Impact of Protocol Overheads on Network Throughput over High-Speed Interconnects: Measurement, Analysis, and Improvement*

Hyun-Wook Jin
Dept. of Computer Science and Engineering
Konkuk University
1 Hwayang-dong, Gwangjin-gu
Seoul 143-701, Korea
jinh@konkuk.ac.kr

Chuck Yoo
Dept. of Computer Science and Engineering
Korea University
1, 5Ka, Anam-Dong, Sungbuk-Ku
Seoul 136-701, Korea
hxy@korea.ac.kr

Abstract

Although extremely high-speed interconnects are available today, the traditional protocol stacks such as TCP/IP and UDP/IP are not able to utilize the maximum network bandwidth due to inherent overheads in the protocol stacks. Such overheads are a big obstacle for high-performance computing applications to exploit high-speed interconnects in cluster environments. To address this issue, many researchers have been presenting analyses of protocol overheads and suggesting a number of optimization approaches to harness the TCP/IP suite over high-speed interconnects. However, to the best of our knowledge, there is no study that analyzes and optimizes the protocol overheads thoroughly in an integrated manner. In this paper, we exploit a set of protocol optimization mechanisms in an integrated manner while dealing with the full spectrum of the protocol layers from the transport layer to the physical layer. To evaluate the impact of each protocol overhead, we apply the optimization mechanisms one by one and perform detailed analyses at each step. The thorough overhead measurements and analyses reveal the dependencies between protocol overheads. With our

* This work was supported by the Faculty Research Fund of Konkuk University in 2006.

comprehensive optimizations, we show that UDP/IP can utilize more than 95% of the maximum network throughput a Myrinet-based experimental system can provide.

Keywords: UDP/IP, Myrinet, Clusters, High-Speed Interconnects, and Programmable Network Interface Cards

1. Introduction

The network physical bandwidth is getting dramatically larger and has reached more than a Gbps bandwidth [1, 2, 25, 26]. Many high-performance computing systems (e.g., clusters [3] and grids [4]) employ such high-speed networks [5-7, 31]. On the other hand, with the rapid increase of data volume to process, efficient data exchange between the computing nodes becomes more critical for many parallel/distributed applications. Accordingly, many data-intensive applications wish to take the advantage of high bandwidth of modern interconnects. Most of existing applications and middleware use IP-based transport protocols such as TCP and UDP to transmit the data between nodes. The reason for this is two fold: (i) the IP network is the most widely accepted network infrastructure in the world and (ii) the IP-based transport protocols support the *de-facto* standard interface, *sockets*. The traditional IP-based protocols, however, cannot deliver the maximum network throughput to the applications due to their heavy overheads. For example, we have observed that UDP/IP only achieves a throughput under 20% of the maximum throughput provided by Myrinet [1] on an Intel Pentium III platform.

Many researchers have tried to reduce the protocol overheads to utilize the maximum physical network bandwidth. One such approach is to design and implement a new high-speed user-level protocol [27-30]. The basic idea of the user-level protocols is to provide new light-weight transport protocols at the user-level and bypass the traditional protocols implemented in the operating system kernel. This can provide an easy way to

achieve zero-copy data transmission with minimal packet handling overhead. In addition, since the user-level protocols target only the LAN environments (i.e., clusters) they use very simple network layers. However, the user-level protocols introduce new API. Since most of existing applications have been implemented on top of the sockets API, this approach requires modifications of existing applications. In addition, their simple network layers cannot provide the compatibility for the IP networks.

On the other hand, to utilize the high-speed networks keeping the advantages of the IP-based protocols, there has been a significant research on optimizations of the IP-based protocols. For instance, the researchers [8-13, 32, 35] have tried to reduce the per-byte overheads by enhancing the network buffer management. Another approach is to deploy the TCP Offload Engine (TOE) [34] that offloads the processing of TCP/IP protocols from the host processor to the hardware on the Network Interface Card (NIC) while preserving the sockets interface. Since both approaches of optimizing host-based protocols and offloading the protocols allow existing applications to run over IP networks without any modification, they have a lot of potential.

However, there has been no study of in-depth overhead analysis performing comprehensive optimizations across several layers. Since the protocol stacks consist of several protocol layers and have dependency between the layers, it is highly desired that the optimization study of the TCP/IP stacks deals several layers in an integrated manner. To achieve this goal we need to measure the performance in details, implement several kinds of optimization schemes for each layer, and analyze their impact on other layers. This is not a trivial research challenge. To tackle this research issue, we take following approaches in this paper:

- **Optimizations over multiple layers:** The protocol stacks consist of several layers such as transport, network, data link, and physical layers. Accordingly, we consider multiple layers to identify the bottleneck and exploit several solutions to fully utilize the physical network bandwidth.

- **Stepwise optimizations:** In order to elucidate the impact of each overhead, the overheads have to be removed in a stepwise manner. Our optimizations start from higher protocol layers and move to lower layers of which performance impacts are shown in each optimization step.
- **Analysis of dependencies between overheads:** Removing some overheads can make an impact on other overheads as well. Such dependencies are very important factors when we try to optimize the protocol stacks because the dependency information can provide a way to fully optimize the protocol stacks. We may achieve a partial optimization without the dependency information.

We can classify the protocol overheads of an end node into the host, NIC, and link overheads by the viewpoint of where the overheads are induced [33]. The host overhead is the time that the host processor spends to perform the protocol stacks in the operating system's kernel and the device driver. The kernel usually implements the transport and network layers. The NIC overhead is generated by the NIC itself, which corresponds to the data link layer. The link overhead is the time to send/receive a packet to/from the physical network link, which is concerned with the physical layer. In this paper, we first focus on the host overhead such as the (de)fragmentation overhead of the network layer and the checksum computation and data copy overheads of the transport layer because the host overhead is dominant compared with NIC and link overheads. Next, we deal with the NIC overhead such as per-DMA, per-packet, and per-byte overheads of the data link layer. Finally, we study the link overhead.

Our thorough and comprehensive analyses of the protocol overheads contribute to the understanding of how much bandwidth is encroached by each overhead and what factors make performance bottlenecks on a high-speed interconnect. In addition, the mechanisms proposed and implemented in this paper reveal useful design considerations that help develop or optimize a protocol for high performance computing systems. Our performance evaluation presents that by removing protocol overheads across several layers we can utilize

more than 95% of the maximum network throughput can be provided by a Myrinet-based experimental system. While this work has been done with the Myrinet-based system, our ideas and the results are applicable to most of modern systems. To the best of our knowledge, this is the first work that carries out in-depth overhead analyses and comprehensive optimizations side by side in an integrated manner while dealing with the full spectrum of the protocol layers from the transport layer to the physical layer.

The remainder of this paper is organized as follows: Section 2 briefs the data path of an UDP/IP packet and gives an overview of Myrinet. Sections 3, 4, and 5 analyze the host, NIC, and link overheads and minimize them, respectively. We generalize some design and performance issues based on our study and discuss them in Section 6. Section 7 describes related work. Finally the paper concludes in Section 8.

2. Background

In this section, we describe the data path of UDP/IP packets on an end node, which is applicable to the most of contemporary systems. We also brief the overview of Myrinet.

2.1 Data Path of UDP/IP Packet

In this section, we briefly describe the protocol layers of UDP/IP and the data path. If an application requests a send through the system call, the UDP layer copies the packet from the user buffer into the kernel buffer. The data checksum computation is also done in this layer. Then the IP layer performs the packet fragmentation if the data size is larger than the maximum transmission unit (MTU) size. The most of modern operating systems including Linux perform the data copy, checksum computation, and fragmentation at the

same time for better performance. After constructing the UDP/IP packet header, the device driver makes a descriptor for the packet and passes it to the NIC. The NIC performs DMA to move the data indicated by the descriptor from the kernel buffer to the NIC memory. Then, the NIC ships the data with a MAC header to the network.

When an application calls a receive system call, if there is any data that already has been received in the socket buffer queue, the data is copied to the user buffer by the UDP layer. Otherwise, the application is forced to block. When a packet arrives from the network to the NIC memory, the NIC moves the packet to the network buffer in the kernel space using DMA. After the completion of the DMA operation, the NIC triggers an interrupt. The interrupt handler handles this interrupt and put the descriptor of the packet into the backlog queue. After that, to reduce the number of interrupts of notifying the DMA completion, the interrupt handler usually polls the DMA completion queue for some more time and handles more packets if there are. This is known as NAPI in Linux. Then the bottom half identifies the packet type and passes the packet to the appropriate protocol. In the present case, the protocol is IP, which performs the defragmentation. If the defragmentation is finished successfully, the IP layer passes the packet to the UDP layer. After the checksum computation, the UDP layer wakes the application that has been waiting and copies the user data in the packet to the user buffer.

