

Performance Analysis of System Overheads in TCP/IP Workloads

Nathan L. Binkert, Lisa R. Hsu, Ali G. Saidi
Ronald G. Dreslinski, Andrew L. Schultz†, Steven K. Reinhardt

Advanced Computer Architecture Lab
EECS Department, University of Michigan
1301 Beal Ave, Ann Arbor, MI 48109-2122

{binkertn, hsul, saidi, rdreslin, alschult, stever}@eecs.umich.edu

Abstract

Current high-performance computer systems are unable to saturate the latest available high-bandwidth networks such as 10 Gigabit Ethernet. A key obstacle in achieving 10 gigabits per second is the high overhead of communication between the CPU and network interface controller (NIC), which typically resides on a standard I/O bus with high access latency. Using several network-intensive benchmarks, we investigate the impact of this overhead by analyzing the performance of hypothetical systems in which the NIC is more closely coupled to the CPU, including integration on the CPU die. We find that systems with high-latency NICs spend a significant amount of time in the device driver. NIC integration can substantially reduce this overhead, providing significant throughput benefits when other CPU processing is not a bottleneck. NIC integration also enables cache placement of DMA data. This feature has tremendous benefits when payloads are touched quickly, but potentially can harm performance in other situations due to cache pollution.

1. Introduction

Networking bandwidth is one of the few technologies that has outstripped Moore's Law in recent years. From 1995 to 2002, the IEEE Ethernet standard evolved from a top speed of 100 Mbps to 10 Gbps (10GigE), a hundred-fold improvement, while in the same period the 18-month doubling rate of Moore's Law indicates a mere 25x increase in transistor density (traditionally correlated with CPU performance). As a result, the host computer systems at the endpoints of these high-speed Ethernet connections are no longer able to keep up with the network data rate.

There are already a number of bandwidth-hungry applications that can readily make use of 10 Gbps networking, such as cluster and grid computing systems and file servers on local area networks. (Most current PC systems ship with 1 Gbps Ethernet adapters; only ten of these are needed to saturate a 10GigE server system.) Many emerging technologies such as network-attached storage (e.g., iSCSI), telepresence-based remote conferencing, and video streaming will further increase demand for very high-speed network connections. For industry to deliver production computer systems capable of meeting these demands as they materialize, system designers must begin addressing this problem right away.

The primary advances in end-host networking performance in the past several years have involved minor enhancements to the network interface controller (NIC), allowing it to offload simple tasks from the CPU such as checksum generation and segmentation of large transfers. Some portions of industry are pushing this direction further by offloading much of the TCP protocol stack to the NIC. However, networking performance is a system-wide issue, involving the I/O subsystem, memory hierarchy, and CPUs. Attempting to address this system-wide problem with a point solution, as these TCP offload engines (TOEs) do, involves fundamental drawbacks, as will be discussed in Section 2.

In this paper, we take a system-level view of high-bandwidth TCP/IP network performance. Using simulation, we analyze the performance of current and hypothetical future systems under network-intensive micro- and macro-benchmark workloads. We find that the key bottleneck in current systems is not protocol processing itself, but the high latency of communication between the CPU and the NIC. From a system-level perspective, this latency can be solved directly by moving the NIC closer to the CPU, either by placing it

†Currently at the University of California, Berkeley.

on a physically closer interconnect or by integrating directly on die with the CPU.

Once the CPU/NIC communication bottleneck is alleviated, protocol processing and memory copying become performance limiters. Although a dedicated TCP processing engine could help here, we find that moderate increases in CPU bandwidth, as will be available in near-future CMP platforms, is adequate to saturate a 10GigE link in several of our benchmarks.

Where this CPU speed increase is not adequate, it is often due to the CPU stalling while copying data. Given an on-chip NIC that has access to the last level of on-chip cache, we can address copying overheads by placing incoming network data directly in the cache. This technique can also be used with off-chip NICs if the CPU cache allows data to be pushed in asynchronously from an external source [12]. Although we occasionally see higher miss rates due to cache pollution from this technique, we do not observe any negative performance effects in our benchmarks. In general, larger caches and/or faster CPUs increase the likelihood that the processor will touch network data before it is kicked out of the cache. In contrast, smaller caches and slower CPUs are less likely to do so, and thus suffer from pollution effects.

Overall, we find that a handful of system-level architecture changes (closer CPU/NIC coupling, increased CPU bandwidth, and NIC cache data placement) are sufficient to saturate a 10 Gbps Ethernet in all but one of our benchmarks. We feel that these changes represent a far more promising and less radical path to achieving high-bandwidth networking than the current industry direction of TCP offload. We identify more intelligent policies for cache network data placement and more efficient NIC event notification (to reduce interrupt overhead) as priorities for future work in this area.

The remainder of the paper begins with a discussion of related work. Options for NIC placement are investigated in Section 3. We describe our simulation environment in Section 4 and our benchmarks in Section 5. Section 6 presents simulation results. Section 7 covers our conclusions and future work.

2. Related Work

As mentioned in the introduction, much of the effort in enhancing commercial NICs has focused on offloading work from the CPU to the NIC. Minor tasks, such as checksum generation and transmit segmentation are common in current gigabit Ethernet NICs. TCP offload engines (TOEs) take the offload concept to its logical conclusion, moving all TCP/IP processing onto

the NIC [1]. However, there are a number of arguments against the TOE approach, such as the inability of a TOE to stay on the general-purpose CPU performance curve and the difficulty in dealing with bugs in firmware-based TCP code [17]. Offload techniques address the high cost of CPU/NIC interactions by having the CPU interact with the NIC less frequently at a higher semantic level. We take an alternate approach by directly attacking the NIC/CPU communication overhead.

Intel claims that TOEs by themselves do not provide significant performance improvements because they do not address the overhead of data movement between the NIC and main memory [8,21]. As a result, they have recently begun a platform-level attack on I/O performance with “I/O Acceleration Technology” (I/OAT) [8,14]. I/OAT includes header splitting, a hardware copy engine, interrupt moderation, and protocol stack optimizations. Intel researchers have also proposed further system-level enhancements. TCP onloading [21] dedicates a CPU to NIC interaction and protocol processing. The end result is that other CPUs are able to communicate with the dedicated CPU at a lower interaction cost than communicating to the NIC directly. There is a similarity between this and TOE, but because this is strictly a kernel optimization, the protocol processing is still done by the kernel instead of firmware, and major changes to the protocol stack are unnecessary. Direct cache access (DCA) [12] investigates modifying the cache coherence protocol to allow an I/O device to push data into the CPU’s cache. Our approach of placing the NIC on the CPU die directly enables cache placement of NIC data, and we also find substantial benefits from this optimization.

