We need to talk about NICs

Pravin Shinde[*]*, Antoine Kaufmann, Timothy Roscoe, Stefan Kaestle*
Systems Group, ETH Zurich

Abstract

Operating systems fail both to efficiently exploit, and to effectively manage, the considerable hardware resources of modern network interface controllers. We survey the kinds of hardware facilities available and their applicability, and then investigate (and critique) the reasons why OS designers eschew core support for such features.

We then describe Dragonet, a new network stack design based on explicit descriptions of NIC capabilities, aimed at making the best use of today's and tomorrow's networking hardware. Dragonet represents both the physical capabilities of the network hardware *and* the current protocol state of the machine as dataflow graphs. We then embed the former into the latter, instantiating the remainder in software.

1 Introduction

Networks are getting faster. Cores are not. If computers are to handle future bandwidth and latency requirements, they will need a combination of parallelism across cores, and specialized network interface hardware.

Without parallelism, applications cannot scale to handle more data in unit time. Without specialized hardware, processing of incoming network packets before demultiplexing (and of outgoing ones after multiplexing) executes serially in software, leading to basic scalability limits via Amdahl's law. Worse, multiplexing aside, the limits of multicore scaling [9] will ultimately make software protocol processing a bottleneck.

Fortunately, modern network interface controllers are highly complex devices, featuring a bewildering array of functions: multiple receive and transmit queues, TCP offload, traffic shaping, filter rules, virtualization, etc. Most of these aim at improving performance in some common enough cases: increasing bandwidth, improving QoS isolation, reducing CPU utilization, minimizing latency, etc. These also frequently work: Section 2 presents a brief survey of such features, their benefits, and some results of our own confirming the advantages of multiple receive queues for improving performance isolation, while also observing their sensitivity to configuration changes.

Unfortunately, OS support for these features is pitiful. As we discuss in Section 3, excepting some ad-hoc features in Windows [26], an OS protocol stack is based on a simple 1980s NIC and does not cope well with today's plethora of feature sets and their programming interfaces. Indeed, Linux explicitly avoids support for all but the simplest hardware protocol offload, for reasons that make sense from the perspective of the kernel maintainers, but which do not hold in a broader perspective.

The problem is thus forced onto hardware vendors. Support for such functionality is isolated in individual device drivers, resulting in a mess of non-standard configuration tools, arbitrary resource policy hard-coded into drivers, and in some cases completely replicated network protocol stacks. This is madness: the OS should be managing these resources, allocating them appropriately among competing applications based on system-wide policy, and providing appropriate abstractions so that applications can benefit without being rewritten for each new piece of hardware.

We are rethinking the OS network stack around explicit, machine-readable descriptions of network hardware. Whereas to date the only description of a NIC the OS can work with is a driver binary with a generic interface, in Section 4 we show how to describe NICs as dataflow programs operating on packets. We embed the associated graphs into a dataflow graph representation of the current OS network state. The approach has many potential benefits: it addresses most of the issues in Section 3, and can provide detailed performance models to guide OS resource allocation policy.

[*]Pravin Shinde is supported by a Microsoft PhD Fellowship.

2 The state of NIC hardware

There is a long history of offloading protocol processing to NICs. We now survey the broad landscape of modern NIC features, and the performance benefits they promise.

Even simple, old Ethernet NICs (e.g. [37]) provide functionality like multicast address filtering, to reduce CPU load caused by processing unwanted packets. Early NICs also reduced CPU load by offloading IP checksum calculation and validation of packet lengths, protocol fields, etc. The value of this functionality by itself is now debatable as CPUs are much faster, but it remains as a prerequisite for further acceleration.

Starting from relatively simple TCP Segmentation Offload (TSO), vendors have implemented increasingly complex TCP Offload Engines (TOEs), experimenting with different mixes of software and hardware functionality, including the use of embedded processors [18] and intelligent coalescing of interrupts based on packet properties [46]. The problem of handing off connections between OS and offload stacks has also been investigated [19]. TOEs can improve web server capacity by an order of magnitude [10], but they remain controversial due to complexity [44] and limits to applicability [27], an issue we return to in Section 3.

