Performance Optimizations for High-Speed TCP on BCL-4

Gao Fan, Ma Jie, Meng Dan
Institute of Computing Technology, Chinese Academy of Sciences
P.O. Box 2704, Beijing, 100080, P.R.China
Emails: gf@ncic.ac.cn, majie@ncic.ac.cn, md@ncic.ac.cn

Abstract: *This paper presents experiences with TCP/IP optimizations on BCL-4 – a high-performance low-level communication protocol used on a cluster of SMPs called DAWNING-4000. We presently implement the TCP/IP support on BCL-4 with a gigabit-per-second Myrinet network. This implementation (called BCL-4/IP) realizes most of optimizations for TCP/IP, including extended MTU, scatter/gather DMA, checksum offloading, interrupt suppression, PIO/DMA tradeoff and using the unreliable protocol in the basic BCL protocol layer. Our experiment results demonstrate that the improvement of TCP performance over Fast Ethernet by orders of magnitude is possible. The one-way latency decreases from 63.9 us to 38.8 us, while the bandwidth increases from 94 Mb/s to 931 Mb/s. BCL-4/IP enables most of communication software such as WWW, NFS and FTP applications, which are based on TCP/IP, to obviously benefit from the high-speed Myrinet network. The binary compatibility with these applications is also realized on BCL-4/IP.*

Keywords: *Network performance, TCP/IP optimizations, Cluster, BCL-4/IP, Myrinet.*

1. INTRODUCTION

To boost the communication performance of distributed applications, many cluster systems employ system area networks (SANs) as a key component that can operate at gigabit speed. However, as the raw-hardware performance of such SANs (e.g. Myrinet, Gigabit Ethernet, SCI, etc.) becomes higher, the software overheads becomes the main part of the total time to transmit a message. As a result, a number of high-performance low-level communication protocols have been designed and implemented to decrease the overhead of the end-system on the transmission path, using some special mechanism such as user-level communication. These low-latency protocols avoid operating system intervention while at the same time providing user-level messaging services across high-speed networks. [1,3,10,11,12,13,14]

On the other hand, when directly using these protocols, SANs have not yet become essential requirements, in part because existing applications can not directly benefit from the potential performance coming along with SAN [15]. Many of such applications are based on TCP/IP protocol rather than directly using message passing libraries such as MPI or PVM. In this context, to gain the performance for most of existing applications, we should make a decision between re-implementing these TCP-based applications and providing the TCP/IP supporting capability for SANs. The latter is the choice of *BCL-4* – a VIA-like low-level high-performance communication system for our ongoing DAWNING-4000 cluster project.

Moreover, the TCP/IP performance is often limited by the network hardware and host-side overheads. This performance bottleneck can be overcome by adopting the high-speed interconnects such as Myrinet [9] instead of conventional ethernet hardware. Besides, high performance of TCP/IP can be achieved by reducing host-side overheads via various optimizations above and below the TCP/IP protocol stack.

As the TCP/IP supporting component of *BCL-4*, *BCL-4/IP* is an effort to harness the power of our high-performance *BCL* communication protocol and high-speed Myrinet SAN. Besides the high-speed network hardware, *BCL-4/IP* can improve communication performance by reducing host-side overheads via multiple optimizations for TCP. Because *BCL-4/IP* supports the standard TCP network interface and BSD Sockets API, the cluster nodes with different OS (e.g. AIX and Linux) can efficiently communicate each other. This feature is especially useful for applications running over a cluster of clusters [2]. This article will introduce our effort to add TCP/IP supporting layer, expatiate on our efficient optimizations and analyze the optimization results for TCP/IP.

The rest of this paper is organized as follows: section 2 introduces our efforts to optimize TCP/IP performance over *BCL-4/IP*. In section 3 we give an overview of *BCL-4* protocol and show the design and implementation of TCP/IP support on *BCL-4* in detail. Then the evaluation of the *BCL-4/IP* performance result with our TCP optimizations is shown and discussed in section 4. At the end of this article, we draw our conclusions and present our ongoing research in section 5.

2. TCP/IP OPTIMIZATIONS

TCP is in fact not the source of the overhead often observed in packet processing, and that it could support very high speeds if properly implemented [7]. Data transmission using TCP involves the host operating system facilities for memory and process management, as well as the TCP/IP protocol stack and the network device and its driver [6]. There are two kinds of transmission overheads over the TCP data path: per-packet overheads and per-byte overheads. The former involves overheads to execute the TCP/IP protocol code, allocate and release memory buffers, and interrupts from network device for packet arrival and transmit completion, the latter includes overheads to move data within the end system and to compute checksums to detect data corruption in the network [6].

