Latency Analysis of UDP and BPI on Myrinet

Hyun-Wook Jin and Chuck Yoo
Department of Computer Science and Engineering
Korea University
{hwjin, hxy}@os.korea.ac.kr

Abstract

High-speed networks such as ATM, Myrinet, and Gigabit Ethernet are available today, and many researchers make efforts to enhance the performance of end-to-end communication on these high-speed networks. One of the efforts is to develop new light-weight communication primitives for high-speed networks. However, the latency of the new primitives has not been characterized thoroughly, partly because existing measurement methodologies do not take into account the features of high-speed networks. Therefore, there are only incomplete comparisons of the new primitives and traditional protocols, and they cannot really prove the usefulness of new primitives. In order to address this issue, this paper suggests a new measurement methodology and uses the methodology to perform a detailed latency analysis of UDP and a light-weight primitive, called BPI, on Myrinet. Our results clearly show the difference of per-byte overhead between BPI and UDP. A surprising result is that BPI is found to be slower than UDP for 4KB or larger data size.

1. Introduction

With the advances in high-speed networks (e.g., ATM, Myrinet [1], and Gigabit Ethernet), network software gets more attention because it plays a key role in taking full advantage of the fast network hardware. However, the current state of network software is not ready to fully utilize the network hardware, and there are many efforts to enhance the performance of network software at the system level.

These efforts can be classified into two basic approaches. The first approach is the optimization of traditional protocols, such as TCP and UDP. This approach generally tries to minimize the per-byte overhead [2] of traditional protocols because the size of data in modern systems (e.g., multimedia systems) is getting larger than that of traditional systems. The second is the development of new light-weight communication primitives. Most of the new primitives, such as FM (Fast Messages) [3], U-Net [4], and Myrinet Software [5], bypass the kernel in order to reduce communication overhead.

The new light-weight communication primitives are less general and portable than traditional protocols but these have the potential to achieve the best performance on a specific network hardware. Although light-weight communication primitives are attractive, these primitives need to prove their usefulness and be compared thoroughly with traditional protocols.

The goal of this paper is to analyze the latency of a light-weight communication primitive in details and to compare it with a traditional protocol. The latency measurement is performed on Myrinet because it is the fastest LAN available today. The measured light-weight communication primitive is BPI [5] that is designed to support high-speed communication on Myrinet. Because BPI does not support reliable delivery, it is compared with UDP rather than TCP. In addition, this paper suggests a new measurement methodology for the latency analysis of high-speed networks. The measurement is performed at the firmware level of NIC (Network Interface Card), which is more advanced than the existing measurements that are performed at the device driver level. As a result, this paper shows what factors influence the performance of a light-weight communication primitive and what should be considered in designing a new light-weight communication primitive.

The rest of this paper is organized as follows. Section 2 describes related works. Section 3 introduces a new measurement methodology to measure the latency on high-speed networks. Section 4 explains Myrinet Software. Section 5 presents experiment environment in details. Sections 6 and 7 show the measurement results and the comparison of BPI and UDP. Finally, the paper concludes in Section 8.

2. Related Works

This research was supported by Institute of Information Technology Assessment under contract C1-98-229.

This section discusses the existing methodologies and their suitability for the measurement of high-speed networks. We also explain how latency analysis has been done.

2.1 Measurement Methodologies

The NICs of high-speed networks, such as Fore ATM [6] and Myrinet, contain an on-board co-processor and firmware that runs on the NIC. The firmware performs various important tasks related to end-to-end communication. For example, it executes header manipulation operations and DMA for data send or receive. In some cases, it also performs multiplexing, network reconfiguration, and so on. Therefore, the NIC firmware of high-speed networks also should be considered as a layer of the protocol stack.

In order to measure the latency, typically the kernel instrumentation (e.g., do_gettimeofday() or `rdtsc` assembly instruction) is used. But the kernel instrumentation is restricted to the measurement of host side latency, and it cannot measure the latency of NIC firmware. That is, the measurement method based on kernel instrumentation alone cannot analyze the full spectrum of the latency of high-speed networks accurately.

One approach to measure the latency of NIC firmware is to use a logic analyzer. But it requires a special hardware board to connect the lines of a logic analyzer to the I/O bus. Therefore, measurement using logic analyzer is very cumbersome and needs hardware expertise.

