading have produced some controversy about its benefits.**

This paper describes a model to simulate the protocol offloading by using the simulator Simics. It can be used for a functional system simulation, including the application program, the operating system, the protocol stack and the device drivers, since network-oriented applications require this kind of full-system simulation. Nevertheless, as Simics does not provide a detailed network I/O model, in this paper we describe the way we have overcome this problem.

Index Terms—**full-system simulation, simics, offloading,**

I. INTRODUCTION

In high performance computing (HPC) platforms, network communication is a key factor that determines the performance. A communication bottleneck may lead to a significant loss of overall platform performance. With the availability of high bandwidth network links such as Gigabit, 10-Gigabit or 40-Gigabit Ethernet, an order-of-magnitude gain in network performance is expected. This is the performance achieved with Myrinet or SCI networks, specialized for cluster and high-performance parallel platforms environments. Nevertheless, processing protocols such as TCP/IP requires significant CPU resources. This way, the communication bottleneck has been moved to node's CPU. So, the network interface (NI) performance is determinant in the overall communication path performance, being specialized software and hardware (offload engines) the technology that reduces the processing constraints (overhead associated to protocol processing) due to context switching, multiple data copies and interrupt mechanisms. As a solution to reduce the communication overhead, offload protocol processing from the CPU to the network interface card (NIC) has been proposed.

This way, the CPU does not have to process the communication protocols, and the NIC can directly interact with the network without the CPU participation, thus allowing both the decrease in the interrupt processing cost for the CPU, and the decrease of protocol latency for short control messages (like ACKs) that, this way, do not have to go to the main memory through the I/O bus.

Nevertheless, besides the works showing the advantages of protocol offloading, some recent papers have presented results arguing that this technique does not benefit applications. Particularly, TCP/IP offloading has been highly controversial because, as some studies have demonstrated, TCP/IP processing costs are small compared to data transference overheads and the costs of interfacing the protocol stack to the NIC and the operating system. The advantage on using TCP/IP is the connectivity with other systems and other networks through routers, switches, etc, since it has been the main transport protocol for the Internet Protocol Stack for years.

The continuous increase in the network data rates and the interest on IP storage due to the emergence of network storage protocols, such as iSCSI, to replace storage-specific networks (SCSI, FiberChannel,…) by commoditized networks (Ethernet) have contributed to maintain the debate about TCP/IP offload. Thus, models such as LAWS [20] and EMO [21] have been proposed to provide a frame for interpreting the experimental results and to drive the discussions over offload technologies, allowing us the exploration of the offloading space of design in stream-oriented (in the case of LAWS) and message-oriented (in the case of EMO) applications.

In order to evaluate computer architecture proposal and check its performance, simulation is the most frequent technique. So it is necessary to have a simulation platform that allows us to model the architecture in a real way. This way, the simulator has to simulate the machine with sufficient detail, and it must be possible to drive it with realistic workloads such as web server, databases or even benchmarks. So, simulators that run only user-

mode, single-threaded workloads are not adequate [24]. A full-system simulator allows the execution of complete software stacks from real systems without any modification. Simics [1,4] is a full-system simulator that fulfils these requirements and it is the simulation tool we have used in our work. Nevertheless, Simics presents some difficulties for a network-oriented simulation.

In this paper, after a brief introduction to protocol offloading and the controversy around its usefulness (Section 2), we describe the way we have used Simics to analyze protocol offloading (Section 3). Section 4 provides the experimental results and the conclusions are given in Section 5.

II. PROTOCOL OFFLOADING

Over the last years, and due to the advances in network technology, much research work has been done in order to optimize the network processing in end-systems. In fact, many actual vendors are designing and manufacturing Offload engines, also called TOE (TCP Offload Engines) in the case of TCP protocol, and in general, other NICs (Network Interface Card) with Network processors (NPs) that provides the Network Interfaces with some processing capabilities. The vendors argue that this processing capabilities on the NICs are necessary to grant data flows at network speed with the latest technology (10 Gigabit Ethernet or 40 Gigabit Ethernet, for instance). Thus, much research work has been done in order to improve the communication performance.