Multicore processors have also led to NIC features to aid scalability: multiple hardware queues for send and receive, Receive-Side Scaling (RSS), and configurable per-queue interrupt routing. The value of such features has been shown in many scenarios, for example, Routebricks provides a detailed analysis for software routers [7]. Affinity accept [34] extends this work to connection setup packets, deriving further benefits at the cost of minor changes to POSIX socket semantics.

Networks are now so fast that the memory system can become the bottleneck. Direct Cache Access (DCA) delivers incoming packets of a flow to the data cache of a selected core, bypassing RAM and thereby reducing memory bandwidth and latency, although performance is highly sensitive to correct configuration [12, 21].

At the same time, the CPU-intensive nature of cryptographic operations have led to some NICs integrating SSL and other crypto accelerators onto the data path [5]. Increasingly important network protocols like RDMA [36] and iSCSI [31] are designed assuming hardware acceleration, and supercomputers employ NICs with hardware support for MPI.

The rise of virtualization has also led to NIC support for virtual devices that can be directly mapped into virtual machines [33, 35]. Plentiful hardware queues help to provide quality of service and isolation. Modern virtualization-aware NICs [42] also provide onboard IOMMUs for address translation and protection on DMA transfers, per-flow queuing, traffic shaping on transmit, and even built-in switching functionality [39].

Given this complexity, it is unsurprising that some NICs are now fully programmable processors themselves [30], and are even used to prototype TCP offload engines [1] and self-virtualizing interfaces [35], as well as applications like intrusion prevention systems which perform wire-speed inline packet filtering [4]. On the other hand, configurable hardware such as FPGAs are used for niche applications such as algorithmic trading [23] and are integrated in some high-end NICs [43].

While undoubtedly useful, realizing the full benefit of hardware NIC features is difficult. In an arbitrary, but illustrative, example, each graph in Figure 1 shows the result of the same experiment run on different Linux servers using an Intel 82599 10GbE NIC [17] with support for hardware queues. We run two `iperf` [45] server processes on different cores, while the `iperf` clients execute on other machines. One client measures TCP throughput for 60 seconds, while the other adds cross-traffic load to the experiment after 10 seconds. The `DEFAULT` data shows the behavior of regular Linux trying to assign dedicated queues automatically, `OPTIMAL` the performance of manually dedicating queues and setting interrupt forwarding to the appropriate cores, and `SUBOPTIMAL` the impact of suboptimal interrupt affinity.

We make three observations: First, hardware features can dramatically affect performance: in the first plot, the default configuration deals poorly with cross-traffic. Second, Linux's policy can do well in some cases but fails to consistently find good configurations during a run (the second plot). Finally, the third plot shows poor performance of all the test configurations for at least one server.

In summary, hardware vendors incorporate ever-more sophisticated capabilities into NICs, with demonstrable benefits, but in most cases such benefits are not automatic, and careful system-specific configuration may be needed to fully realize them.

3 The state of OS network support

Modern OSes do not provide a good framework for NIC accelerators, and TCP Offload is a good example which illustrates the wider issues surrounding OS support for new NIC functionality. Linux does not support TOEs, for reasons the developers have made clear [44], and Mogul [27] also surveys similar arguments. We recap, and critique, these arguments here.

Deployment issues: These take many forms: Intelligent NICs use proprietary firmware the OS vendor cannot maintain, leading to security bugs and poor RFC compliance, configuration requires vendor-specific tools, support can only come from the hardware vendor (often a small company likely to go under), etc. Consequently, the OS is not granted the visibility into, and control over

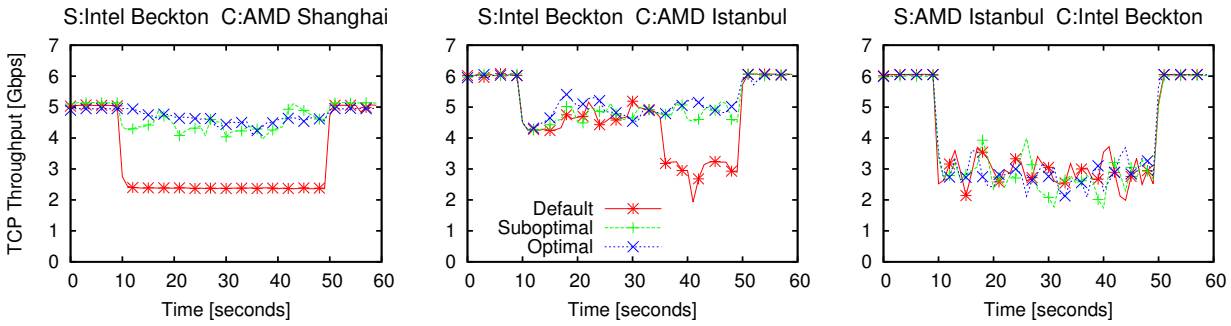

Figure 1: Impact of dedicating hardware queues and setting interrupt affinity on TCP throughput.