To maximize the communication performance, we adopt several optimizations for TCP, they are:

1) *Scatter/Gather DMA Support:* The main idea of zero-copy mechanism resides in eliminating all data copying on the communication path. To achieve the efficient compatibility with most of existing applications, we maintain the traditional BSD Sockets API without any modifications or extensions to OS kernel. But *BCL-4/IP* can avoid other internal data copying such as the copy from kernel buffers to DMA-able buffers by supporting scatter/gather DMA. Because the data for an IP frame may span multiple regions of memory and be separated from the packet head, the IP data can be DMAed to NIC (Network Interface Card) directly. *BCL-4/IP* does not need to manage any DMA-able buffers in driver layer when sending and does not need to copy data from driver buffers to kernel buffers when receiving. This measure reduces the per-byte overheads.

2) *Checksum Offloading:* To detect data corruption in the network, TCP/IP generates an end-to-end checksum for each packet covering all protocol headers and data [6]. In most implementations, this checksum requires host CPU to load all of the data through the memory hierarchy for a sequence of add operations [6]. This per-byte overhead may be eliminated to improve the TCP/IP performance. Because of the hardware checksum supported by the Myricom's LANai adapter and the ACK/NAK mechanism provided by reliable *BCL-4* protocol (see also section 2.7), we can consider that *BCL-4/IP* packets are transferred on a reliable network and each packet received by the receiving buffer is correct. (The NIC firmware of *BCL-4* will discard the current packet when finding out any data corruption with this packet). Therefor *BCL-4/IP* can bypasse the software checksum to enhance the TCP/IP communication performance.

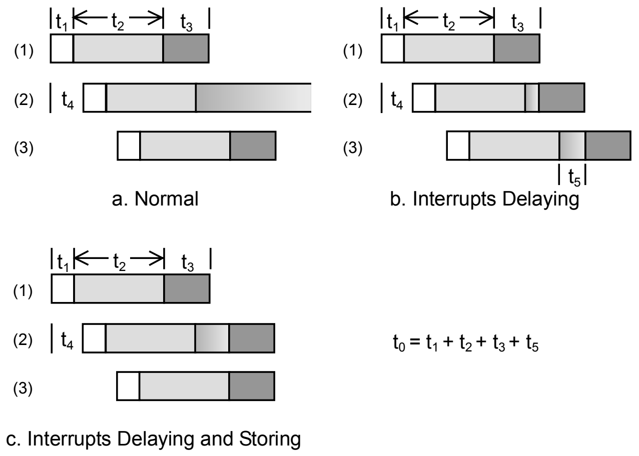

Figure 1. Interrupt Delaying and Storing on Sending

3) *Interrupt Delaying and Storing:* Interrupts to notify the hosts of transmit-complete events and packet arrival can impose a significant overhead [6]. The on-card *MCP* (Myrinet Control Program) of *BCL-4* can store and delay interrupts triggered by more packets received or transmitted when the host are handling the current interrupt. When storing the interrupts, the *MCP* stores the relative sending/receiving events in the sending/receiving event queue. And then the interrupt handler can deal with multiple event notifications

during the next interrupts handling process. This reduces the total number of interrupts delivered by *MCP* and limits this per-packet overhead generated by interrupt handling. When sending, *Interrupt Storing* can reduce the average IP packets handling time (the time from *BCL-4/IP* receiving an IP packet from IP layer till the NIC completing the packet transmission and releasing relevant data buffers). This improves the potential packets transmitting capacity under the current flow control mechanism of *BCL-4/IP* (see also section 3.4). When receiving, *Interrupt Delaying and Storing* can avoid the drop of incoming packets that come from the lost of interrupts (shown as Figure 1.a.2). Figure 1 gives a sending example to compare the 3 different schemata. In this Figure, t_1 means the time for *BCL-4/IP* to handle and transfer an IP packet from upper layer to NIC hardware, t_2 denotes the time for the sender to wait the sending-complete interrupt, and t_3 is the time overhead for interrupt handling. (BCL-4/IP releases the relevant packet buffer during t_3 time.) The t_1, t_2 and t_3 can be considered constant factors on a given platform. Then t_0 is the total time for *BCL-4/IP* to transmit an IP packet while t_4 is the time slice between two consecutive packets from upper layer. The factor t_5 denotes the additional time for a sender to continue to wait after the NIC has completed the relevant packet sending, this is caused by the interrupt handling process triggered by the last packet sending operation. As shown in Figure 1, *Interrupt Delaying and Storing* can eliminate the infection of time waste. We will discuss the effect of *Interrupt Delaying and Storing* in more detail at following sections.

