Hpcbench – A Linux-Based Network Benchmark for High Performance Networks

Ben Huang, Michael Bauer, Michael Katchabaw
Department of Computer Science, The University of Western Ontario
{huang|bauer|katchab}@csd.uwo.ca

Abstract

In recent years, Linux-based clusters have become more prevalent as a basis for High Performance Computing (HPC) systems. Network performance analysis is crucial to the management and administration of such clusters. To assist in this process, we developed Hpcbench [8] to measure UDP, TCP and MPI communications over high performance networks. Hpcbench records and tracks experiment results and system statistics, facilitating detailed analyses of network behaviour. In this paper, we introduce the design and prototype implementation of Hpcbench, and demonstrate Hpcbench in evaluating the network performance of three high performance interconnects in HPC clusters: Gigabit Ethernet, Myrinet, and Quadrics' QsNet.

1. Introduction

Being able to solve larger, more complex problems in a shorter period of time is a key motivating factor in building a High Performance Computing (HPC) system. Modern computers, even personal computers, are becoming more and more powerful. As well, high-speed, low-latency network products are becoming increasingly available and less expensive. It is now possible to build powerful platforms for high performance computation from off-the-shelf computers and network devices. In particular, the Linux-based commodity cluster constructed by general purpose computers is a popular model and seems to be a trend for future HPC.

Commodity cluster computing can be characterized as being cost effective, flexible, extensible, easy to maintain and to be integrated. However, since these commodity clusters are built with a variety of standalone computers and network devices, they do not necessarily guarantee high performance. The performance power of a standalone computer usually depends on its operating system, CPU and memory speed, and a variety of other factors. For HPC clusters, network communication is another key factor for cluster performance—a communication bottleneck in an HPC cluster may lead to a significant loss of overall performance.

Traditional network benchmarks do not necessarily work well in evaluating the performance of HPC environments, as the network interconnecting a cluster plays a more critical role in supporting the applications distributed across the compute nodes in the environment. Furthermore, some functionality and parameters required for detailed performance evaluations cannot be found in currently available network benchmarking tools. To better understand network behavior in HPC environments, we developed a Linux-based network benchmark tool set, named Hpcbench [8]. Hpcbench is able to accurately and effectively measure UDP, TCP and MPI communication throughput and latency in high performance networks, recording the detailed communication parameters and kernel statistics under a variety of parameters. The name "Hpcbench" was chosen as this tool set was designed specifically for HPC systems and high performance networks.

In this paper, we first provide a survey of present network benchmarking tools and related work in Section 2, and justify the need to develop a new benchmark tool for HPC environments. In Section 3, we discuss the design and implementation of Hpcbench. In Section 4, we demonstrate Hpcbench in testing and analyzing HPC networks through a collection of experiments. Our experiments were based on three of the most commonly used interconnects in HPC systems: Gigabit Ethernet, Myrinet [4] and Quadrics' QsNet [11]. Finally, in Section 5, we provide concluding remarks, and discuss directions for future work in this area.

2. Related work

To measure network performance metrics, such as throughput and latency, two types of measurement are commonly used: active and passive. Active measurement is based on introducing a workload into the network, usually through the execution of a special application, and collecting data on the performance of the workload. Typically, this is done in a client/server model, in which the client and server exchange probe packets across the network, with both the client and server collecting timing measurements on the data transferred. Passive measurement does not depend on interjecting new workloads into a network to measure its performance. Instead, it probes existing network traffic to compute various network performance metrics.

To study the high performance networks one would find in an HPC environment, it is preferable to use an active measurement model as it allows direct and repeatable measurements of network performance and behaviour without the inaccuracies or assumptions introduced by other approaches. While this can cause disruptions to the HPC environment during experimentation, this inconvenience is well worth the better quality results that can be obtained in the process. There are many existing tools available that involve active measurements as described above. Some of the most frequently used tools include Udpmon[9], Netperf[7], Iperf[5] and NetPIPE[13]. However, all of these tools were designed as general purpose network benchmarking tools, and have their own limitations and restrictions that make them unsuitable for HPC environments. For example, Udpmon measures UDP communication using hardware-dependent assembly language to access an Intel CPU cycle counter for high-precision timing, and therefore it can only be used on IA32/IA64 platforms. Iperf supports multi-threading and parallel TCP streams, but does not measure network latency. NetPIPE works well with connection-oriented protocols, such as TCP and MPI, but not UDP.

