Title: Analyzing Performance and Power Efficiency of Network Processing over 10GbE

Article Type: Special Issue: CommArch for Scalable Sys

Keywords: 10GbE, Network Processing, Instrumentation, Performance, Power, NIC.

Abstract: Ethernet continues to be the most widely used network architecture today for its low cost and backward compatibility with the existing Ethernet infrastructure. Driven by increasing networking demands of cloud workloads, network speed rapidly migrates from 1Gbps to 10Gbps and beyond. Ethernet's ubiquity and its continuously increasing rate motivate us to fully understand high speed network processing performance and its power efficiency.

In this paper, we begin with per-packet processing overhead breakdown on Intel Xeon servers with 10GbE networking. We find that besides data copy, the driver and buffer release, unexpectedly take 46% of processing time for large I/O sizes and even 54% for small I/O sizes. To further understand the overheads, we manually instrument the 10GbE NIC driver and OS kernel along the packet processing path using hardware performance counters. Our fine-grained instrumentation pinpoints the performance bottlenecks, which were never reported before. In addition to detailed performance analysis, we also examine power consumption of network processing over 10GbE by using a power analyzer. Then, we use an external Data Acquisition System (DAQ) to obtain a breakdown of power consumption for individual hardware components such as CPU, memory and NIC, and obtain several interesting observations. Our detailed performance and power analysis guides us to design a more processing- and power-efficient server I/O architecture for high speed networks.
Ethernet is the most widely used network architecture today. Its speed has been increased at a very unprecedented rate, migrating from 1Gbps to 10Gbps and beyond. This paper aims to understand high speed network processing performance on mainstream servers and then pinpoint its bottlenecks. Moreover, as power has become one major concern in IT industry, we also study power efficiency of high speed network processing and then breakdown power consumption over individual hardware components by using an external Data Acquisition System. Our detailed performance and power analysis leads us to several interesting and useful observations.

The major contribution of this paper is summarized as follows:

- We do per-packet processing overhead breakdown on Intel Xeon servers with 10GbE network to understand network processing overhead.

- To further pinpoint performance bottlenecks, we manually instrument NIC driver and OS kernel using hardware performance counters along the packet processing path. Our instrumentation locates the bottlenecks and points out several observations, which have never been reported before.

- We study power efficiency of high speed network processing over mainstream servers and then setup real environment using an external Data Acquisition System to obtain power consumption breakdown over individual hardware components.
Dear Reviewers,

Thanks for your valuable comments and feedbacks. We carefully went over them and revised our submission according to your comments. See my response as follows:

Reviewer 1:

1: This is an enhanced version of the HotI '10 paper by the authors. However, this paper is not cited at all in this submission. There seems to be a lot of overlap between this submission and the HotI '10 submission. The differences between these submissions need to be stated clearly.

Response: We already cited our Hot'10 paper in the new version and stated their differences clearly in related work. There are lots of difference between HotI '10 paper and this paper. The previous HotI paper focused on understanding energy efficiency of network processing over Intel Xeon processors and Sun Niagara 2 processors. But this paper conducted more extensive studies of network processing and covered both very detailed performance and power analysis.

2: In Introduction, the authors have mentioned that Ethernet is replacing InfiniBand, Fibre Channel, etc. This statement is not completely true.

Response: Thank you for pointing out the overstatement. I have removed this sentence in the paper.

3: Instead of performance counters, PMU should have been used.

Response: Thanks. I already changed to PMU in the new version.

4: Badly written cache organization. Not sure if 8MB LLC can be used instead of two 4MB LLC.

Response: I have explained cache organization more clearly in the new version of paper. Most of Intel Xeon Quad-Core processors in the market are using two LLCs (like Penryn, Clovertown, Westmre etc.), and they are widely used in servers.

5: Fig. 4 needs description.

Response: I have written more detailed description for Fig.4 in the new version.

6: For performance and power measurements different types of processors are used. It would have been better to carry out the study on the same processors.

Response: Thanks for your points. We conducted our power studies in Intel Labs using their platforms and tools. Although we don’t have detailed power breakdown data for the processors I used for performance analysis, they are in the same family of Xeon processors and thus we believe they exhibit similar behavior.
7: Do we see similar power and performance behavior for other benchmarks and applications?

Response: TCP/IP has been widely used and becomes a major communication infrastructure. It has been reported widely that TCP/IP processing itself consumes lots of CPU cycles on servers. TCP/IP processing is complex and involves several hardware components and system components (driver, system call, OS kernel, TCP/IP stack, NIC, IO bus and CPU etc.) Thus, this paper targets at TCP/IP processing itself by using micro-benchmark and isolating application behavior.

8: Table1 - There fine grained instrumentation method does not take into consideration the overhead of the calls to INSTRUMENT(). Nested INSTRUMENT() calls are used.

Response: We did take instrumentation overhead into consideration. One instrument call only takes ~70 CPU cycles and does not affect performance in our experiments.

9: No description on type and environment of their simulation results based on Simics.

Response: I added description about Simics in the new version.

10: Results are averaged over how many runs? Just saying "MANY" does not give a proper overview.

Response: Sorry for ambiguity. We run 5 times to get average number in our experiments.

11: A very important question is left unanswered. Why CPU power consumption rises for small I/O sizes, even when packets are grouped together to avoid overhead of per packet processing.

Response: Good question. Higher CPU power consumption with small I/O sizes is incurred from higher CPU utilization. In our experiments, we noticed that small I/O sizes incurred higher kernel overheads due to more frequent context switches among software interrupt, hardware interrupt and applications.