2.2 Overview of Myrinet

Myrinet [1] is a quite popular high-speed interconnection network technology developed by Myricom Incorporation. Myrinet can provide 1.28Gbps on Myrinet-1280 link and 2.0Gbps on Myrinet-2000 link in one direction. In this paper, we consider both link types and show their impact on network throughput. Recently

Myricom has released Myri-10G that supports 10Gbps data rate providing the compatibility with 10 Gigabit Ethernet. One of the interesting features of Myrinet is the programmability of the NIC firmware, which allows developers to add new functions or offload protocol stacks into the NIC firmware. Though many other NICs are also programmable, usually the tools and interfaces for the firmware programming are not opened to the general system developers. We utilize this NIC programmability feature of Myrinet in this paper to optimize the protocol stacks. In addition to the IP-based protocols, Myrinet also provides the high-speed user-level protocol called *GM*. As we have discussed in the introduction, in this paper we focus on the IP-based protocols (i.e., UDP/IP) over Myrinet.

3. Host Overhead: Transport (UDP) and Network (IP) Layers

This section deals with the host overheads such as (de)fragmentation, checksum computation, and data copy overheads, which are overheads in the transport and network layers. The workstations on which we analyze the overheads and throughput use an Intel Pentium III 450MHz processor and an M2F-PCI32C (LANai-4) Myrinet NIC set to a 32bit 33MHz PCI slot. Workstations are directly connected via an M2F-CB-25 Myrinet cable. The operating system installed is Linux (kernel version 2.2), and we adopt Myrinet Software (version 3.22c) [16] for the device driver and the firmware of the NIC.

We measure the throughput using *ttcp* [17] that calculates the throughput at the receiver side by the difference of the first packet arrival time and the last packet arrival time for a large volume back-to-back packet transmission. In addition, to measure the pure overhead for specific operations (e.g., fragmentation, data copy, checksum computation, DMA, etc.), we inserted code lines that generate timestamps by reading the timer into the corresponding kernel and firmware functions. We also implemented a simple interface to

extract these timestamps and calculate the average for each overhead. When we measure the overheads we run the standard latency test in a ping-pong fashion in which for given iterations the client sends a packet and waits a reply packet of the same size from the server.

3.1 (De)fragmentation Overhead

The data larger than the MTU size is fragmented into several packets by IP to transmit the data over the network, which results in a per-packet overhead. This processing overhead increases in proportion to the number of fragments. In addition, the packet header per fragment wastes the network bandwidth.

We measure the (de)fragmentation overhead to see its impact. The solid lines in Figures 1(a) and (b) show the fragmentation and defragmentation overheads, respectively. We vary the data size from 32B to 32KB. It is difficult to extract the pure fragmentation overhead on the sender because, as described in Section 2.1, UDP/IP implemented in Linux performs the fragmentation concurrently with the data copy and the checksum computation. Thus the overhead in Figure 1(a) includes the overhead for the data copy and checksum computation as well.

Since the default MTU size of Myrinet is 4KB, we can observe that the overheads labeled *4KB MTU* in Figure 1 suddenly increase when the data size is a multiple of 4KB. The fragmentation and defragmentation overheads are around 31μs and 9μs per fragment, respectively. More importantly, the defragmentation routine of the Linux kernel version 2.2 performs the copy operation that moves all received fragments into a large buffer in order to merge fragments. This copy operation induces about 10μs overhead per 1KB. As a result, in Figure 2, the throughput labeled *4KB MTU* reaches its peak at 4KB while it remains slightly below the peak maximum with larger packet sizes than the MTU.

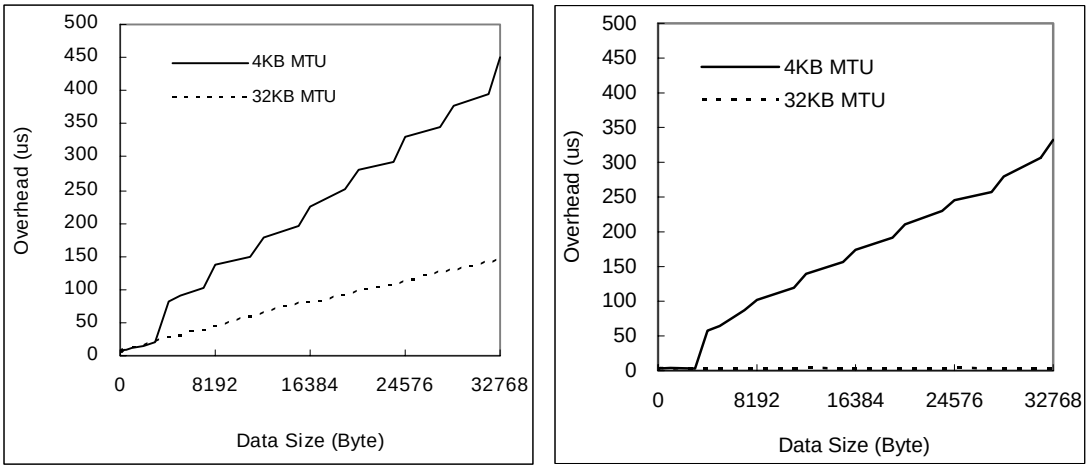

Figure 1 Fragmentation and defragmentation overheads: (a) Fragmentation overhead on the sender and (b) Defragmentation overhead on the receiver

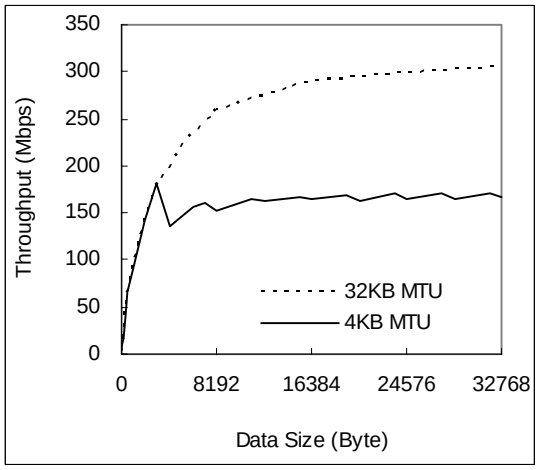

Figure 2 Throughput comparison of UDP/IP with 4KB and 32KB MTU

A notable characteristic of high-speed interconnects is their large MTU size. The jumbo frame of Gigabit Ethernet is 9000B. Myrinet also can have a larger MTU size than 4KB with the modification of the Myrinet software. In order to remove the (de)fragmentation overhead, we enlarge the Myrinet MTU size through the modification of the device driver and the NIC firmware. We need to consider two factors when we try to

increase the MTU size. First, the NIC has a relatively small memory size. In the case of M2F-PCI32B Myrinet NIC, the memory (i.e., SRAM) size on the NIC is only 512KB in which Myrinet Software allocates 4 send buffers and 10 receive buffers. Therefore, when the MTU size is 32KB, 448KB (= 4 × 32KB + 10 × 32KB) is required for the send and receive buffers, leaving only 64KB memory space. The 60KB of the rest memory space is assigned to the NIC firmware and the remainder is used for the stack and shared data between the host and NIC. Second, we should consider the size of the socket buffer that is the kernel buffer allocated per connection to store the network data. For instance, Linux kernel version 2.2 allocates 32KB as the socket buffer. Therefore, even if the MTU size is larger than the socket buffer size, applications cannot take advantage of the excessively large MTU size. Due to these two limitations we decide to increase the MTU size up to 32KB.

As we can see, the overhead labeled *32KB MTU* in Figure 1 does not show (de)fragmentation overhead. In the case of Figure 1(a), only the data copy and checksum computation overheads remain. Consequently, Figure 2 shows that the throughput with the 32KB MTU size increases continuously even for data sizes larger than 4KB. The improvement over the 4KB MTU size reaches 84% at 32KB.

3.2 Checksum Computation Overhead

The UDP layer performs the checksum computation for the whole network data and header (i.e., the UDP header and pseudo header). In Linux, the checksum computation of the sender side is performed during the copy operation, which maximizes the cache effect. The solid line of Figure 3(a) shows the data copy and checksum computation overheads on the sender. On the other hand, the receiver performs the checksum computation separately with the data copy, which is shown in Figure 3(b). One of interesting results in Figure 1 is that the checksum computation overhead on the receiver is high up to 132% in comparison with that of

the sender. This is because the checksum computation on the receiver is performed with uncached data that has not been accessed after being moved from the NIC to the host memory while the sender computes the checksum with cached data. Since the *ttcp* benchmark uses the same buffer for every packet transmission the data on the sender side is always present at the cache for the throughput test. Thus the data copy and checksum computation of the sender are performed with a cached data. This is also true for many other applications because the data is usually touched or generated before the transmission.