4) *Extended MTU:* Another way to limit per-packet overheads is to use larger message frames, reducing the total number of transmitting packets. The MTU is the typical unit of packet that travels between the host and the network adapter. Generally, using larger MTU can reduce the total number of travelling packets. This means fewer interrupts and less overhead to process the TCP/IP protocol. Nowadays many network adapter vendors introduce the MTU of "jumbo" frame size up to 9000 bytes to exceed the 1500-byte MTU limit of IEEE 802.3 standard. Myrinet interconnects support long packets (unlimited MTU), link-level error and end-to-end flow control that is properly handled by deadlock free routing in the wormhole switches [8]. Under this circumstance *BCL-4* can support up to 128k bytes packet size by which *BCL-4/IP* can reduce the overheads from the fragmentation and de-fragmentation of IP packets. Currently the maximum MTU of *BCL-4/IP* is 32000 bytes.

5) *PIO vs. DMA:* There are two methods to access the NIC: *PIO* (Programming IO) and *DMA* (Direct Memory Access). The former, by which the message data is directly wrote into the sending request, costs less CPU time at startup time but the latter can achieve a higher bandwidth when transferring large messages. So when small messages are sent, *BCL-4/IP* employs the *PIO* mode to lower the latency, while when transferring larger messages, we use the *DMA* mode to enhance the bandwidth performance.

6) *Buffer Posting vs. Copying: BCL-4* adopts a particular memory registering operation called *Buffer Posting*. That is, the buffers used by the receiver/sender can be pre-registered for DMA operation. *Buffer posting* is the key element of *BCL* that enables the zero-copy mechanism, but it is a relatively expensive operation for small messages. Instead, we can copy the outgoing/incoming data into/from the pre-posted DMA-able buffers. However, because the cost of memory copying increases more rapidly than that of *buffer posting*, we employ a hybrid scheme where data is copied into the pre-posted buffer only if its size is relatively small (otherwise there will be new buffer posted for the data).

7) *Reliable BCL vs. Unreliable BCL: BCL-4* basic protocol adopts ACK/NAK mechanism to guarantee reliable and ordered data communication. That is, *BCL-4* realizes the timeout/retransmission mechanism below the NIC level. Although TCP is a reliable protocol, realizing the reliable communication mechanism on a lower protocol layer can reduces the possibility of the TCP timeout occurrences, while the expense is increasing the average communication overhead. For example, when a transmission error occurs in the hardware layer, the *MCP* on NIC will find it first (thanking for the smaller timeout value of *BCL* than that of TCP) and retransmit it before the TCP timeout occurs. On the other hand, *BCL-4* also provides unreliable mode – basic *BCL* protocol just transmits packets without any verification. The reliability is the business of the upper layer. Out of question, unreliable

BCL can yield higher average latency performance on a reliable physical network.

3. IMPLEMENTION ON BCL-4

3.1 DAWNING-4000 Super-server

DAWNING-4000 is our ongoing high-performance CLUMPS project. Its current version is based on Linux2.4, which will consist of 128 Intel-based SMP nodes. And our ongoing Power-based IBM AIX version will come out soon. All nodes of DAWNING-4000 are connected with Fast Ethernet and Myrinet. The Myrnet NIC is supported by *BCL-4* protocol.

3.2 BCL Protocol

BCL is a VIA-like low-level high-performance communication interface [4]. Our ongoing semi-user-level *BCL* protocol [4] version is *BCL-4*, which is implemented on Myrinet.

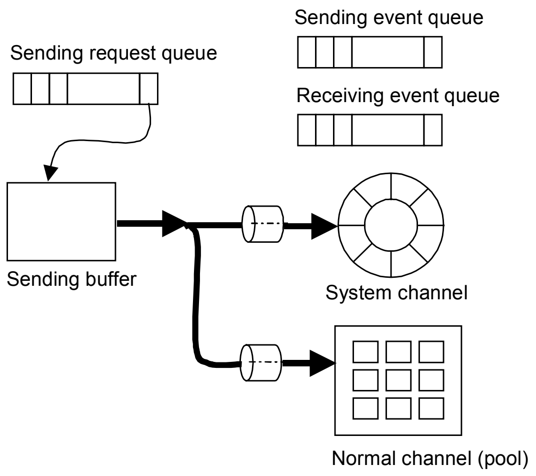

Figure 2. Communication Mechanism of BCL-4

BCL-4 is divided into three parts, *BCL* library, NIC device driver (*DD*) and on-card control program (*MCP*).

BCL library includes all implementations of *BCL API* for user-level applications. The popular message passing libraries such as MPI or PVM can employ it to increase the performance of parallel applications. User applications can use these APIs directly too.

DD is a NIC driver that transfers the message between applications and *MCP*. *DD* also provides some *sys_calls* to control the hardware and posts some operation requests to the memory space on the NIC card. Besides, *DD* executes several functional operations such as the physical memory address conversion in the OS kernel.

MCP manages all the inter-node packet transfers, starting a sending operation by reading the send request, sending or receiving packets with DMA engines and noticing the user process about the transferring completion.