Closer integration of network interfaces with CPUs has been a prominent theme of work in the area of fine-grain massively parallel processors (MPPs). Henry and Joerg [9] investigated a range of placement options, including on- and off-chip memory-mapped NICs and a NIC mapped into the CPU’s register file. Other research machines with tight CPU/network integration include *T [20] and the J-Machine [6]. Mukherjee and Hill [18] propose several optimizations for NICs located on the memory bus that can participate in cache-coherent memory transactions. Many of their optimizations could be used with our integrated NICs as well. We see our future work, in part, as an attempt to apply some of the ideas from this custom MPP domain to the more commercially significant area of TCP/IP networking. Now that TCP/IP over off-the-shelf 10GigE can provide bandwidth and latency competitive with, and often better than, special-purpose high-speed interconnects [7], a single CPU device with

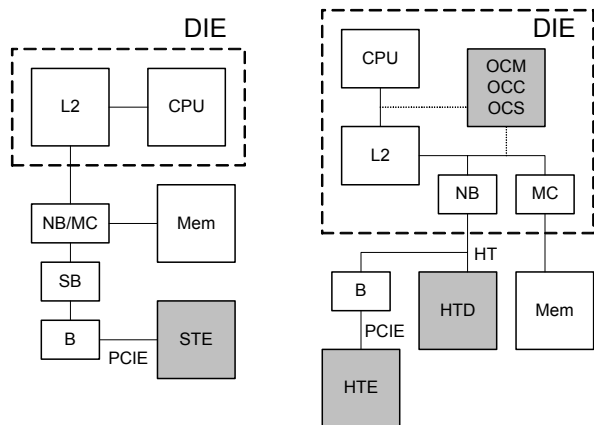


Figure 1. NIC placement options.

Shaded boxes represent different NIC locations

NB: North Bridge, SB: South Bridge, B: I/O Bridge, MC: Mem Controller

integrated 10GigE support could serve as both a data-center server and a node in a high-performance clustered MPP.

Integrated NIC/CPU chips targeted at the embedded network appliance market are available (e.g., [3]); our work differs in its focus on integrating the NIC on a general-purpose end host, and on performance rather than cost effectiveness. Reports indicate that the upcoming Sun Niagara processor, a multithreaded chip multiprocessor designed for network workloads and scheduled to ship in 2005, will have multiple integrated 1 Gbps Ethernet interfaces.

3. NIC Placement Options

To investigate the impact of changing the location of the NIC in the memory hierarchy, we chose a set of five configurations as shown in Figure 1. The first two configurations we model are aggressive I/O designs that we expect to be common in the near future. The first system, standard PCI Express (STE), has an off-chip memory controller and a dedicated PCI Express x4 channel for the 10GigE NIC hanging off an I/O bridge. The HyperTransport PCI Express (HTE) configuration represents a similar system with an on-chip memory controller, but with one fewer chips separating the NIC from the CPU.

The third configuration we model, HyperTransport direct (HTD), represents a potential design for systems that implement a high-speed I/O interconnect via HyperTransport-like channels. This configuration is similar to attaching the NIC directly to a 6.4GB/s HyperTransport channel.

The remaining configurations integrate the NIC onto the CPU die. We look at three on-chip configura-

tions: on-chip memory-bus-attached (OCM), on-chip cache-attached (OCC), and on-chip split (OCS). OCM attaches the NIC to the memory bus, on the other side of the L2 cache from the CPU. This configuration provides very high bandwidth similar to HTD but even lower latency due to the elimination of a chip crossing. OCC goes one step further and attaches the NIC to the bus between the L1 and L2 caches. This configuration reduces latency even further, but more importantly allows incoming NIC DMA data to be written directly into the L2 cache. OCS is a hybrid of the two configurations, where the NIC is logically attached to both the cache bus and the memory bus. In this setup, the NIC splits header from payload data (actually the first cache block from the remainder of the packet) and places the former in the cache and the latter in memory. The goal is to provide low-latency access to the header control information, which is likely to be processed quickly by the kernel, without polluting the cache with large data transfers that may not be referenced as quickly.

We believe the area and complexity required for this integration are modest. The only required on-chip components are those which enqueue and dequeue packet data into and out of the network FIFOs and the I/O bus interface. The physical layer signalling logic, or even the bulk of the FIFO buffer space could remain off chip. Signalling between the on-chip and off-chip portions of the NIC could use dedicated pins or could be multiplexed over an I/O channel (e.g., HyperTransport). This setup differs from the HyperTransport-attached NIC in that CPU/NIC interactions are on-chip, and only latency-tolerant transfers between on-chip and off-chip network FIFOs go across the HyperTransport link.

4. Simulator Platform and Configuration

Alternative NIC architectures are often evaluated by emulation on a programmable NIC or by hardware prototyping. While these approaches allow modeling of different NICs in a fixed system architecture, unfortunately they do not lend themselves to modeling a range of system architectures as we have described in the previous section. We thus turn to simulation for our investigation.

Because the bulk of network processing activity occurs in the OS kernel, network-oriented system architecture research requires a full-system simulator capable of running both OS and application code. Additionally, the simulator must have a reasonably detailed timing model of the memory and I/O systems and of the network device. The following sections (4.1-4.3) describe these aspects of our simulator platform in

turn. Section 4.4 discusses the system parameters we use in this paper.

4.1 The M5 Simulator

Network-intensive applications spend the majority of their time in OS kernel code—including driver code to communicate with the NIC, the network stack, and the code to copy the packets to/from user processes—rather than user-level application code. Conventional architectural simulators, which execute only user-level application code and functionally emulate kernel behavior, do not provide meaningful results for these workloads. Of the handful of existing full-system simulators [16,22,24], none provide detailed network I/O modeling nor are easily extendable to do so.