Although these tools work reasonably well for the specific tasks for which they were designed, they are limited in functionality and lack a feature set capable of supporting many of the interesting experimental scenarios for HPC environments. For example, none specialize in high performance interconnects, capable of testing all three of the most common communication protocols in commodity clusters: UDP, TCP and MPI.

3. Design and implementation of Hpcbench

With this in mind, the best option was to implement our own network benchmarking tool, Hpcbench, focusing specifically on HPC environments. Hpcbench was designed to measure the high-speed, low-latency communication networks in Linux-based HPC systems. The objectives of Hpcbench include high accuracy and efficiency; support for UDP, TCP and MPI communications; tunable communication parameters of interest in HPC environments; and detailed recording of test results, settings, and system information.

Hpcbench was written in C and uses BSD socket and MPI APIs. It is comprised of three independent sets of benchmarks measuring UDP, TCP and MPI communications. As the benchmarks are based on a common infrastructure, the implementation and usage of each benchmark are quite similar, allowing us to easily compare results for different communication protocols.

3.1. Communication model

For UDP and TCP communication tests, similar to Netperf's design, we employ a client/server model that uses two channels during testing: a control channel and a test channel. The first is a reliable TCP connection for critical data communication for controlling the test run, while the second is used for carrying test data packets (UDP or TCP). This two-channel design makes it easier to control the tests and gather results. The control channel is used by Hpcbench solely for control of the tests and only involves data transfer before and after each test; thus, it does not introduce additional traffic or overhead during actual testing. The communication model of Hpcbench is illustrated in Figure 1.

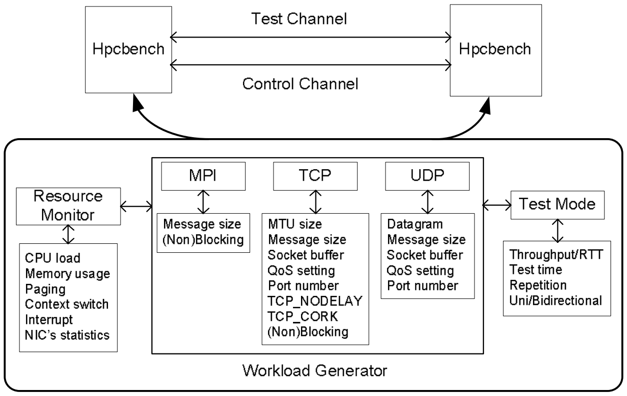

Figure 1. Hpcbench Communication Control

Another reason for two communication channels in Hpcbench is for test configuration purposes. As shown in Figure 1, Hpcbench supports many test modes for various protocols, with numerous tunable parameters for each protocol; all of this must be configured for each test. For example, some socket options, such as a socket's buffer size setting (configurable using the *setsockopt()*

function), should be set before establishing a connection for test data packets. With only one communication channel between the client and server, the server process must be initialized with a long and cumbersome argument set according to the client's test setting. Furthermore, in such a case, the server process would need to be restarted with different parameters every time the test parameters change. With two communication channels, the client is able to send all the test parameters to the server by first establishing the control channel, and then creating the test channel with the desired options. Thus, the server process does not have to be concerned with test parameter settings during initialization, and can easily have parameters changed during or between test runs without restarting.

3.2. Timers and timing

Precise timing is important for any kind of accurate performance measurement. There is currently a high resolution timer project for Linux [1] that is implementing the POSIX 1003.1b clocks and timers with nanosecond precision. The project is under continuous development, but only supports a few platforms, such as Intel platforms that include a high resolution timer (TSC register) readable by the *rdtsc* assembly instruction.

Without the general availability of a high resolution function such as *clock_gettime()* in most systems, the standard *gettimeofday()* function call is used in UDP and TCP communication timing. In MPI testing, the *MPI_Wtime()* function call is used instead, as it is designed to be able to select the best timer available in a system for MPI communication. Both methods provide microsecond resolution in most cases. In Alpha systems, however, the *gettimeofday()* function call has been found experimentally to achieve only about 1ms resolution [8].

A relatively low-precision timer may affect the accuracy of latency and throughput tests. For instance, in a Gigabit Ethernet, a round trip time can be less than 50µsec, and a 1MByte message can be sent out in less than 0.01 second. Consequently, test times must be made long enough to minimize the effects of a timer's resolution. To do so, we repeat transmissions of the same message several times when measuring network throughput and network latency. In Hpcbench, the number of transmission repetitions to configure for testing is computed by an estimation test conducted before actual testing begins.