BCL defines port for inter-process communication. As shown in Figure 2, each port has a sending request queue, a receiving buffer pool and the corresponding event queues. Receiving buffer pool is composed of several channels. [4]

Currently there are two kinds of channels in *BCL-4*: system channel and normal channel. System channel is designed to transmit small messages. Each system channel has a buffer ring that is organized as a FIFO queue to receive the incoming small message. Normal channel is designed to transfer messages with rendezvous semantics [4]. Like the system channel, each port has a set of normal channels. The receiving process needs to post a data buffer in its user/kernel space and post the buffer to the receive channel before the remote sending process start transmission.

When sending, the sender traps into OS kernel, completes virtual-to-physical address translation and pins down the user-buffer for sending data [4]. Then it issues a sending request (the request includes physical addresses and lengths of all physical pages consisted of the user buffer) into the sending request queue on NIC. After being noticed with the new sending request, *MCP* DMAs the message data from sending buffer to NIC and transmits them to destination NIC through Myrinet physical link. When a sending operation is complete, a send event will be generated. The receiving channel at the destination should be ready before the message arriving. When a message arrives, a receiving event will be put into the receiving event queue to notice the receiver process.

3.3 TCP/IP Support

Our approach to support TCP/IP on *BCL-4* is illustrated in Figure 3. As shown in it, we insert an *IP-to-BCL* layer between IP and *BCL* and implement it as a function module in the *DD* driver.

The *IP-to-BCL* layer is an extension to *BCL DD* part in which the main part of TCP/IP supporting implementation resides. *BCL-4/IP* can make full use of the existing data structure and the communication mechanism of *BCL* protocol (see also the 3.2 section and Figure 2) through the *IP-to-BCL* layer. *BCL-4* reserves an *IP-port* to transfers the IP packets between the IP layer and the *MCP*. In this layer, an IP address is translated to the destination node number in *BCL-4* context.

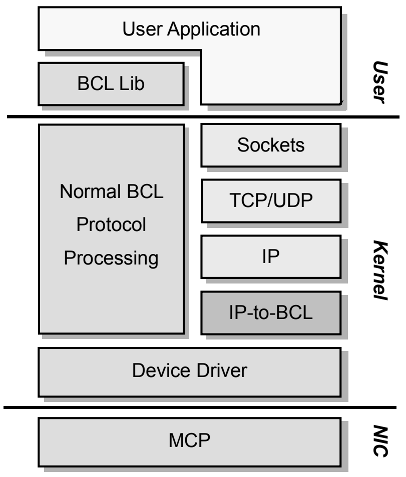

Figure 3. BCL-4/IP Architecture

As shown in Figure 4, *DD* (with the *IP-to-BCL* layer implementation) introduces two message queues to transfer IP packets: *send_ring* and *recv_ring*.

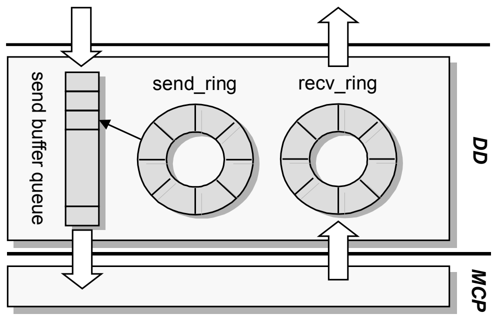

Figure 4. BCL-4/IP Communication Mechanism

When sending, *DD* receives an IP packet and links its data buffer into the *send buffer queue*. Then *DD* generates the physics memory page table of its data (used by the scatter/gather DMA operation). After registering them to the *send_ring*, *DD* composes a send request to *MCP*. When sending complete, *MCP* triggers a *sent-interrupt* and posts a sending complete event into the *sending event queue*. In the *sent-interrupt* handling process, *DD* access the *send buffer queue* to release the relevant data buffer indexed in the *send_ring*.

At the receiver side, *DD* allocates all the DMA-able buffers to full-fill the *recv_ring* and posts them to the normal channel of IP-port at the initial time. After receiving a message for IP-port, *MCP* DMAs it into the *recv_ring* and refresh the *receiving event queue* to notice *DD*. Because of the *Interrupt Delaying and Storing* technique (see also section 2.3) *DD* can deal with several sending/receiving events once in the interrupt handling process.

In summary, there are two paths to access the network communication in *BCL-4*: using the *BCL API* or using the standard BSD Sockets API through the *IP-to-BCL* layer. The latter is implemented by *BCL-4/IP*.

3.4 Flow Control

Flow control is used to avoid buffer overflow and system deadlock. In *BCL-4/IP*, the sending process will be blocked when the *send_ring* is full. Since the unit in *send_ring* is unavailable during the relevant IP packet handling time (see also section 2.3), the potential packets transmitting capacity will be limited by both average IP packets handling time and the *send_ring* size of *BCL-4/IP*. At the receiving side, *MCP* simply discards packets when there is no enough buffers (an overflowed packet is handled as an error packet in basic *BCL* protocol). Using reliable *BCL* protocol, *MCP* will re-transmit them when timeout, otherwise it will lead to a TCP timeout.