2.2 Latency Analyses

There have been many latency analyses of traditional protocols. One of the detailed latency analyses of UDP is performed on top of FDDI by [7]. The analysis is focused to the latency of the host side, and it does not reflect the latency of the NIC side although it uses a logic analyzer. Another latency analysis is performed with TCP on ATM [8]. It uses a high-resolution clock on TurboChannel card for the measurement, but it analyzes the latency at the device driver level as in [7]. [9] shows the latency of TCP over Gigabit Ethernet is lower than TCP over Myrinet. However, we believe that they need to measure the detailed latency of the protocol stack to justify their result because the physical bandwidth of Gigabit Ethernet is smaller than that of Myrinet.

The latency analysis for a new light-weight communication primitive has focused on round-trip or one-way latency [3][4], so that such analysis cannot show the detailed latency of each layer or a specific operation. Although [10] shows the latency of U-Net/MM in details by using kernel instrumentation and oscilloscope, there is no comparison with the latency of a traditional protocol, such as TCP or UDP.

In summary, most detailed latency analyses are limited to the traditional protocols and performed at the device driver level, which means they do not consider the effect of the NIC firmware on end-to-end communication. In addition, there is no comparison between the detailed latency of a new light-weight communication primitive and that of a traditional protocol.

3. New Measurement Methodology

In order to take the NIC firmware into account, the first problem to be addressed is how to measure the latency of the host program and NIC firmware by the same clock. There are two clocks: one is the system clock and the other is the NIC clock.

If the system clock is used, NIC has to interrupt the host every time the measurement code of NIC firmware wants to read the clock. Then, interrupt handler reads the system clock and passes the clock value to the NIC firmware via PIO or DMA. By the clock value passed, the NIC firmware can measure its latency. However, in this case, the measurement overhead is too high because it requires interrupts. Furthermore, the measurement overhead in the NIC firmware is significantly larger than that in the host program.

Because the method based on the system clock has the undesired effects, using NIC clock is desirable. This paper suggests a new methodology summarized as follows.

- The NIC clock is mmapped (memory mapped) into the host address space. Now, the reading of mmapped clock from the host side simply becomes a matter of referencing a pointer. One thing to be careful is that the clock should be mmapped into the uncacheable area. If the clock is mmapped into the cacheable area, the latest clock value is not read.
- The measurement code is inserted into the kernel and NIC firmware. By reading the NIC clock through the measurement code, time stamp can be generated from the host and NIC sides. In this manner, we can measure the latency in host and NIC with the same clock.
- Reading the clock is done when the I/O bus is idle. This makes the overhead of reading the mmapped clock ignorable. For example, assume that the latency of DMA needs to be measured. We put the measurement code before and after of DMA operation, so that clock is not read while the I/O bus is busy.

As described, the measurement method based on the NIC clock is suitable for the detailed latency measurement of high-speed networks without additional equipment like

logic analyzer, and the measurement overhead can be minimal. The detailed procedure of applying the new measurement methodology to Myrinet is described in Section 5.

4. Myrinet Software

Figure 1 shows the protocol stack of Myrinet Software. As depicted in the figure, Myrinet Software supports not only light-weight communication primitives such as API and BPI, but also traditional protocols such as UDP. API and BPI allow an application to communicate over the network directly from user space.

BPI offers more flexible interface to an application than API does. For instance, BPI translates the virtual address of data buffer into the physical address for DMA between the host and NIC memory. For the purpose of the address translation, BPI allocates a copy block in the kernel memory and mmaps the area into the user space when BPI is initialized. The copy block is a physically linear area of the host memory, and BPI knows the physical base address of the copy block. On the sender side, BPI copies the data into the copy block and calculates the physical address by the offset from the base address. The receiver side is also handled similarly. BPI uses DMA to/from the copy block without kernel intervention. U-Net also has a similar light-weight primitive.

In contrast, Myrinet API has no interface to translate the address for DMA so that the applications that use API have to handle complicated DMA setup. It means that BPI is the preferred interface for an application. Therefore, this paper measures the latency of BPI rather than API, and BPI is compared with UDP because BPI does not provide reliable delivery.