In order to clearly understand small packet processing overhead, we actually disabled Nagle algorithm in the sender size (coalescing many small packets into a larger packet of MTU size). In the receiver side, Large receive offload (coalescing packets into one larger packet) is enabled but it works for all packet sizes.

12: Solutions are suggested without any analysis. For instance, Rate adaption NIC does not even talk about its probable side-effects. What would be the overhead of transition and its effect on performance.

Response: thanks and we already added more discussion in the paper.
Reviewer 2:

1: The experiments are done between two servers, and eight connections are established in a unidirectional manner. The problem is that this does not reflect the bidirectional data transfers involved in real data center applications or in HPC environments. The unidirectional test and analysis is the initial step, but it does not provide the real picture. The authors should experiment with bidirectional traffic in a real-life environment with much more number of connections, generated by a larger number of nodes to evaluate the real impact on network processing.

Response: Thanks. Actually we did bidirectional experiments to understand achievable network bandwidth and CPU utilization. We realized that they exhibit similar behavior to unidirectional experiments. Thus, in order to very clearly understand systematic behavior of TCP/IP network processing (which is already complicated), we focused on unidirectional experiments in this paper and pinpointed performance and power bottlenecks, to motivate new I/O architectural designs.

2. In addition, the paper should extend the work to real data center applications (TPC-W, etc.) or HPC applications in MPI. It would have been more interesting to see the issues involved in network processing when the network in under real scenarios.

Response: Good suggestions. Since TCP/IP processing over 10GbE is very complicated and involves many architecture and system designs (driver, system call, OS kernel, TCP/IP stack, NIC, IO bus, IO controller and CPU etc.), this paper attempts to pinpoint the bottlenecks of TCP/IP processing and to motivate new I/O architecture designs. By doing this, we can isolate behavior from applications and clearly understand the network processing capability provided by OS and hardware. But yes, we will extend our work to include some real data center applications.

3: The paper targets TCP/IP, yet studies have shown that in the near future, majority of Internet traffic will be datagram based. Even in the HPC domain, recent research has shown the scalability and performance of MPI over datagrams. So, the study of UDP/IP would have made more sense, or at least would have nicely complemented the current work.

Response: Good suggestions. TCP/IP processing is still dominant in many areas (it was reported that up to 90% network traffics are TCP traffic ) including data centers, due to backwards compatibility and reliable in-order delivery features. For instance, the popular MapReduce model widely used by Google, Facebook and Yahoo leverages TCP/IP to communicate each other. But we will consider UDP/IP as well.
4: For the performance and power sections, two different server platforms have been used. I am not sure why. It would have made more sense to have the same experimental framework in order to be able to relate the findings in the performance section to those of power. There are no real application scenarios for the power section either.

Response: Thanks for your points. We conducted our power studies in Intel Labs using their platforms and tools when I was an intern there. Although we don’t have detailed power breakdown data for the processors I used for performance analysis, they are in the same family of Xeon processors and thus we believe they exhibit similar behavior.

5: Any insights as to the differences in the findings when TCP/IP offloading or segmentation offloading is supported in the adapter?

Response: In our experiments, we already enabled TSO (TCP segmentation offload), CRC offloading, Large receive offload. Regarding TOE (TCP offload engine), we don’t present it in this paper mainly because that TOE itself needs extensive OS kernel changes and has been criticized for a long time due to its hard programmability and maintenance, security issue etc. We did TOE performance analysis in our previous experiments and found that TOE does not improve performance but with slightly less CPU utilization. That is because TOE still moves data from NIC buffers to memory and gets OS kernel, driver and system calls involved. Our performance analysis in this paper shows that TCP/IP protocol itself is not a big overhead, indicating TOE’s limited performance improvement as well.

6: How about jumbo frames? Any insights or analysis on such results when jumbo frames is allowed?

Response: Good points. We targeted our research to widely used Ethernet frame sizes and have not covered Jumbo Frames yet which is limited in a small set of interconnected servers.

7: The bandwidth results for 8 connections are shown in the power section; it would be good to show the latency results as well, possibly in the performance section.

Response: We do have ping-pong latency results with various I/O sizes by running NetPIPE benchmark. However, since the paper focuses on pinpointing CPU performance bottlenecks of network processing over 10GbE and characterizing corresponding power consumption, I am hesitating a little bit to put latency results in the picture. But I would like to listen to your suggestions and would definitely add it if you think it is reasonable.
8. While I agree with the authors that Ethernet is dominant in data centers, I don't buy the argument that it is replacing other high-performance networks in the HPC domain, as suggested in the Introduction.

Response: Thank you for pointing out the overstatement. I have removed this sentence in the paper.

9. On the experimental platform, it appears the servers are directly connected together, without a switch. This has to be mentioned explicitly in the paper.

Response: thanks and already fixed.

10. Figure captions should be complete. For example, Figure 6 caption says "Data copy breakdown". Include receiver side, 16KB I/O, etc., in the caption to make the figures more informative. Use a more informative title as well for the figures.

Response: Fixed.

11. Is LSO used at the sender side? There are some results for the send side in the power section, but not in the performance section.