As a result, we developed our own simulator, called M5, to meet our specific needs [2]. M5's Alpha ISA CPU timing model has roots in the SimpleScalar [4] simulator but has been largely rewritten. We added numerous full-system capabilities including privileged instructions, virtual/physical address translation, processor-specific control registers, and a Compaq Alpha Tsunami chipset model. SimOS [22] provided a valuable reference implementation and was the source of the Alpha PAL code that M5 executes. We model the Tsunami platform with enough fidelity to boot unmodified Linux 2.4 and 2.6 kernels. We used Linux 2.6.8.1 to generate the results for this paper.

We enhanced our detailed CPU timing model to capture the primary timing impact of system-level interactions. M5 executes actual Alpha PAL code in privileged mode to handle traps and interrupts, flushing the pipeline where appropriate. For memory barrier instructions, we drain the pipeline and stall until outstanding accesses have completed. Write barriers prevent reordering in the store buffer. Uncached memory accesses (e.g. for programmed I/O) are performed only when the instruction reaches the commit stage and is known to be on the correct path.

To provide deterministic, repeatable simulation of network workloads, as well as accurate simulation of network protocol behavior, M5 models multiple systems and the network interconnecting them in a single process. Implementing this capability was simplified by the object-oriented design of M5—creating another system requires simply instantiating another set of objects modeling another CPU, memory, disk, etc.

We have validated M5 against two Compaq Alpha XP1000s (w/500MHz and 667MHz CPUs). Five of six benchmark/system combinations have simulated network bandwidth within 15% of the real system. CPU utilization breakdowns (user, driver, stack, etc.) also

correlate well. This level of accuracy is reasonable given that we did not try to model that specific platform precisely, but merely tuned our generic cpu, cache, bus, and memory parameters to match. More information on the validation of M5 is available [23].

4.2 Memory and I/O System Model

The memory and I/O systems are key determinants of networking performance. We use a handful of simple components to construct system models representing those of Section 3.

To emulate all of the interconnects in the system, we use a single bus model of configurable width and clock speed. This model provides a split-transaction protocol, and supports bus snooping for coherence. Our DRAM, NIC, and disk controller models incorporate a single slave interface to this bus model, capable of variable transaction sizes up to 64 bytes. The cache model includes a slave interface on the side closer to the CPU and a master interface on the side further from the CPU. Note that this model is optimistic for bidirectional point-to-point interconnects such as PCI Express and HyperTransport, as it assumes that the full bidirectional bandwidth can be exploited instantaneously in either direction.

We also have a bus bridge model that interfaces two busses of potentially different bandwidths, forwarding transactions in both directions using store-and-forward timing. This model is used for the I/O bridges (labeled B in Figure 1) and for the memory controller. In our model the memory controller simply bridges between two busses (for example, the front-side bus and the controller/DRAM bus). The DRAM timing is modeled in the DRAM object itself.

4.3 Ethernet Device Model

Our simulated Ethernet NIC is modeled after the National Semiconductor DP83820 [19] Gigabit Ethernet device. The model is sufficiently faithful to support the off-the-shelf Linux DP8820 device driver. We fix a bug in the real DP83820 that prevents its DMA engine from writing to arbitrarily aligned blocks. This common feature allows the kernel to align buffers based on packet payloads rather than headers. The model focuses its detail on the main packet data path and the system-side I/O interface. Three logical blocks comprise the device model: the device itself, the physical interface, and the link.

The device portion of the model manages device programming registers, DMA to and from device buff-

Table 1. Simulated System Parameters

Frequency	4 GHz, 6 GHz, 8 GHz, or 10 GHz
Fetch Bandwidth	Up to 4 instructions per cycle
Branch Predictor	Hybrid local/global (ala 21264).
Instruction Queue	Unified int/fp, 64 entries
Reorder Buffer	128 Entries
Execution BW	4 insts per cycle
L1 Icache/Dcache	128KB, 2-way set assoc., 64B blocks, 16 MSHRs Inst: 1 cycle hit latency Data: 3 cycle hit latency
L2 Unified Cache	4MB and 16 MB, 8-way set assoc. 64B block size, 25 cycle latency, 40 MSHRs
L1 to L2	64 bytes per CPU cycle
L2 to Mem Ctrlr	4 bytes per CPU cycle
HyperTransport	8 bytes, 800 MHz
Main Memory	65 ns latency for off-chip controller, 50 ns on-chip

ers, interrupts, and the assembling and buffering of packet data. The device model fully participates in the memory system timing, interfacing to the bus model described in Section 4.2 for both DMA transactions and programmed I/O requests to device control registers. DMA transactions are fully coherent with the cache hierarchy. The device also implements interrupt moderation (aka interrupt coalescing) to reduce the interrupt rate. At high bandwidths, interrupting the CPU on each packet is impractical. We use a fixed-delay scheme which puts an upper bound on the rate of common device interrupts at the cost of some additional latency under light loads. We experimented with different rates and found that constraining the interval between interrupts to a minimum of 10 μ s was reasonably effective.

The physical interface model moves data from the transmit buffer to the link or passes data from the link to the receive buffer. Since there is no buffering in the physical interface and it represents negligible delay, its timing is not modeled.

The Ethernet link models a lossless, full-duplex link of configurable bandwidth. The latency of a packet traversing the link is simply determined by dividing the packet size by that bandwidth. Since we are essentially modeling a simple wire, only one packet is allowed to be transmitted in each direction at any given time.

4.4 Memory Latencies

The parameters we used in modeling the configurations of Section 3 are listed in Table 1. We are primarily interested in the relative behavior of these systems, rather than their absolute performance, so some of these parameters are approximations. In addition to the

Figure 2. Uncached access latencies latency \pm standard deviation (all in ns)

Location	Alpha DP264	Pentium III 933MHz	Pentium 4 3GHz	M5 Simulator
Peripheral	788 \pm 40	835 \pm 54	803 \pm 24	773 \pm 8.6
Off NB	—	423 \pm 49	392 \pm 21	381 \pm 7.2
On NB	475 \pm 26	413 \pm 61	216 \pm 16	190 \pm 2.0
On Die	—	132 \pm 52	82 \pm 3	30 \pm 2.9

parameters shown, we also add a bridge-dependent latency penalty for each bus bridge that connects devices on separate chips. These bridge latencies were tuned based on measurements taken from real systems using hardware performance counters and a custom Linux kernel module.

