Implementation and Analysis of Asynchronous UDP over Myrinet

Jee-Hyun Park, Hyun-Wook Jin, and Chuck Yoo

▶ **Technical area :** High-speed Network / LAN Protocol

▶ **Affiliation :** Department of Computer Science

▶ **Keywords :** Asynchronous UDP, UDP/IP, Myrinet, Copy Overhead, DMA, Device Driver, anode

▶ **Address :** 1, 5ka, Anam-Dong, Sungbuk-Ku, Seoul 136-701, Korea

▶ **Telephone Number :** +82-2-922-6341

▶ **Fax Number :** +82-2-922-6341

▶ **E-mail Address :** Jee-Hyun Park(jhpark@os.korea.ac.kr)

Hyun-Wook Kim(hwjin@os.korea.ac.kr)

Chuck Yoo(hxy@os.korea.ac.kr)

Implementation and Analysis of Asynchronous UDP over Myrinet

Jee-Hyun Park, Hyun-Wook Jin, and Chuck Yoo

Department of Computer Science

Korea University

{jhpark, hwjin, hxy}@os.korea.ac.kr

Abstract

To achieve high-speed communication, it is well-known that the per-byte overhead needs to be minimized. The advances of gigabit networks highlight the importance of so-called light-weight primitives in order to utilize the available bandwidth. Asynchronous UDP [1] is one of such primitives proposed recently.

This paper evaluates Asynchronous UDP over Myrinet. We describe the design and implementation on Linux and Myrinet MCP. The results of performance measurements show that Asynchronous UDP is faster than the optimized UDP of Linux up to 45 percents.

1. Introduction

Gigabit networks (e.g., Myrinet [2] and Gigabit Ethernet) have entered into the mainstream, but the perceived throughput at the application level has not sufficiently increased. Many studies diagnose that the overheads in legacy network protocol prevent the bandwidth of network from being fully utilized [3][4][5]. Particularly, as multimedia applications demand a large amount of network bandwidth, the per-byte overheads such as copy operation and checksum calculation are regarded as a major cause of the less - than expected performance over high-speed network.

Therefore, many researchers have tried to reduce the per-byte overhead in order to take advantage of the gigabit bandwidth. Trapeze messaging system [6] and Copy emulation [7] suggest a software solution based on the idea of page-remapping [8]. However, the weakness of page-remapping is copy-on-write (COW). When the data buffer is reused, COW is unavoidable [8]. Because COW has as much (or more) cost as data copy, their performance may drop sharply if COW occurs frequently.

Asynchronous UDP [1], which is derived from traditional UDP, also is a software solution, but it does not have constraints such as COW. Furthermore, it is integrated with the socket API so that existing applications can easily use Asynchronous UDP without any special library. [1] also shows that Asynchronous UDP utilizes 94% of the bandwidth of ATM.

A contribution of [1] is that a variant of traditional protocol achieves a throughput close to the maximum bandwidth of high-speed network. The goal of this paper is to apply Asynchronous UDP to Myrinet, the fastest network available today. We show the implementation and analyze the performance.

The rest of this paper is organized as follows. Section 2 explains the background for Asynchronous UDP and Myrinet. Section 3 details the design and implementation of Asynchronous UDP over Myrinet. The performance measurements and analysis are shown in Section 4. Section 5 concludes this paper.

2. Background

2.1 Asynchronous UDP

Asynchronous UDP is introduced in [1]. We briefly summarize it here as related to this paper. For more details on Asynchronous UDP, refer to [1].

The semantics of *send()* and *recv()* system calls of UNIX guarantees that the data send or receive is completed when the calls return. We call this synchronous mechanism because the data buffer can be reused once the call returns. This mechanism gives the application much flexibility, but it comes with the expensive copy operation because kernel has to copy the data in user buffer into kernel buffer or vice versa.

Asynchronous UDP extracts the synchronous property from the traditional UDP and eliminates the redundant data copy. In the send side, Asynchronous UDP just builds a header for data in the kernel and gives network interface card (NIC) the descriptor about the header and user data. Then, it returns immediately. The data on user buffer is directly moved into network buffer via DMA (Direct Memory Access).