There has been a significant research on improving the checksum computation. Partridge and Pink [9] have combined the checksum computation and the data copy to reduce the number of data touch and maximize the cache effect, which reduces the checksum computation overhead up to 75%. Braden et al. [22] and Kay and Pasquale [11] have suggested optimization algorithms for different machine architectures and improved the throughput up to 33%. Their suggestions are already employed in most of modern protocol implementations including Linux kernel we use; however, Figure 3 shows that the checksum computation overhead still induces a large overhead.

In order to offload the checksum computation from the host processor, we utilize the NIC's checksum hardware. Many high-speed NICs include checksum hardware, which is generally integrated with the DMA engine and does not generate an additional overhead. For example, Myrinet has CKS register [19] in LANai-4 and DMA control block [20] in LANai-9 for the hardware checksum. Upon completion of DMA between the host and NIC memories, the CKS register and DMA control block would contain the checksum result.

One of important design issues for the checksum computation offloading is how to get the result of the checksum computation for user data, UDP header, and pseudo header excluding the others. DMA performs for the whole data and headers because it follows the IP layer. Thus the DMA engine computes the checksum with user data, UDP header, and pseudo header but also with whole IP header including option headers and

Ethernet header. In order to solve this problem, our modified NIC firmware performs DMAs for the header and data separately and uses the hardware checksum result of only the data part. Then the firmware computes the addition of the hardware checksum result of the data part to the UDP header and pseudo header. The checksum result is put into the UDP header for sending data or reported to the host with interrupt for receiving data.

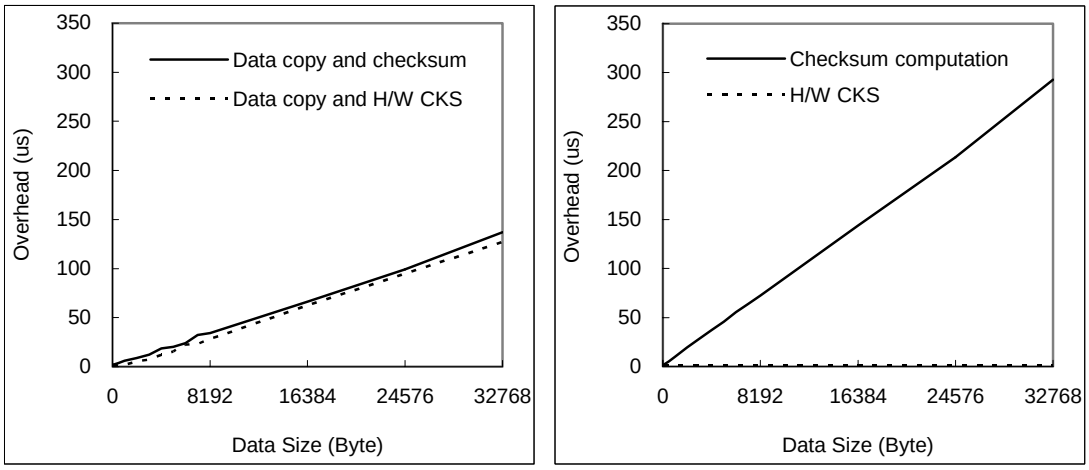

Figure 3 Checksum computation overheads: (a) sender and (b) receiver

A drawback is that such a hardware-assisted checksum computation may partially break down the layering of the protocol stack since the data checksum computation is the role of the transport layer. Despite this disadvantage, the hardware checksum can remove the checksum computation overhead dramatically as shown in Figure 3(b)-*H/W CKS*. Figure 3(a)-*Data copy and H/W CKS* presents that the hardware checksum on the sender does not prominently reduce the overhead. This is due to the fact that the implementation of the checksum computation on the sender side is highly optimized as previously described. Hence we can obtain a larger performance gain when we apply the hardware checksum to the receiver.

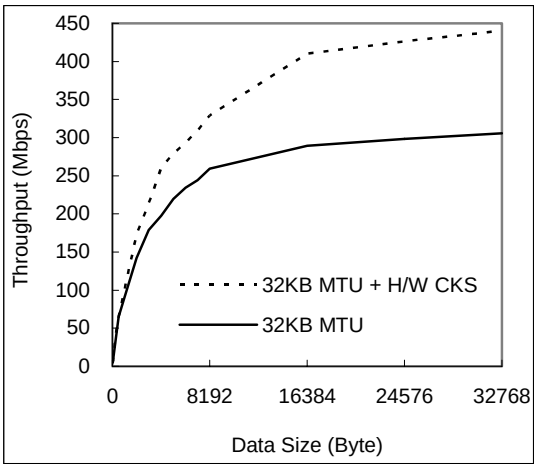

Figure 4 Throughput improvements by eliminating the checksum computation overhead

Figure 4 shows that UDP/IP with the hardware checksum achieves 44% higher throughput than the UDP/IP of the improved version of removing (de)fragmentation overhead in Section 3.1. *Trapeze* [14] is another example that offloads the checksum computation with hardware assistance in the NIC. They append the checksum result on the end of the packet. Therefore, an end node that performs this checksum offloading mechanism is not interoperable with the end nodes that implement only standard protocols. On the other hand, the hardware checksum implemented in this paper is transparent to the corresponding node because this hardware checksum preserves the header structure and its semantics.

3.3 Data Copy Overhead

In addition to the checksum computation, UDP performs the copy operation that moves the data between the user and kernel buffers. Page-remapping [8, 10, 12, 14] is a representative mechanism to eliminate the copy operation. This mechanism, however, can cause a copy-on-write as a penalty of removing the data copy

overhead, which is much more severe than the copy overhead. In addition, page-remapping requires to map a page even for the packets much smaller than the page size (e.g., 4KB).

We eliminate the data copy by using the pointer of user buffer and directly moving the data between the user and NIC buffers bypassing the kernel buffer. When an application calls the send system call, our mechanism lets the system call return without copying data into the kernel buffer. What it does is to save the packet information, such as the pointer to the user buffer and its length. To save this information, we extend the *sk_buff* structure in the Linux kernel. Then, the data in the user buffer are directly moved to the NIC memory by DMA when NIC is available to process the packet. When the data arrives from the network, NIC moves it directly to the user buffer without copying it to the kernel memory. We refer the details of this zero-copy mechanism to Yoo et al. [21].

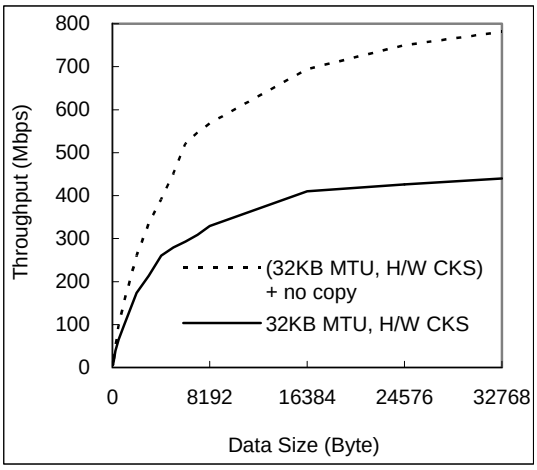

Figure 5 Throughput improvements by eliminating the data copy overhead

Figure 5 presents the throughput comparison between UDP/IP employing the hardware checksum and that additionally removing the copy overhead. An interesting result in the figure is that the throughput of UDP/IP with both hardware checksum and zero-copy is much higher than with the hardware checksum only. Comparing the throughput with 32KB MTU in Figure 2, we get the 156% higher throughput when we exploit

both hardware checksum and zero-copy, while we only achieve the 44% higher throughput with the hardware checksum alone. The reason why the removal of the copy operation (after the checksum computation) results in even higher improvement is due to the cache effect. Since the received data is not cached, any operation to touch the data in the first time takes longer. That is, the checksum computation by UDP/IP on the receiver side is done with the uncached data while the following copy operation is done with the data cached during the checksum computation. However, if the DMA engine on NIC performs the checksum computation, which does not result in caching data, the copy operation on the host has to be done with the uncached data. Therefore, on the receiver, the data copy overhead becomes large with the hardware checksum as much as the traditional checksum computation overhead as shown in Figure 6. Thus the removal of both checksum computation and the copy operation eliminates the overhead of touching uncached data and leads to a high throughput.