hardware required to provide solid OS functionality. We remark that GPUs face a similar visibility challenge, and researchers are exploring better ways to integrate them in the OS [38].

From a *research* perspective this is politics and business relationships, and it is myopic to thus exclude a large area of the design space: we should think about appropriate designs, not short-term irritations having little to do with technology. Consequently, in our work we deliberately bracket these considerations. Instead of shying away from programming any hardware that is not a CPU, OS researchers should pursue OS designs that aim to manage them [32].

Any performance gain is short term: The argument goes that cores will always get fast enough to render any hardware feature irrelevant for performance.

While valid once upon a time, this thinking is an outdated relic of the "free lunch" software era. Until the physics of logic gates changes, cores will not get faster.

Multiplexing is an essential part of protocol processing, and any attempt to move this into software results in a serialization point where Amdahl's law fundamentally limits scalability. Unlike 20 years ago with multimedia networks, sitting around and waiting will not render intelligent NICs redundant. Moreover, with processor architecture moving towards large sets of specialized processing elements [9], we can view NIC hardware as one subset of this trend. We need to understand how to write OS software for this kind of system.

The benefits are marketing fiction: Mogul [27] observes that hardware interfaces were often poorly designed for performance under many common workloads. For example, RDMA shows highly idiosyncratic performance characteristics: the OS overhead of descriptor management can frequently outweigh the gain from the hardware [11]. Buffer and flow management and interface complexity reduce the utility of intelligent NICs.

Of course, hardware features rarely improve performance in all cases and must be used judiciously. One can even build models to predict the value of protocol offload for different applications [40]. Also, sometimes, NICs just get it wrong.

However, this is no reason to ignore hardware which can deliver benefits in many cases. OS designers should critique such hardware more effectively and encourage designers to listen to their concerns [28]. This will not happen while OS maintainers ignore the hardware and leave the market to a small user group with a narrow set of requirements such as high frequency trading [23] and high performance computing [47].

The hardware lacks functionality: Modern OS protocol stacks are feature-rich, with sophisticated filtering and scheduling, and uniform configuration interfaces. In contrast, NICs which try to assume such functions never support the full capabilities of the OS stack, have hardware-specific resource limits (e.g. on flows), can suffer remote resource exhaustion attacks, and often require special tools for configuration.

This is true, but these are research challenges, not reasons to avoid modern hardware. We propose here one framework which aims to work around limitations, and move processing between hardware and software based on requirements, not ease of design: some users may care more about packet scheduling than filters, others vice versa, for example. The problem should be one of *placement* of functionality for a given workload, which we address explicitly in Section 4.

Integration is a nightmare: Supporting a complex piece of NIC hardware in a modern OS is an engineering effort requiring "massive, heavily invasive hooks into the network stack" [44]. Worse, having some flows or state handled by hardware and some by the OS eliminates the global system view otherwise enjoyed by the kernel, making it hard to manage resources.

We turn this argument around and claim it shows a basic flaw in current protocol stacks. An OS which manifestly fails to exploit the hardware to deliver good performance in a range of scenarios is simply a poorly-designed OS. The discussion also generalizes beyond Linux and TOE to most other NIC features and OSes.

For example, virtualization support on NICs [8] could deliver benefits to a single OS via reduced CPU usage and performance isolation, as with other virtualization support [3].

In response to the lack of core OS support and the difficulty of integration, NIC vendors have three choices. First, they can *encode policy in the driver* and hide it from the OS - for example, Intel 10GbE Linux drivers monitor TCP connections and try to dedicate a queue to a connection after 20 packets have been sent. As we have seen, this is sometimes good and sometimes bad, in that the OS has no control over this policy and cannot connect it with other resource allocation decisions.