4. PERFORMANCE RESULT AND ANALYSIS

The testing platform consists of 32 Linux workstations now. Each node is a 2-way 1GHz PIII (Coppermine) SMP with 66MHz PCI Bus, which is running Linux2.4 kernel. Myrinet [9] M3S-PCI64B NICs are used on each node and interconnected by M3S-SW16-8S switches.

To test the performance and quantify the impact of our optimization efforts, we use the netperf v. 2.1p13, a standard tool to benchmark TCP/IP performance, to test the bandwidth performance of *BCL-4/IP* and *tcplatency*, an application developed by ourselves to measure the latency performance. We also port RPC (Remote Procedure Call) and FTP (File Transfer Protocol) applications on *BCL-4/IP* to evaluate the *BCL-4/IP* performance over user applications.

In addition, we have added a particular module into *DD* to measure the IP data deliver rate between upper layer and *IP-to-BCL* layer (see also section 4.3).

4.1 Latency

The first tests are raw point-to-point latency tests. All the latency results are measured by a half of round-trip time in a ping-pang test. Figure 5 compares the latency of *BCL-4/IP* (using reliable and unreliable basic *BCL* protocol) with that of Fast Ethernet.

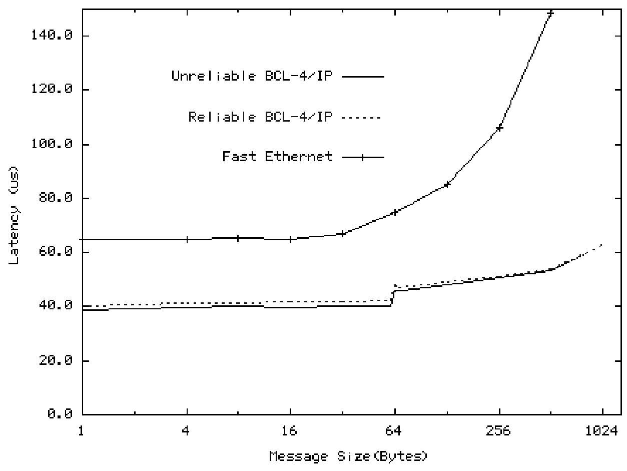

Figure 5. Latency of TCP on BCL-4/IP

As shown in Figure 5, *BCL-4/IP* performs an extremely lower latency than that of Fast Ethernet. The lowest latency result is 38.8 us on unreliable *BCL-4/IP* while that of Fast Ethernet is 63.9 us. The impact of using unreliable basic *BCL* protocol can also be noticed clearly. But compared with the total latency result, this improvement is not very obvious since the primary latency overhead is resided in the upper layer such as Sockets API and TCP/IP protocol processing.

Noticeable feature of the line labeled *Reliable/Unreliable BCL-4/IP* is the jump at message size of 64 bytes. This is the effect of the optimization to make a tradeoff between *PIO/DMA* (see also section 2.5). This "jump" reflects the remarkable difference of the latency performance between *PIO* and *DMA*.

4.2 Bandwidth

Figure 6 shows TCP/IP bandwidth on *BCL-4/IP* as a function of the message size. All the test results in this figure are come from netperf. We can obviously consider the effect of *Extended MTU* optimization via this figure. Comparing the line labeled *1500 MTU with csum* with the line labeled *32k MTU with csum*, the peak bandwidth increases extremely from 392 Mb/s to 809 Mb/s – yielding about a 100 per cent improvement. This result shows that moving from a 1500-byte MTU to a 32-kbyte MTU enables *Extended MTU* optimization of *BCL-4/IP* to result in a notable effect. By Figure 6 we can also estimate the impact of *Checksum Offloading* optimization. Considering the line labeled *32k MTU without csum* (*Checksum Offloading*) and the line labeled *32k MTU with csum* (no *Checksum Offloading*), the bandwidth performance jumps by 100 Mb/s on average when the message size exceeds 512 bytes. Since the checksum is a bytes-sensitive operation, the effect of *Checksum Offloading* can be noticed more distinctly over longer IP packets. When transmitting data smaller than 256 bytes, *BCL-4/IP* can hardly benefit from the *Checksum Offloading* optimization. Similarly, the advantage of *Checksum Offloading* is relatively slight with smaller MTU size. This feature can be observed in Figure 7.