The receive side of Asynchronous UDP is more complicated because the read request can be issued either before or after a packet arrives. When a packet arrives earlier than the request, Asynchronous UDP does what the traditional UDP normally performs – copies the packet into kernel buffer. However, when an anticipated packet has not yet arrived, Asynchronous UDP records the location of user buffer and returns immediately. Since the Asynchronous *recv()* does not sleep, the caller continues to do other computation or to request another receiving operation. When a packet arrives, kernel first finds the corresponding user buffer and DMAs it directly to user buffer. Figure 1 depicts the difference between traditional UDP and Asynchronous UDP.

The performance measurements show that Asynchronous UDP utilizes 94.2% of the physical bandwidth of ATM, and it is faster by 65% than traditional UDP on ATM. Moreover, Asynchronous UDP requires only 1/4 CPU resource compared with the traditional UDP.

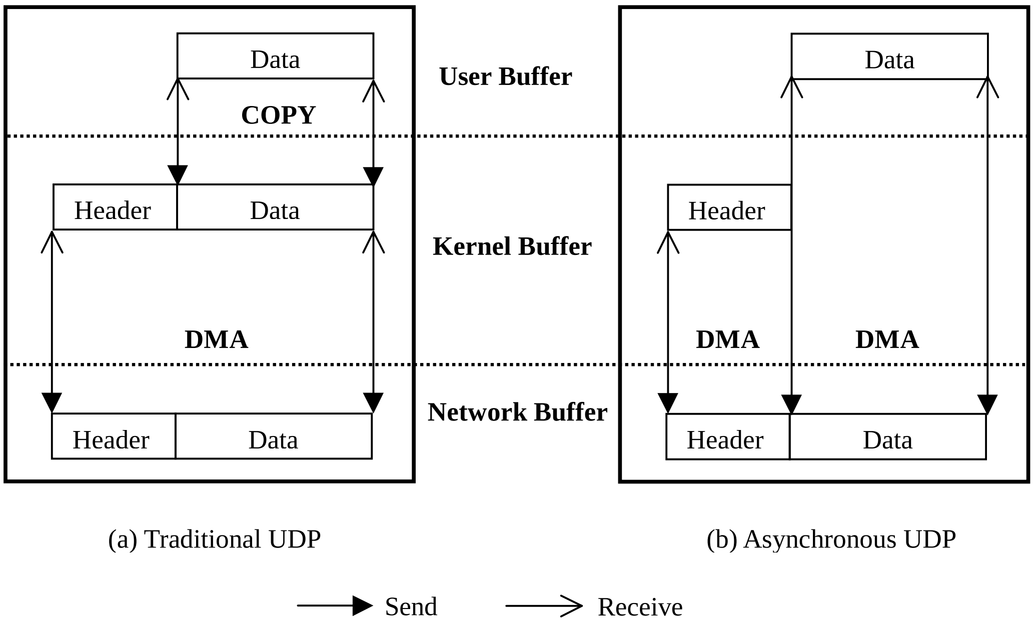

Figure 1. Traditional UDP vs. Asynchronous UDP

2.2 Myrinet

Figure 2 shows the protocol stack over Myrinet. The most conspicuous feature of Figure 2 is the MCP (Myrinet Control Program). MCP is the firmware running on Myrinet NIC and mainly responsible for the data movement from kernel area to physical network and vice versa.

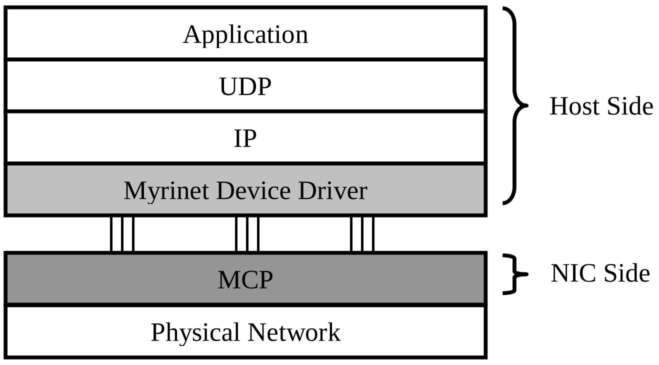

Figure 2. Protocol stack over Myrinet

As shown in Figure 2, the Myrinet device driver takes charge of mediating packets between UDP/IP and MCP. However, because the Myrinet device driver runs on the host side and MCP does on Myrinet NIC, there should be some way to communicate send and receive requests. For this, MCP allocates a memory block called channel [9]. Myrinet device driver can access the channel by memory mapping. MCP supports several channels, but the first channel is reserved for traditional protocols such as TCP and UDP. Each channel has three queues including a send queue and a receive queue.