A synchronization step is also conducted before the start of each test run to ensure that both the client and the server have a common baseline for timing. Another synchronization step is used after data transmission as necessary for unidirectional communication tests. (For further details, the reader is urged to refer to [8].)

3.3. Test logs

The more information we can obtain from testing, the better we can study and analyze the performance and the behaviour of our networks. Unlike most benchmarks that only provide basic results, Hpcbench records a considerable amount of information collected during experimentation, including all socket parameters, test time, process time, CPU and other resource utilization, interrupts, MAC layer network statistics, and so on.

To trace and log system information, virtual files in the /proc directory are parsed before and after each test. These virtual files are actually mapped from different kernel memory areas, providing a mechanism for accessing system statistics easily. By parsing these files, we are able to obtain a considerable amount of system and network information.

It is important to note, however, that recorded system information from these files is not necessarily accurately synchronized to each test because of delays in accessing and parsing the files, and the deviation of updates to those virtual files. While these statistics only provide a rough view of system performance and behaviour, they still provide much needed information to help in the analysis of test results.

4. Hpcbench network performance testing

In this section, we demonstrate the usefulness of Hpcbench in studying and analyzing the network performance of three high performance interconnects: Gigabit Ethernet, Myrinet and Quadrics' QsNet. To avoid unwelcome loads and interactions that could complicate results, all experiments were conducted using dedicated and completely idle machines in our experimental environment. All MPI communication is based on MPICH 1.2.5 [6] built with default settings.

4.1 Testbed introduction

Our testbed includes two Linux-based clusters named "mako" and "hammerhead" in SHARCNET [2], a distributed HPC Network in Ontario, Canada.

The hammerhead cluster consists of 28 Compaq ES40 Alpha SMP servers. Each server is configured with 4 x

833MHz Alpha EV6 processors, 4GB RAM, an Alteon AceNIC Gigabit Ethernet Adaptor (PCI 64bits/33MHz), a Quadrics QSW Elan3 PCI Network Adaptor, and Redhat Linux 7.2 as its operating system, with kernel 2.4.21-3.7qsnet #9 SMP. For its interconnects, it uses a Nortel Passport 8600 switch (Gigabit Ethernet) and a Quadrics' QsNet switch.

The mako cluster consists of eight HP DL360 Intel Xeon SMP servers. Each server is configured with 4 x 3GHz Intel Xeon Hyperthreading processors, 2 GB RAM, a Broadcom Tigon 3 Gigabit Ethernet Adaptor (PCI-X 64bits/100MHz), a Myricom PCI-X Network Adaptor, and Redhat Linux 9.0 as its operating system, with kernel 2.4.20-8smp #1 SMP. For its interconnects, it uses an HP ProCurve 2800 (Gigabit Ethernet) switch and a Myrinet switch.

We evaluate UDP, TCP and MPI communication over Gigabit Ethernet in this experimentation. Since TCP/IP communication is not configured for our Myrinet and QsNet environments, we instead test only their MPI communication based on the same version of MPICH.

4.2 Throughput on Gigabit Ethernet

We select two idle nodes in two clusters and test their UDP, TCP and MPI throughput over Gigabit Ethernet.

Figure 2 shows the results of UDP unidirectional throughput versus datagram size, measured by Hpcbench in an exponential test mode with default socket buffering. The Intel cluster achieved 961Mbps peak UDP throughput, while the Alpha cluster gave 530Mbps peak UPD throughput. In high throughput situations, the UDP sender's CPU load was approximately 10% in the Intel system compared to 15% for the Alpha system; the receiver's CPU load was approximately 15% in the Intel system and 20% for the Alpha system.