Table 2 presents timings for devices on three real machines in six different configurations, reflecting the different NIC locations that we studied. The “peripheral” timing corresponds to our PCIE configuration, where the device is on a commodity I/O bus with multiple bus bridges between the device and the CPU. The “off NB” (north bridge) location is similar to our HTE configuration, where a standard I/O bus is connected to the NB directly. The HTD configuration could be realized by integrating the NIC onto the NB (“on NB”). It is interesting to note that the peripheral and off-NB latencies in Table 2 have not varied by more than 10% over several years. Even the on-NB latency is only reduced by approximately 2x for more than a 3x change in CPU frequency.

OCM timing is approximated by measuring the latency to a device integrated on the CPU die. The Alpha EV6 has no such devices, but both Pentium chips have an integrated local I/O APIC. These devices have an access time of 3x-4x more than the on-chip access time we modeled. However, we believe these devices were not designed to minimize latency, and that an integrated NIC could be designed with an access time closer to that of a cache.

We measured the memory access latencies for our configurations with an on-chip and off-chip memory controller. With our memory configurations an on-chip memory controller has an access latency of 50ns and an off-chip memory controller has a latency of 65ns. In both cases our results are similar to published numbers of 50.9ns and 72.6ns respectively [13].

5. Benchmarks

For this evaluation, we used several standard benchmarks: netperf [10], a modified SPEC WEB99 [25], and NFS with a Bonnie++ [5] stressor. All bench-

marks were simulated in a client-server configuration. The system under test was modeled with detailed CPU timing, memory timing, and peripheral device interaction. The other system(s) stressing the system under test were operated artificially quickly (functionally modeled with a perfect memory system) so as not to become the bottleneck. In addition to the simple client-server setup, we ran the netperf benchmark in a network address translation (NAT) configuration with the client system accessing the server through a NAT gateway, which was the system under test.

Netperf is a simple network throughput and latency microbenchmark suite developed at Hewlett-Packard. We focus on two of the many microbenchmarks that are included in netperf: stream, a transmit benchmark, and maerts, a receive benchmark. In both of these, the netperf client opens a connection to the machine running the netperf server and sends or receives data at the highest rate possible. The benchmark itself has a very short inner loop, basically filling up a buffer and calling the write syscall. This benchmark consequently has very little user time, and spends most of its time in the TCP/IP protocol stack of the kernel or in the idle loop waiting for DMA transactions to complete.

SPECWEB99 is a well-known webserver benchmark that stresses the performance of a server machine. Simulated clients generate web requests to form a realistic workload for the server, consisting of static and dynamic GET and POST operations over multiple HTTP 1.1 connections. Specweb also contains dynamic ad rotation using cookies and table lookups. The original benchmark is intended to be tuned to determine the maximum number of simultaneous connections a server is able to handle. However, iterative tuning is impractical for our purposes due to the relative slowdown of our simulated environment. We created our own benchmark client that generates Specweb client requests using the same statistical distribution as the original clients, but without the throttling. Our version thus continuously generates packets until the link bandwidth is saturated. We use the Apache http server, version 2.0.52, with a maximum of 100,000 clients and 50 threads per child.

NFS is a ubiquitous network file system from Sun that enables users to see a distributed file system network locally on their individual machines. Thus, a simple copy of one file to another location incurs network traffic. To remove the disk subsystem from this benchmark, the system under test (server) was run with a RAM disk that it exported over NFS while the client ran Bonnie++, a simple benchmark for testing hard

drive and file system performance. The Bonnie++ tests we utilized consist of a series of large block writes.

For netperf over NAT, we simply placed another machine between the server and client to act as a NAT machine. The NAT machine translates the IP addresses on the client's requests from a private network addresses to its own address, then reverses the translation on responses. When conducting these experiments, the system under test was the NAT machine.

For all of these experiments, we use the standard 1500 byte maximum transmission unit (MTU) as that is standard on the Internet today. While many networking papers vary the MTU in their experiments, 1500 bytes is by far the dominant MTU in modern networks and systems.

The time required for full-system cycle-level simulation is immense and, in general, is orders of magnitude slower than a real system's performance. Thus running any benchmark to completion is impractical. To address this we used a combination of functional fast-forwarding, warm-up and sampling to obtain the results herein.

We modified our benchmarks to insert special M5-defined instructions that cause the simulator to checkpoint its state. We use these instructions to take checkpoints only after the workload enters its steady state.

From a checkpoint we warm-up the state in the caches and TLB for 1 billion cycles using a simple CPU model. After the warm-up is complete we switch to a detailed cycle-level CPU model and run for another 200 million cycles. The warm-up period was of lower effective performance than the detailed simulation so that the TCP protocol could adapt quickly to the bandwidth change [11].

The results presented in the next section were composed from a single sample of 200 million cycles from the above mentioned benchmarks. However, we did run many of the configurations for an extended period of time taking as many as 50 samples of 200 million cycles each. These runs show a coefficient of variation of $5\% \pm 2.5\%$ for the macrobenchmarks and under 1% for the microbenchmarks.

6. Results

We first examine the behavior of our simulated configurations using the netperf microbenchmark, varying CPU speed and L2 cache size. We then use our application benchmarks (web, NFS, and NAT) to explore the performance impact of our system configurations under more realistic workloads.