Second, they can expose the functionality to specially-written applications using a *non-standard control interface* [15] or, in some cases, a new, separate network stack [41]. This works for niches (Finance, HPC), but is no long-term solution. Vendors realize this, but it can be the best option to gain revenue today.

Third, they can *hide complex functionality and policy in hardware* behind a compatibility interface. This is perhaps the worst possible option [28], since it prevents a more adventurous OS design from exploiting or managing the hardware effectively at all.

The objections above should be research challenges, and OS researchers should step up to the plate and provide a solution. We should describe NIC capabilities and limitations to the OS and equip it with an infrastructure to map application requirements and system-wide policies to the available hardware. We argue this can give better performance for applications, and a freer environment for NIC designers to innovate.

4 Dragonet: talking about NICs

We are designing *Dragonet*, a new network protocol stack initially targeting the Barrelfish research OS [2]. Our goals are to, first, *integrate new hardware features* while minimizing driver bloat; second, *regain a global view* of system resources and give the OS control of allocation policy; third, *manage user expectations* by switching between hardware and software implementation as requirements change; and finally *deliver understandable performance*: the models used by the OS to manage resources should explain how the performance of the stack changes.

Dragonet is inspired by much previous work, such as the use of dataflow graphs to represent protocol stack state [13,20,24], and systems which represent and schedule flows as first-class entities in the system [6, 29]. The key novel feature of Dragnonet is how we propose to exploit specialized hardware using graph embedding.

Physical Resource Graphs: Hardware features in Dragonet are described explicitly by a *Physical Resource Graph* (PRG). This captures the NIC capabilities and can be viewed as a dataflow program which the hardware executes when sending or receiving a packet. Currently we write PRGs using a Haskell-based domain-specific language; Figure 2 shows a simplified PRG (generated automatically from the specification) for the receive path of an Intel 82576 1GbE NIC [16], a relatively simple NIC which nonetheless supports RSS and checksum offload. The PRG for the NIC 82599 used in Section 2 is much more complex, and we omit it for space reasons.

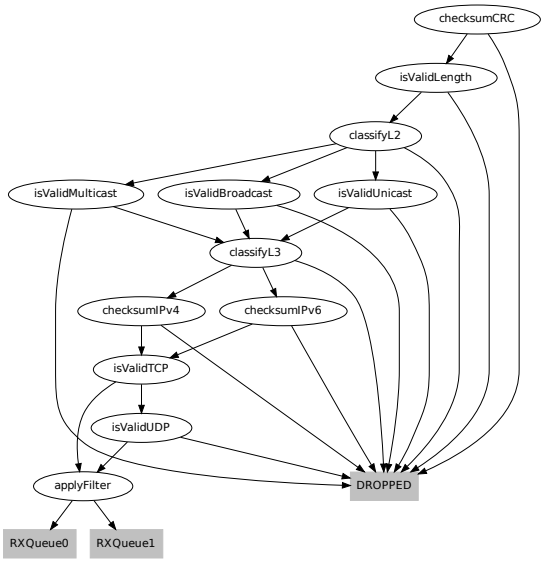

Figure 2: Intel 82576 PRG for the receive path

PRGs are a critical feature of Dragonet: they allow the OS to reason about the diverse capabilities of NIC hardware without a priori knowledge. The PRG is part of the OS-driver interface, putting resource policy back in the core OS (where it belongs), while hardware configuration remains in the driver (where it should be). PRG arcs and nodes can be annotated with processing costs such as latency to guide OS policy decisions.

The Logical Protocol Graph: Like the x-kernel [13], Dragonet views the state of the OS stack as a graph of protocol operations, the *Logical Protocol Graph* (LPG). Figure 3 shows the LPG for the receive path of the experiment in Section 2.

The LPG captures the global OS network state at a high level, and can be viewed as a dataflow program executed whenever a packet enters the system (from either an application or the network). For correctness, the OS must instantiate the function of each LPG node either in hardware or software. Data must be passed appropriately between these nodes. The LPG changes as connections (or connectionless endpoints) come and go, and so in-

stantiation must be incremental.