Response: LSO is used at the sender side. Since the receiver side is more CPU-intensive, we focused our performance analysis on receiver side. But we do have performance data for sender side as well.
See paper's highlights as follows:

- Our per-packet processing overhead breakdown points out three major overheads.
- Our fine-grained instrumentation in OS locates bottlenecks and reveals several observations.
- We find that unlike 1GbE NICs (~1W), 10GbE NICs have almost 10 Watts idle power dissipation.
- Our measurement shows that network processing over 10GbE has high power consumption.
- The power breakdown reveals that the major contributor is CPU, followed by main memory.
Analyzing Performance and Power Efficiency of Network Processing over 10GbE

Guangdeng Liao, Laxmi Bhuyan
Department of Computer Science and Engineering
University of California, Riverside, USA
{gliao, bhuyan}@cs.ucr.edu

Abstract

Ethernet continues to be the most widely used network architecture today for its low cost and backward compatibility with the existing Ethernet infrastructure. Driven by increasing networking demands of cloud workloads, network speed rapidly migrates from 1Gbps to 10Gbps and beyond. Ethernet's ubiquity and its continuously increasing rate motivate us to fully understand high speed network processing’s performance and its power efficiency.

In this paper, we begin with per-packet processing overhead breakdown on Intel Xeon servers with 10GbE networking. We find that besides data copy, the driver and buffer release, unexpectedly take 46% of processing time for large I/O sizes and even 54% for small I/O sizes. To further understand the overheads, we manually instrument the 10GbE NIC driver and OS kernel along the packet processing path using hardware performance counters (PMU). Our fine-grained instrumentation pinpoints the performance bottlenecks, which were never reported before. In addition to detailed performance analysis, we also examine power consumption of network processing over 10GbE by using a power analyzer. Then, we use an external Data Acquisition System (DAQ) to obtain a breakdown of power consumption for individual hardware components such as CPU, memory and NIC, and obtain several interesting observations. Our detailed performance and power analysis guides us to design a more processing- and power-efficient server I/O architecture for high speed networks.
1. Introduction

Ethernet continues to be the most widely used network architecture today for its low cost and backward compatibility with the existing Ethernet infrastructure. It dominates in large scaled data centers and is even competing with specialized fabrics such as InfiniBand [15], Quadrics [25], Myrinet [5] and Fiber Channel [7] in high performance computers. As of 2011, Gigabit Ethernet-based clusters make up 44.2% of the top-500 supercomputers [33]. Driven by increasing networking demands of cloud workloads such as video servers, Internet search, web hosting etc, network speed rapidly migrates from 1Gbps to 10Gbps and beyond [11]. Ethernet's ubiquity and its continuously increasing rates require us to clearly understand performance of network processing (or TCP/IP packet processing) over high speed networks and its power efficiency.

It is well recognized that network processing consumes a significant amount of time in network servers, particularly in high speed networks [3, 17-19, 22, 31]. It was reported that network processing in the receive side over 10Gbps Ethernet network (10GbE) easily saturates two cores of an Intel Xeon Quad-Core processor [18]. Assuming ideal scalability over multiple cores, network processing over upcoming 40GbE and 100GbE will saturate 8 and 20 cores, respectively. In the past decade, a wide spectrum of research has been done in this topic to understand its overheads [17-19, 22, 23, 31]. Nahum et al. [23] used a cache simulator to study cache behavior of the TCP/IP protocol and showed that instruction cache has the greatest effect on network performance. Zhao et al. [31] revealed that packets and DMA descriptors exhibit no temporal locality. Makineni et al. [22] conducted architectural characterization of TCP/IP processing on the Pentium M with 1GbE. However, they built their studies on cache simulators or used low speed networks, and did not conduct a system-wide architectural analysis for high speed network processing on mainstream server platforms.

Besides a lack of detailed performance analysis, high speed network processing has not been investigated carefully from the power perspective. As the concern on power and energy management has been arousing great interests in data centers with thousands of interconnected servers in recent years [1, 9,
it also becomes critical to understand the power efficiency of network processing over high speed networks like 10GbE on mainstream platforms.

In this paper, we begin with per-packet processing overhead breakdown by running a network benchmark over 10GbE on Intel Xeon Quad-Core processor based servers. We find that besides data copy, the driver and buffer release, unexpectedly take 46% of processing time for large I/O sizes and even 54% for small I/O sizes. To understand the overheads, we manually instrument the driver and OS kernel along the packet processing path using hardware performance counters (PMU) [14]. Unlike existing profiling tools attributing CPU cost such as retired cycles or cache misses to functions [24], our instrumentation is implemented at the fine-grained level and can pinpoint data incurring the cost. Through the above studies, we obtain several new findings: 1) the major network processing bottlenecks lie in the driver (>26%), data copy (up to 34% depending on I/O sizes) and buffer release (>20%), rather than the TCP/IP protocol itself; 2) in contrast to the generally accepted notion that long latency Network Interface Card (NIC) register access results in the driver overhead [3-4], our results show that the overhead comes from memory stalls to network buffer data structures; 3) releasing network buffers in OS results in memory stalls to in-kernel page data structures, contributing to the buffer release overhead; 4) besides memory stalls to packets, data copy implemented as a series of load/store instructions, also has significant time on L1 cache misses and instruction execution. Moreover, keeping packets in caches after data copy, which will not be reused, pollutes caches. Prevailing platform optimizations for data copy like Direct Cache Access (DCA) are insufficient for addressing the copy issue.