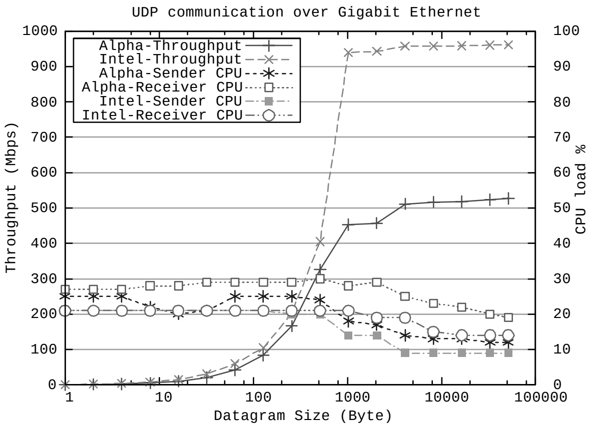

Figure 2. UDP Throughput on Gigabit Ethernet

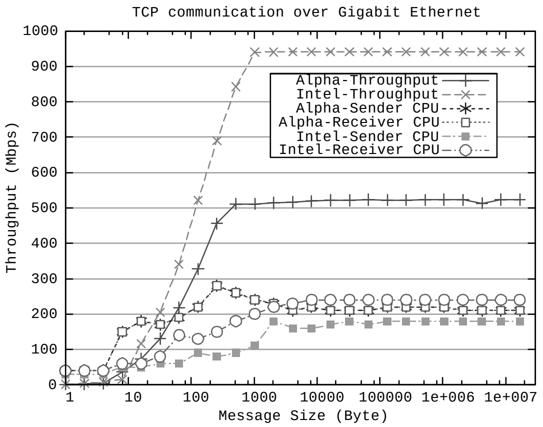

Figure 3. TCP Throughput on Gigabit Ethernet

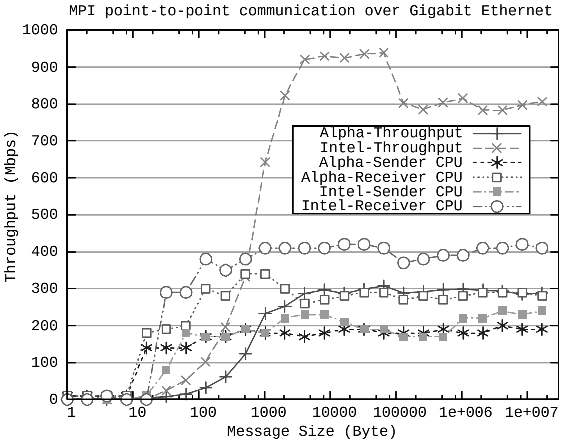

Figure 4. MPI Throughput on Gigabit Ethernet

Figure 3 shows the TCP unidirectional throughput on the Gigabit Ethernet environment as measured by Hpcbench. When message size is greater than 2KB, the Intel system delivered a stable throughput around 940Mbps and the Alpha system delivered about 520Mbps throughput. At the same time, the CPU usage varied from 18% to 29% in both systems (the receivers had slightly higher load than the senders).

Figure 4 shows the MPI point-to-point communication over Gigabit Ethernet measured by Hpcbench. The Intel system delivered a peak 938Mbps unidirectional throughput, which dropped to around 800Mbps when message size was larger than 64KB. The Alpha system only provided about 310Mbps peak MPI throughput. It is likely that the poor performance in this case came from inefficiencies in the implementation of TCP-based MPICH for that platform, because TCP communication in the same system has much higher throughput.

The above experiments demonstrated that the Intel Xeon system performed much better than the Alpha system, with higher throughput and lower CPU usage.

To help discover the cause of the observed poor performance in the Alpha cluster, we used Hpcbench to conduct further UDP tests to locate the bottleneck. UDP was used for further testing rather than TCP because UDP communication has less protocol overhead than TCP and other connection-oriented protocols, and UDP does not utilize transmission control. Usually, UDP can reflect true network behaviour more closely than TCP.

Using Hpcbench, UDP experiments with larger socket buffer sizes were conducted. This testing found that the Alpha cluster could provide a maximum 650Mpbs UDP unidirectional throughput with a better selection of buffer size. From detailed log files generated by Hpcbench, there was significant data loss was observed in the sender during the UDP processing when the socket buffer was too large (1MB, for example). At the same time, however, no data loss was observed in the network according to the log data collected by Hpcbench. Consequently, it is reasonable to believe that the bottleneck in the Alpha cluster came from the relatively slow sender. For further details, please refer to [8].

4.3 Throughput using Myrinet and QsNet

In the Intel cluster, we conducted experiments to test MPI point-to-point communication over Myrinet using Hpcbench. The results are shown below in Figure 5.