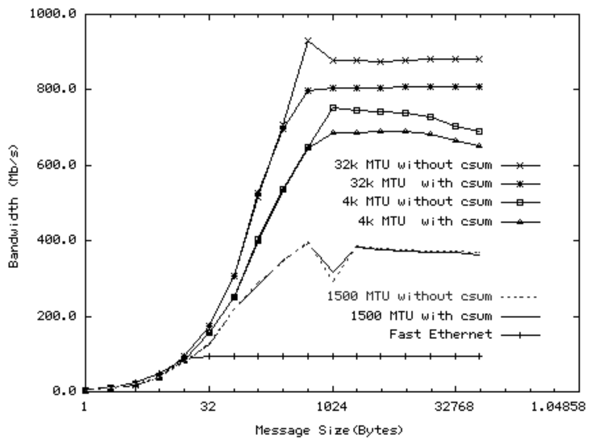

Figure 6. Bandwidth of TCP on BCL-4/IP

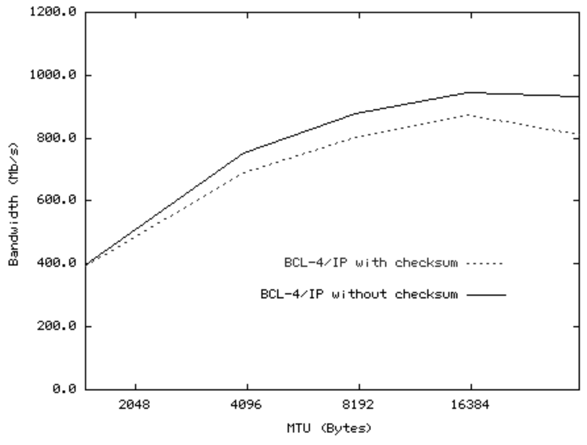

Figure 7. Peak Bandwidth of Various MTU Size

In Figure 7, TCP/IP peak bandwidth on *BCL-4/IP* is shown as a function of the MTU size. The data clearly shows the bandwidth improvement from increasing MTU size. The peak bandwidth improves extremely within 8-kbyte MTU size and the advantage of *Checksum Offloading* increases continuously in the range of 1500 – 32k-byte MTU size. That is the reason for *BCL-4/IP* to set the default MTU size of 8 kbytes. Beyond the 8-kbyte MTU size, per-byte host-side overheads such as memory-copy and software checksum with large packet size dominate the bandwidth performance. The benefit of increasing MTU is neutralized by such kinds of overheads. That is one of the reasons why the line labeled *BCL-4/IP without checksum* slightly descends while the difference between the two lines in Figure 7 increases after reaching 16-kbyte MTU size. We can make an assumption that the "sweet spot" of 8-kbyte MTU size would shift to a higher place if we can ulteriorly reduce the host-side per-byte overheads extremely.

4.3 IP Data Deliver Rate and Flow Control

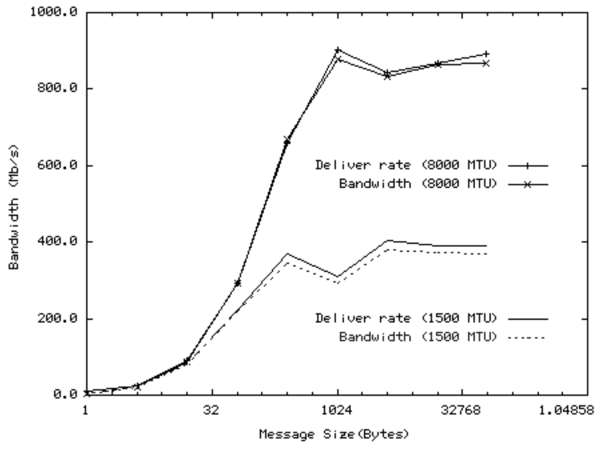

Figure 8. IP Data Deliver Rate

IP data deliver rate between the upper layer and *IP-to-BCL* layer means the transmission bandwidth between the OS kernel and *DD*. Clearly, the total TCP transmission bandwidth can not exceed this factor. It can be calculated by the formula below:

$$W = N / T_n \qquad (1)$$

Where W is the IP data deliver rate value (Mb/s), N denotes the total packet length that *DD* receives from upper layer during a given period, and T_n means the time value of this period. W reflects the bandwidth capacity of layers above the *IP-to-BCL* layer to issue IP packets without the intervention from lower layer such as sender blocking caused by *BCL* flow control mechanism. This factor is primarily dominated by the data moving overheads in above and within TCP stack such as memory-copy and checksum. Therefor n and T_n can be obtained by experiments. We do not intend to discuss the detailed method of our experiments in this paper. Here we only give the measurement and calculation results in Figure 8.

Comparing the experiment results with the total TCP bandwidth using the same MTU size (lines with label *Bandwidth*), we can consider that the total TCP bandwidth of *BCL-4/IP* almost saturate the bandwidth capacity of upper layers (we do not detect any lower level blocking occurrence at sender side). It means the current bottleneck shifts to upper layers. To further improve the bandwidth performance of *BCL-4/IP*, we should widen this upper-layer bottleneck first.