MCP (Myrinet Control Program) in Figure 1 is the firmware of Myrinet NIC, and it is divided into two layers that are named MCP-host and MCP-net. MCP-host layer covers the data movement between the host and NIC. MCP-net layer is responsible for the data movement between the NIC and network. MCP is executed independently from the host program. Thus, MCP catches send requests from the host program through polling. Furthermore, because MCP-host layer and MCP-net layer work asynchronously, they detect an event that occurs in the other party by polling. We refer readers to [1] and [5] for the details of Myrinet.

5. Experiment Environment

This section describes the experiment environment where our new measurement methodology is implemented.

5.1 Implementation of Measurement Methodology

As clock, the RTC (Real Time Clock) register that is a counter of a rate 2MHz on Myrinet NIC is used. This register can be read easily in MCP. The host program can also plainly read the register because it is mmapped into the host memory in initialization (① in Figure 2). The mmapped RTC is uncached; thus the undesirable cache effect is prevented. Therefore, we can measure the accurate latency with 0.5μs resolution without additional overhead.

The measurement code in MCP writes time stamps into a pre-allocated area in NIC memory (② and ③ in Figure 2). The area can be allocated by pushing down the base address of the stack to the low address. The measurement code in host program writes time stamp into the host memory. When the data transfer is completely finished, the host program gathers (④ in Figure 2) the time stamps from the host and the NIC area to evaluate the result (⑤ in Figure 2). The host program can access the NIC area through the base address of mmapped NIC memory.

For the measurement of UDP, a mechanism that reads the RTC in the kernel is needed. We chose to use an ioctl function that returns a pointer to the RTC. A new system call is also added to pass the time stamps to the application.

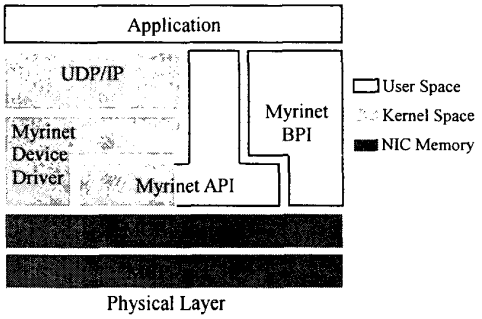

Figure 1. Protocol stack of Myrinet Software

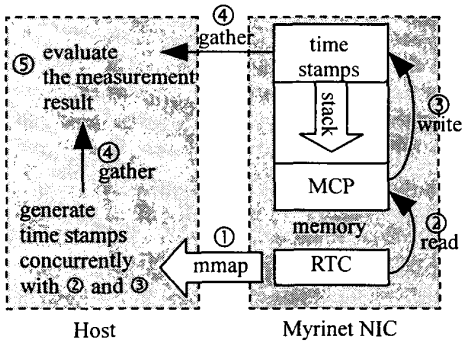

Figure 2. Reading and gathering of RTC value

5.2 Configuration

All of our experiments run on a pair of the workstations that use 200MHz Intel Pentium-Pro processor, VS440FX motherboard (Intel 440FX PCIset), and M2F-PCI32B Myrinet NIC. The software used is Linux (kernel version 2.0.30) and Myrinet Software (version 3.09). The network topology is constructed as a simple LAN where the two workstations are connected through a switch.

Because BPI does not guarantee reliable data transfer nor flow control, the loss of data may occur according to the network pattern. Because the measurement goal is not throughput but latency, the data loss is eliminated as follows.

In the sender side, the continuous data loss in BPI occurs when the copy block is full. It happens when an application requests MCP to send data more frequently than the MCP's processing capability. In the receiver side of BPI, if the receive buffer is not allocated from the copy block before data is arrived, the received data is lost.

To avoid data loss, the send and receive patterns are set as follows. An application calls the send function after MCP completely processes the previous send request. That is, MCP sustains a state that only holds one request at any given time. On the receiver side, application prepares to receive before the arrival of a packet. If no packet arrives, the application polls until data is received.

Because UDP also does not guarantee reliable delivery, a sufficient interval between the send requests is ensured. The waiting time for packet arrival is excluded from the measurement.

Another factor that we consider in measurements is cache effect. Therefore, we construct the application that sends different data each time. If the application sends the same data repeatedly, the latency could be decreased because the same data is cached, but we believe that it is not the case in typical real systems.