In addition to the above anatomized performance analysis, we also examine power consumption of network processing over 10GbE across a range of I/O sizes on Intel Xeon platforms by using a power analyzer [26]. To further understand insights of the power consumption, we set up our experimental environment with an external Data Acquisition System (DAQ) [8] and obtain the power consumption of individual hardware components (e.g. CPU, main memory, NIC). We find that unlike 1GbE NICs, which has a typical power consumption of about 1 Watt, 10GbE NICs have almost 10 Watts idle power dissipation. Our measurement shows that network processing over 10GbE has significant dynamic power
consumption. Up to 23 Watts and 25 Watts are dissipated in the receiving and transmitting processes without any computation from the application. The power breakdown demonstrates that CPU is the largest contributor to the power consumption and its power consumption reduces as the I/O size increases. Following CPU, memory is the second contributor but its power consumption grows as the I/O size increases. Compared to CPU and memory, the NIC has small dynamic power consumption. All of these point out that: 1) improving CPU efficiency of network processing has the highest priority, particularly for small I/O sizes; 2) a rate-adaptive energy management scheme is needed for modern high speed NICs.

The remainder of this paper is organized as follows. We revisit network processing in Section 2 and then present a detailed processing performance overhead analysis over 10GbE in Section 3, followed by detailed power studies in Section 4. Finally, we cover related literature and conclude our paper in Section 5 and 6, respectively.

![Figure 1. Network processing overview](image)

### 2. Network Processing

Unlike traditional CPU-intensive applications, network processing is I/O-intensive. It involves several platform components (e.g. NIC, PCI-E, I/O Controller or Bridge, memory, CPU) and system components (e.g. NIC driver, OS). In this section, we explain how network processing works on mainstream platforms. A high-level overview of network receiving process is illustrated in Figure 1. In the receive side, the
packet processing begins when the NIC hardware receives an Ethernet frame from the network. The NIC extracts the packet embedded inside the frame by removing the frame delineation information and updates a data structure, called a descriptor, with the packet information. Then, the NIC copies the incoming data into pre-allocated packet buffers in main memory using DMA operation over the PCI-E interface. Once the packet is placed in memory, the NIC generates an interrupt to CPU to kick off processing of the received packet. Then, CPU reads packets from main memory and does protocol processing. Therefore, the network receiving process involves various hardware components such as NIC, memory and CPU etc.

Similarly, the packet processing in the transmit side involves several hardware components. It starts when an application wants to transmit data by passing a data buffer to the TCP/IP stack. The TCP/IP stack copies data into kernel buffers and does protocol processing. The NIC driver triggers a DMA engine in the NIC to transfer packets from memory to NIC buffers and then the NIC sends them over Ethernet.

3. Understanding Network Processing Overhead over 10GbE

In this section, we analyze prevalent I/O architecture and present detailed CPU/NIC interaction in mainstream servers. Then we conduct extensive experiments to obtain per-packet processing overhead breakdown on Intel Xeon servers over 10GbE across a range of I/O sizes in Subsection 3.2. In Subsection 3.3, we manually instrument the 10GbE NIC driver and OS kernel along the packet processing path to locate the processing bottlenecks. Since the processing in the receive side is more complicated than the transmit side and consumes significant overheads, this section presents results from the receive side.

3.1. CPU/NIC Interaction

In the receive side, an incoming packet starts with the CPU/NIC interaction. The RX descriptors (typically 16 bytes each), organized in circular rings, are used as a communication channel between the driver and the NIC. The driver tells the NIC through these descriptors, where in the memory to copy the incoming packets. To be able to receive a packet, a descriptor should be in “ready” state, which means it has been initialized and pre-allocated with an empty packet buffer (SKB buffer in Linux) accessible by the NIC [30]. The SKB buffer is the in-kernel network buffer to hold any packet up to MTU (1.5 KB). It
contains an SKB data structure of 240 bytes carrying packet metadata used by the TCP/IP protocol and a DMA buffer of 2 KB holding the packet itself.

The detailed interaction is illustrated in Figure 2. To transfer received packets, the NIC needs to fetch ready descriptors from memory over PCI-E bus to know the DMA buffer address (step 1). When the NIC receives Ethernet frames from the network (step 2), it transfers the received packets into corresponding DMA buffers (denoted as buf in Fig. 2) using DMA engine (step 3). Once the data is placed in memory, the NIC updates descriptors with packet length and marks them as used (step 4). Then, the NIC generates an interrupt to kick off network processing in CPUs (step 5). In the CPU side, the interrupt handler in the driver reads the NIC register to check the interrupt cause (in step 6). If legally, the driver reads descriptors to obtain packet’s address and length, and then maps the packet into SKB data structures (step 7). After the driver delivers SKB buffers up to the protocol stack, it reinitializes and refills used descriptors with new allocated SKB buffers for incoming packets in the near future (in step 8). Finally, the driver re-enables the interrupt by setting the NIC register (step 9). After the driver, SKB buffers are delivered up to the protocol stack. Once the protocol stack finishes processing, applications are scheduled to move packets to user buffers. Finally, the SKB buffers are reclaimed into OS [6, 30].
3.2. Per-Packet Processing Overhead Breakdown

To understand network processing overheads, we conducted extensive experiments on two mainstream servers over 10GbE across a range of I/O sizes. Both SUT (System under Test) and stress machines are Intel servers, which contain two Quad-Core Intel Xeon 5335 processors [14]. Each core is running at 2.66GHz frequency and each processor has 2 last level caches (LLC) of 4MB each shared by 2 cores. The servers are directly connected by two PCI-E based Intel 10Gbps XF server adapters [16] without using a switch. They ran Linux kernel 2.6.21 and Intel 10GbE NIC driver IXGBE version 1.3.31. We retain default settings of the Linux network subsystem and the driver, unless stated otherwise. Note that LRO [12], a technique to amortize the per-packet processing overhead by combining multiple in-order packets into a large packet, is enabled in the driver. TSO is enabled in hardware NIC to segment large TCP packets. Stream hardware prefetcher employing a memory access stride based predictive algorithm is configured in the servers [14]. Since we aim to clearly understand network processing behavior over 10GbE and isolate application’s noise, in the experiments the micro-benchmark Iperf with 8 TCP connections is run to generate network traffic between servers (SUT is a receiver). We find from our experiments that one Xeon core with 4MB LLC achieves ~5.6Gbps throughput and two cores equipped with two 4MB LLCs are saturated to obtain line rate throughput. The high processing overhead motivates us to breakdown the per-packet processing overhead.

