Netgauge: A Network Performance Measurement Framework

Torsten Hoefler[1,2], Torsten Mehlan[2], Andrew Lumsdaine[1], and Wolfgang Rehm[2]

[1] Open Systems Lab, Indiana University, Bloomington IN 47405, USA,
{htor,lums}@cs.indiana.edu
[2] Chemnitz University of Technology, 09107 Chemnitz, Germany,
{htor,tome,rehm}@cs.tu-chemnitz.de

Abstract. This paper introduces **Netgauge,** an extensible open-source framework for implementing network benchmarks. The structure of Netgauge abstracts and explicitly separates communication patterns from communication modules. As a result of this separation of concerns, new benchmark types and new network protocols can be added independently to Netgauge. We describe the rich set of pre-defined communication patterns and communication modules that are available in the current distribution. Benchmark results demonstrate the applicability of the current Netgauge distribution to to different networks. An assortment of use-cases is used to investigate the implementation quality of selected protocols and protocol layers.

1 Introduction

Network performance measurement and monitoring plays an important role in High Performance Computing (HPC). For many different network protocols and interconnects, numerous tools are available to perform functionality and correctness tests and also benchmark two fundamental network parameters (i.e., latency and bandwidth) . While these features are generally useful for system administrators and end users, for high-end performance tuning, the HPC community usually demands more detailed insight into the network.

To satisfy the demands of application and middleware developers, it is often necessary to investigate specific aspects of the network or to analyze different communication patterns. The simplest measurement method is a "ping-pong" benchmark where the sender sends a message to a server and the server simply reflects this message back. The time from the send to the receive on the sender is called Round Trip Time (RTT) and plays an important role in HPC. Another benchmark pattern, called ping-ping, can be used to analyze pipelining effects in the network (cf. Pallas Microbenchmarks [1]). However, more complicated communication patterns are necessary to simulate different communication situations, such as one-to-many or many-to-one patterns to analyze congestion situations.

The most advanced network benchmarks are parametrizing network models, such as LogP [2], LogGP [3] or pLogP [4]. These require special measurement

methods such as described in [4–7]. Those models can be used to predict the running time of parallel algorithms. The LogP model family can also be used to predict the potential to overlap communication and computation for different network protocols (cf. Bell et al. [7]).

1.1 Related Work

Existing portable tools, such as netperf or iperf, can readily measure network latency and bandwidth. However, it is often necessary to measure other network parameters (e.g., CPU overhead to assess the potential for computation and communication overlap or performance in threaded environments). Such parameters can be measured, but the tools to do so are generally specific either to the parameter or hardware being measured. HUNT[8], White's implementation[9] and COMB[10] are able to assess the overlap potential of different network protocols. Other tools like Netpipe [11, 12] simply benchmark latency and bandwidth for many different interconnection benchmarks. Another interesting tool named coNCePTuaL [13, 14] can be used to express arbitrary communication patterns, but the support for different low-level networks is limited and the addition of new networks is rather complicated. The MIBA [15] tool reports a number of different detailed timings, but is specific to the InfiniBand network.

Our work combines the advantages of previous work and enables users to easily implement their own communication patterns and use low-level communication modules to benchmark different network stacks in a comparable way. The interface of the communication modules is kept as simple as possible to ensure an easy addition of new networks or protocols.

1.2 Challenges and Contributions

After presenting a motivation and an overview about related projects, we will discuss the challenges we faced in the design and implementation phase and then highlight the main contributions of our project to the scientific community.

Challenges

The biggest challenge we faced was to design a single interface between the communication layer and the benchmark layer that supports many different communication networks. A main distinction has to be made between one-sided protocols where the sender writes directly into the passive receiver's memory and two-sided communication where the receiver fully takes part in the communication. Other issues come up when networks with special requirements, (i.e., the communication-memory must be registered before any communication can take place) need to be supported. We describe our methodology to deal with the unification of those different interfaces and networks into a single simple-to-use interface that supports easy addition of new benchmark patters. Therefore, we avoid all semantics that are not required for our benchmarking purposes (such as message tagging), even though they may be helpful or even required for real parallel applications. Furthermore, in order to reflect real-world applications as accurately as possible, the benchmark implementor must be able to "simulate" application behavior in his implementation by using the abstract communication

interface provided. Portability is achieved by limiting the number of external dependencies. Thus, Netgauge is written in plain C and needs only MPI (a portable MPI implementation is available with Open MPI [16]).