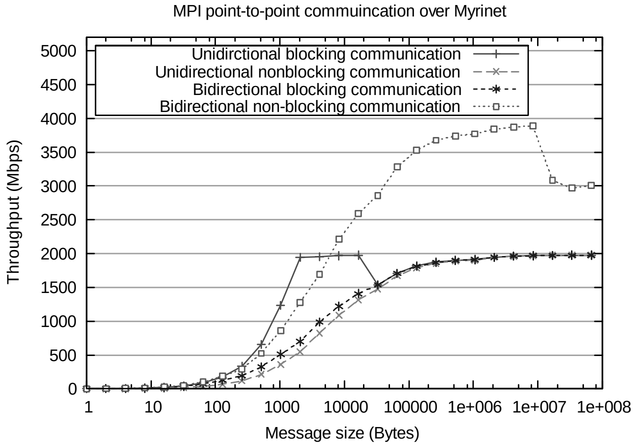

Figure 5. MPI Communication Throughput on Myrinet

As shown in Figure 5, unidirectional MPI throughput over Myrinet could reach 1995Mbps, while non-blocking (synchronized MPI_Isend and MPI_Irecv) bidirectional throughput achieved 3885Mbps. In contrast, QsNet in the Alpha cluster delivered a maximum of 1600Mbps unidirectional MPI throughput, while the non-blocking bidirectional throughput dropped to around 1250Mbps when message exceeded 500KB, as shown below in Figure 6. This may have occurred because the Alpha machines were not fast enough to handle the heavy load.

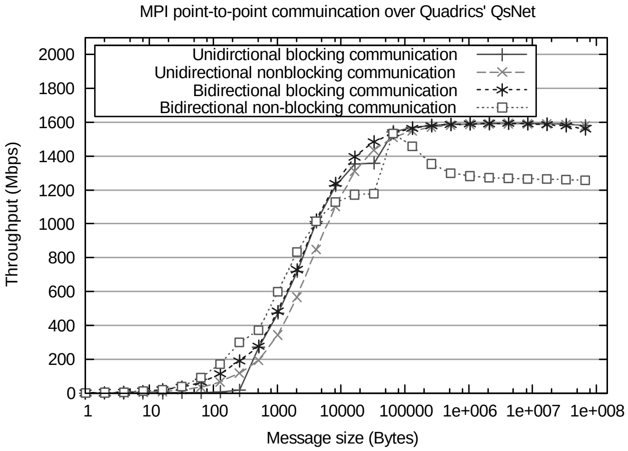

Figure 6. MPI Communication Throughput on QsNet

Consulting the data logs recorded by Hpcbench, we observed that the communication on Myrinet and QsNet only consumed a single CPU's clock cycles in a 4-processor SMP system. In fact, communicating processes completely occupied CPU1's utilization, resulting in a sharp 25% overall CPU load in both Intel and Alpha clusters. This could come from the fact that both Myrinet and QsNet technologies use a zero-copy (OS-bypassing) technique for message passing between two nodes. Unlike traditional interrupt-driven TCP/IP communication, there is no interrupt interaction between the Linux kernel and the NIC in Myrinet and QsNet communications. Instead, the data goes directly from user space into the NIC without kernel processing. Consequently, different CPUs in an SMP system are unable to cooperate to handle one communication session. During benchmarking, the sender application attempted to send as much data as possible to the NIC, and used as many CPU resources it could get to do so. A similar situation also occurred on the receiver side.

We also conducted multi-link communication experiments over Myrinet using Hpcbench. When the number of communication links exceeded the number of CPUs, we found that the Intel Xeon system had reached a 100% CPU load in both the sender and the receiver, while the overall throughput in the network remained nearly the same. On the other hand, the kernel is involved in TCP/IP communications. As a result, multi-link communication can introduce a high system load, but will not overwhelm the system for 100% usage. When the network becomes saturated or congested, the kernel will block application transmission to ensure that communication can be serviced properly.

4.4 Communication latency

High throughput does not necessarily imply low network latency. The overall performance of some applications is very sensitive to network latency. Hpcbench measures network latency in terms of round trip time (RTT) from the application layer, using various underlying transports, giving us UDP RTT, TCP RTT and MPI RTT. The traditional ping utility instead evaluates the ICMP RTT with a relatively coarse resolution (milliseconds), and may not work properly or accurately enough in a low-latency network.

Figure 7 shows the Intel Xeon cluster's RTTs for different protocols on the Gigabit Ethernet with message (datagram) size less than the Maximum Transmission Unit (MTU, 1500Byte in our experimental testbed).