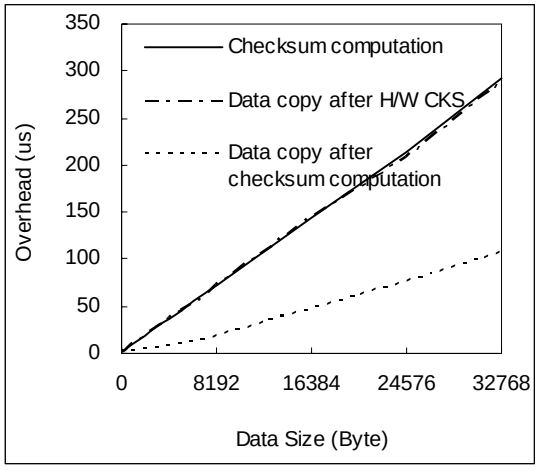

Figure 6 Comparison of data copy overheads after the traditional host-based checksum computation and after the NIC-based hardware checksum computation on the receiver

4. NIC Overhead: Data Link Layer

So far, our main focus has been the host overhead (i.e., network and transport layers) because it was the dominant overhead but the host overhead now has been significantly reduced in Section 3. With the mechanisms implemented in Section 3, we measure the host, NIC, and link overheads as shown in Figure 7. In the figure we can clearly see that the NIC overhead of the data link layer now becomes the new dominant overhead for most data sizes while the host overhead is flat over the data size.

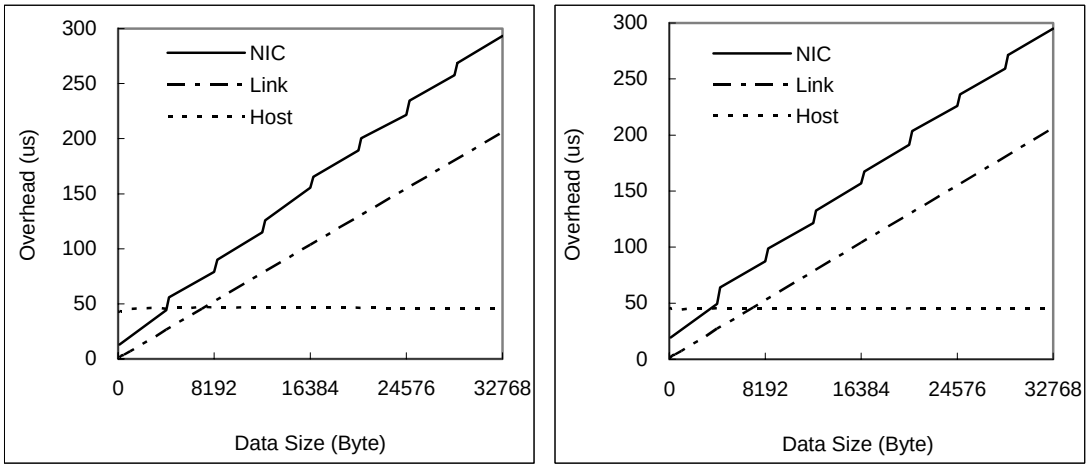

Figure 7 Comparison of NIC, link, and host overheads after the host overhead optimizations: (a) sender and (b) receiver

The NIC overhead consists of per-DMA, per-packet, and per-byte overheads. The per-DMA overhead is the initialization overhead for each DMA operation. The per-packet overhead includes the cost required generating each packet (i.e., frame) on the sender and the cost for searching of a corresponding receive buffer. The per-byte overhead is induced during the data movement between the host and NIC memories. Following subsections analyze and try to reduce each of them.

4.1 Per-DMA Overhead

DMA is a very efficient way to move data between the host and NIC memories, but the DMA initialization overhead is large and results in the per-DMA overhead. In addition, DMA can perform only for a physically linear buffer. If the network buffer is not physically contiguous, the NIC needs a scatter/gather to move a chunk of data. A scatter/gather is a list of vectors, each indicating the location and length of one linear physical memory segment. Several vectors in a scatter or gather induce multiple DMA initializations proportional to the number of vectors.

In the case of traditional protocols, the kernel allocates a physically linear memory area for the network buffer. On the other hand, it is to be noted that the zero-copy UDP/IP described in Section 3.3 uses the memory area in user space as the network buffer, which is not guaranteed to be physically linear. Therefore, the network buffer crossing the page boundary leads to multiple DMA initializations.

To reduce the number of DMA initializations, we implement a new memory allocation function (i.e., *malloc( )*) that returns physically linear memory area to the application and override the existing *malloc( )* function. The new function internally calls a new system call that allocates a physically linear memory area in the kernel, maps this area into the user space allowing the application to use, and returns the buffer pointer. Using this new function, applications can use the physically linear buffer for the network buffer without any modifications and avoid multiple DMA initializations.

Figure 8 shows the dependence of DMA overhead on the linearity of the network buffer. We measured the overhead with a page-aligned buffer so that the number of vectors in a scatter/gather can easily be estimated. As shown in the figure, DMA overheads are the same up to 4KB regardless of the buffer linearity due to the 4KB page size of Linux. That is, obviously, only one vector is required for data sizes smaller than

4KB. However, we can see that the DMA overhead of the nonlinear buffer is gradually getting larger than that of the linear buffer as the data size grows. This is due to the DMA initialization needed for every 4KB memory segment of the nonlinear buffer. Figure 8 also shows that the linear buffer reduces the DMA overhead by 30μs at 32KB in each side. We find that, as shown in Figure 9, this difference translates into 10% increase in the throughput of UDP/IP.

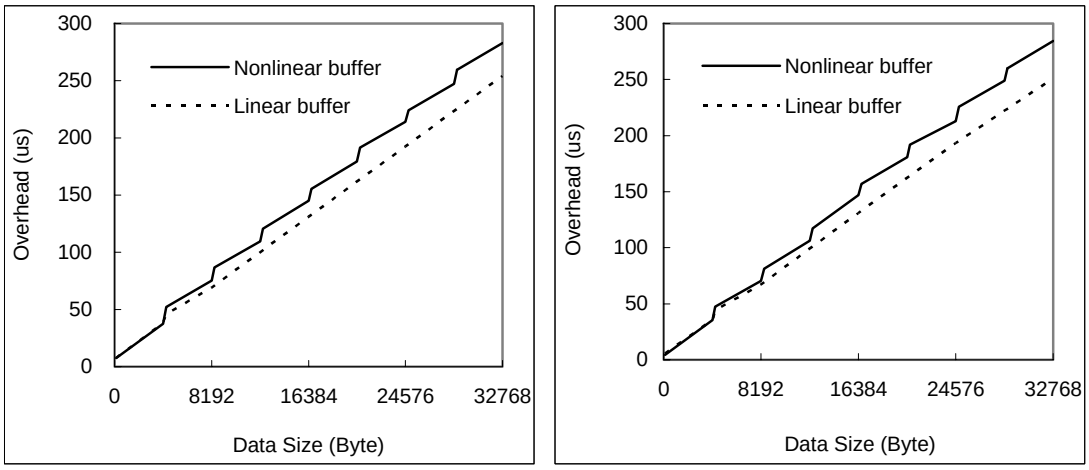

Figure 8 DMA overhead depending on the linearity of network buffer: (a) sender and (b) receiver

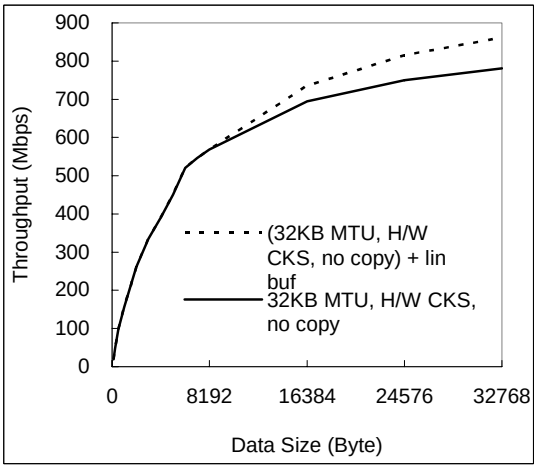

Figure 9 Throughput comparison of UDP/IP with nonlinear and linear network buffer

4.2 Per-Packet Overhead

In this section, we try to reduce the per-packet overhead on NIC. First we see the dependency of NIC's per-packet overhead on the NIC processor speed by adopting another Myrinet NIC equipped with a faster processor. Then we study on the overhead of searching for corresponding receive descriptor.