Contributions

Our main contribution to the scientific community is the design and implementation of an abstract interface to separate the communication part from the benchmark part in a network benchmark. We show that our framework is able to support many different kinds of network benchmarks (including those that simulate applications). The interface is kept as simple as possible to allow an easy addition of benchmarks (for researchers interested in performance of applications or communication algorithms) or communication modules (for researchers interested in the performance of a particular network or hardware, e.g., different parameters).

Merging those two groups makes Netgauge a useful tool for research. For example, if a new network has to be tested, the researcher needs only implement a communication module and will then immediately have all benchmarks available, including network parameter assessment of well-know network models, flow control tests and, of course, also the simple latency and bandwidth tests. Another scenario supports more theoretical research in network modeling. If a new network model is designed, the researcher simply implements the parameter assessment routine as a benchmark and thus measures the parameters for all networks supported by Netgauge.

The current version of Netgauge ships support for many communication networks and benchmark algorithms that can be used as templates for the addition of new modules. Netgauge also offers an abstract timer interface which is able to support system-dependent high-performance timers (e.g., RDTSC [17]) to enable very accurate time measurements.

2 The Performance Measurement Framework

Netgauge uses a component architecture [18] similar to LAM/MPI or Open MPI which consists of different "frameworks". A framework defines a particular interface to the user or other frameworks and a "module" is a specific instance of a framework.

Fig. 1 shows the overall structure of Netgauge. Different types of benchmarks are implemented in the "pattern" framework. The low-level communication layer is represented by the "communication" framework. The pattern implementor is free to use the functionality offered by the communication framework to implement any benchmark pattern. The communication framework abstracts the different network types. Netgauge currently supports a simplified two-sided interface similar to MPI. This interface assures an easy implementation of new communication modules.

2.1 The Pattern Framework

The pattern framework is the the core of every benchmark with Netgauge. A pattern interface is very simple and consists of a name for the pattern, a short

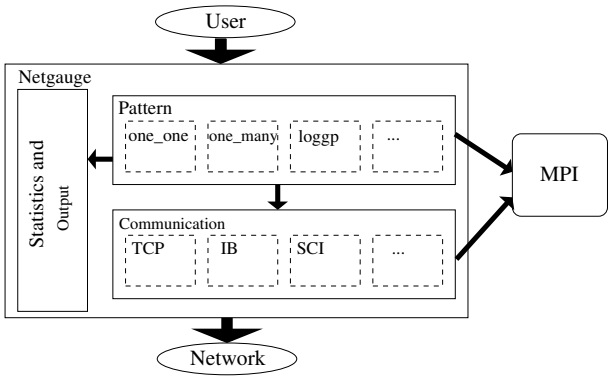

Fig. 1. Netgauge Framework Structure

description, the requirement flags and a benchmark function pointer. The user selects a specific pattern via the command line and Netgauge checks that the communication module supports all requirements indicated by the pattern's flags (e.g., if the pattern requires non-blocking communication). If so, Netgauge calls the specified module's benchmark function and passes a reference to the user-selected communication module.

2.2 The Communication Framework

The communication framework is used by the user-selected pattern module to benchmark the communication operations. The framework interface contains a name, a short description, flags and different communication functions. A module can also indicate a maximum message-size (e.g., for UDP) and how many bytes it adds as additional header (e.g., for RAW Ethernet). Different module flags indicate if the module offers reliable transport, channel semantics and/or requires memory registration. Optional `init()`, `shutdown()` and `getopt()` functions can be offered to initialize, shut the module down or read additional module-specific command line options. All optional function-pointers may be set to `NULL` which means that the module does not offer this functionality.