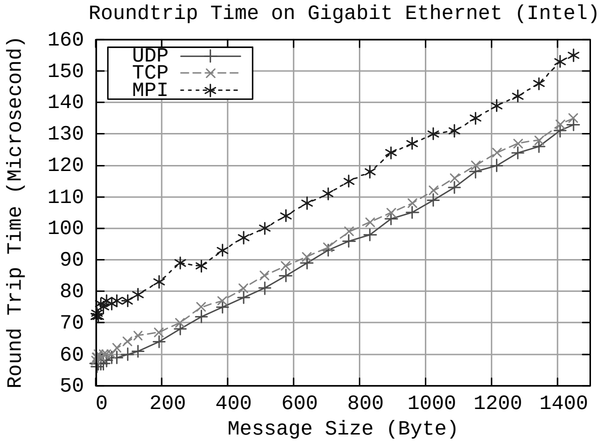

Figure 7. Gigabit Ethernet RTT in the Intel Cluster

The results show that UDP and TCP's RTTs were around 56-60µsec for tiny messages (1~32 Byte), implying around a 28µsec one way network latency. MPI RTTs, on the other hand, started from nearly 72µsec, making it a higher latency protocol than TCP and UDP in our experimentation, according to Hpcbench.

Figure 8 demonstrates that the network latency in the Alpha cluster was much higher than that of the Intel cluster according to experimentation with Hpcbench. The minimum RTTs in the Alpha cluster were about 240µsec for UDP and TCP, and about 350µsec for MPI communication.

From the statistics recorded by Hpcbench, we observed that the interrupt coalescence technique [12] was used in the Alpha cluster but not in the Intel cluster. Interrupt coalescence was used by Alpha machines' Gigabit Ethernet network cards to reduce the system load for network communication. Unfortunately, this technique can also introduce significantly larger network latency. We can also see that all curves in Figure 8 are not linearly smooth, in part due to the effect of interrupt coalescence.

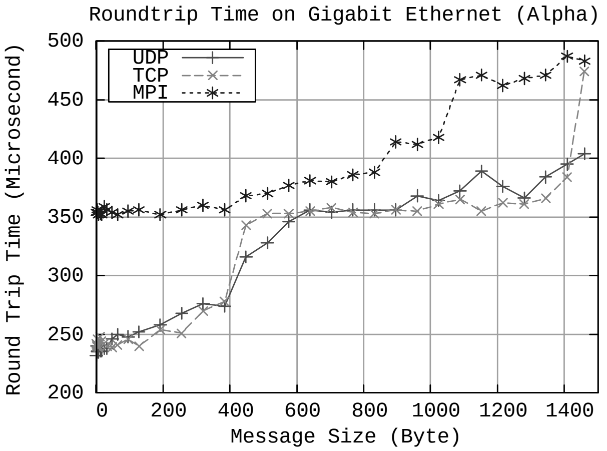

Figure 8. Gigabit Ethernet RTT in the Alpha Cluster

Figure 9 shows MPI communication latency in Myrinet and QsNet. The RTTs with tiny messages could be as low as 14µsec in both interconnects, which is significantly lower compared to Gigabit Ethernet.

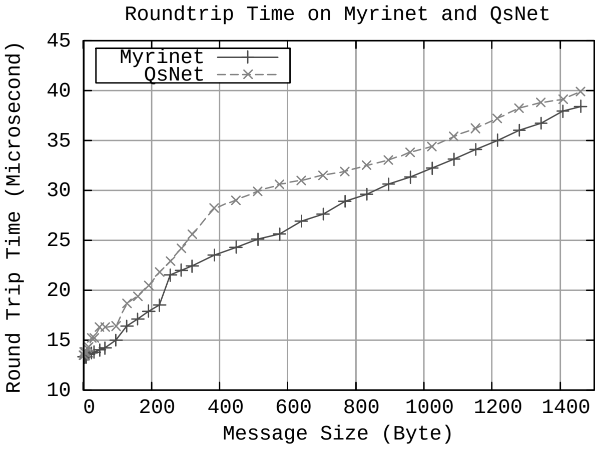

Figure 9. MPI RTT on Myrinet and Quadric's QsNet

5. Conclusions and future work

In this paper, we introduced Hpcbench, a Linux-based network measurement tool set designed for high performance networks. Hpcbench can accurately evaluate UDP, TCP, and MPI communication throughput and latency with a variety of configurable parameters. For each test, Hpcbench is able to record detailed communication settings and performance results, as well as system information, such as system load, interrupts, MAC layer statistics, and so on. This logged information can be quite helpful in analyzing and understanding network performance and behaviour, as well as the interactions between the kernel and network subsystem. Our work also provides several performance analysis methodologies for high performance network systems.