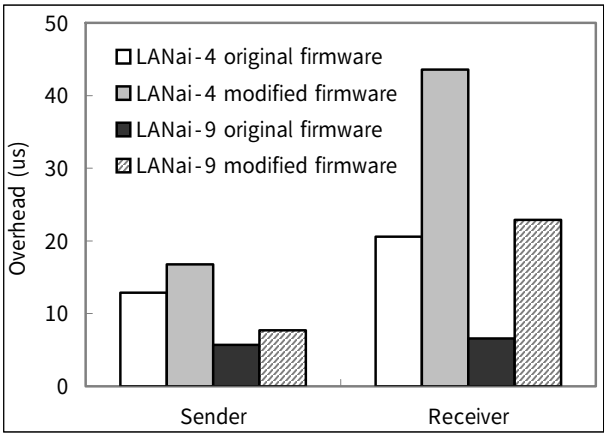

Figure 10 NIC overheads on LANai-4 and LANai-9 when a 32B data is transmitted

Since the NIC processor is order of magnitudes slower than the host processor, adding new functions to the NIC firmware affects the NIC overhead sensitively. Figure 10 compares the NIC overheads without firmware modification and with firmware modification on LANai-4 NIC when a 32B data is transmitted through UDP/IP. It is to be noted that the firmware modifications to implement checksum offloading and zero-copy (described in Sections 3.2 and 3.3, respectively) increase the NIC overhead on LANai-4 by 23μs on the receiver side. To see the impact of the firmware modifications on a faster NIC processor, we adopt M2L-PCI64B Myrinet NIC, which has a 133MHz LANai-9 RISC (about 4 times faster than LANai-4). We use GM (version 1.4) [18] for the device driver and the NIC firmware of LANai-9 NIC. Figure 10 shows that the introduced overhead by the firmware modifications has been reduced significantly. Overall, the LANai-9 NIC

reduces the per-packet overhead by more than 1/2 on both sender and receiver, respectively. We can observe that Figure 11 shows the throughput of the improved UDP/IP on the LANai-9 NIC.

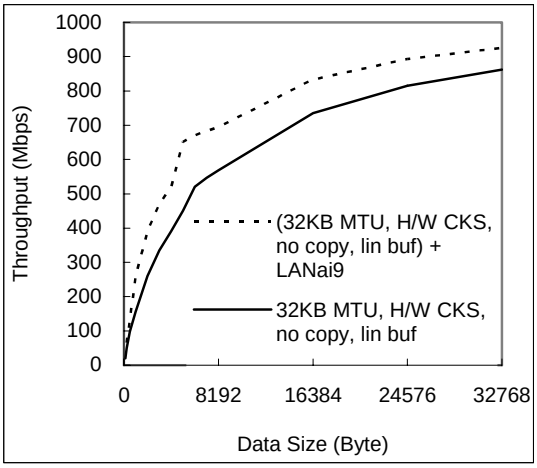

Figure 11 Throughput comparison of UDP/IP on LANai-4 and LANai-9 based NICs

Although the faster NIC processor has reduced the NIC overhead a lot, the modified firmware still shows a high overhead on the receiver side and, comparing with sender side, has been less affected by the NIC processor speed. This is mainly because the modified firmware on the receiver generates an additional interrupt to search a corresponding receive buffer for every messages. An important role of the NIC firmware is to decide the host memory location to DMA the received packet. In the traditional implementation of UDP/IP over Myrinet, the device driver pre-allocates a set of kernel buffers in the initialization phase and pre-posts them. The firmware then simply DMAs received packets to the pre-posted buffers. Then the kernel takes care of demultiplexing of the packet across processes and copies the packet to the final user buffer. Thus, the firmware does not need to interrupt the host to know where the received packet should be placed. On the other hand, the modified firmware for zero-copy UDP/IP in Section 3.3 has to know the final user buffer because the packet does not go through the intermediate kernel buffer. Since the basic zero-copy implementation keeps the receive requests into the kernel internal queue, the firmware generates an interrupt

to query the corresponding final user buffer. In addition, the firmware should wait until the host returns the final buffer information before issuing the DMA. Therefore, although this design can provide an easy implementation of synchronization between the host and NIC, it causes a large overhead.

To remove this additional interrupt, we move the queue that stores applications' receive requests into the NIC. With this new design, the firmware can access the queue directly without interrupting the host. Moreover, since the firmware can immediately get the final buffer information, we can remove the waiting time in the firmware. The host accesses the queue in the NIC through PIO (Programmed I/O). As a result, we can reduce the NIC overhead on the receiver up to 16μs.

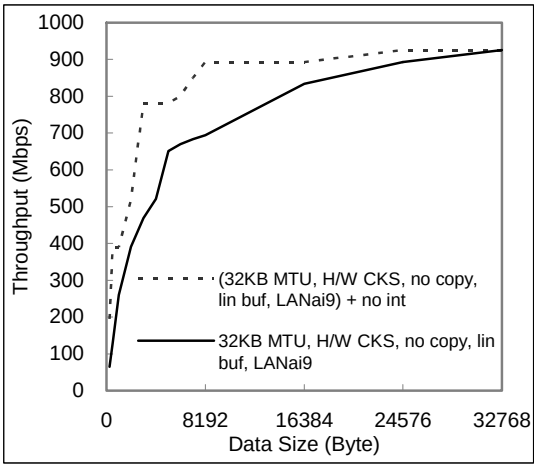

Figure 12 Throughput comparison of UDP/IP with interrupt based and non-interrupt based buffer searching

By removing the interrupt, we can improve the throughput by 200% for small and medium packet sizes as shown in Figure 12. Since the interrupt and interrupt handling can be overlapped with a large data transmission, we cannot see a huge benefit with new firmware on the large data sizes. The maximum throughput of the improved UDP/IP is 926Mbps, which is a remarkable performance on a 32bit 33MHz PCI platform with Myrinet comparing with 808Mbps by Gallatin et al. [23] and 302Mbps by Barak et al. [13].

4.3 Per-Byte Overhead

The per-byte overhead of the NIC is the DMA overhead of moving data byte by byte, which depends highly on the performance of the I/O bus (i.e., PCI bus). Figure 13 compares DMA overheads on 32bit 33MHz (Intel 440BX Chipset) and 64bit 66MHz (AMD 762 Chipset) PCI systems. The 64bit 66MHz PCI platform is equipped with an AMD Athlon MP 2000+ processor. The figure shows that the DMA overhead of the 64bit 66MHz PCI system is significantly smaller than the other. Now the link overhead becomes the dominant overhead compared with the host and NIC overheads as shown in Figure 14.

As shown in Figure 15, the UDP/IP with 64bit 66MHz PCI bus achieves over 1Gbps at 4KB data size and reaches its maximum throughput of 1.22Gbps at 7KB data size. We can observe that the throughput does not increase any more after the 7KB data size because the UDP/IP utilizes over 95% of the physical network bandwidth of 1.28Gbps.

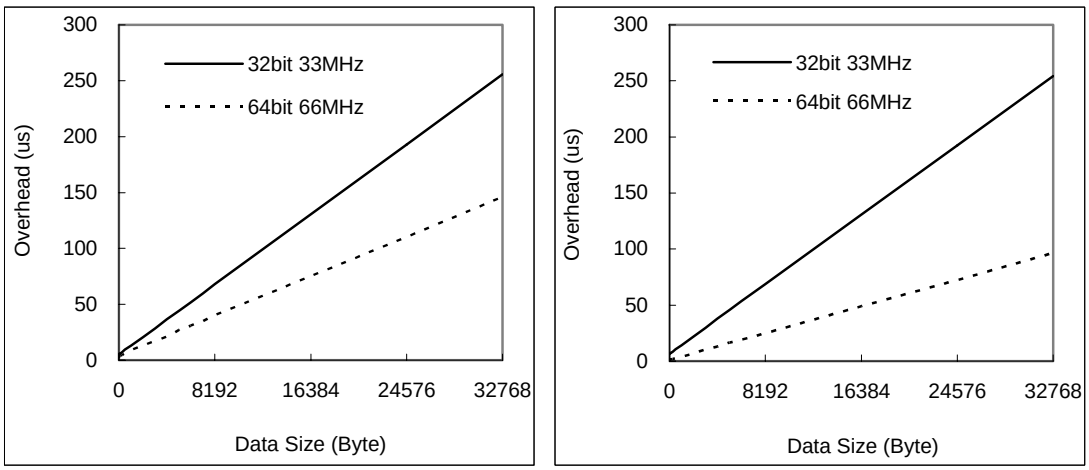

Figure 13 DMA overheads on 32bit 33MHz and 64bit 66MHz PCI buses: (a) sender and (b) receiver