6. Measurement Results

In characterizing the latency of BPI and UDP, data transfer is subdivided as in Table 1. Each symbol represents the latency of its category. In the table, symbol *(c)* means the copy overhead at a specific category: for example, *B(c)* and *U(c)*. The symbol *N* is the time interval between the time when the host program triggers a request and the time when MCP detects the request. The symbol *M(i)* means the time taken for MCP-host to detect an event by MCP-net or vice versa. Therefore, *M* can be written as $M = M(h) + M(i) + M(n)$.

	Symbol	Category
Host	*B*	BPI
	U	UDP
	I	IP
	DD	Device Driver
	(c)	Copy
Notice	*N*	Notice
NIC	*M*	MCP
	M(h)	MCP-host
	M(i)	Interval
	M(n)	MCP-net

Table 1. Latency symbol and category

6.1 BPI

Figure 3 depicts the subdivided latency of the protocol stack using BPI. On the sender side, BPI copies the data to the copy block (*B(c)* in Figure 3) and inserts send request into the Tx queue that is shared with MCP. When MCP detects the request by polling the Tx queue (*N* in Figure 3), the data of copy block is DMA'ed into the NIC memory (*M(h)* in Figure 3). MCP-net detects that the data is in the NIC memory (*M(i)* in Figure 3) and sends the data to the network (*M(n)* in Figure 3). On the receiver side, incoming data is received to the NIC memory by MCP-net (*M(n)* in Figure 3). MCP-host detects the received data (*M(i)* in Figure 3) and DMA'es the data into the copy block (*M(h)* in Figure 3). Then, BPI passes the pointer of the received data to the application (*B* in Figure

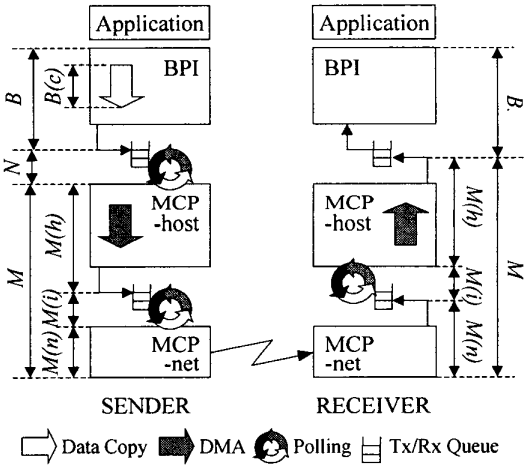

Figure 3. BPI send/receive

byte	*B*	*N*	*M*
32	13.4	4.8	20.7
64	13.3	4.8	20.7
128	15.9	4.9	21.6
256	20.9	5.0	22.0
512	30.9	4.9	24.4
1024	50.8	4.8	29.4
2048	91.9	4.8	44.3
4096	169.5	4.8	77.9
8192	332.5	4.8	130.7

Table 2. BPI send latency (μs)

byte	*B*	*M*
32	14.5	34.1
64	14.5	34.6
128	14.5	34.8
256	14.4	35.8
512	14.4	38.0
1024	14.5	42.3
2048	14.5	49.4
4096	14.5	65.7
8192	14.5	96.8

Table 3. BPI receive latency (μs)

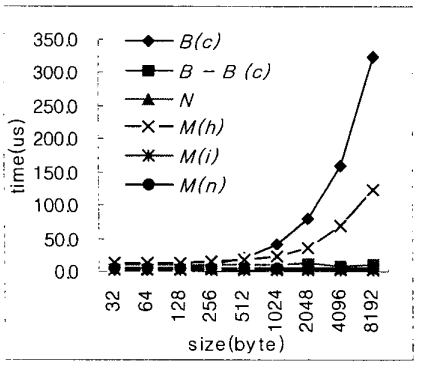

Figure 4. Subdivided latency of BPI send

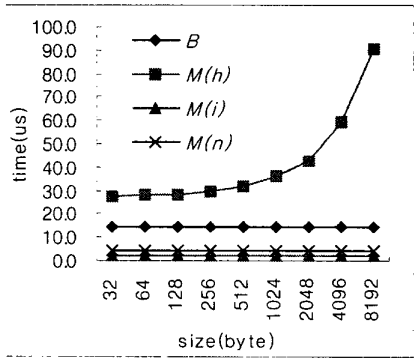

Figure 5. Subdivided latency of BPI receive

3).