The send queue contains gathers, and each gather has the information about the data in kernel buffer. In handling the traditional UDP, kernel copies user data into the kernel buffer (Figure 3 ①), and the Myrinet device driver makes a gather and puts it to the send queue in channel 0. The gather points to the kernel buffer as shown in Figure 3. Then, MCP pops the gather and initiates DMA (Figure 3 ②). When DMA completes, MCP sends the data over Myrinet (Figure 3 ③).

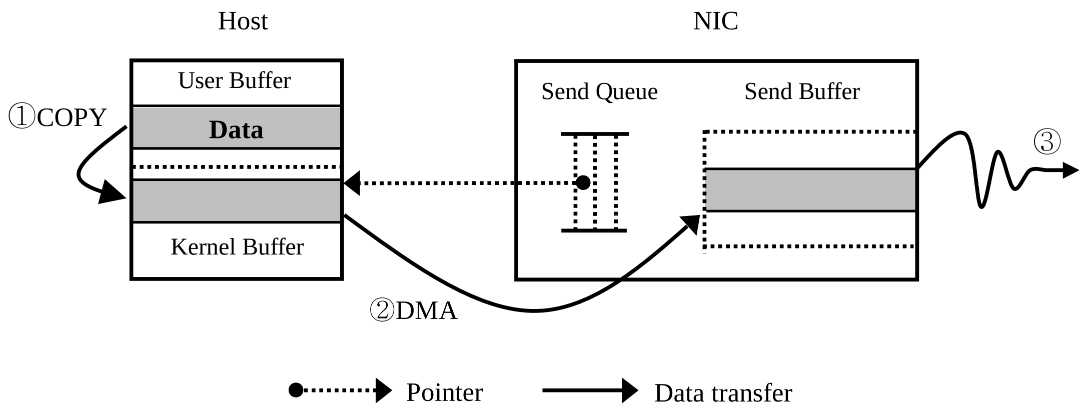

Figure 3. Traditional data send over Myrinet

The receive queue manages scatters. A scatter is similar to a gather, but it is used for receiving data and holds the address of kernel buffer. When Myrinet is initialized, the Myrinet device driver allocates some memory in kernel and splits it into several chucks. Then, Myrinet device driver makes scatters, each of which points to a memory chuck in the kernel and inserts them to the receive queue. When MCP receives a packet from the Myrinet (Figure 4 ①), MCP takes out a scatter on the receive queue and copies the packet into the kernel buffer described in the scatter via DMA (Figure 4 ②). After DMA completes, MCP generates an interrupt to notify the arrival of a packet to the kernel (Figure 4 ③). If an application already issued *recv* (), kernel immediately delivers the packet to the application by copying it (Figure 4 ④).

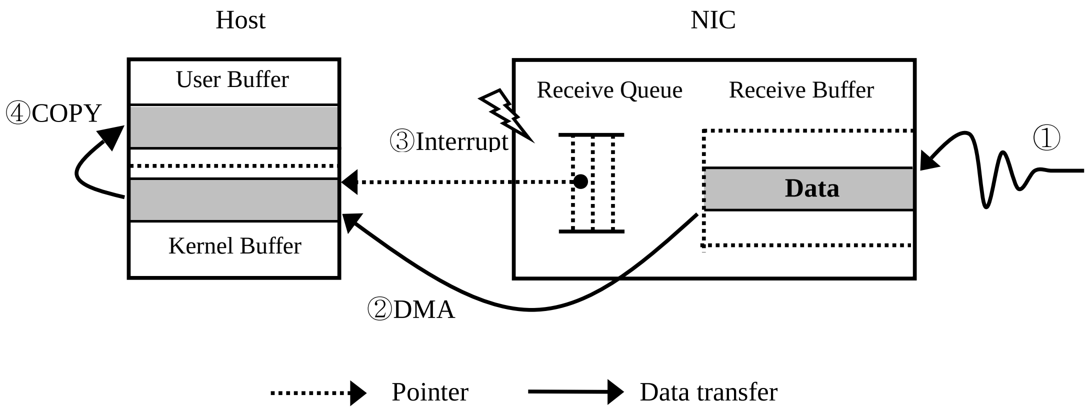

Figure 4. Traditional data receive over Myrinet

3. Asynchronous UDP over Myrinet

Unlike page-remapping, Asynchronous UDP does not depend on the virtual memory system. It removes data copy between user buffer and kernel by using DMA directly to and from Myrinet NIC.