This research can be classified into two complementary alternatives. One of these alternatives searches for decreasing the software overhead in the communication protocols either by optimizing the TCP/IP layers, or by proposing new and lighter protocols. Moreover, these new protocols usually fall into one of two types: the protocols that optimize the operating system communication support (GAMMA [14], CLIC [15]), and the user level network interfaces [16], such as the VIA (*Virtual Interface Architecture*) standard [17].

The other researching alternative in this field tries to take advantage of the improvement of the NIC resources. Many of the interconnection networks used in current cluster-based computing systems include NICs with programmable processors (also called *Intelligent* NICs, INICs), and much research has been done towards the use of these processors to offload processing from the host CPU. This way, the CPU is free from communication overheads and it is possible a faster implementation of more flexible communication systems. It is also possible to include network processors (NP) [5, 6, 18] in the NIC

There are some advantages that offloading communication functions to the NIC can provide:

1. An increment in the CPU cycles available for application processing
2. The protocol latency can be reduced as short messages, such as the ACKs, do not need to travel through the E/S bus; and (b) the CPU does not have to process interrupts for context changing to attend the received messages.
3. DMA efficiency improvement due to short messages assembling.
4. Bus contention can be reduced due to less use of I/O system bus.
5. Improvement in the efficiency of the communication protocols due to the possibility of dynamic protocol management.

Nevertheless, some works [9-12] provide experimental results to argue that protocol offloading, in particular TCP offloading, does not clearly benefit the communication performance of the applications. On the one hand, the reasons for this scepticism are the difficulties in the implementation, debugging, quality assurance and management of the offloaded protocols [9]. 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 [12] (cited in [9]). Protocol offloading requires the coordination between the NIC and the OS for a correct management of resources such as the buffers, the port numbers, etc. In case of protocols such as TCP, the control of the buffers is complicated and could hamper the offloading benefits (for example, the TCP buffers must be held until acknowledged or pending reassembly). 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 [9]. Probably, these are not definitive arguments with respect to the offloading usefulness but they counterbalance the possible benefits.

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 processors in the NIC and, moreover, the increment in the CPU speeds according the Moore's law tends to maintain or even to increase this ratio. Thus, the part of the protocol that is offloaded would require more execution time in the NIC than in the CPU, and the NIC could appear as 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) available in the NIC could imply restrictions in the system scalability (for example, limitations in the size of the IP routing table).

These problems 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, it seems that this technique is more adequate in applications requiring high bandwidths, low latencies, and long-term connections [9]. For example RDMA (Remote Direct Access Memory) is a protocol that allows 0-copy packet transference to the corresponding memory buffer. As the RDMA component DDP (Direct Data Placement) requires an early demultiplexing of the input

packets, its implementation in the NIC can be advantageous.

Nevertheless, there are other works that demonstrate the benefits of TCP offloading. In [8] 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-emulated offload.

The paper by Shivam and Chase [20] describes the LAWS model to characterize the protocol offloading benefits in Internet services and streaming data applications. The LAWS model is based on four parameters: the ratio of CPU speed to NIC speed (***Lag ratio***); the ratio of application processing needs to communication processing overhead (***Application ratio***); the ratio of host saturation bandwidth to raw network bandwidth (***Wire ratio***); and the ratio of overhead for communication with offload to overhead for communication without offload (***Structural ratio***). As the ratios used by LAWS are independent of a particular protocol this model is extensible. Among the conclusions of the analysis provided in [20] for through-limited applications (Internet servers) is that offloading is more worthwhile as application and structural ratios decrease, for example, with faster networks (application ratio decrease) and improvements in the offloading techniques (structural ratio decrease).