Every module must at least offer blocking send and receive calls. The `sendto()` function accepts destination, a buffer and a message-size as parameters. There is no tag and so the ordering of messages is significant for the message matching. The `recvfrom()` function gets a source, a buffer and a size as parameters and blocks until some data is received. The `recvfrom()` function returns the number of received bytes. Macros to send or receive a complete message (`send_all()` and `recv_all()`) are also available.

An optional non-blocking interface can be used to benchmark asynchronous communication or analyze overlap capabilities of the communication protocols. This interface is the same as the blocking interface except it contains an additional request handle. Some of the patterns that require non-blocking communication (e.g., `1:n`) will not work with communication modules that do not offer this functionality.

Memory Registration Issues

Some types of networks need registered memory areas for communication. Typically the interface to those networks exposes functions to register memory. The communication functions have to ensure data transmission only from registered memory to registered memory. Thus the communication module has a member function to allocate memory for data transmission. Patterns have to use this function instead of malloc(). The communication module has to perform all the tasks to setup memory for communication. In many cases this includes the use of some hash table to store additional information (i.e. LKEY and RKEY for InfiniBand).

2.3 Control Flow

Netgauge's main routine is the only part that interacts directly with the user. All modules for all frameworks are registered at the beginning of the program. The selected module is initialized and its parameters are passed in the module's `getopt()` function (this enable communication module implementors who add network-specific command line arguments). General internal variables like e.g., maximum message size and test repetitions are set and the user-selected pattern module is called. The pattern module performs all benchmarks and may either print the results directly or use helper functions provided by the statistics and output module to process and output data. All modules may use MPI calls (e.g., to exchange address information or parameters in order to establish initial connections). However, the implementor should try to avoid excessive MPI usage so that the modules can also be used in environments without MPI (which is supported by a basic set of modules).

2.4 Other Available Communication Patterns

Simple Microbenchmark Patterns

This section describes simple patterns that are available in Netgauge.

Pattern 1:1 The main purpose of the one–to–one communication pattern, shown in Fig. 2, is to test a bisectional bandwidth[3] of a given network. This pat-

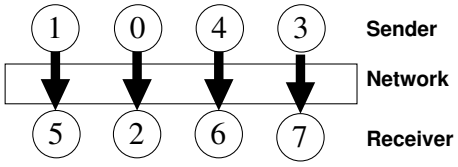

Fig. 2. The one–to–one communication pattern of Netgauge

tern communicates data between an even number of Netgauge processes. The setup stage splits all processes randomly into a group of servers and a group of clients of equal sizes. One member of each group gets associated with exactly one partner of the other group. After these steps the benchmark performs a barrier synchronization and synchronously starts sending data between all pairs

[3] the bisection is determined randomly, i.e., client/server pairs are chosen randomly

of server and client processes. Thus, one may assume that all pairs communicate approximately at the same time. The special case of two communicating processes represents the well–known ping–pong test.

Pattern 1:n and n:1 The communication patterns one–to–many and many–to–one, shown in Fig. 3 perform an asymmetric network benchmark. In contrast

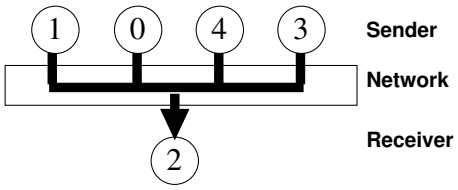

Fig. 3. The many–to–one communication pattern of Netgauge

to the most conventional approaches many processes send data to a single process or a single process sends data to many processes. This pattern is most suitable to determine the behavior of the network's flow control or congestion control. Netgauge randomly selects one of the processes to be the server and the remaining processes (n) work as clients. Before the actual test starts, a barrier synchronization is done. All client processes start sending data to the server process which receives the entire data volume of the clients and sends a reply as soon as it received all data. The current implementation uses non-blocking communication to ensure a realistic benchmark. The clients measure the time of this operation. In each subsequent round, more data is sent, up to the user specified limit. Thus, this time measurement allows the user to rate the efficiency of flow control and congestion control.