Note that a graph representation in the OS does not imply strictly structuring packet handling *code* using an operator for each node, as in the x-kernel [13], Click [20], or P2 [24]. Starting from an LPG, many software options are possible, including a layered structure similar to existing stacks in Linux, Windows, etc. However systems like Melange [25] have shown that incorporating semantic information into the network stack can result in more efficient implementations. This is in part due to the possibility of global optimizations of memory management.

The LPG has two further important properties. First, arc annotations can express properties of flows similar to PRG, such as latency guarantees or priorities. Our first non-trivial annotations will use Network Calculus [22] to specify connection requirements.

Second, we can perform semantics-preserving transformations on the LPG, much as the relational algebra specifies how query plans can be transformed without changing query semantics. In a software-only implementation, this allows us to use cost-based optimization to lay out optimal protocol stacks, for example to drop incoming packets early. It can also be extended to encapsulate the cost of traversing inter-core links, and thereby aid in placing functionality in a NUMA machine, for example.

In practice, we find it is easy to express simple rewrite rules on LPGs, but a complete semantic treatment is highly complex, and an interesting area of research.

Our design allows us to fully integrate general purpose cores embedded on NICs into Dragonet. Consequently, we can schedule software-provided module implementations on such cores to minimize communication cost.

PRG embedding: At its simplest, exploiting a NIC's hardware becomes a matter of embedding as much of the PRG as possible into a valid transformation of the LPG, and then instantiating the rest of the LPG in software. Figure 3 shows an example LPG where nodes which can be mapped to hardware capabilities are shaded. Abstract graph embedding is NP-complete, but in our case there are a limited number of options (for example, the network itself is a fixed node in both graphs) and early experiments suggest it is highly tractable.

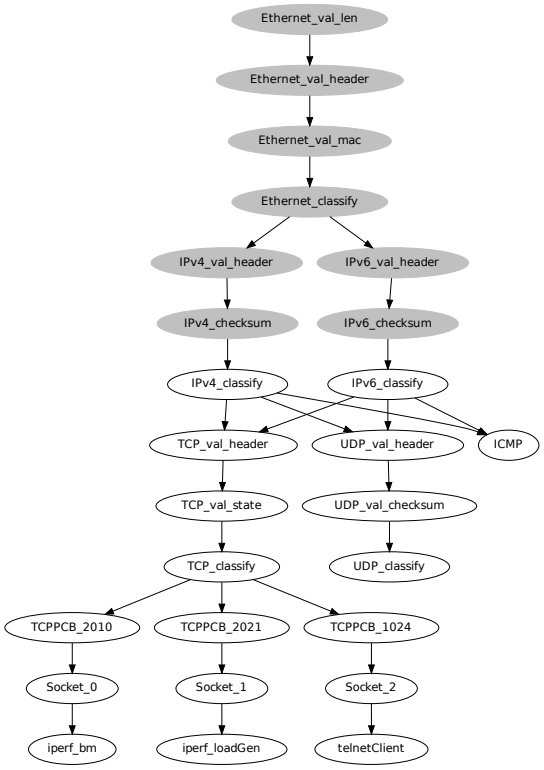

Figure 3: Example for a LPG. The gray boxes represent modules mapped to hardware provided features.

5 Conclusions and Challenges

The Dragonet approach overcomes the limitation of fixed boundaries between hardware and OS, and between application and OS. By making these boundaries flexible, applications can use the hardware better whenever it is available, and the OS has precisely the global view of system state which is denied to Linux (and other OSes) with current designs.

Naturally, there are many unsolved challenges. To talk about NICs usefully, the PRG also requires a performance model of the hardware and a way to calculate the performance characteristics of a path through the graph. NICs have hardware limits (queues, flow table entries, filter rules, etc.), and Dragonet has to deal with intelligently spilling these to software. Some NICs can also be configured in different ways, complicating the corresponding PRGs. Also, NIC semantics and the LPG don't always match: for example, many perform only partial flow classification using hashing. Finally, some hardware doesn't fit our current core/fixed function dichotomy, for example micro-engines [14].

Nevertheless, Dragonet explores one way to build a stack which assumes and embraces a wide variety of complex NIC features, rather than over-abstracting them away. We claim this is an important problem: if OS designers do not fix their designs, we face the uncertain prospect of NIC designers burying their clever ideas in hardware without any OS policy control at all.