![Figure 3. Per Ethernet packet processing overhead breakdown](image-url)
We use the tool Oprofile to collect system-wide function overheads while Iperf is running over 10GbE. We group all functions into components along the network processing path: the NIC driver, IP layer, TCP layer, data copy (copy data from kernel buffers to user buffers after TCP/IP protocol processing), buffer release (OS reclaim SKB buffers after data is copied out to user buffers), system call and Iperf. All other supportive kernel functions such as scheduling, context switches etc. are categorized as others. Per-packet processing time breakdown is calculated and illustrated in Figure 3. Note that I/O sizes are not packets over Ethernet and large I/Os larger than MTU are segmented into several Ethernet packets (<=MTU).

We obtain the following observations from Fig.3: 1) the overhead in data copy increases as the I/O size grows and becomes a major bottleneck with large I/Os (>256 bytes); 2) the driver and buffer release consume ~1200 cycles and ~1100 cycles per packet, respectively, regardless of I/O sizes. They correspond to ~26% and 20% of processing time for large I/Os and even higher for small I/Os; 3) the TCP/IP protocol processing overhead is substantially reduced because LRO coalesces multiple packets into one large packet to amortize the overhead. Fig.3 reveals that besides data copy, high speed network processing over mainstream servers has another two unexpected major bottlenecks: the driver and buffer release.

3.3. Fine-grained Instrumentation

The Oprofile in Subsection 3.2 does profiling at the coarse-grained level and attributes CPU cost such as retired cycles and cache misses to functions. It is unable to identify data or macro incurring the cost. In order to locate the cost, we manually did fine-grained instrumentation inside functions. The environment in Subsection 3.2 is used. Table 1 shows one instrumentation example in the driver. We first measured the function's cost and then did fine-grained instrumentation for every code segment if the function has considerable cost. We continue to instrument each code segment with considerable cost until we locate the bottlenecks. Our instrumentation is applied to all functions along the processing path. Most of events are collected including CPU cycles, instruction and data cache misses, LLC misses, ITLB misses and DTLB misses etc. Since large I/Os include all three major overheads, this subsection presents the detailed
analysis for the 16KB I/O. Notes that each instrumentation call takes ~70 CPU cycles in our platform and the overhead has been considered in our results.

Table1. Fine-grained instrumentation

<table>
<thead>
<tr>
<th>Coarse-grain</th>
<th>Fine-grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUMENT(Counter1)</td>
<td>ixbbe_clean_tx_irq()</td>
</tr>
<tr>
<td>ixbbe_clean_tx_irq()</td>
<td></td>
</tr>
<tr>
<td>INSISTRUMENT(Counter2)</td>
<td></td>
</tr>
</tbody>
</table>

3.3.1. Driver

The driver comprises of three main components: NIC register access (step 6 and 9), SKB conversion (step 7) and SKB buffer allocation (step 8), as shown in Fig.2. Existing studies [3-4] claimed that NIC register access contributes to the driver overhead due to long latency traversal over PCI-E bus, and then proposed NIC integration to reduce the overhead. In this subsection, we architecturally breakdown the driver overhead for each packet and present results in Figure 4. In contrast to the general accepted notion that the long latency NIC register access results in the overhead [4], the breakdown reveals that the overhead comes from SKB conversion and buffer allocation. Although NIC register access takes ~2500 CPU cycles on mainstream servers, ~60 packets are processed per interrupt over 10GbE (~7 packets/interrupt over 1GbE) substantially amortizing the overhead. In addition, Fig.4 also reveals that L2 cache misses mainly result in the SKB conversion overhead and long instruction path is the largest contributor of the SKB buffer allocation overhead.

Since L2 cache misses in SKB conversion constitute ~50% of the driver overhead, we do detailed instrumentation to identify data incurring those misses. We group data in the driver into various data types (SKB, descriptors, packet headers and other local variables) and measure their misses. The result presented in Figure 5 reveals that SKB is the major source of the memory stalls (~1.5 L2 misses/packet on SKB). Different from prior studies [3-4], the memory stalls to packet headers are hidden and
overlapped with computation because the recent driver uses software prefetch instructions to preload headers before they are accessed. Unfortunately, SKB access occurs at the very beginning of the driver and software prefetch instructions cannot help. Although DMA invalidates descriptors to maintain cache coherence, the memory stalls to descriptors are negligible (~0.04 L2 misses/packet). That is because each 64 bytes cache line can host 4 descriptors of 16 bytes each and hardware prefetchers preload several consecutive descriptors with a cache miss. To understand the SKB misses, we instrument kernel to study its reuse distance over 10GbE. It is observed that SKB has long reuse distance (~240K L2 access), explaining the misses.