Nevertheless, LAWS is stream-oriented and useful in applications where the end-to-end throughput depends on network bandwidth or processing overhead more than on latency. Thus, it is not very useful to understand the latency minimization in the parallel applications. In [21], Gilfeather and Maccabe propose EMO (Extensible Message-oriented Offload model) to analyze the performance of various offload strategies for message oriented protocols. It is based on the cycles of protocol processing, *C*, on the NIC and CPU (at the OS, and application levels); the CPU cycles, *O*, to move data and control between the NIC, the OS and the application; the time, *L*, to move data and control between the NIC, the OS and the application; and the rate, *R*, of the CPU and the processor in the NIC. The benefits of offloading techniques, such as splintered TCP [28], are also justified in [21], and the circumstances in which offloading significantly reduces latency and allows the application control over protocol overhead are also described. It is possible to map EMO onto LAWS by taking into account that while EMO considers an arbitrary amount of time for a specified amount of bytes, LAWS considers an arbitrary number of bytes in a specified amount of time.

III. TOWARDS A REALISTIC 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 it 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 [2]. There are not many simulators with these characteristics. Some examples are M5 [3], SimOS [26], and some other simulators based on Simics [1,4], such as GEMS [24] and TFsim [25].

Simics [1,4] 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 modelled. Nevertheless, Simics is a functional simulator and does not provide an accurate timing model. Precisely, in [22] Simics is extended with detailed timing models. In [23] some are reported with respect to the capabilities of Simics in the model of x86 processors (out-of-order microarchitectural issues such as branch prediction, use of reorder buffer, the number of functional units, etc. are not modelled) and 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. On the other side, Simics is a fast functional simulator that makes it possible to simulate complex applications, operating systems, network stack protocols, and other real workloads.

In this paper, we consider the use of Simics for protocol offloading evaluation. In order to do this, we need a network interface model that processes the protocol instead the main CPU of the system. Simics has several limitations that are necessary to have into account to simulate 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 [2].

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 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 could be modified, but the lower bound is determined by the CPU speed. So, it is necessary to ensure that the minimum latency we are simulating is enough to allow the maximum bandwidth needed for our simulation 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 used Simics instead of simulators such as M5 [3] or SimOS [26] 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 modelling devices. Furthermore, because of its C++ features, the debugging process in simulators such as M5 and SimOS is harder compared with that in Simics, that also provides effective tools for debugging and profiling. There are other simulators, such as GEMS [24] or TFSim [25], based in Simics, 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, we have built a Simics simulation model by defining two customized machines and a standard Ethernet network connecting them in the same way 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 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 as using a crossover Ethernet cable in the real world.
We have used two models in our simulations. The first one corresponds to the 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) having a scaled model avoiding to slow down the simulation work. We have used Simics NIC gigabit models in the BCM5703 PCI based Ethernet card included in our system. We have assumed a memory 10 times slower than the CPU. The model is shown in Figure 1.

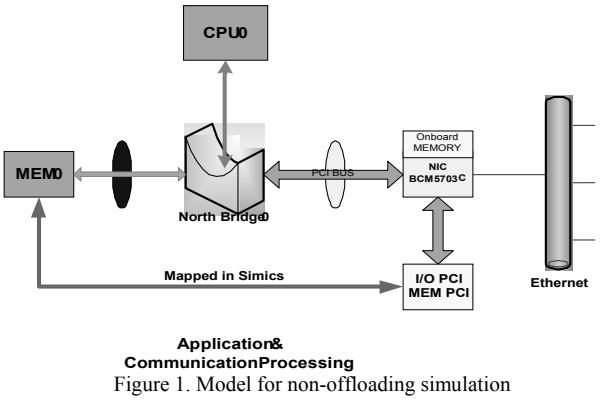

Figure 1. Model for non-offloading simulation

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 4.

In order to offload the protocols, and so discharge the protocol processing work from the CPU, we have used the model shown in Figure 2, corresponding to a system in which one of the processors has been isolated from the other and the NIC is connected directly to this CPU in order to improve the parallelism between application and network processes. In Simics, by default, the bridges merely act as connectors. In this case, they do not model any contention at simulation time. The way to simulate a contention model is through timing models, by connecting a timing model to each entry of the bridge where access contention is possible. This is not a precise way to model contention, but it provides an adequate simulation of the contention behaviour. Thus, in our model (Figure 2), 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, transferences between CPUs and memory would not hold any other transference.