Acknowledgments
We thank Gerd Zellweger and Kornilios Kourtis for their contributions and helpful suggestions.

References

[1] Ang, B. An evaluation of an attempt at offloading TCP/IP protocol processing onto an i960RN-based iNIC. *Computer Systems and Technology Laboratory, HP Laboratories* (2001).

[2] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., and Singhania, A. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)* (Big Sky, Montana, USA, 2009).

[3] Belay, A., Bittau, A., Mashtizadeh, A., Terei, D., Mazières, D., and Kozyrakis, C. Dune: safe user-level access to privileged CPU features. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI)* (Hollywood, CA, USA, 2012).

[4] Bruijn, W. D., Slowinska, A., Reeuwijk, K. V., Hruby, T., Xu, L., and Bos, H. Safecard: a gigabit IPS on the network card. In *Proceedings of 9th International Symposium on Recent Advances in Intrusion Detection* (2006).

[5] Burnside, M., and Keromytis, A. Accelerating application-level security protocols. In *The 11th IEEE International Conference on Networks (ICON)* (2003), IEEE, pp. 313–318.

[6] Burnside, M., and Keromytis, A. D. High-speed I/O: the operating system as a signalling mechanism. In *Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications (NICELI)* (Karlsruhe, Germany, 2003).

[7] Dobrescu, M., Egi, N., Argyraki, K., Chun, B., Fall, K., Iannaccone, G., Knies, A., Manesh, M., and Ratnasamy, S. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)* (Big Sky, Montana, USA, 2009), vol. 9.

[8] Dong, Y., Yang, X., Li, X., Li, J., Tian, K., and Guan, H. High performance network virtualization with SR-IOV. In *The 16th International Symposium on High-Performance Computer Architecture* (Bangalore, India, January 2010).

[9] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., and Burger, D. Dark silicon and the end of multicore scaling. *IEEE Micro 32*, 3 (May 2012).

[10] Feng, W., Balaji, P., Baron, C., Bhuyan, L., and Panda, D. Performance characterization of a 10-gigabit ethernet TOE. In *Proceedings of 13th IEEE Symposium on High Performance Interconnects* (2005).

[11] Frey, P. W., and Alonso, G. Minimizing the hidden cost of RDMA. In *29th IEEE International Conference on Distributed Computing Systems* (2009).

[12] Huggahalli, R., Iyer, R., and Tetrick, S. Direct cache access for high bandwidth network I/O. In *Proceedings of the 32nd Annual IEEE International Symposium on Computer Architecture (ISCA)* (2005).

[13] Hutchinson, N. C., and Peterson, L. L. The X-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering 17*, 1 (January 1991).

[14] Intel Corporation. *Intel IXP2400/IXP2800 Network Processor Programmer's Reference Manual*, November 2003.

[15] Intel Corporation. *Assigning Interrupts to Processor Cores using an Intel 82575/82576 or 82598/82599 Ethernet Controller*, September 2009. Application Note.

[16] Intel Corporation. *Intel 82576 Gigabit Ethernet Controller Datasheet*, December 2010. Revision 2.61.

[17] Intel Corporation. *Intel 82599 10 GbE Controller Datasheet*, December 2010. Revision 2.6.

[18] Jang, H., Chung, S.-H., Kim, D. K., and Lee, Y.-S. An efficient architecture for a TCP offload engine based on hardware/software co-design. *Journal of Information Science and Engineering 509* (2011).

[19] Kim, H., and Rixner, S. TCP offload through connection handoff. *ACM SIGOPS Operating Systems Review 40*, 4 (2006).

[20] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. *ACM Transactions on Computer Systems 18*, 3 (August 2000).

[21] Kumar, A., Huggahalli, R., and Makineni, S. Characterization of direct cache access on multicore systems and 10GbE. In *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture* (2009).

[22] Le Boudec, J.-Y., and Thiran, P. *Network Calculus*, vol. 2050 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

[23] Lockwood, J., Gupte, A., Mehta, N., Blott, M., English, T., and Vissers, K. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *Proceedings of the 20th Annual Symposium on High-Performance Interconnects (HOTI)* (2012).