The measurement results of *B*, *N*, and *M* of the sender side are shown in Table 2. *B* and *M* increase as the data size gets larger. The reason is because *B(c)* and *M(h)* increase as the data size does in Figure 4. Also note that the slope of *B(c)* is steeper than that of *M(h)*. This is the major reason why *B* is larger than *M* for 512B or larger data size in Table 2. However, other categories of the latency at Figure 4 are invariant. This means that the performance of the memory system has strong influence on the BPI performance. Latency *N* in Table 2 has a constant value because our measurement forces MCP to process one request at a time to prevent data loss as mentioned in Section 5.2.

Table 3 is the measurement results of *B* and *M* in the receiver side using BPI. *M* increases in proportion to the data size. The increase of *M* is due to the DMA operation of *M(h)* as shown Figure 5, which is the per-byte overhead of the receiver side. Note that *B* is invariant. This is because BPI receive simply returns the pointer of the data located in the copy block. Therefore, on the receiver side, there is no data copying in BPI layer unlike the sender side.

6.2 UDP

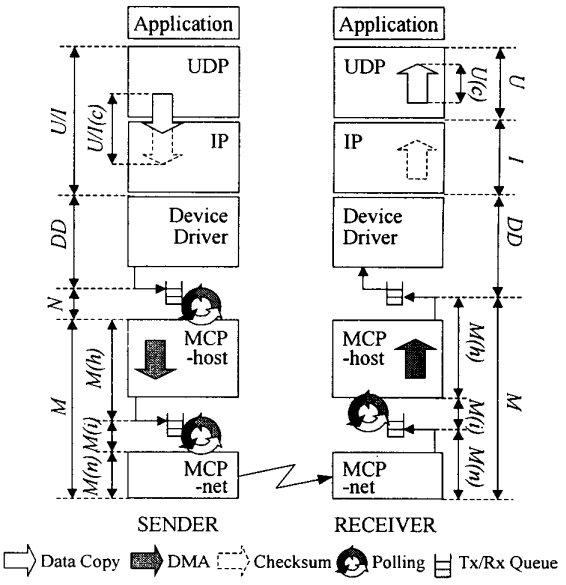

Figure 6. UDP/IP send/receive

byte	*U/I*	*DD*	*N*	*M*
32	39.4	29.7	5.3	21.3
64	38.3	29.8	5.4	21.3
128	39.6	30.5	5.3	23.2
256	40.2	30.8	5.4	24.6
512	43.7	32.6	5.4	28.9
1024	48.3	34.5	5.4	37.6
2048	57.6	37.6	5.4	59.9
4096	71.0	47.5	5.3	104.2
8192	110.7	49.2	5.4	209.8

Table 4. UDP/IP send latency (μs)

byte	*U*	*I*	*DD*	*M*
32	25.4	9.0	44.1	35.2
64	27.4	10.0	44.1	35.6
128	27.4	10.1	43.5	34.8
256	28.0	10.3	43.6	35.9
512	28.8	18.0	43.5	38.0
1024	31.9	20.5	44.3	42.2
2048	37.1	33.0	44.1	49.4
4096	47.4	62.0	44.7	65.5
8192	66.6	107.0	44.9	96.5

Table 5. UDP/IP receive latency (μs)

Figure 6 shows the subdivided latency of UDP. On the sender side, an application requests a send operation via the kernel through a system call. UDP/IP (*U/I* in Figure 6) copies the data from the user space into the kernel space with the checksum calculation (*U/I(c)* in Figure 6) and constructs its header. After that, the data is passed to the device driver (*DD* in Figure 6) and then sent to the network through MCP (*M* in Figure 6). On the receiver side, an application calls a receive system call. If there is any data already received in the socket buffer queue, the data is copied to the user space by UDP. If the socket buffer queue is empty, the application is forced to block. MCP (*M* in Figure 6) triggers an interrupt when a packet has arrived from the network. The device driver (*DD* in Figure 6) handles this interrupt, which identifies the packet type and passes it to the appropriate protocol. In the present case, the protocol is IP (*I* in Figure 6) that processes the packet including the checksum calculation. Subsequently, the packet is delivered to the above layer, that is UDP (*U* in Figure 6). At this time, the application that has been waiting is wakened, and the data is copied