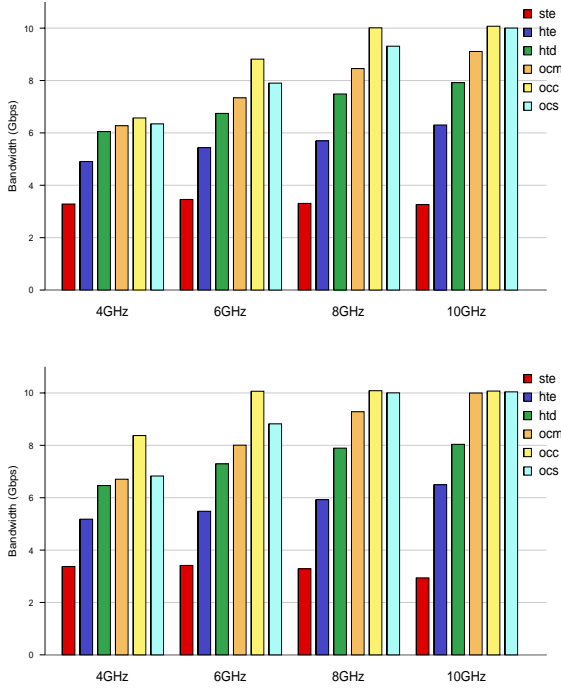


Figure 3. TCP receive microbenchmark performance. (Top: 4MB, Bottom: 16MB)

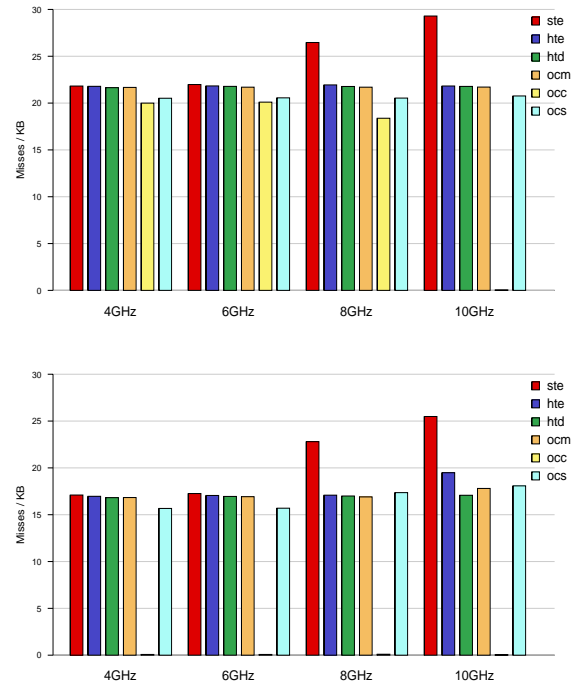


Figure 4. TCP receive microbenchmark misses/KB. (Top: 4MB, Bottom: 16MB)

6.1 Microbenchmark Results

Figure 3 plots the achieved bandwidth of our TCP receive microbenchmark across our six system configurations, four CPU frequencies (4, 6, 8, and 10 GHz) and two cache sizes. Although the higher CPU frequencies are not practically achievable, they show the impact of reducing the CPU bottleneck. In the future, their comparative performance will likely be achievable through multiprocessing, either through plentiful coarse-grained connection-level parallelism as is seen in the macrobenchmarks below or through advancements in protocol stacks that enable the processing of packets in parallel.

We first note that the STE configuration has no CPU bottleneck given that an increase in CPU performance does not impact bandwidth. It is likely that the latency between the CPU and the NIC alone accounts for the majority of the bottleneck given the fact that the HTE configuration also has nearly flat performance, albeit at a higher level. Conversely, the integrated NICs universally provide higher performance at higher CPU speeds, though their advantage over the direct HyperTransport interface is slight at lower frequencies when the benchmark is primarily CPU-bound. Comparing the top and bottom graphs in Figure 3, we see that the

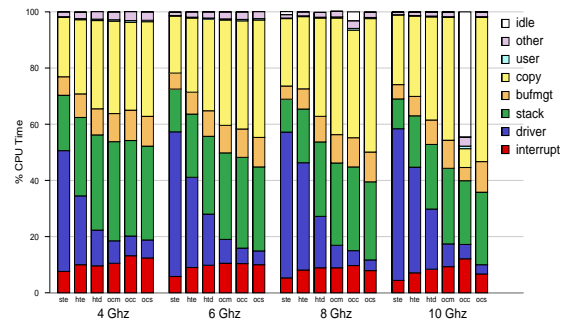


Figure 5. TCP receive microbenchmark CPU utilization. (4MB shown)

more tightly coupled interfaces also get a larger boost from larger LLC sizes.

In some situations, the in-cache DMA configuration (OCC) provides higher performance than OCM and OCS. The explanation for this difference can be seen in Figure 4, which shows the number of last-level cache misses per kilobyte of network bandwidth for these configurations. Because OCM and OCS NICs write payload data to memory, the CPU will always miss in the cache when accessing that data. The result is a minimum of 16 cache misses per kilobyte of data transferred, as seen in the configurations without cache placement. Cache placement alone is not a guarantee that these misses can be alleviated—it is necessary for

the cache to be large enough to hold the network receive buffers until the CPU accesses them. In this case, OCC dramatically reduces the number of cache misses incurred. Interestingly, this condition is a function of both the cache size and the CPU speed: a faster CPU is better able to keep up with the network and thus requires less buffering. Because our microbenchmark does not perform any application-level processing, the cache pollution induced by OCC when the cache is too small does not negatively impact performance. We will see a counterexample when we look at the macrobenchmarks below.

Figure 5 presents the breakdown of CPU utilization for the same configurations just described. Clearly, moving the NIC closer to the CPU drastically reduces the amount of time spent in the driver, the most dominant bottleneck, as it reduces the latency of accessing device registers. This translates directly to the higher bandwidths of Figure 3. However, most cases are still CPU-bound as the driver cost is replaced by CPU time in copy due to increased network bandwidth. However, when OCC is able to eliminate cache misses on network data (only with the 10 GHz CPU on this 4 MB cache), it drastically reduces the time spent in copy because the source data is in the cache rather than in memory. Looking at Figures 3 and 5 together shows that although some other configurations manage to saturate the network, only OCC at 10GHz does this with significant CPU capacity remaining to perform other tasks.

Figure 5 also illustrates a potential pitfall of integration: over-responsiveness to interrupts. Because the CPU processes interrupts much more quickly with the on-chip NIC, it processes fewer packets per interrupt, resulting in more interrupts and higher interrupt overhead. It is likely that this issue can be addressed using a more sophisticated interrupt moderation scheme.

Figure 6 presents the performance, cache, and CPU utilization results for the TCP transmit microbenchmark at a 4MB last-level cache size. In this microbenchmark, the sending CPU does not touch the payload data. The result is similar to what one might see in a static-content web server. Since the payload is not touched, a larger cache does not affect the results.