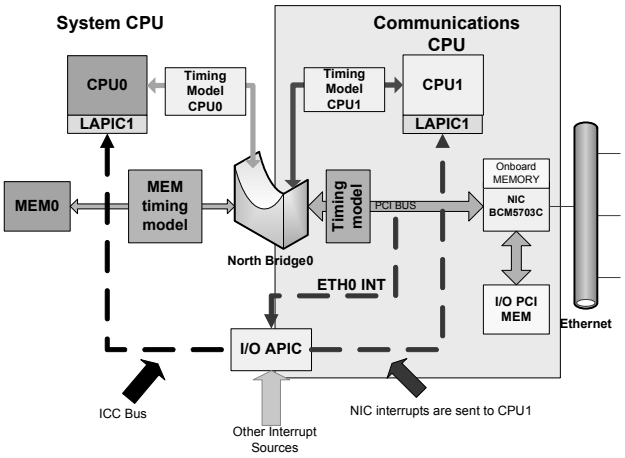

Figure 2. Model for offloading simulation

On the other hand, in Simics, PCI I/O and memory spaces are mapped in the main memory (MEM0). So, at hardware level, transferences between these memory spaces would not necessarily require a bridge because Simics allows us the definition of a full-custom hardware architecture. 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 Figure 2 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 reach a faster simulation.

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 which will allow us the necessary support for our system architecture and the implementation of the required changes.

On the operating system layer, we call a TCP/IP thread and it is bounded to the processor in the NIC. The above architecture makes possible the TCP/IP stack execution without disturbing the remaining processors of the host (CPU0 in this case). This way, we have isolated the processor in the NIC to remove it from cross-call participation. Furthermore, since the processor in the NIC and the host processor have their own memory spaces that can be accessed concurrently, the host processor is not slowed down by memory or bus traffic.

IV. EXPERIMENTAL RESULTS

In order to evaluate protocol offloading, we have used several Simics and operating system features, in a similar way that in [8]. This way, using the kernel 2.6 facilities such as cpuset objects, it is possible to assign a CPU to the communication subsystem, isolating it from any other load, running no other threads and receiving no interrupts. The *cpuset* objects make it possible to avoid attaching processes to the isolated CPU's. They are Linux lightweight objects that allow us the machine partition. This applies to the CPUs as well as to the memory. Thus, the memory used by a *cpuset* object can be restricted to some of the nodes of a NUMA system.

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

The goal using a full-system simulator is to examine the effect of parameters that could modify the offloading performance, such as the difference between speeds in the host and in the NIC CPUs.

In order to test our model and evaluate offloading, we have used *netpipe* [27], 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. For UDP measurements we have used *netperf* [29].

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 are using MTU 9000 (jumbo frames). Also we have used standard TCP windows, which sometimes produce oscillations in the throughput. This could be avoided using oversized TCP windows (i.e.: 256 Kbytes), but the maximum attainable throughput should not be affected.

In the following graphs (Figure 3 and 4), we illustrate some experimental results using TCP or UDP protocols and a gigabit network. In Figure 3, the loads of CPU0 in the no-offloaded and offloaded cases are compared.

TABLE I.
LATENCY FOR DIFFERENT OFFLOADING ALTERNATIVES

Offloading	Latency (μs)
1	24.4
2	31.68
4	42.53
6	46.61
10	120.54
No Offloading	66.9