![Figure 4. Architecture breakdown (RX, 16KB I/O) Figure 5. L2 cache miss sources in step7](image)

### 3.3.2. Data Copy

After protocol processing, user applications are scheduled to copy packets from SKB buffers to user buffers. Data copy incurs mandatory cache misses on payload because DMA triggers cache invalidation to maintain cache coherence, and thus consumes a large number of CPU cycles. We study its architectural overhead breakdown as shown in Figure 6. 16KB I/O is segmented into small packets of MTU each in the sender and they are sent to the receiver. Fig.6 shows that L2 cache misses are the major overhead (~50%, ~3.5 L2 misses/packet), followed by data cache misses (~27%, ~50 misses/packet) and instruction execution (~20%). Although DCA implemented in Intel recent platforms avoids L2 cache misses, it is unable to reduce overheads in L1 cache misses and a series of load/store instructions execution (total ~47%). Due to the small L1 cache size, routing network data into L1 caches would pollute caches and
degrade performance [18, 29]. Moreover, since packets become obsolete after data copy [30], loading them into L1 caches or keeping them in L2 caches may evict other valuable data to incur cache pollution. Hence, more optimizations are needed to fully address the data copy issue.

![Figure 6. Data copy breakdown (RX, 16KB I/O)](image)

![Figure 7. Buffer release breakdown (RX, 16KB I/O)](image)

![Figure 8. L2 miss sources (RX, 16KB I/O)](image)

### 3.3.3. Buffer Release

SKB buffers need to be reclaimed after packets are copied to user applications. SKB buffer allocation and release are managed by slab allocator [6]. The basis for this allocator is retaining an allocated memory that used to contain a data object of certain type and reusing that memory for the next allocations for another object of the same type. Buffer release consists of two phases: looking up an object cache controller and releasing the object into the controller. In the implementation of slab allocator, the page data structure is used to keep cache controller information and read during the object cache controller lookup. This technique is widely used by mainstream OS such as FreeBSD, Solaris and Linux etc.
Figure 7 shows architectural overhead breakdown for buffer release. We observe from Fig.7 that L2 cache misses are the single largest contributor to the overhead (~1.6 L2 cache misses/packet). Similarly, we analyze data sources of L2 cache misses and present results in Figure 8. The figure reveals that L2 cache misses are from the 128 bytes in-kernel page data structures. The structure reuse distance analysis shows that it is reused after ~255K L2 cache access, explaining the cache misses.

3.4. Optimization Discussion

The above studies reveal that besides memory stalls to itself, each packet incurs several cache misses on corresponding data and has considerable data copy overhead. Some intuitive solutions like having larger LLC (>8MB for 10GbE) might help to some extent, but it has major limitation. We extend Simics by enhancing it with detailed cache, I/O timing models and effects of network DMA. We extend the DEC 21140A Ethernet device with the support of interrupt coalescing using Device Modeling language to simulate a 10GbE Ethernet NIC. Our simulation results based on Simics simulating 4 cores sharing one LLC, without considering application memory footprint, 16MB LLC is needed to avoid those cache misses of packet processing over 10GbE. When network jumps to 40GbE and beyond, increasing LLC becomes an ineffective solution.

In order to address the cache misses on SKB and in-kernel page data structure, one straightforward approach is to extend existing DCA infrastructure to deliver both packets and corresponding missed data into caches when NIC receives a packet. But this approach would stress PCI-E interconnect bus and thus degrade PCI-E efficiency of packet transfers. This PCI-E issue can be naturally avoided if the NIC is integrated into CPU die to eliminate PCI-E bus. In contrast to redesigning hardware infrastructure, one alternative solution is to redesign system software to change data access pattern. We can dedicate a continues memory range to incoming packets and reuse them as soon as possible, thus avoiding cache misses related to SKB buffer management (SKB access in driver, SKB buffer release). All above approaches only aim at cache misses. To address the data copy issue, besides with the support of DCA, we can add one hardware on-chip cache copy engine to move packets insides caches. The hardware copy engine avoids a series of load/store instructions and bypasses L1 cache misses. We believe that a new I/O architecture including the above optimizations is needed for high speed networks in future.
4. Understanding Power Efficiency of Network Processing

The above section performs a detailed performance analysis of network processing on mainstream servers over 10GbE. In this section, we extend our studies to another important aspect: power consumption. In Subsection 4.1, we present our experimental methodology to measure power consumption of network processing. In Subsection 4.2, we show extensive power results.

4.1. Power Measurement

We first use two Intel Xeon machines to conduct experiments across various I/O sizes. The Intel server is a two-processor platform based on the Dual Core Intel Xeon processor 5500 series with 8 MB of L3 cache per processor [14]. Both of the machines are equipped with 4GB DDR3 DRAM. The two machines are connected by two PCI Express based Intel 10Gb/s XF server adapters [16]. Each server has an Intel 10GbE server adapter (a.k.a Oplin), which is connected to hosts through PCI-E x8. This provides a 16+16 Gbps full-duplex I/O fabric that is fast enough to keep up with the 10+10 Gbps full-duplex network port.

In our experiments, the system power consumption is measured by the tool Power Analyzer Model 380801 from Extech Instruments [26]. The tool connects the wall power supply into its input terminal and the tested server power leads into its output terminal. The software component of this device is called Power Analyzer (version 2.11 was used). We determine the power consumption of network processing over 10GbE by using a subtractive method. We first measure the server idle power consumption as baseline. Then we measure the total power while running networking workloads. Since idle power stays the same for different workloads, it is not workload dependent. We subtract the idle power from the total power to get the dynamic power consumption while processing TCP/IP packets (in other words, the dynamic power consumption means the deltas above the idle).