3.1 Asynchronous *send()*

The key idea is not to copy the user data to the kernel buffer but to leave the data in the user space. For this, the structure of gather needs to be changed. Remember that the traditional UDP puts header and user data on the same kernel buffer, and Myrinet device driver makes a gather that points to the kernel buffer. However, Asynchronous UDP builds only header in the kernel buffer and leaves the data in user buffer. So, we need to build a gather that points to at least two locations. One is the kernel buffer address where the header for user data lies. The other is the user buffer address. After making a gather as shown in Figure 5, the Myrinet device driver puts it on the send queue and knocks MCP to request the data send. If DMA is available, MCP immediately starts DMA for the header and for the user data and makes them into one packet in the send buffer inside Myrinet NIC (Figure 5 ①). Then, MCP sends the packet over Myrinet (Figure 5 ②).

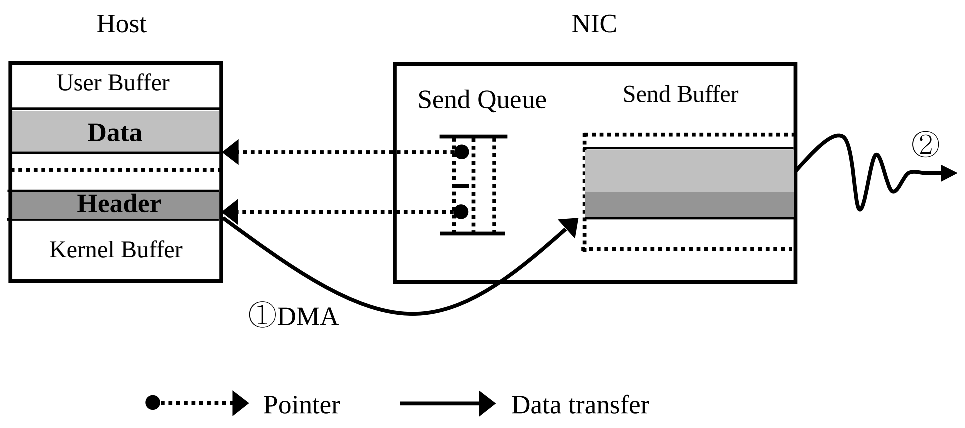

Figure 5. Asynchronous data send over Myrinet

3.2 Asynchronous *recv()*

When a packet arrives, there are two possible cases depending on whether the asynchronous read request is made. If the read request is not made yet, the packet needs to be copied into kernel. This is the same as the processing of the traditional UDP read request. If the request is already made, we need a structure in the kernel that stores information such as data buffer address so that MCP can move the packet directly to the data buffer.

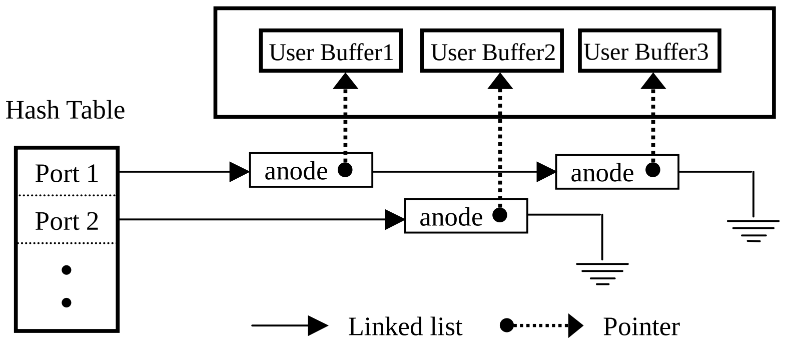

Figure 6. *anode* structure

We call the structure *anode*. Figure 6 illustrates how *anode* is designed - it is hashed by the port number of the application and points to the data buffer in user space. *Anode*s can naturally be allocated in the Myrinet NIC memory because MCP has to handle the arrived packet. The kernel in the host can access *anode*s in the NIC memory by mapping the NIC memory. However, we decide to allocate *anode*s in the kernel memory. The reason is because the NIC memory has the limited size so that a large number of asynchronous read requests can exhaust the NIC memory.