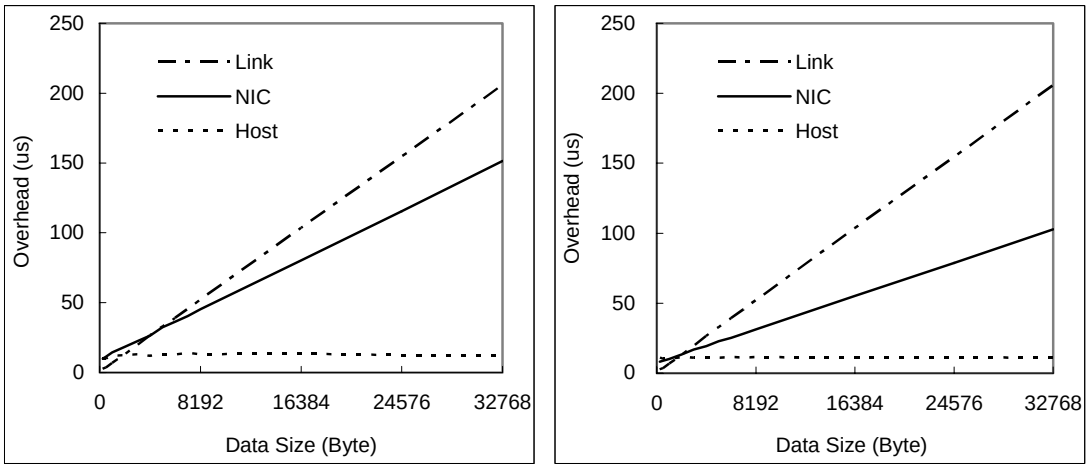

Figure 14 Comparison of link, NIC, and host overheads: (a) sender and (b) receiver

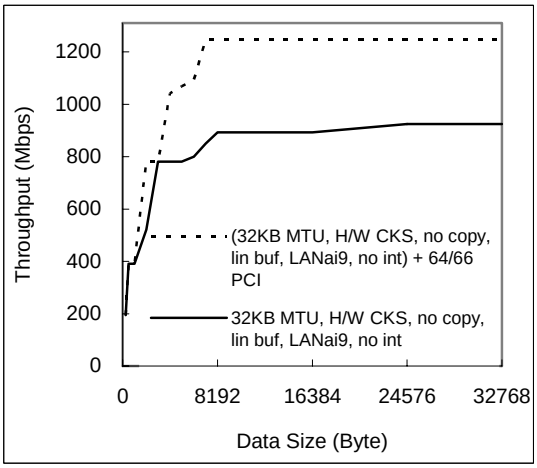

Figure 15 Throughput comparison of UDP/IP on 32bit 33MHz and 64bit 66MHz PCI platforms

An unexpected result in Figure 13(a) is that the DMA overhead is not reduced as much as we would expect. We have expected that the DMA overhead can be reduced by 1/4 in theory because both the width and speed of the PCI bus is doubled. The only other thing that can affect the DMA overhead except the width and speed of PCI bus is the PCI chipset. So we measured the DMA overhead on various PCI chipsets to see the effect. Experimental results in Figure 16 are measured on 32bit 33MHz PCI systems, and chipsets listed are Intel PCI chipsets. We can observe that the DMA overhead varies by the chipset although the width and speed

of the PCI bus is fixed. In addition, chipsets such as 430VX and 440FX show asymmetric performance on the sender and receiver like AMD 762 chipset in Figure 13. As a result, we believe that the unexpected result shown in Figure 13(a) is due to the inefficient chipset. The AMD 762 chipset is sufficient for the 1Gbps performance but not for over 2Gbps as will be discussed in Section 5.

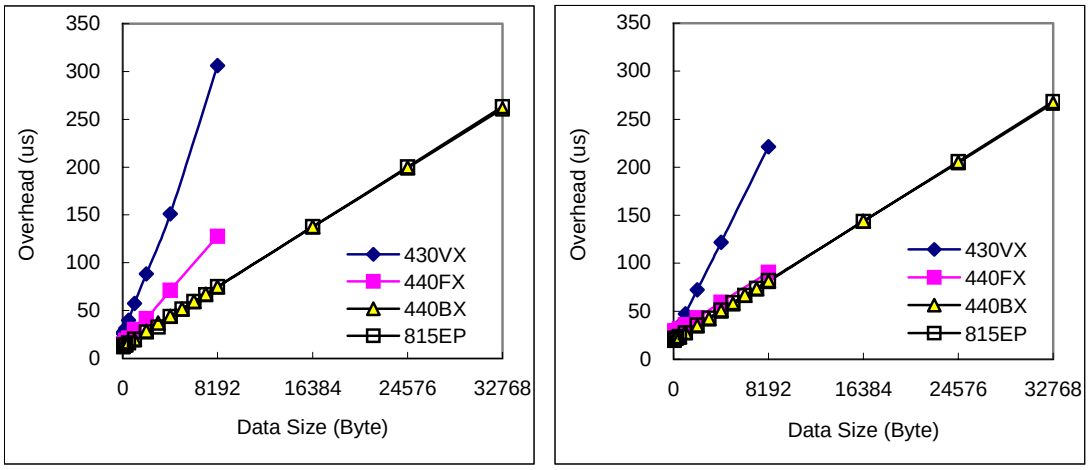

Figure 16 DMA overhead depending on the PCI chipset: (a) sender and (b) receiver

5. Link Overhead: Physical Layer

Finally, this section analyzes the link overhead of the physical layer, which is decided by the physical network media. To reduce the link overhead, we switched the Myrinet NIC and cable to M3F-PCI64C and Myrinet-2000 fiber cable, respectively. Myrinet-2000 fiber can provide 2Gbps in each direction, which is about 1.6 times higher than Myrinet-1280 of maximum 1.28Gbps bandwidth (i.e., the configuration of previous sections).

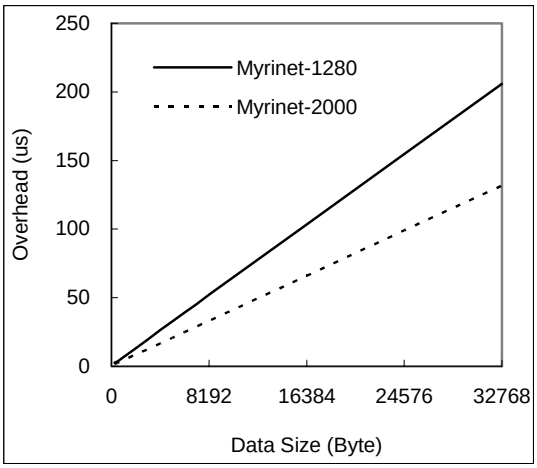

Figure 17 Link overhead on Myrinet-1280 and Myrinet-2000 links

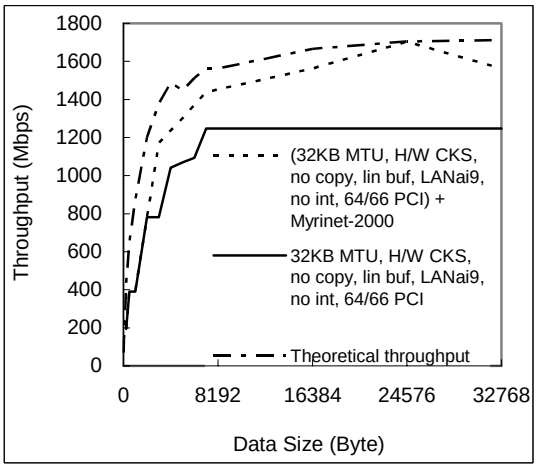

Figure 18 Throughput comparison of UDP/IP on Myrinet-1280 and Myrinet-2000 links.

Figure 17 compares the link overhead of Myrinet-2000 with that of Myrinet-1280. The figure shows that the link overhead is reduced by about 64%. Figure 18 shows the throughput improvement on Myrinet-2000. The reason why the throughput on Myrinet-2000 is the same as that on Myrinet-1280 up to 2KB is that the dominant overhead is not the link overhead but the NIC overhead for smaller data sizes than 2KB as shown in Figure 14. We can see that the throughput on Myrinet-2000 link reaches 1.66Gbps at 24KB, which is 83% of

the physical bandwidth provided by Myrinet-2000. As we have discussed in Subsection 4.3, the physical bandwidth of 2Gbps cannot be fully utilized due to the limitation of the PCI chipset on the experimental systems. *Theoretical throughput* in Figure 18 represents the theoretical maximum throughput that the employed PCI system can provide. It is to be noted that the improved UDP/IP achieves near theoretical maximum throughput by 99%.

6. Discussion

Though in this paper we study on the network throughput using Myrinet-based systems, the suggestions and the measurement results can be generalized easily and applicable to other interconnects. In this context, we would like to discuss some more issues that are not mentioned in the previous sections. Specifically we discuss: i) incorporation of zero-copy mechanism into the TCP/IP stacks, ii) network throughput over 10Gbps interconnects, and iii) recent developments in TCP/IP implementations.

6.1 Incorporation of Zero-Copy Mechanism into the TCP/IP Stacks

As will be discussed in Section 7, many researchers have suggested several zero-copy mechanisms. However, these mechanisms are not still incorporated in the most of TCP/IP implementations. This is mainly due to their implementation complexity and mismatch with the BSD sockets interface's semantics. Since we are dealing with UDP in this paper, we can avoid troublesome cases but the zero-copy mechanisms bring tricky issues with TCP. For example, in the zero-copy TCP transmission, the user buffer on the sender should be locked until the acknowledgement is received. Some zero-copy mechanisms that block the access on the

buffer may hamper the progress of execution. Other mechanisms may result in the copy-on-write, which leads a worse performance than the traditional copy-based transmission. Thus on the high delay or lossy network link, the zero-copy TCP transmission may not be a good idea. The similar issues are on the receiver side.