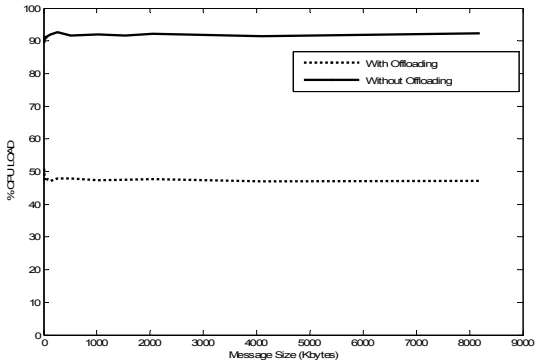

Figure 3. CPU load curves with and without offloading

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

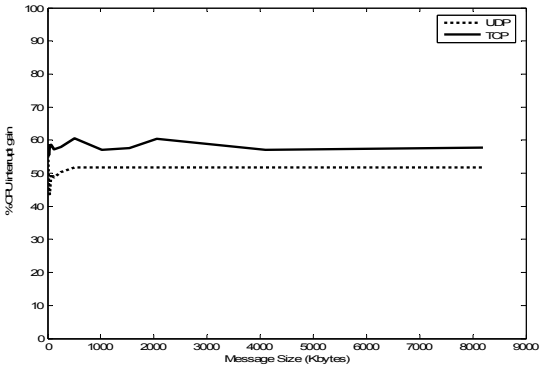

Figure 4. Decrease in the number of interrupts

The curves of Figure 4 provide the benefit on CPU time associated to the interrupts when the communication protocol is offloaded. Thus, a 50% in the figure means that when the protocol is offloaded, CPU0 receives half of the interrupts that it received in the non-offloading case, and a 0% that with and without offloading, CPU0 receives the same number of interrupts.

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, as more cycles are required for protocol processing, higher is the improvement in the time spent in interrupt servicing (less interrupts and less CPU time spent processing them). Thus, as TCP requires more CPU cycles than UDP, the benefits are more apparent in the case of TCP.

The results obtained with *netpipe* are shown in Figures 5 to 8. These graphs provide the throughput for each transfer block size and the maximum attainable throughput as well as the latency.

Figure 5 shows the Round Trip Time *vs* Block Size and Table 1 provides the latency, both measured with *Netpipe*. In the notation, *Offload x*, *x* is the number of delays, with respect to a reference value, to access memory from CPU1. So, Offload 2 means twice more delays in memory accesses from CPU1 than Offload 1. As we can see, memory latency is an important bottleneck in the communications path, and its effect is more noticeable for small packets. Therefore, the implementation of techniques that improves memory accesses, such as DMA or any improved DMA technique [30] are necessary in order to obtain benefits from any protocol offloading technique.

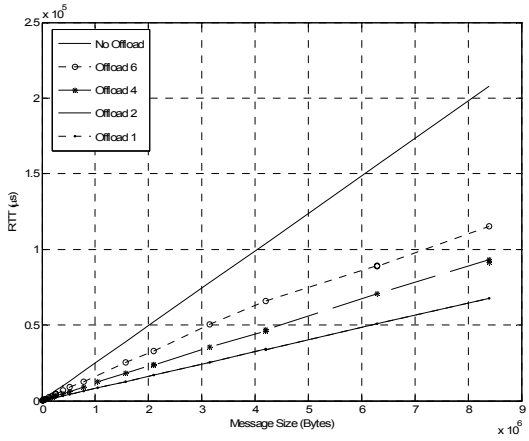

Figure 5. RTT on Gigabit Ethernet with and without offloading

The experiments we have performed were done over Gigabit Ethernet, by using TCP as transport protocol. Higher throughputs do not imply lower latencies. In TCP, the latency depends on a lot of TCP configuration factors. In the test we have performed, TCP parameters have been optimized, and so, this is the reason why the improvement obtained with offloading could be less noticeable then other effects (such as throughput improvement). This can be seen in Figure 5. However, in Table 1, we can see the latency improvement and in Figure 6a the saturation point in both offloaded and non-offloaded TCP cases, in which we can see how the saturation point depends on the offloading capabilities. In figure 6b we can see the improvement in latency. Nevertheless, latency and saturation point have a heavy dependence on TCP configuration such as TCP buffers, or the use of Nagle Algorithm.

To obtain the results that show the improvements caused by offloading, we have modelled the effect of having a non-ideal connection between the CPU0 and CPU1. In order to simulate this, we have introduced the corresponding timing models in the NIC bus and in the memory accesses from the processor of the NIC.