Network Model Parametrization Patterns

Model parametrization patterns are much more complex than microbenchmark patterns. They use elaborate methods to assess parameters of different network models. They are not described in detail due to space restrictions but references to the original publications which describe the methodology in detail are provided.

Pattern LogGP The LogGP pattern implements the LogGP parameter assessment method described in [19] for blocking communication. This method uses a mixture of 1:1 and 1:n benchmarks with inter-message delay to determine the parameters g, G and o as exactly as possible. The parameter L can not be measured exactly (cf. [19]).

Pattern pLogP The pLogP pattern implements the measurement of o_s and o_r in terms of Kielmann's pLogP model. Details about the measurement method can be found in [4].

2.5 Available Communication Modules

Two-sided Communication Modules

Two-sided communication modules implement two-sided communication protocols where both, the sender and the receiver are involved in the communication.

The protocol is relatively simple and maps easily to Netgauge's internal communication interface. The receiver has to post a receive and the sender a matching send. It is important to mention that the simplified module interface does not offer message tagging, i.e., the message ordering determines the matching at the receiver.

Module MPI Netgauge is able to use the well–known Message Passing Interface to perform the actual data transmission. Since MPI is frequently used in parallel programs this feature is useful for many users. The MPI module of Netgauge makes use of the conventional send and receive calls, known as point–to–point communication in the MPI specification. Blocking semantics are supported by calls to `MPI_Send()` and `MPI_Recv()`. The functions `MPI_Isend()` and `MPI_Irecv()` are used to implement the non–blocking interface. Because of its simplicity and complete implementation the MPI module serves as baseline for implementing other network modules. The semantics of the corresponding module functions of Netgauge are very close to those of MPI.

Socket-Based Modules Several modules supporting standard Unix System V socket-based communications are available. The "TCP" module creates network sockets prepared for the streaming protocol [20]. The "UDP" module implements communication with UDP [21] sockets. It is able to send data up to the maximum datagram size (64 kiB). The "IP" module of Netgauge sends and receives data using raw IP sockets [22]. The raw Ethernet ("ETH") module opens a raw socket and sends crafted Ethernet packets to the wire. The opening of raw sockets requires administrator access. Standard Posix `sendto()` and `recvfrom()` calls are used to transmit data..

Support for two special socket-based low-overhead cluster protocols is also available. The Ethernet Datagram Protocol ("EDP") and the Ethernet Streaming Protocol ("ESP") are described in [23, 24] in detail. They aim at the reduction of communication overhead in high-performance computing cluster environments.

This support for many different protocol layers inside the operation system enables the user to determine the quality of the higher-level protocol implementations (e.g., TCP, UDP) by comparing with raw device performance.

Module Myrinet/GM The Myrinet/GM ("GM") module implements an interface to the Glenn's Messages API of Myrinet [25]. It supports RDMA and send/receive over GM. Different memory registration strategies, such as registering the buffer during the send, copying the data into a pre-registered buffer and sending out of a registered buffer without copying are supported.

Module InfiniBand The communication module for the native InfiniBand [26] interface ("IB") invokes the API from the OpenFabrics project to transmit data over InfiniBand network adapters. Currently the module supports the four transport types of InfiniBand: Reliable Connection, Unreliable Connection, Reliable Datagram and Unreliable Datagram. The unreliable transport types do not protect against packet loss since additional protocol overhead would affect

the performance measurements. Blocking send posts a work queue element according to the InfiniBand specification and polls for completion of this request and returns. The blocking receive function works in a similar way.

One-sided Communication Modules

In one–sided communication the sender directly writes to the memory of the passive receiver. To test for new data, the receiver may poll for a flag, although this action does not belong to the task of pure data transmission. Usually one–sided communication requires synchronization between sender and receiver. The sender has to get the information about the memory area of the receiver. Moreover, some flag may be required to notify the receiver about the completion of the data transfer. The protocol of Netgauge maintains sequence numbers for both the sender and the receiver. The sender increments the sequence count of a public counter variable at the receiver after data transmission. Accordingly, the receiver checks the public sequence count against a local sequence count to test for completion of the data transmission. The protocol is illustrated in Fig. 4.