To overcome such limitations of the traditional sockets interface, the asynchronous sockets and RDMA over IP [48] have been proposed. Windows provides the asynchronous sockets interface in WinSock. RDMA over IP is a new protocol layer over TCP with new asynchronous communication API, which enables the zero-copy transmission over TCP/IP. Based on the wide acceptance of these approaches, we believe that the asynchronous communication can fully exploit the zero-copy transmission without side effects; however, instead of introducing new interface, extending existing sockets interface might be better approach for existing applications. The asynchronous communication interface also makes the implementation issues of the copy avoidance techniques simple. For instance, with asynchronous communication interface, a zero-copy implementation does not need to consider the copy-on-write case because the application should not touch the user buffer before the completion of communication on the buffer by its definition.

6.2 Network Throughput over 10Gbps Interconnects

In Sections 3, 4, and 5, we have measured the performance with the Myrinet NIC model that supports the firmware source code and its development tools, such as cross compiler and debugger, and the workstations that is available when the NIC starts to be available at the market. The basic idea behind such system setup is to see the performance on the contemporary system setup when the new interconnect becomes available though achieving 1 or 2Gbps network throughput is not a difficult issue any more with the state-of-the-art system architecture.

Then an important question is how much the performance results presented in this paper are applicable to the modern interconnect systems such as 10 Gigabit Ethernet [26] and InfiniBand [25]. Especially it is interesting to see whether the traditional TCP/IP implementation running on a fast processor still far underutilizes the bandwidth provided by emerging interconnects. Balaji et al. [46, 47] have measured the TCP/IP throughput over InfiniBand and 10 Gigabit Ethernet, respectively, in which we can observe that the TCP/IP utilizes less than 20% of 10Gbps. For example, Figure 19 shows the TCP/IP throughput over InfiniBand with the Intel Xeon 2.4GHz processor based systems. It is to be noted that the reason of such low throughput is the high overheads of the protocol stacks. Thus a faster processor cannot resolve simply the protocol stacks' overheads. A rough estimation of the processor processing speed required to handle one bit per second of TCP/IP packet is one hertz [34, 49]. Thus we need a 20GHz processor to handle the TCP/IP traffic over a full-duplex 10Gbps network link. This suggests that the performance gap between the processor speed and the network bandwidth still exists and leads the very low performance of TCP/IP even on the up-to-date systems.

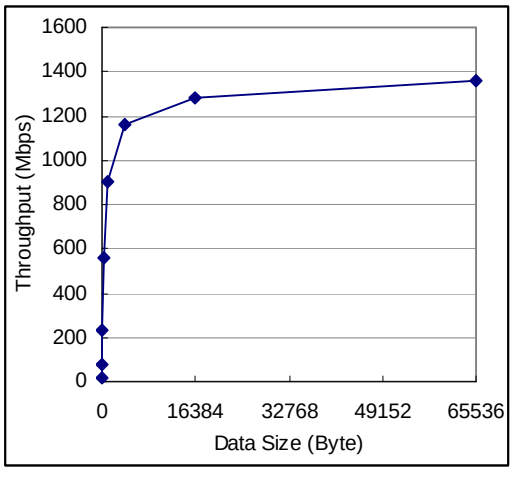

Figure 19 TCP/IP Throughput over InfiniBand (Courtesy of Ref. [46])

6.3 Recent Developments in TCP/IP Implementations

Recently there have been developments in operating systems with respect to the efficient TCP/IP implementation. For instance, the latest version of Linux avoids the copy operation during the defragmentation discussed in Section 3.1. Instead copying, the IP layer manages the linked list of the fragments and passes the head pointer of the list to the transport layer. This will definitely help to increase the throughput for medium and large messages as we already have shown in Figure 2.

The NICs that support the hardware checksum computation become available today. To support such NICs and exploit their advantage, the operating system has to be aware about the feature. Linux have expanded the *sk_buff* structure to let the protocol stacks know if the NIC perform the checksum computation. However, performing the checksum computation on NIC does not guarantee always better performance. There can be three different approaches to implement the checksum computation on NIC: i) by firmware, iii) by an ASIC, and ii) by DMA engine. If the firmware performs the checksum computation, since the NIC processor is much slower than the host processor, it will give worse performance. On the other hand, if a special hardware only for the checksum computation does, it may improve the performance but the overhead for the computation will be still there. Thus, since its logic is not complicate, it might be better to integrate the hardware with the DMA engine as discussed in Section 3.2.

To reduce the number of interrupts, Linux introduces NAPI recently. The basic idea of NAPI is that, if an interrupt is occurred due to a newly arrived packet, the interrupt handler processes more packets in the arrival queue if there are without additional interrupts. In this manner network throughput can be improved. As we have discussed in Section 2.1, the Myrinet device driver already implemented such idea. Also we have shown the impact of reducing number of interrupts in Section 4.2.

7. Related Work

The protocol overheads in the system are divided into two groups: the overheads that increase in proportion to the packet size in byte and the overheads that scale per packet. There have been significant researches on the reduction of the per-byte overhead of the transport layer such as data copy and checksum computation. To remove the data copy between the user and kernel buffers, the memory mapping mechanism has been widely used [8, 10, 12, 13, 14, 23]. There are also hardware-assistant zero-copy approaches [39, 40] in which the NIC has a multi-port memory that can be accessed from host directly. A similar but software approach has been introduced in [21]. The copy overhead during the defragmentation at the network layer has been addressed by Kurmann et al. [42]. Many researchers also have tried to reduce the checksum computation overhead. In [11, 22], researchers have suggested code optimization for checksum computation. To maximize the cache effect, it has been proposed to perform the checksum computation during the copy operation [37]. Utilizing the DMA engine on the NIC is also suggested in [23] though this approach is not compatible with existing protocol stack. To reduce the per-packet overheads, header prediction [37] and efficient PCB lookup [44, 45] have been proposed, which are incorporated in most of modern operating system implementations. Also Massalin and Pu [43] have optimized the I/O code execution time with the kernel code synthesis mechanism. Though the researchers have showed prominent performance improvements by reducing the per-byte or per-packet overheads, these are partial optimizations due to the lack of considerations for multiple protocol layers.

There are some works focusing on the NIC overhead. Shivam et al. [24] and Willmann et al. [36] have showed that the multi-CPU NIC can also improve the NIC's per-packet processing performance. They try to distribute the major steps of send and receive paths to achieve a balance of work on the multiple processors.

The research by Feng et al. [15] has considered several layers to optimize TCP on 10-gigabit Ethernet but they have focused on tuning of given parameters or options. In addition, Smith and Traw [41], Chase et al. [14], and Jin et al. [32] have also tried to reduce the overheads over multiple layers. Although the researchers have considered multiple layers, they do not show the effect of each layer because they are concerned with the final throughput achieved without detailed analysis of the overhead of each layer. Accordingly, they also do not reveal the dependency characteristics between the protocol overheads. Partridge and Pink [9] and Wolman et al. [38] have analyzed and reduced the protocol overheads in a stepwise manner. However, their focus is the latency of only host side while this paper studies NIC and link overheads as well.

Overall, our work is differentiated in terms of considering full spectrum of protocol layers in an integrated manner and revealing the dependencies between the protocol overheads through actual implementation and detailed analysis.

8. Conclusions

In order to elucidate the impact of protocol overheads upon the high-speed network throughput, we thoroughly analyze the critical protocol overheads of UDP/IP over Myrinet and remove them step by step. We deal with the protocol overheads from the transport layer to the physical layer by dividing protocol overheads into the host, NIC, and link overheads. We present the improvement of throughput in each step so that we clearly demonstrate how much bandwidth is wasted by each overhead and what factors have to be considered for a high-speed network protocol.

Our study starts with the analysis of the host overhead such as the (de)fragmentation of the network layer, checksum computation, and data copy of the transport layer. We avoid the (de)fragmentation of IP by a large

MTU size. And we implement the hardware checksum and zero-copy mechanisms to remove the checksum computation and data copy overheads from UDP. A notable result is that we can significantly improve the throughput by removing both checksum computation and data copy. The reason is that the first operation that scans received data causes a large overhead because of the lack of the cache effect. Therefore, we can obtain a high throughput when we remove all data touching operations.