Figure 7 shows the throughput improvement for different offloading configurations. In Figure 8, it is shown the effect of the different speeds of CPU0 (host CPU) and CPU1 (NIC CPU).

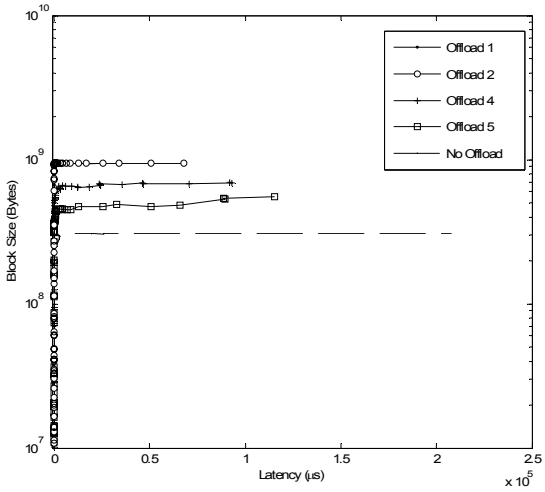

Figure 6a. Saturation points for different offloading alternatives

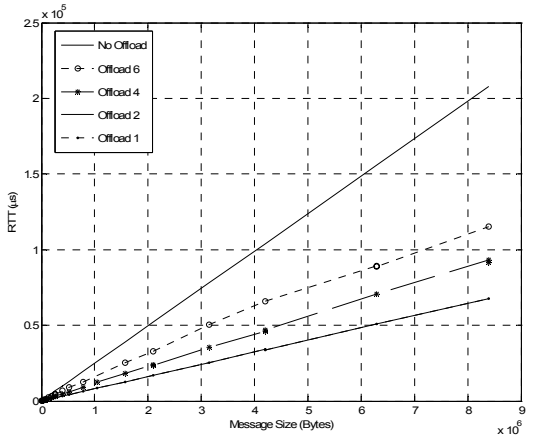

Figure 6b. Saturation graph. Latency detail

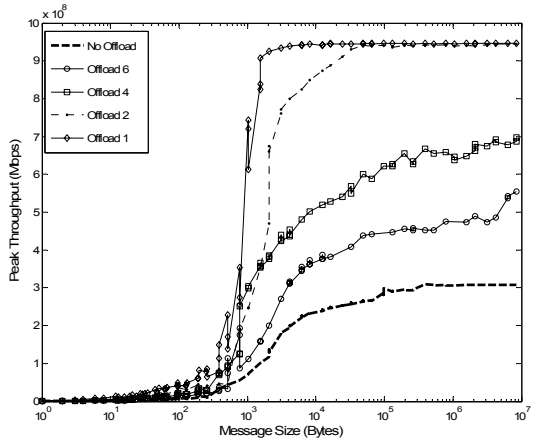

Figure 7. Throughput improvement

In Figure 7, the throughputs for different latency values in the NIC accesses are shown and as we can see, the memory latency is decisive in the performance obtained. The lower throughputs obtained in the case of small block sizes are due to the ACKs required by any TCP block transference.

Figure 8 shows the effect of the technology of 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 interesting. In order to do the corresponding simulations, we have modified the step rate of the NIC processor. The curves in Figure 8 correspond to the communication performance for NIC processors running at 75%, 50% and 25% of the host CPU.

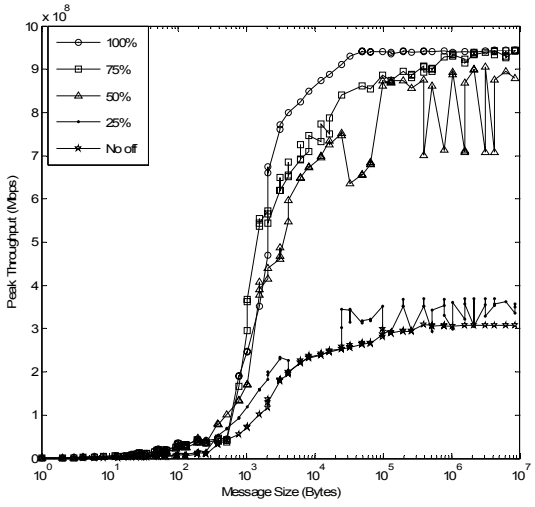

Figure 8. Throughput for different NIC processor speeds

As we can see form Figure 8, the speed of the processor at the NIC (CPU1) affects in a decisive way to 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 (CPU1) is sufficient fast compared with the system CPU (CPU0). Otherwise, offloading could even diminish the performance.