To demonstrate Hpcbench, we conducted experiments to test the network performance of Gigabit Ethernet, Myrinet and Quadrics' QsNet. In these experiments, we investigated and analyzed both throughput and latency using UDP, TCP, and MPI communication mechanisms as transports. These experiments also showcased Hpcbench's ability to tune a variety of parameters affecting overall performance, including packet size, buffering, and blocking versus non-blocking communication primitives, as well as its capacity to collect various system and network statistics and operational information of import to later analyses. The results of these experiments were quite interesting, validating the operation of Hpcbench and demonstrating its usefulness in conducting network performance analyses and measurements in HPC environments. With Hpcbench, we can uncover behaviours and relationships in our networks that are not at all obvious, yet still have an impact on application performance.

Currently, Hpcbench can be freely downloaded from its SourceForge website at: http://hpcbench.sf.net. This website also includes important documentation and test examples. By making Hpcbench available in this fashion, it is hoped that we can facilitate further research and development of improved cluster technologies.

In the future, there are many interesting directions for our work to take. While Hpcbench already supports a large number of variables and protocol options to tune for experimentation, there is always more that can be added, for example support for other MPI methods of communication besides point-to-point. Naturally, as the feature set of Hpcbench increases, more work will be required in terms of interfacing with the tool set and in managing the large sets of experiments that are possible.

Currently, the tracing of MAC layer statistics and other low-level network information is only possible with Gigabit Ethernet and other TCP/IP-based networks. For proprietary technologies, such as Myrinet and QsNet, it is possible to trace the information of the network interface card only with vendor-dependent APIs. Extensions to Hpcbench to include this support are also part of our future work.

Another interesting topic for future study with Hpcbench is the relationship between network performance and computational performance. For instance, in a Gigabit Ethernet cluster, different network interface cards can lead to different network throughput and latency. How does the computational capacity of a HPC cluster change with different network behavior introduced by different NICs or different underlying technologies? It is important to conduct this kind of experimentation to have better guidelines for building commodity HPC systems in the future.

6. Acknowledgments

We would like to thank the administrators of SHARCNET for their support in making test systems available, especially Baolai Ge and Gary Molenkamp at the University of Western Ontario, and John Morton at the University of Guelph.

7. References

[1] G. Anzinger, et al. "Functional Specification for the High-res-timers Project". *Technical Documentation*. January 2005.

[2] G. Baolai. "SHARCNET Resources for Computing and Programming". *SHARCNET HPC Workshop Series*, February 2005

[3] N. Boden, et al. "Myrinet: A Gigabit-per-Second Local Area Network". *IEEE Micro, Vol. 15, No. 1*, 1995.

[4] B. Chun et al. "Virtual Network Transport Protocols for Myrinet". *Technical Report. UC Berkeley*, 1998.

[5] M. Gates, et al. "Iperf User Docs". *Technical Documentation*. March 2003.

[6] W. Gropp, et al. "A High-performance, Portable Implementation of the MPI Message Passing Interface Standard". *Parallel Computing, 22, 6*. September. 1996.

[7] Hewlett-Packard Company, Information Networks Division. "Netperf: A Network Performance Benchmark". *Technical Documentation*. February 1995.

[8] B. Huang, "Network Performance Studies in High Performance Computing Environments". *Masters Thesis. University of Western Ontario*, Canada. March 2005.

[9] R. Hughes-Jones. "UDPMon: A Network Diagnostic Progam". *Technical Documentation*. March 2004.

[10] R. Hughes-Jones, et al. "Performance Measurements on Gigabit Ethernet NICs and Server Quality Motherboards". *First International Workshop on Protocols for Fast Long-Distance Networks*, Geneva, Switzerland, February 2003.

[11] F. Petrini et al. "The Quadrics Network (QsNet): High-Performance Clustering Technology". *IEEE Micro*, January-February 2002.

[12] R. Prasad, et al. "Effects of Interrupt Coalescence on Network Measurements". *Proceedings of Passive and Active Measurement (PAM) Workshop*, April 2004.

[13] D. Turner, et al., *Integrating New Capabilities into NetPIPE*. PVM/MPI 2003.