Next, we focus on the NIC overhead such as per-DMA, per-packet, and per-byte overheads of the data link layer. The per-DMA overhead is the DMA initialization overhead of which multiple occurrences are induced by the zero-copy UDP. We prevent the multiple DMA initializations with the physically linear user buffer. In addition, we show how the fast processor on NIC affects the per-packet overhead. Also we show that the per-byte overhead of the NIC (i.e., DMA overhead) is decided by the PCI system. Our experimental results reveal that not only the physical width and speed of the PCI bus but also the implementation of the PCI chipset influences the per-byte overhead largely. An interesting observation is that the zero-copy implementation affects the NIC overheads such as DMA initialization and destination buffer searching time to a great extent.

Finally, we deal with the link overhead of the physical layer. The high-speed link reduces the overhead for the data movement between the NIC and physical network. In order to analyze the link overhead, we compare the link overheads on Myrinet-1280 and Myrinet-2000 links.

The performance measurements have showed that the improved UDP/IP achieves 926Mbps on a 32bit 33MHz PCI platform with Myrinet-1280, 1.22Gbps on a 64bit 66MHz PCI platform with Myrinet-1280, and 1.66Gbps on a 64bit 66MHz PCI platform with Myrinet-2000. As a result, the improved UDP/IP have utilized more than 95% of the maximum network throughput the experimental systems can provide.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. –K. Su, "Myrinet -- A Gigabit-per-Second Local-Area Network," IEEE-Micro, Vol. 15, No. 1, pp. 29-36, February 1995.

[2] Gigabit Ethernet Alliance, IEEE 802.3z. The Emerging Gigabit Ethernet Standard, 1997.

[3] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW Team, "A Case for Networks of Workstations: NOW," IEEE Micro, February 1995.

[4] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, CA, 1999.

[5] D. Anderson, J. Chase, S. Gadde, A. Gallatin, K. Yocum, and M. Feeley, "Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet," Proceedings of the 1998 USENIX Technical Conference, June 1998.

[6] D. M. Halstead, B. Bode, D. Turner, and V. Lewis, "Giga-Plant Scalable Cluster," Proceedings of the USENIX Extreme Linux Technical Conference, pp 10-15, June 1999.

[7] E. Arnould, F. Bitz, E. Cooper, H.T. Kung, R. Sansom and P. Steenkiste, "The Design of Nectar: a Network Backplane for Heterogeneous Multicomputers," Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 205-216, April 1989.

[8] P. Druschel and L. L. Peterson, "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility," Proceedings of 14th ACM SOSP, pp. 189-202, 1993.

[9] C. Partridge and S. Pink, "A faster UDP," IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 429-440, August 1993.

[10] H. J. Chu, "Zero-Copy TCP in Solaris," 1996 Winter USENIX, 1996.

[11] J. Kay and J. Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP," IEEE/ACM Transactions on Networking, Vol. 4, No. 6, pp. 817-828, December 1996.

[12] J. C. Brustoloni, P. Steenkiste, "Copy Emulation in Checksummed, Multiple-Packet Communication," IEEE Infocom '97, April 1997.

[13] A. Barak, I. Gilderman, I. Metrik, "Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN," Computer Communications, Vol. 22, pp. 989-997, 1999.

[14] J. Chase, A. Gallatin, and K. Yocum, "End-System Optimizations for High-Speed TCP," IEEE Communications Magazine, Vol. 39, No. 4, pp. 68-75, April 2001.

[15] W. –C. Feng, J. Hurwitz, H. Newman, S. Ravot, R. L. Cottrell, O. Martin, F. Coccetti, C. Jin, X. Wei, and S. Low, "Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters, and Grids: A Case

Study," Proceedings of SC2003: High-Performance Networking and Computing Conference, November 2003.

[16] Myricom Inc., Myrinet User's Guide, http://www.myri.com, 1996.

[17] USNA, TTCP: A test of TCP and UDP Performance, December 1984.

[18] Myricom Inc., The GM Message Passing System, http://www.myri.com, January 2000.

[19] Myricom Inc., LANai 4, http://www.myri.com, February 1999.

[20] Myricom Inc., PCI64 Programmer's Documentation, http://www.myri.com, May 2001.

[21] C. Yoo, H. –W. Jin, and S. –C. Kwon, "Asynchronous UDP," IEICE Transactions on Communications, Vol. E84-B, No. 12, pp. 3243-3251, December 2001.

[22] R. Braden, D. Borman, and C. Partridge, "Computing the Internet Checksum," IETF Networking Working Group Request for Comments: 1071, September 1988.

[23] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds," Proceedings of 1999 USENIX Technical Conference (Freenix Track), June 1999.

[24] P. Shivam, P. Wyckoff, and D. K. Panda, "Can User Level Protocols Take Advantage of Multi-CPU NICs?," Proceedings of International Parallel and Distributed Processing Symposium (IPDPS '02), April 2002.

[25] InfiniBand Trade Association, http://www.infinibandta.org.

[26] IEEE, IEEE Std 802.3ae-2002, Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10Gbps Operation, August 2002.

[27] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," Proceedings of 15th ACM SOSP, pp. 40-53, December 1995.

[28] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, A. M. Berry, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," IEEE Micro, Vol. 8, pp. 66-76, March-April 1998.

[29] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," Proceedings of SC95, 1995.

[30] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high performance networking on myrinet," Proceedings of the International Parallel Processing Symposium Workshop on Personal Computer Based Networks of Workstations, 1998.

[31] H. Shah, D. Minturn, A. Foong, G. McAlpine, R. Madukkarumukumana, and G. Regnier, "CSP: A Novel System Architecture for Scalable Internet and Communication Services," Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS01), 2001.

[32] H. –W. Jin, P. Balaji, C. Yoo, J.-Y. Choi, and D. K. Panda, “Exploiting NIC Architectural Support for Enhancing IP based Protocols on High Performance Networks,” Journal of Parallel and Distributed Computing, Vol. 65, No. 11, pp. 1348-1365, November 2005.

[33] H. –W. Jin, C. Yoo, and J. –Y. Choi, “Firmware-Level Latency Analysis on a Gigabit Network,” The Journal of Supercomputing, Vol. 26, No. 1, pp. 59-75, August 2003.

[34] 10 Gigabit Ethernet Alliance, “Introduction to TCP/IP Offload Engine (TOE),” Version 1.0, April 2002.

[35] C. M. Woodside and J. R. Montealegre, “The Effect of Buffering Strategies on Protocol Execution Performance,” IEEE Transactions on Communications, Vol. 37, No. 6, June 1989.

[36] P. Willmann, H. -Y. Kim, S. Rixner, and V. Pai, “An Efficient Programmable 10 Gigabit Ethernet Network Interface Card,” Proceedings of HPCA-11, February 2005.

[37] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, “An Analysis of TCP Processing Overhead,” IEEE Communications Magazine, Vol. 27, No. 6, pp.23-29, June 1989.

[38] A. Wolman, G. Voelker, and C. Thekkath, “Latency Analysis of TCP on an ATM Network,” 1994 Winter USENIX, January 1994.

[39] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, “Afterburner,” IEEE Network, Vol. 7, No. 4, pp. 36-43, July 1993.

[40] D. Banks and M. Prudence, “A High-Performance Network Architecture for a PA-RISC Workstation,” IEEE Journal on Selected Areas in Communication, Vol. 11, No. 2, pp. 191-202, February 1993.

[41] J. Smith and C. Traw, “Giving Applications Access to Gb/s Networking,” IEEE Network, Vol. 7, No. 4, pp. 44-52, July 1993.

[42] C. Kurmann, M. Muller, F. Rauch, and T. Stricker, “Speculative Defragmentation A Technique to Improve the Communication Software Efficiency for Gigabit Ethernet,” Proceedings of 9th IEEE Symposium on High Performance Distributed Computing (HPDC), August 2000.

[43] H. Massalin and C. Pu, “Threads and Input/Output in the Synthesis Kernel,” Proceedings of ACM Symposium on Operating Systems Principles (SOSP), pp. 191-201, 1989.

[44] J. Mogul, “Network Locality at the Scale of Processors,” Proceedings of ACM SIGCOMM ’91, September 1991.

[45] P. McKenney and K. Dove, “Efficient Demultiplexing of Incoming TCP Packets,” Proceedings of ACM SIGCOMM ’92, August 1992.

[46] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda, “Sockets Direct Procotol over InfiniBand in Clusters: Is it Beneficial?” IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 04), March 2004.

[47] P. Balaji, H. V. Shah and D. K. Panda, “Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck,” Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2004). September 2004.

[48] A. Romanow and S. Bailey, “An Overview of RDMA over IP,” Proceedings on First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet2003), February 2003.

[49] S. Makineni and R. Iyer, “Architectural Characterization of TCP/IP Packet Processing on the Pentium M Microprocessor,” Proceedings of 10th International Symposium on High Performance Computer Architecture (HPCA-10), pp. 152-161, February 2004.