At lower CPU frequencies, on-chip NICs exhibit a noticeable performance improvement over direct HyperTransport; because transmit is interrupt intensive, low-latency access to the NIC control registers speeds processing. Again, we see that faster processors increase the utility of in-cache DMA, as they have fewer outstanding buffers and are thus more likely to fit them all in the cache. Although all of the configura-

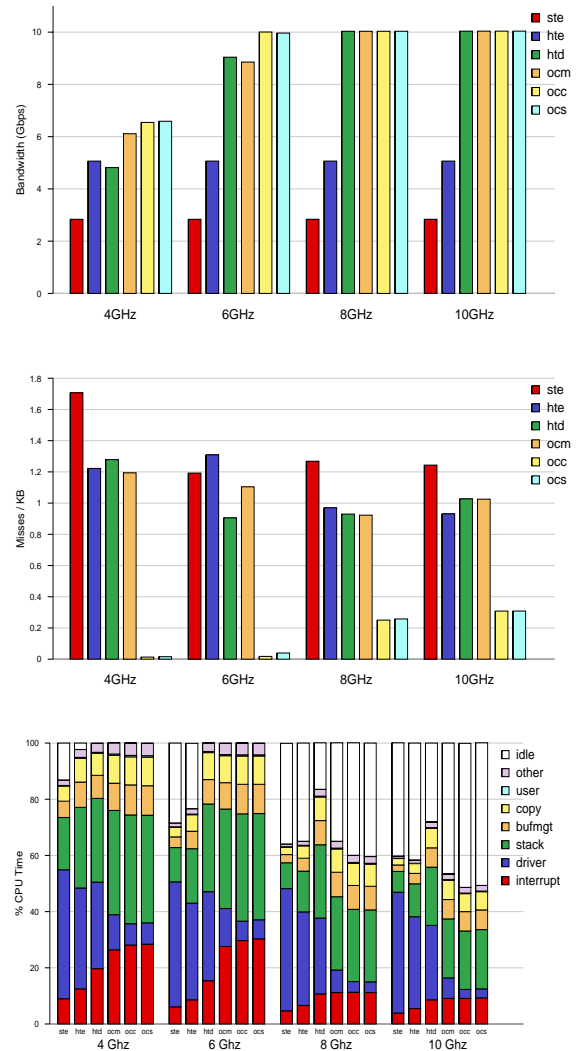


Figure 6. TCP transmit microbenchmark results.

tions have some idle time, with the faster CPUs the on-chip NICs have a distinct advantage over HTD. When looking at the cache performance results, DMA data placement affects only headers and acknowledgment packets, giving OCC and OCS similar behavior. Both incur significantly fewer misses than OCM, though this translates to only a slight decrease in CPU utilization due to the already low miss rate. (Note the difference in scale on the misses/KB graphs of Figures 4 and 6.)

The high idle time in STE and HTE is due to poor overload behavior; note that the link bandwidth is only a fraction of what HTD and the on-chip interfaces achieve. We are investigating whether this behavior is due to our device model, the NS83820 driver, or is inherent in Linux 2.6.

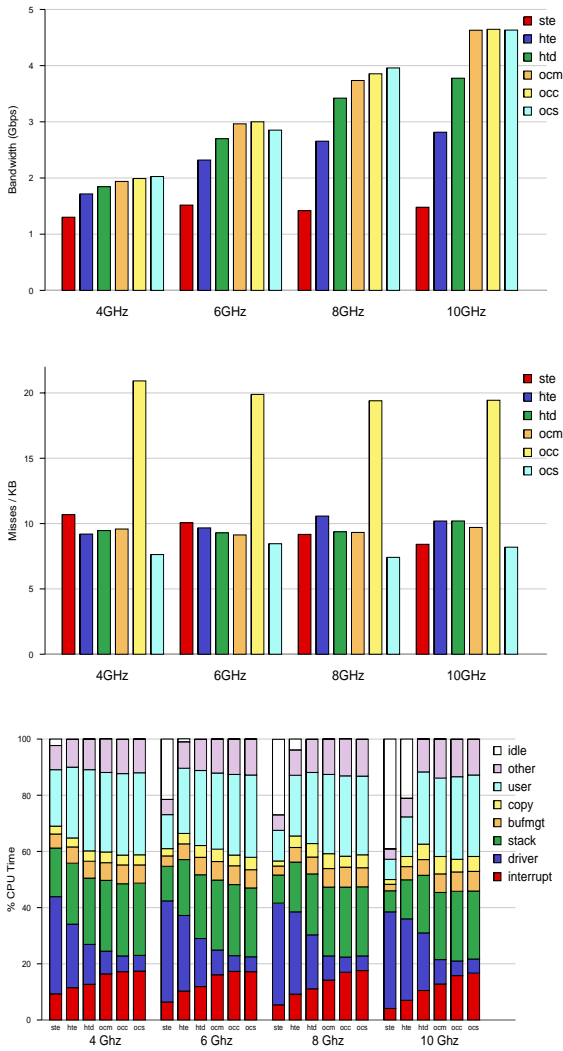


Figure 7. Web server benchmark.

6.2 Macrobenchmark Results

While the microbenchmark results provide valuable insight into the fundamental behavior of our configurations, they do not directly indicate how these configurations will impact real-world performance. To explore this issue, we ran the three application-level benchmarks described in Section 5: the Apache web server, an NFS server, and a NAT gateway. Although we ran with both 4 MB and 16 MB caches, we present only the 4 MB results here. For each benchmark, we show network throughput, L2 cache misses per kilobyte of network data transmitted, and a breakdown of CPU time.

The web server results are shown in Figure 7. In this test, we can see that the 4 GHz runs are CPU lim-

ited and only very minor performance improvements are realized by tighter integration of the NIC. On the other hand, the 10 GHz runs are network bound and achieve marked improvement in bandwidth when the NIC is tightly integrated. While a 10 GHz CPU may never be realized, a web server benchmark is highly parallel, and this single-cpu 10 GHz system performance could likely be achieved by a chip multi-processor system. Another thing that stands out in these graphs is that OCC has the opposite effect on misses/KB when comparing the web server benchmark to the microbenchmarks. This result is unsurprising since the working set of this application is non-trivial, unlike the microbenchmarks. Thus, the OCC configuration pollutes the cache and reduces cache performance. In this case, the additional misses appear to occur to user data not related to networking, as the fraction of time spent copying does not increase even though misses/KB increases. Because of the pollution, OCS's header-splitting approach achieves the lowest cache miss rate.