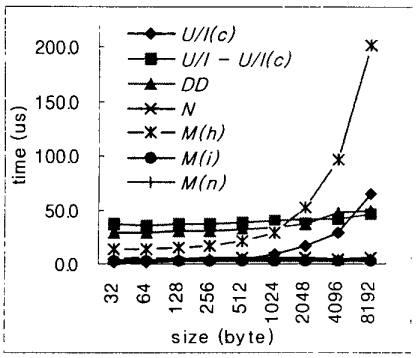

Figure 7. Subdivided latency of UDP/IP send

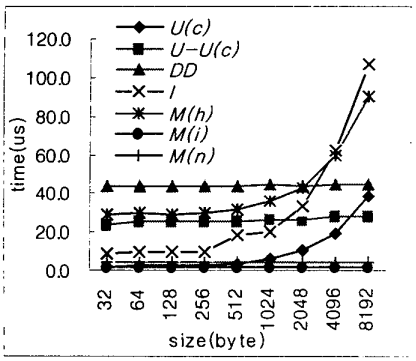

Figure 8. Subdivided latency of UDP/IP receive

(*U(c)* in Figure 6) to the user space.

Table 4 shows the measurement results of the subdivided latency of UDP send. Figure 7 graphs the details. Two categories increase in proportion to the data size: *U/I(c)* and *M(h)*. An interesting difference between Figures 4 and 7 is that there is a great disparity between the copy overheads, *B(c)* and *U/I(c)*. The reason is explained in Section 7.

Table 5 and Figure 8 show the latency of UDP receive at each category. Per-byte overheads are *U(c)*, *I*, and *M(h)* in Figure 8. *U(c)* and *M(h)* are due to the copy operation and DMA operation. The increase of *I* over the data size is caused by the checksum calculation of IP.

7. BPI versus UDP

Figures 9 and 10 depict the comparison of total latencies between BPI and UDP. On the sender side, BPI has less latency than UDP for 2KB or smaller data size. However, for larger than 4KB, BPI takes longer than UDP. On the receiver side, BPI has smaller latency than UDP for all data sizes. It is very surprising that UDP takes less time than BPI for large data sends. It is generally believed that the light-weight communication primitives have smaller latency, but this paper found a case that it is not true. The reason is identified as follows.

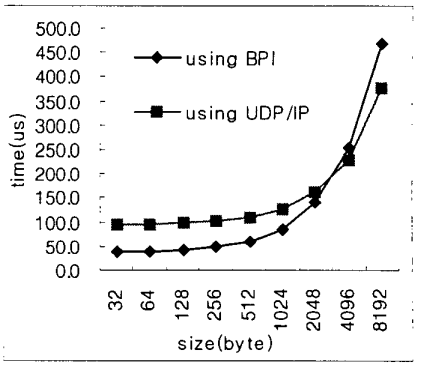

Figure 9. Total latency for send

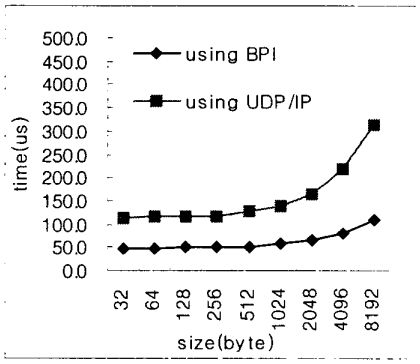

Figure 10. Total latency for receive

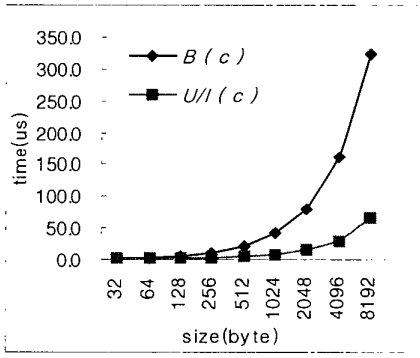

Figure 11. Copy overhead

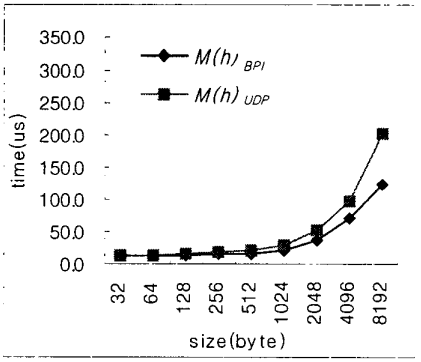

Figure 12. DMA overhead

As mentioned in Section 6, there are two dominant overheads in the sender side: 1) copy overhead of BPI and UDP layer and 2) DMA overhead of MCP-host layer. We need to examine each one carefully.