On the other hand, considering Figure 1 in section 2, we can calculate W by this formula:

$$W = n / t_4 \qquad (2)$$

The factor n is the average packet data size. On Linux SMP platform, generally the kernel process can avoid the intervention of interrupt handling process because of the multiple CPU architecture (an interrupt signal can only interrupt one CPU at a time). In this case, t_4 is dominated by W (when transferring long enough packets, n is the current MTU size). To achieve efficient flow control mechanism, we should to determine the appropriate *send_ring* size. While all elements in *send_ring* are being held, the sending process will be blocked. So the appropriate size is dominated by the number of used *send_ring* units at the same time. Defining this number as M, we can obtain it by several expressions. In the case of Figure 1.a (normal interrupt handling):

$$M_a = t_0 / t_4 + x = (t_1 + t_2 + t_3) / t_4 + x \qquad (3)$$

Where x is the number of packets that remain in *send_ring* caused by the interrupt lost (see also Figure 1.a.2). While in the case that $t_4 > t_0$, M can always be 1 (any packet can be completely handled before the next packet arriving). In the case of Figure 1.b (interrupt delaying):

$$t_5(i) = t_5(i-1) + t_3 - t_4(i) \qquad (4)$$

Where $t_{4/5}(i)$ means the $t_{4/5}$ value of the i'th packet. When $t_4 > t_3$, t_5 will always be 0. Then:

$$M_b = t_0 / t_4 = (t_1 + t_2 + t_3) / t_4 \quad (5)$$

While $t_4 < t_3$, from formula (4), we can see that t_5 and $t_0$will increase continuously. This means whatever the *send_ring* size we define the *send_ring* will definitely be sometime blocked. Unfortunately, this will badly reduce the TCP/IP performance.

Thankfully, in the case of Figure 1.c (interrupt delaying and storing), all delayed packets will be handled together with the following interrupt-handling process. Then M_c can be simply calculated by formula (5). On our current experiment platform, we have measured t_0/t_4 using various MTU and message sizes and found the current range of M is between 1 and 4. That means we can only select a 4-unit *send_ring* on our current system to avoid low-level blocking in general (currently we use 64-unit *send_ring as default – quite a waste of kernel space*). In the case that W is relatively low, determining the *send_ring* size to select appropriate flow control capacity seems not very critical. But after performing further optimizations to enhance the IP data deliver capacity of upper layer (increasing W and decreasing t_4), this problem will become quiet a challenge. Via the method used in this section, we can precisely predict the sending capacity with *BCL-4* flow control mechanism from given *send_ring* size in the future.

4.4 RPC Performance

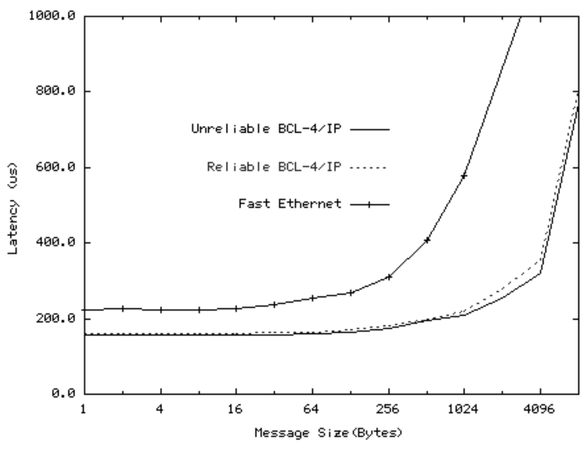

Figure 9. RPC Latency

Like a local function call, RPC can make a network function call transmit its calling arguments to the target procedure on remote side and wait for the returned response. The primary advantage of RPC is that it separates user applications from particular physical and logical network interface. RPC applications can benefit from the high-speed TCP on *BCL*/4-IP.

To process the experiment, we have used *drpc* tool kit, which is developed by our *DAWNING-4000* project group to test and apply the RPC mechanism into the cluster management and task schedule subsystem. *Drpc* selects the standard Sockets API to implement its network communication and the message size transferred by TCP/IP denotes the argument size of a RPC call. We measure the average elapsed time for a single RPC call of a variety of argument sizes using multiple network interfaces. The experiment results are shown in Figure 9.

Since RPC is a latency-sensitive application, our experiment shows that the peak RPC performance takes about 223 us on Fast Ethernet and 153 us on *BCL-4/IP*, improving the latency performance by about 69 per cent.

4.5 FTP Performance

Table 1 compares the measured bandwidth reported by the FTP client (ncftp on Linux 2.4) for two files of different sizes on various network interface.