When a packet arrives, the kernel searches *anode*s to find the owner of the packet. And the kernel makes a scatter from the *anode* and puts it into the receive queue. Now, a challenge is how to inform the kernel of the arrival of packets. MCP can detect the arrival easily, but since the kernel is running on the host, it needs to be informed. We choose to use interrupt. This means that the order of generating interrupts from MCP is changed. In other words, normally MCP generates an interrupt after it moves the arrived packet to kernel memory as shown in Figure 4. To support asynchronous read, MCP generates an interrupt right after a packet arrives (Figure 7 ① and ②). As the interrupt handling, the kernel scans *anode*s and puts a scatter to the receive queue, and then MCP uses the address in the scatter to do DMA directly to the data buffer (Figure 7 ③). When DMA is done, a flag is set so that the kernel can notice the end of DMA.

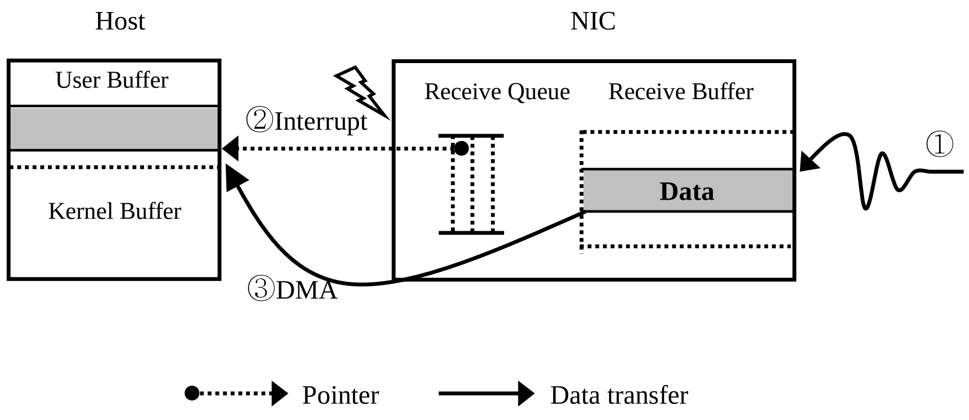

Figure 7. Asynchronous data receive over Myrinet

Another issue we faced is that the receive queue of channel 0 is filled with scatters made by the Myrinet device driver so that we cannot use channel 0. Fortunately, Myrinet supplies other extra channels so that scatters made from *anode*s can be put in the receive queue of a separate channel.

4. Experimental Results

To analyze the performance of Asynchronous UDP over Myrinet, we experimented the following two cases. Each machine is equipped with Intel Pentium processor and 32MB RAM.

- Case 1 : sender (90MHz CPU) is slower than receiver (120MHz CPU).
- Case 2 : sender (120MHz CPU) is as same as receiver (120MHz CPU).

M2F-PCI32C Myrinet NIC is installed to all hosts. Sender and receiver are connected directly with M2F-CB-25 Myrinet LAN cable. The operating system used is Linux (kernel version 2.0.27).

We measured the unidirectional point-to-point throughput using ttcp which is a well-known benchmark program. To support Asynchronous UDP, ttcp is slightly modified. The throughput is measured in three different types of experiments as follows.

- Experiment A : traditional UDP for both the sender and receiver.
- Experiment B : Asynchronous UDP for the sender and traditional UDP for the receiver.
- Experiment C : Asynchronous UDP for both the sender and receiver.