V. CONCLUSSIONS

In this paper we have considered the use of Simics to analyze protocol offloading. Although Simics presents some limitations and it is possible to use other simulators for our purposes, the resources provided by Simics for device modelling and the debugging facilities, make Simics an appropriate tool. Moreover, it allows a relative fast simulation of the different models.

The simulation results obtained shows 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 modelled. Thanks to the Simics model, it is possible to analyse the most important parameters and the conditions in which offloading determines greater improvements in the overall communication performance.

The results obtained in the experiments we have done show throughput improvements in all the cases where the host 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 behaviour we have observed in our experiments coincides with the analyses and conclusions provided in papers such as [20]. This situation constitutes an evidence of the correctness of our Simics model for protocol offloading. Nevertheless, we are now working in the quantitative analysis of our experimental results according to the expressions provided by the LAWS model [20] and the mapping of EMO model [21] onto LAWS.

ACKNOWLEDMENTS

This work has been funded by projects TIN2004-01419 (Ministerio de Ciencia y Tecnología, Spain) and TIC-1395 (Junta de Andalucía, Spain).

REFERENCES

[1] Magnusson, P. S.; et al.:"Simics: A Full System Simulation Platform". IEEE Computer, pp.50-58. February 2002.
[2] Binkert, N.L.; Hallnor, E.G.; Reinhardt, S.K.:"Network-oriented full-system simulation using M5". Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CECW). February, 2003.
[3] M5 simulator system Source Forge page: http://sourceforge.net/projects/m5sim
[4] Virtutech web page: http://www.virtutech.com/
[5] Chakraborty, S. et al.:"Permance evaluation of network processor architectures: combining simulation with analytical estimation". Computer Networks, 41, pp.641-645, 2003.
[6] Thiele, L. et al.:"Design space exploration of network processor architectures". Proc. 1st Workshop on Network Processors (en el 8th Int. Symp. on High Performance Computer Architecture). February, 2002.