Instead of using real world workloads, we have elected to use widely-used network micro-benchmark Iperf for our evaluation. The choice of the micro-benchmark gives us a more controlled test environment and helps us easily understand the power consumption from network processing [17, 18, 19]. In our
platforms, 8 concurrent connections are run to generate 10Gbps traffics among the transmitter and the receiver. All the results are the average data among five runs on each I/O size.

While the power analyzer tool gives the system total power consumption, it is unable to provide detailed component power information. To track the power consumed by various hardware components involved in network processing such as CPU, memory, NIC etc, we add sensing resistors to +12V, +5V, +3.3V power supply, as well as each DIMM and the 10GbE PCI-E NIC card. The 12V rail supplies power to CPU, memory, board components as well. We only care about the 12 V rail from 8-pin connectors since it goes to CPU and memory. After subtracting the power consumption on each DIMM, we obtain the CPU power consumption. We measure the VDC on each sensing resistors, since we know the resistance value of each sensing resistor, we compute the current from it. Based on the power supply voltage, we are able to determine the power consumption on each hardware component. In our experiments, we use an external 2645A Data Acquisition System (DAQ) [8] that is connected to hardware components we want to measure as shown in Figure 9. We sample voltage value of multiple channels simultaneously with 1 sample per second. Fluke DAQ software (version 4.0) is used to configure the instrument and monitor/collection the instrument's online values and alarms.

![Figure 9. Power measurement configuration](image-url)
4.2. Evaluation Results

We start with the measurement of the system idle power consumption. In our baseline, we find that 10GbE NICs have high idle power dissipation, ~10 Watts in our NICs [16]. In addition, we study the dynamic power consumption from running Iperf over 10GbE. The dynamic power consumption is obtained by subtracting the idle power from the measured system power while running TCP/IP. Figure 10 shows the network receiving throughput and the corresponding dynamic power consumption along various I/O sizes. The figure shows that TCP/IP has high power consumption in the receive side. With large I/O sizes (>1KBytes), the server achieves a 9.49Gbps network throughput by dissipating ~17 Watts power. When we come to small I/O sizes (<1KBytes), the server obtains much lower network throughput but consumes significantly higher power. For instance, at the 64 Bytes I/O size, the server only has 0.73 Gbps network throughput while dissipating ~23 Watts power. In our experiments, we notice that small I/O sizes incurred higher kernel overheads due to more frequent context switches among software interrupts, hardware interrupts and applications.

![Figure 10. Throughput & dynamic power (RX)](image)

To understand insights of the dynamic power consumption, we breakdown it into individual hardware components involved in the network receiving process, such as CPU, memory, NIC. The power breakdown methodology is described in Section 4.1 and the result is illustrated in Figure 11. The figure
demonstrates that CPU is the largest power contributor, regardless of I/O sizes, and takes 70% to 82% of the power consumption. For the large I/O sizes (>2KBytes), CPU dissipates ~11.5 Watts for packet processing. The CPU power consumption increases to ~18 Watts when we use small I/O sizes. Following CPU, the memory is the second largest power contributor. The percent of the memory power consumption increases as the I/O size goes up, from 11% with the 64 Bytes I/O size to 15% with the 64KBytes I/O size. For each packet, two packet data relevant memory accesses are required in the receive side: one packet memory write from the DMA engine and one packet memory read from the CPU during packet processing (note that as shown in Section 3, each packet also incurs two another packet data irrelevant memory accesses on SKB and page data structures). Large I/O sizes have more memory accesses to packets and incur higher power consumption. Compared to CPU and memory, while the NIC has considerable idle power dissipation, it has much smaller dynamic power consumption, 1.09 Watts to 1.29 Watts, corresponding to 5% to 8% of the whole power consumption.

![Figure 11. Dynamic power breakdown (RX)](image)

In addition to the receive side, we also study the performance and the power dissipation of the transmit side. The results are shown in Figure 12. Similar to the receive side, high power consumption is needed in the transmit side and small I/O sizes have higher power dissipation (~22 Watts) than large I/O sizes (~18.5 Watts). One power spike occurs at the I/O size 1KBytes (~24.2 Watts) because the CPU utilization is highest at 1KBytes among all I/O sizes in our experiments. We break down the power
consumption to understand power consumption of individual hardware components as shown in Figure 13. The figure points out that CPU dissipates 75% to 84% of the dynamic power, corresponding to ~13 Watts with the 64 KBytes I/O size and ~18 Watts with the 64 Bytes I/O size. As the I/O size increases from 64 Bytes to 64KBytes, the memory power consumption grows from ~1.8 Watts to ~3.1 Watts. That is because large I/O sizes incur more memory accesses than small I/O sizes. Similar to receiving packets, the NIC power consumption slightly increases from ~1 Watts to ~1.26 Watts as the I/O size increases.

![Figure 12. Throughput & dynamic power (TX)](image)

![Figure 13. Dynamic power breakdown (TX)](image)
4.3. Optimization Discussion

The above analysis shows that 10GbE brings high power cost including idle power consumption and dynamic power consumption. We discuss several architectural power optimizations along three dimensions: NIC, CPU and memory. To reduce the NIC power cost, we propose using a rate-adaption NIC into CPUs. It intelligently adjusts its processing capability based on packet arriving rates. For instance, the NIC can reduce its 10Gbps MAC processing capability to 1Gbps or even to 100Mps if there are no arriving/sending packets or low arriving/sending rates. By using the rate-adaption NIC, the server can substantially reduce its idle power cost. Notes that the rate adaption NIC might introduce some packet losses during its transition from low rate to high rate. Fortuantely, network stacks are able to recover those lost packets through packet retransmission.