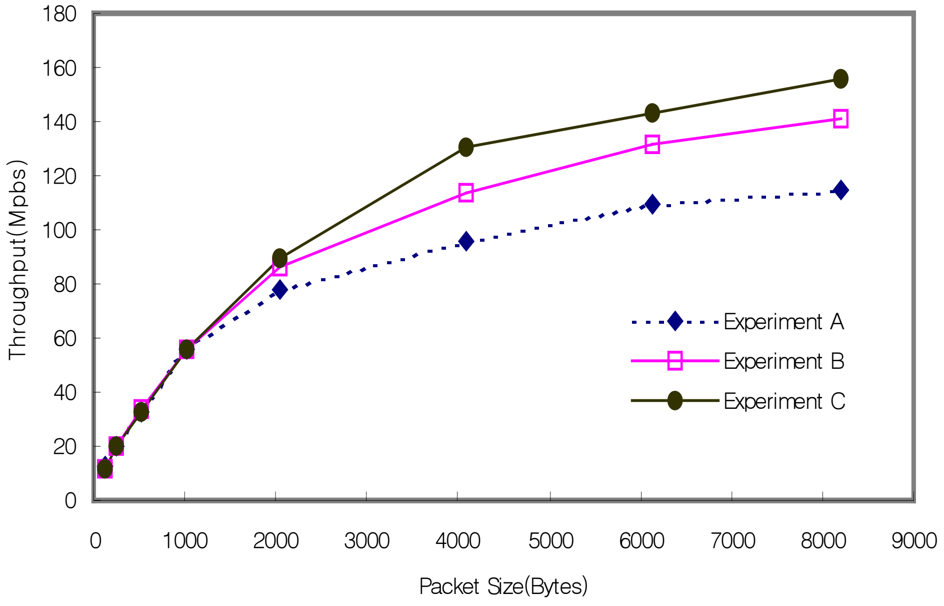

Figure 8. Results of Experiment A, B and C for Case 1

4.1 Case 1

The throughput is determined by two constraints : one is how fast packets arrive at network buffer, the other is how efficiently packets on network buffer are transferred to user buffer. In our experiments, the former strongly relies on the data pumping ability of sender because hardware configurations such as NIC and physical link are the same. The latter is governed by the capability of receiver. Even though the sender pumps the data fast enough, there is no gain in performance if the receiver spends much time in processing it.

Figure 8 shows the results of Experiments A, B and C for Case 1. The throughput of Experiment B is better than that of Experiment A. Considering that both Experiment A and B use traditional UDP to receive a packet, the major reason for the improved performance is the interval of packets sent. Asynchronous UDP used in Experiment B sends data much faster than the traditional UDP, which leads to the higher performance.

Though both Experiment B and C use Asynchronous UDP to send packets, the throughput of Experiment C is better than that of Experiment B. The performance difference is due to the fact that Asynchronous UDP is used in the receiver so that the packet processing in Experiment C is more efficient. Table 1 shows that the throughput of Asynchronous UDP applied to both sender and receiver (Experiment C) is better than traditional UDP by up to 36.2 percents.

Bytes	Experiment A	Experiment C	
	Throughput (Mbps)	Throughput(Mbps)	Improvement(%)
128	12.2	11.7	-4.0
256	19.5	19.5	0.0
512	32.6	32.6	0.0
1024	55.8	55.8	0.0
2048	78.1	89.0	13.9
4096	95.6	130.2	36.2
6144	109.6	143.0	30.5
8192	114.9	156.3	36.0

Table 1. Improvement ratio of Experiment C over A of Case 1

Unfortunately, the throughput of Asynchronous UDP is not better than that of traditional UDP for the packet of 128 bytes. The overheads such as managing *anode* cut down the benefit obtained from removing the copy overhead. However, the throughput continues to increase, as the packet size gets large.

4.2 Case 2

The experiment result of Case 2 is shown in Figure 9. Compared with Case 1, Case 2 shows some interesting result. Experiment A and B have almost the same throughput, unlike Case 1.