Figure 8 shows NFS server performance for the various configurations. Again, the 4 GHz runs are largely CPU bound and do not exhibit significant performance improvement with the on-chip NICs. Here, the interplay between network buffer sizing and CPU speeds is clearly illustrated. When looking at the bandwidth and misses/KB graphs for the 4 GHz CPU, OCC clearly pollutes the cache. (Recall that the client is doing block writes, which stress receive performance on the server.) However, at 10GHz, despite having the same cache size, OCC is a boon to performance. The 10GHz machine requires less buffering, resulting in a smaller working set. As with the microbenchmark, moving the NIC closer to the CPU drastically reduces the amount of time spent in the driver since it reduces the latency of accessing device registers. In addition, the time spent copying is similarly proportional to the misses. In nearly all cases, these effects result in improved bandwidth due to the loosening of the CPU bottleneck.

Figure 9 shows the NAT gateway performance. In this case, we are running the TCP receive microbenchmark between two hosts on either side of the gateway. The poor performance in the slower configurations is due to poor behavior under overload conditions.

The misses/KB graph shows that the OCC configuration eliminates all misses, while OCS eliminates misses on only header data, as we expect. For the 10 GHz CPU configuration, OCC and OCS are able to saturate the network link with CPU cycles to spare.

Overall, tighter integration is clearly a performance win in all cases, but the best performing config-

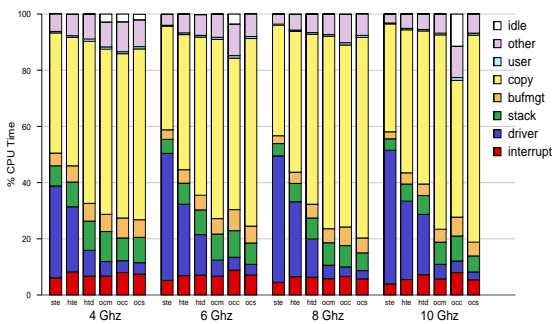
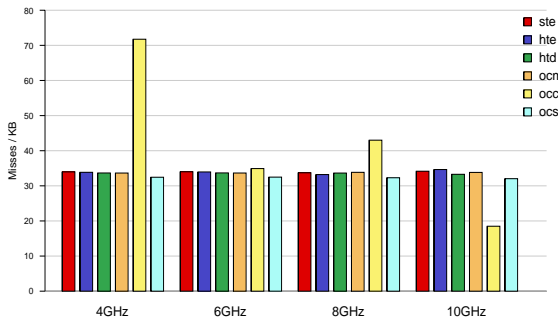
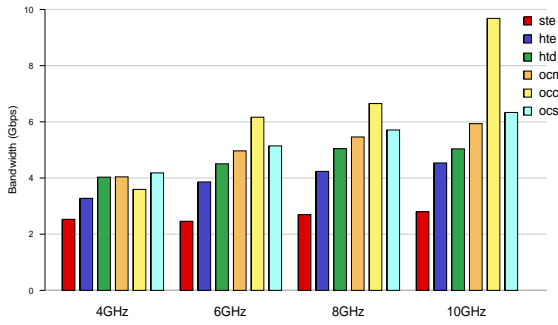


Figure 8. NFS server benchmark.

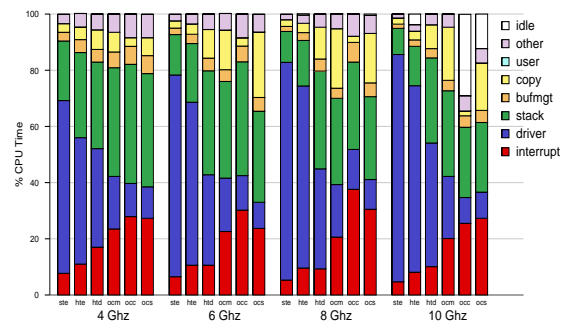
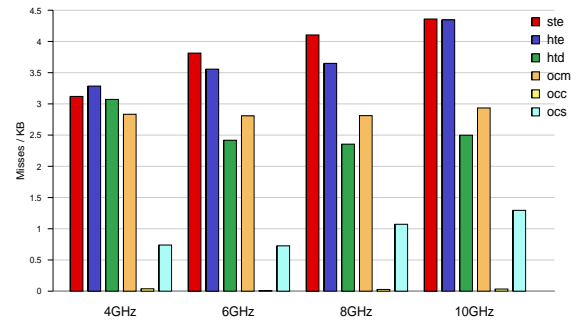
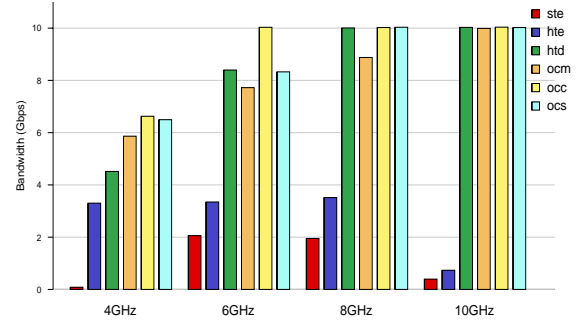


Figure 9. NAT gateway benchmark.

uration varies depending on the application. DMAing directly to the cache can yield huge performance differences in some circumstances while hurting performance in others. The header splitting configuration is a first step in attempting to mitigate the problem of cache pollution while achieving some of the benefit, but it is likely that more can be done to optimize performance.

7. Conclusions and Future Work

We have simulated the performance impact of tighter coupling between a 10 Gbps Ethernet NIC and the CPU, and find that this option provides higher bandwidth and lower latency than current implementations. Moving the NIC onto the CPU die itself provides a major opportunity for closer interaction with the on-chip memory hierarchy. Our results show a dramatic

reduction in the number of off-chip accesses when an on-chip NIC is allowed to DMA network data directly into an on-chip cache.

We have begun to investigate the potential for NIC-based header splitting to selectively DMA only packet headers into the on-chip cache. Clearly there is room for more intelligent policies that base network data placement on the expected latency until the data is touched by the CPU, predicted perhaps on a per-connection basis. The on-chip cache could also be modified to handle network data in a FIFO manner [26].