[24] Loo, B. T., Condie, T., Hellerstein, J. M., Maniatis, P., Roscoe, T., and Stoica, I. Implementing declarative overlays. In *Proceedings of the 20th ACM symposium on Operating Systems Principles (SOSP)* (Brighton, United Kingdom, 2005).

[25] Madhavapeddy, A., Ho, A., Deegan, T., Scott, D., and Sohan, R. Melange: creating a "functional" internet. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (Lisbon, Portugal, 2007).

[26] Microsoft Corporation. Information about the TCP chimney offload, receive side scaling, and network direct memory access features in windows server 2008. Microsoft Knowledge Base article 951037, http://support.microsoft.com/kb/951037, revision 7.0, February 2011.

[27] Mogul, J. C. TCP offload is a dumb idea whose time has come. In *Proceedings of the 9th International Workshop on Hot Topics in Operating Systems (HotOS)* (Lihue, Hawaii, 2003).

[28] Mogul, J. C., Baumann, A., Roscoe, T., and Soares, L. Mind the gap: reconnecting architecture and OS research. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems* (Napa, California, 2011).

[29] Mosberger, D., and Peterson, L. L. Making paths explicit in the scout operating system. In *Proceedings of the 2nd USENIX symposium on Operating systems design and implementation* (1996).

[30] Netronome Systems, Inc. *Netronome Flow Driver: Programmer's Reference Manual*, 2008. v.1.3.

[31] Network Working Group. RFC 3720: Internet Small Computer Systems Interface (iSCSI). http://tools.ietf.org/html/rfc3720.

[32] Nightingale, E. B., Hodson, O., McIlroy, R., Hawblitzel, C., and Hunt, G. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)* (Big Sky, Montana, USA, 2009).

[33] PCI-SIG. IO virtualization. http://www.pcisig.com/specifications/iov/.

[34] Pesterev, A., Strauss, J., Zeldovich, N., and Morris, R. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM European Conference on Computer Systems (Eurosys)* (2012).

[35] Raj, H., and Schwan, K. High performance and scalable I/O virtualization via self-virtualized devices. In *Proceedings of the 16th ACM International Symposium on High Performance Distributed Computing (HPDC)* (Monterey, California, USA, 2007).

[36] RDMA Consortium. Architectural specifications for RDMA over TCP/IP. http://www.rdmaconsortium.org/.

[37] Realtek Semi-Conductor Co., Ltd. *RTL8029AS: Realtek PCI Full-Duplex Ethernet Controller with built-in SRAM*, January 1997.

[38] Rossbach, C. J., Currey, J., Silberstein, M., Ray, B., and Witchel, E. Ptask: operating system abstractions to manage gpus as compute devices. In *Proceedings of the 23rd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)* (Cascais, Portugal, 2011), pp. 233–248.

[39] Santos, J., Turner, Y., Janakiraman, G., and Pratt, I. Bridging the gap between software and hardware techniques for I/O virtualization. In *Proceedings of the USENIX'08 Annual Technical Conference* (2008).

[40] Shivam, P., and Chase, J. S. On the elusive benefits of protocol offload. In *Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications (NICELI)* (Karlsruhe, Germany, 2003).

[41] Solarflare Communications, Inc. *Onload User Guide*. 9501 Jeronimo Road, Irvine, California 92618, 2010. Version 20101221.

[42] Solarflare Communications, Inc. *Solarflare SFN5122F Dual-Port 10GbE Enterprise Server Adapter*, 2010.

[43] Solarflare Communications, Inc. SFA6900 ApplicationOnload Engine Overview. http://www.solarflare.com/Content/UserFiles/Documents/Solarflare_AOE_Overview.pdf, 2012.

[44] The Linux Foundation. toe. http://www.linuxfoundation.org/collaborate/workgroups/

networking/toe, November 2009. Retrieved January 2013.

[45] University of Illinois. iperf version 2.0.5. http://sourceforge.net/projects/iperf/, 2010.

[46] Wang, W., Wang, J., and Li, J. Study on enhanced strategies for TCP/IP offload engines. In *Proceedings of 11th IEEE International Conference on Parallel and Distributed Systems* (2005), vol. 1.

[47] Woodall, T. S., Shipman, G. M., Bosilca, G., Graham, R. L., and Maccabe, A. B. *High Performance RDMA Protocols in HPC*. 2006.