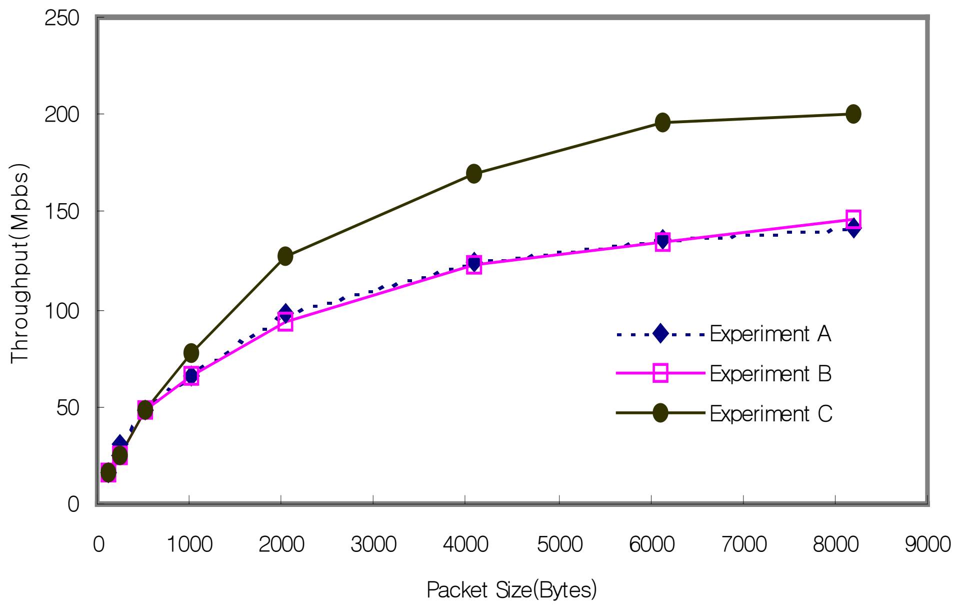

Figure 9. Results of Experiment A, B and C for Case 2

The reason is that the receiver on Experiment A and B is a bottleneck when sender and receiver have the same CPU power. Therefore, Asynchronous UDP in the sender does not make any difference. Experiment C applies Asynchronous UDP to not only sender but also receiver. So, the bottleneck of receiver is eliminated. Thus, Experiment C shows the best performance among three experiments. Table 2 shows that the throughput of Asynchronous UDP is better than that of traditional UDP up to 44.9 percents.

The maximum throughput of Experiment C on Case 1 is 156.6 Mbps and on Case 2 it is 200.5Mbps. So the throughput of Asynchronous UDP on Case 2 is improved by 28.3%, compared with Case 1. Considering that there is no difference between Experiment C of Case 1 and Case 2 except for the CPU speed, CPU is very important on throughput. We believe that the performance of Asynchronous UDP could increase even further if faster CPU is used.

Bytes	Experiment A	Experiment C	
	Throughput(Mbps)	Throughput(Mbps)	Improvement(%)
128	16.3	15.5	-5.0
256	30.9	24.4	0.0
512	48.8	48.8	0.0
1024	65.1	78.1	20.0
2048	97.7	126.5	35.5
4096	124.2	170.1	39.2
6144	136.3	195.3	44.9
8192	142.0	200.5	37.2

Table 2. Improvement ratio of Experiment C over A of Case 2

Conclusion

We present the implementation of Asynchronous UDP over Myrinet and show that it achieves better performance up to 45 percents than the optimized UDP of Linux. The improvement ratio gets bigger with larger packet sizes. In addition to the performance advantage, Asynchronous UDP is an extension of UDP so that applications can use it easily because it can be integrated with the socket API. We believe that this paper demonstrates the potential of Asynchronous UDP as a viable alternative for high-speed communication.

References

[1] Chuck Yoo, Soon-Cheol Kwon, Hyun-Wook Jin, “Asynchronous UDP,” submitted for publication, 1999.

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su, “Myrinet -- A Gigabit-per-Second Local-Area Network,” *IEEE-Micro*, Vol.15, No.1, pp.29-36, February 1995.

[3] David D. Clark, Van Jacobson, John Romkey, Howard Salwen, “An Analysis of TCP Processing Overhead,” IEEE Communication Magazine, pp. 23-29, June 1989.

[4] Hyun-Wook Jin and Chuck Yoo, “Latency Analysis of UDP and BPI on Myrinet,” *Proceedings of 18th International Performance, Computing, and Communication Conference (IPCCC’9)*, pp.185-191, February 1999.

[5] Hyun-Wook Jin, Jee-Hyun Park, Chuck Yoo, “Firmware-Level Latency Analysis on a Gigabit Network,”, submitted for publication, 1999.

[6] Trapeze/IP: TCP/IP at Near-Gigabit Speeds, by Andrew Gallatin, Jeff Chase, and Ken Yocum. 1999 USENIX Technical Conference (Freenix Track). June 1999.

[7] J. Carlos Brustoloni, Peter Steenkiste, "Copy Emulation in Checksummed, Multiple-Packet Communication," IEEE Infocom '97, April 1997.

[8] Chu, Hsiao-keng Jerry, "Zero-Copy TCP in Solaris," 1996 Winter USENIX, 1996.

[9] Myricom Inc., Myrinet User’s Guide, http://www.myri.com:80/scs/documentation/mug/, 1996.