[7] Cruz, R.L.:"A calculus for network delay". IEEE Trans. on Information Theory, Vol.37, No.1, pp.114-141, 1991.
[8] Westrelin, R.; et al.:"Studying network protocol offload with emulation: approach and preliminary results", 2004.
[9] Mogul, J.C.:"TCP offload is a dumb idea whose time has come". 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.
[10] Reginier, G. et al.:"TCP onloading for data center servers". IEEE Computer, pp.48-58. November, 2004.
[11] Clark, D.D. et al.:"An analysis of TCP processing overhead". IEEE Communications Magazine, Vol. 7, No. 6, pp.23-29. June, 1989.
[12] O'Dell, M. :"Re: how bad an idea is this?". Message on TSV mailing list. November, 2002.
[13] IETF RDDP working group documents: http://www.ietf.org/ids.by.wg/rddp.html
[14] Ciaccio, G.: "Messaging on Gigabit Ethernet: Some experiments with GAMMA and other systems". Workshop on Communication Architecture for Clusters, IPDPS, 2001.
[15] Díaz, A. F.; Ortega, J.; Cañas, A.; Fernández, F.J.; Anguita, M.; Prieto, A.: "A Light Weight Protocol for Gigabit Ethernet". Workshop on Communication Architecture for Clusters (CAC'03) (IPDPS'03). April, 2003.
[16] Bhoedjang, R.A.F.; Rühl, T.; Bal, H.E.: "User-level Network Interface Protocols". IEEE Computer, pp.53-60. November, 1998.
[17] http://www.vidf.org/ (Virtual Interface Developer Forum, VIDF, 2001).
[18] Papaefstathiou, I.; et al.: "Network Processors for Future High-End Systems and Applications". IEEE Micro. September-October, 2004.
[19] Hyde, D.C.:"Teaching design in a Computer Architecture Course". IEEE Micro, pp.23-27, May-June, 2000.
[20] Shivam, P.: Chase, J.S.:"On the elusive benefits of protocol offloas". SIGCOMM'03 *Workshop on Network-I/O convergence: Experience, Lesons, Implications (NICELI)*. August, 2003.
[21] Gilfeather, P.; Maccabe, A.B.:"Moeling protocol offload for message-oriented communication". Proc. of the 2005 IEEE International Conference on Cluster Computing (Cluster 2005) , 2005.
[22] Alameldeen, A.R. et al.: "Simulating a $2M Comercial Server on a $2K PC". IEEE Computer, pp.50-57. February, 2003.
[23] Villa, F.J.; Acacio, M.E.; García, J.M.:"Evaluating IA-32 web servers through simics: a prectical experience". Journal of Systems Architecture, 51, pp.251-264, 2005.
[24] Martin, M.M.; et al.: "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset ". Computer Architecture News (CAN), 2005.
[25] Mauer, C.J.; et al.:"Full-System Timing-First Simulation". ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, June, 2002.
[26] Rosenblum, M.; et al.:"Using the SimOS machine simulator to study complex computer systems". ACM Tranns. On Modeling and Computer Simulation, Vol.7, No.1, pp.78-103. January, 1997.
[27] Q.O. Snell, A.R. Milker, and J.L Gustafson, "NetPIPE: Network Protocol Independent Performance Evaluator", Ames Laboratory / Scalable Computing Lab, Iowa State., 1997.
[28] Gilfeather, P.; Maccabe, A.B.:"Splintering TCP". http://www.cs.unm.edu/~pfeather/papers/ucf02.pdf
[29] Netperf Homepage, http://www.netperf.org
[30] Intel I/O AT Acceleration Technology. http://www.intel.com/technology/ioaccelertation/index.htm, 2007

Andres Ortiz Garcia received the Ing. degree in Electronics Engineering from the University of Granada in 2000. From 2000 to 2005 he was working as Systems Engineer with Telefonica, Madrid, Spain, where his work areas were high performance computing and network performance analysis. Since 2004 he has been with the Department of Communication Engineering at the University of Malaga as an Assistant Professor. His research interests include high performance networks, mobile communications, RFID and embedded power-restrained communication devices.

Julio Ortega received the B.Sc. degree in electronic physics in 1985, the M.Sc. degree in electronics in 1986, and the Ph.D. degree in 1990 (Ph.D. Award of the University of Granada), all from the University of Granada (Spain). He was with the Open University, UK, and the Department of Electronics, University of Dortmund, Germany, as invited researcher in 1992 and 1993, respectively. Currently, he is full professor at the Department of Computer Architecture and Technology of the University of Granada. His research interests include parallel processing and parallel computer architectures, evolutionary computation and optimization.

He has led projects in the areas of parallel algorithms and parallel platforms for optimization, and communication architectures in clusters.

Antonio F. Diaz 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.

Alberto Prieto received the B.Sc. degree in electronic physics from the Complutense University, Madrid, Spain, in 1968, and the Ph.D. degree from the University of Granada, Spain, in 1976.

He is currently a full professor and Director of the Department of Computer Architecture and Technology of the University of Granada. He has been a visiting researcher in several centers including the University of Rennes, France, the Open University, UK, the Institute National Polytechnique of Grenoble, France, the University College of London, UK, and the Department of Electronics at the University of Dortmund, Germany. His research interests are in the area of Computer Architecture and Intelligent Systems.

Dr. Prieto was nominated as a member of the IFIP WG 10.6 Neural Computer Systems and Chairman of the Spanish RIG of the IEEE Neural Networks Council. He received the Ph.D. Dissertations Award of the Citema Foundation National Award in 1976.