Table 1. FTP Performance on *BCL-4/IP*

	FILE 1	**FILE 2**
File size (bytes)	12,328,960	263,475,200
Fast Ethernet	90 Mbps	90 Mbps
***BCL-4/IP* csum**	469 Mbps	510 Mbps
***BCL-4/IP* no csum**	506 Mbps	520 Mbps

As shown in Table 1, *BCL-4/IP* can improve the FTP bandwidth performance by about 5 times over Fast Ethernet. But the FTP performance results in less than about 40 percent with the raw TCP/IP performance (520 Mbps to 931 Mbps). This is because the actual data transfer is limited by the storage file system performance on local host.

5. CONCLUSION AND FUTURE WORK

BCL-4 provides the capacity to build clusters of commodity SMPs with ease. This high-performance communication protocol combined with TCP support part can provide an efficient message passing ability and binary compatibility with most TCP communication software. Besides its high reliability, *BCL-4/IP* provides the high communication performance on TCP with Myrinet interconnects.

Currently we have realized the TCP support of *BCL-4* on both IBM AIX and Linux. In this article we present the impetuses and aims to support the TCP on *BCL-4*, and show the optimizations and corresponding implementation on Linux in detail. Some of these optimizations have made considerable impact to TCP/IP performance on our current systems, while some others have just improved some potential capacities for future improvements. These optimizations and the high-speed Myrinet hardware make the point-to-point TCP bandwidth of *BCL-4* above 900 Mb/s and the latency under 40μs.

To further improve the TCP/IP performance, we plan to process our future work about the following issues:

1) Improving the IP packet deliver capacity above the *IP-to-BCL* layer by further optimizations such as *Zero-copy Socket*, modifying the OS kernel to implement *Scatter/Gather DMA* and to support the zero-copy data path of TCP more efficiently.

2) After that, we plan to introduce the parallel communication mechanism into *BCL-4/IP*. It is a double-card system that uses two Myrinet cards to perform a higher performance since the transmission bandwidth of the current PCI Bus on our platform is still further higher than that of the raw Myrinet NIC hardware.

Furthermore, the TCP performance in heterogeneous network environment (between Linux and AIX clusters) needs to be studied in our future work too.

REFERENCES

[1] M. Baker, "Cluster Computing White Paper", *IEEE Task Force on Cluster Computing,* December 2000.

[2] J. Ma, J. He, D. Meng, and G. Li, "*BCL*-3: A High Performance Basic Communication Protocol for Commodity Superserver DAWNING-3000", *Journal of Computer Science and Technology*, vol. 16(6), pp. 522-530, 2001.

[3] A. Gallatin, J. Chase, K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds", *Proc. The 1999 USENIX Technical Conference,* June 1999.

[4] D. Meng, J. Ma, J. He, L. Xiao, Z. Xu, "Semi-User-Level Communication Architecture", *Proc. of IPDPS 2002.*

[5] J. S. Kim, K. Kim, S. I. Jung, "SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture", *Proc. of CLUSTER'01.*

[6] J. S. Chase, A. J. Gallatin, and K. G. Yocum, "End System Optimizations for High-Speed TCP", *IEEE Commun. Mag.*, vol. 39(4), pp. 68-74, April 2001.

[7] D. D. Clark, V. Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead", *IEEE Commun. Mag.*, vol. 27(6), pp. 23-29, June 1989.

[8] C. Kurmann, M. Müller, F. Rauch, T. M. Stricker, "Speculative Defragmentation – A Technique to Improve the Communication Software Efficiency for Gigabit Ethernet", *Proc. of HPDC 2000*.

[9] N.J. Boden, et al., "Myrinet – A gigabit-per-second-local-area network", *IEEE Micro*, vol. 15(1), pp. 29-38, 1995.

[10] T. von Eicken, D. Culler, S. Goldstein and K. Shauser, "Active Messages: a mechanism for integrated communications and computation", *Proc. of the International Symposium on Computer Architectures*, 1992.

[11] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet", *Supercomputing '95*, http://www.supercomp.org/sc95/proceedings/567_SPAK/SC95.HTM.

[12] M.A. Blumrich, C. Dubnicki, E.W. Felten, K. Li, and M.R. Mesarin, "Virtual Memory Mapped Network Interfaces", *IEEE Micro*, February 1995.

[13] A. Basu, M. Welsh and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces", Presented at *Hot Interconnects V*, August 1997, Stanford University and also a Cornell University, Technical Report, TR97-1620 – http://www2.cs.cornell.edu/U-Net/papers.html

[14] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high performance networking on Myrinet", In *the PC-NOW workshop, IPPS/SPDP 1998*, Orlando, USA, 1998.

[15] M. Fischer, "GMSOCKS – A Diect Sockets Implementation on Myrinet", *Proc. of CLUSTER'01*.