While CPU has become much more power efficient in recent years, the 10Gbps packet processing still dissipates more than 10 Watts. Since the packet processing is a memory-intensive application, using complicated out-of-order cores is not as power efficient as using small in-order cores [27]. Therefore, we propose a single-ISA heterogeneous multi-core architecture, where a small set of cores are small in-order cores but share the same hardware instruction set. All cores and memory banks are organized as NUMA architectures. The small cores and their local memory are dedicated to processing TCP/IP packets. By leveraging better power efficiency of small cores, the server architecture can avoid wasting power and behave more power efficiently. In addition to providing better CPU power efficiency, the performance optimizations discussed in Section 3.4 can improve network processing performance and also naturally translate to better power efficiency.

We find that ~3 Watts in memory are needed while processing packets. That is mainly because that two packet data relevant memory accesses occur for each packet in the receive side (one write from the DMA engine and one read from the CPU), and one packet memory access in the transmit side. In the receive side, prevailing optimizations DCA and Cache Injection are able to avoid those DMA writes by injecting network I/O data into caches, but injected packets are in the "Modified" state and will be written back to
memory when they are replaced. In the transmit side, packets sit in caches with the "Modified" state after the user-to-kernel copy. They need to be written back to memory to keep cache coherency while the NIC is fetching transmission data via DMA operation. In order to avoid those packet memory accesses, we propose cache invalidation instructions used by the protocol stack to actively invalidate packets in caches after processing. This optimization can fully eliminate packet data relevant memory accesses.

5. Related Work

It is well documented that Internet servers spend a significant portion of time processing packets [3, 17-19, 22, 31]. In the past decade, a wide spectrum of research has been done in network processing to uncover its characteristics and to optimize the processing efficiency [2-4, 10, 17-19, 22-23, 31-32]. Nahum et al. [23] used a cache simulator to study cache behavior of the TCP/IP protocol and showed that instruction cache has the greatest effect on network performance. Zhao et al. [31-32] used a cache simulator to study cache behavior of the TCP/IP protocol. They showed that packets show no temporal locality and proposed a copy engine to move packets in memory. Liao et al. [20] conducted power studies of TCP/IP processing over Intel Xeon servers and Sun Niagara 2 processors, but the paper focused on power analysis and did not conduct detailed network processing performance analysis over 10GbE.

In addition, researchers have also proposed several architectural schemes to optimize the processing efficiency. Most of them aimed to reducing the data copy overhead. Zhao et al. [32] designed an off-chip asynchronous DMA engine close to main memory to move data inside memory. The similar idea has been implemented in Intel platforms with the Intel I/OAT technique [2], but has been widely criticized in industry because memory stalls are still incurred when applications read packets from memory. To eliminate the memory stalls, Intel proposed DCA to route network data into CPU caches [10], and implemented it in Intel 10 GbE adapters and server chipsets. Its performance evaluation on real servers has demonstrated overhead reduction in data copy [17-18]. Recently, Tang et al. [29] claimed that DCA might incur cache pollution on small LLC and introduced two cache designs (a dedicated DMA cache or limited ways of LLC) to keep packets. Binkert et al. [3-4] integrated a redesigned NIC to reduce the processing overhead by implementing zero-copy and reducing access latency to NIC registers. However,
all of them did not conduct detailed system-wide performance analysis and are unable to pinpoint real performance bottlenecks of high speed network processing on mainstream servers. Moreover, they also did not study network processing from the power perspective.

6. Conclusion

As Ethernet network becomes ubiquitous and its speed continues to grow rapidly, it becomes critical for us to clearly study high speed network processing on mainstream servers from two important perspective: processing performance and power consumption. In this paper, we first studied the per-packet processing overhead on mainstream servers with 10GbE and pinpointed three major performance overheads: data copy, the driver and buffer release. Then, we carefully instrumented the driver and OS kernel along the packet processing path to do a system-wide architectural analysis. Unlike existing tools attributing CPU cost to functions, our instrumentation was done at the data granularity and can pinpoint data with considerable cost. Our studies reveal several new findings, which have never been reported due to a lack of fine-grained instrumentation.

In addition, we also examined power consumption of network processing over 10GbE on mainstream servers by using a power analyzer. To clearly understand its power consumption, we use an external Data Acquisition System to obtain a breakdown of power consumption for individual hardware components. Our investigation leads to several interesting observations. Given the trend towards rapid evolution of network speed, we envision that our detailed performance and power analysis can guide us to design a more processing- and power-efficient server I/O architecture.

References

[29] D. Tang, Y. Bao, W. Hu et al., DMA Cache: Using On-chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance, HPCA 2010.
Guangdeng Liao received the BE and ME degrees in computer science and engineering from Shanghai Jiaotong University, and the PhD degree in computer science from the University of California, Riverside. He currently works as a research scientist, at Intel Labs. His research interests include I/O architecture, operating system, virtualization and data centers.

Laxmi N. Bhuyan is a professor of computer science and engineering at the University of California, Riverside. His research interests include network processor architecture, Internet routers, and parallel and distributed processing. Bhuyan has a PhD from Wayne State University. He is a Fellow of the IEEE, the ACM, and the American Association for the Advancement of Science.