Another future opportunity lies in the interaction of packet processing and CPU scheduling. We have observed in this work the increasing impact of interrupts on high-bandwidth streaming. Along with this benefit comes an associated penalty for coalescing in a latency-sensitive environment. An on-chip NIC, co-

designed with the CPU, could possibly leverage a hardware thread scheduler to provide low-overhead notification, much like in earlier MPP machines [6, 20].

We have also demonstrated a simulation environment that combines the full-system simulation and detailed I/O and NIC modeling required to investigate these options. We have made this environment publicly available [15] to promote further research in this area.

While a general-purpose CPU is not likely to replace specialized network processors for core network functions, this trend should allow general-purpose systems to fill a wider variety of networking roles more efficiently, e.g., VPN endpoints, content-aware switches, etc. Given the very low latencies integrated NICs can achieve, we also see opportunity for using this “general-purpose” part as a node in high-performance message-passing supercomputers as well, eliminating the need for specialized high-performance interconnects in that domain.

In addition to exploring the above issues, our future work includes expanding our benchmark suite to include additional macrobenchmarks. We currently have a VPN application and an iSCSI-based storage workload under development. A comparison of the performance of an integrated NIC with a TCP offload engine (TOE) is highly desirable, but a TOE model and the associated driver and kernel modifications would be very complex to implement.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. CCR-0219640. This work was also supported by gifts from Intel and IBM, an Intel Fellowship, a Lucent Fellowship, and a Sloan Research Fellowship.

References

- [1] Alacritech, Inc. Alacritech / SLIC technology overview. http://www.alacritech.com/html/tech_review.html.
- [2] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. Network-oriented full-system simulation using M5. In *Proc. Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, pages 36–43, February 2003.
- [3] Broadcom Corporation. BCM1250 product brief, 2003. <http://www.broadcom.com/collateral/pb/1250-PB09-R.pdf>.
- [4] Doug Burger, Todd M. Austin, and Steve Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Technical Report 1308, Computer Sciences Department, University of Wisconsin–Madison, July 1996.
- [5] Russell Coker. <http://www.coker.com.au/bonnie++/>.
- [6] William J. Dally et al. The J-Machine: A fine-grain concurrent computer. In G. X. Ritter, editor, *Information Processing 89*, pages 1147–1153. Elsevier North-Holland, Inc., 1989.
- [7] Wu-chun Feng et al. Optimizing 10-Gigabit Ethernet for networks of workstations, clusters, and grids: A case study. In *Proc. Supercomputing 2003*, November 2003.
- [8] Pat Gelsinger, Hans G. Geyer, and Justin Rattner. Speeding up the network: A system problem, a platform solution. *Technology@Intel Magazine*, March 2005. <http://www.intel.com/technology/magazine/communications/speeding-network-0305.pdf>.
- [9] Dana S. Henry and Christopher F. Joerg. A tightly-coupled processor-network interface. In *Proc. Fifth Int’l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS V)*, pages 111–122, October 1992.
- [10] Hewlett-Packard Company. Netperf: A network performance benchmark. <http://www.netperf.org>.
- [11] Lisa R. Hsu, Ali G. Saidi, Nathan L. Binkert, and Steven K. Reinhardt. Sampling and stability in TCP/IP workloads. In *Proc. First Annual Workshop on Modeling, Benchmarking, and Simulation*, pages 68–77, June 2005.
- [12] Ram Huggahalli, Ravi Iyer, and Scott Tetrick. Direct cache access for high bandwidth network I/O. In *Proc. 32nd Ann. Int’l Symp. on Computer Architecture*, pages 50–59, June 2005.
- [13] Xbit Laboratories. http://www.xbitlabs.com/articles/cpu-display/lga775_19.html.
- [14] Keith Lauritzen, Thom Sawicki, Tom Stachura, and Carl E. Wilson. Intel I/O acceleration technology improves network performance, reliability and efficiency. *Technology@Intel magazine*, March 2005. <http://www.intel.com/technology/magazine/communications/Intel-IOAT-0305.pdf>.
- [15] M5 Development Team. The M5 Simulator. <http://m5.eecs.umich.edu>.
- [16] Peter S. Magnusson et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, February 2002.
- [17] Jeffery C. Mogul. TCP offload is a dumb idea whose time has come. In *Proc. 9th Workshop on Hot Topics in Operating Systems*, May 2003.
- [18] Shubhendu S. Mukherjee and Mark D. Hill. Making network interfaces less peripheral. *IEEE Computer*, 31(10):70–76, October 1998.
- [19] National Semiconductor. DP83820 datasheet, February 2001. <http://www.national.com/ds.cgi/DP/DP83820.pdf>.
- [20] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: A multi-threaded massively parallel architecture. In *Proc. 19th Ann. Int’l Symp. on Computer Architecture*, pages 156–167, May 1992.
- [21] Greg Regnier, Srihari Makineni, Ramesh Illikkal, Ravi Iyer, Dave Minturn, Ram Huggahalli, Don Newell, Linda Cline, and Annie Foong. TCP onloading for data center servers. *IEEE Computer*, 37(11):48–58, November 2004.
- [22] Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Stephen A. Herrod. Using the SimOS machine simulator to study complex computer systems. *ACM Trans. Modeling and Computer Simulation*, 7(1):78–103, January 1997.
- [23] Ali G. Saidi, Nathan L. Binkert, Lisa R. Hsu, and Steven K. Reinhardt. Performance validation of network-intensive workloads on a full-system simulator. Technical Report CSE-TR-511-05, Computer Science and Engineering Division, EECS, The University of Michigan, July 2005.
- [24] Lambert Schaelicke and Mike Parker. ML-RSIM reference manual. <http://www.cse.nd.edu/~lambert/pdf/ml-rsim.pdf>.
- [25] Standard Performance Evaluation Corporation. SPECweb99 benchmark. <http://www.spec.org/web99>.
- [26] Li Zhao, Ramesh Illikkal, Srihari Makineni, and Laxmi Bhuyan. TCP/IP cache characterization in commercial server workloads. In *Proc. Seventh Workshop on Computer Architecture Evaluation using Commercial Workloads*, February 2004.