Comparing the copy overhead of BPI and UDP, Figure 11 shows that the copy operation of BPI takes much longer time than that of UDP. This is due to the cacheability of the BPI copy block. In the initialization of BPI, it allocates an area of the kernel memory to the copy block and mmaps this area into the user space, so that an application can directly access the copy block. When the copy block is mapped, BPI sets the PCD (Page-level Cache Disable) flag [11] of page-table that maps the copy block. It means that the copy block of BPI is uncached because the PCD flag of page-table entry controls the cacheability of individual pages in Pentium-Pro processor. Therefore, the copy from user buffer to the copy block is uncached. This turns out to have a large overhead. Whereas, the copy operation of UDP send takes 1/5 of BPI for 8KB even though it includes the checksum calculation, because it copies data from the host memory to the processor cache. This is the main reason why the total send latency of UDP is less than that of BPI for larger than 4KB as Figure 9.

In contrast, the DMA overhead of BPI is less than that of UDP as in Figure 12. The difference of the DMA latency in Figure 12 is caused by the write-back cache policy of Pentium-Pro processor. There is no implicit write-back overhead when BPI is used because the copy block is uncached. However, in the case of UDP, implicit write-back occurs by the snoop protocol when MCP-host tries to DMA the data into the NIC memory because the data is in the cache. Therefore, as shown Figure 12, $M(h)_{UDP}$ is a little larger than $M(h)_{BPI}$.

On the receiver side, using BPI has less latency than using UDP for all data sizes as shown in Figure 10. This result is opposite to the sender side. The first reason is that the receive latency of MCP-host is the same regardless of BPI or UDP because it is based on DMA from the NIC memory to the host memory. The second is that the BPI receive merely returns the pointer of data, but UDP copies the received data from the kernel space to the user space. Therefore, BPI layer has an invariant latency for receive but the latency of UDP layer increases as data size.

8. Conclusions

This paper presents a new measurement methodology suitable for high-speed networks and applies it to Myrinet. The latency of BPI, a light-weight communication primitive in Myrinet Software, is measured and analyzed in details. As a traditional protocol, UDP is thoroughly analyzed in Linux on Myrinet and compared with BPI.

As a surprising result, we found that BPI has larger latency than UDP for larger than 4KB data size. In other words, BPI is efficient for small data transmission but in the case of large data transmission UDP is more suitable than BPI. This paper identifies that the reason of this result is the cache policy of the copy block of BPI. BPI may outperform if cacheable copy block is used. On the receiver side, BPI has less latency than UDP for all data sizes, because BPI receive dose not copy.

In summary, this paper shows: 1) per-byte overhead is still an important factor to light-weight communication primitives as well as traditional protocols and 2) light-weight communication primitive should be designed to use cacheable memory.

Acknowledgements

The authors thank the anonymous reviewer for their helpful comments and suggestions on this paper.

References

[1] 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.

[2] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications*, pp.23-29, June 1989.

[3] Scott Pakin, Mario Lauria, and Andrew Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," *Supercomputing '95*, December 1995.

[4] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *15th ACM SOSP*, pp.40-53, December 1995.

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

[6] Eric Cooper, Onat Menzilcioglu, Robert Sansom, and Francois Bitz, "Host Interface Design for ATM LANs," *16th Conference on Local Computer Networks*, pp.247-258, October 1991.

[7] J. Kay and J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *USENIX Winter Conference*, pp.249-258, 1993.

[8] Alec Wolman, Geoff Voelker, and Chandramohan A. Thekkath, "Latency Analysis of TCP on an ATM Network," *USENIX Winter Conference*, pp.167-179, January 1994.

[9] Ames Lab, Interconnect Performance, http://www.scl.ameslab.gov/Projects/ClusterCookbook/icperf.html, 1998.

[10] M. Welsh, A. Basu, and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces," *Hot Interconnects V*, August 1997.

[11] Intel Co., *Pentium Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, Order Number: 242692, Intel Co., December 1995.