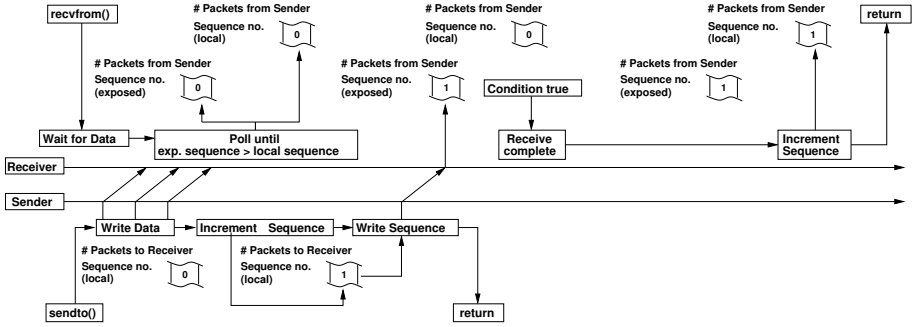

Fig. 4. The protocol for one–sided communication modules

Module ARMCI The "ARMCI" module uses the ARMCI API [27] to communicate. The blocking send interface utilizes ARMCI_Put() to copy data to the remote side and the receive polls the completion flag.

Module MPI-2 One Sided MPI 2.0 features the one–sided communication functions to access remote memory without participation of the target process [28, 29]. The corresponding Netgauge module performs data transmission using these MPI functions. Any send operation corresponds to an access epoch according to the MPI 2.0 specification. In the same way any receive operation is associated with one exposure epoch.

Module SCI The Scalable Coherent Interface ("SCI") [30] is a network type providing shared memory access over a cluster of workstations. Data is transmitted by writing and reading to and from remote memory segments mapped into the address space of a process. The SCI module of Netgauge sends data to a receiver by writing to the remote memory.

3 Benchmark Examples and Results

To demonstrate the effectiveness of Netgauge, we present a small selection of benchmark graphs that we recorded with different transport protocols and patterns.

Two test clusters were used to run the benchmarks. On system **A**, a dual Intel Woodcrest 2 GHz cluster with 1 InfiniBand HCA Mellanox MT25204 InfiniHost III and 2 Gigabit Ethernet ports driven by Intel 82563, we run all tests regarding Ethernet, InfiniBand and ARMCI. System **B**, a dual AMD Athlon MP 1.4 GHz with 1 Myrinet 2000 network card and 1 Gigabit Ethernet port SysKonnect SK-98, was used to benchmark Myrinet.

3.1 Benchmarks of the 1:1 Communication Pattern

The one–to–one communication pattern, described in Section 2.4 provides pairwise communication between an arbitrary number of processes. Fig. 5 shows the

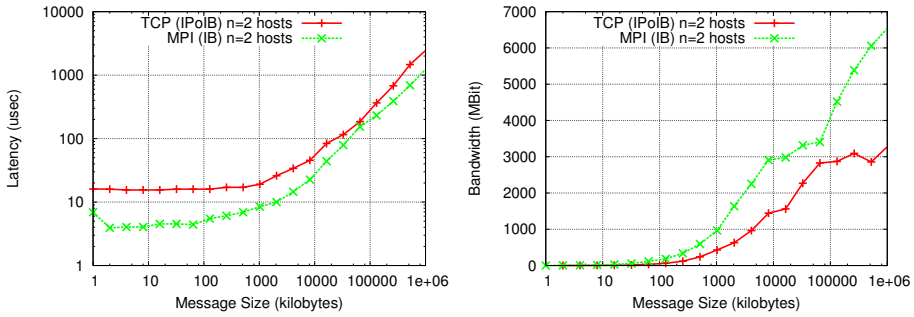

Fig. 5. Comparison of the Latency (left) and Bandwidth (right) of a one–to–one communication via MPI IB and IPoIB

latency and bandwidth measurements of an InfiniBand network. The results are given for MPI over InfiniBand and TCP using IPoIB. This benchmark allows to assess the performance and implementation quality of the different protocol layers. The slope of IPoIB transmission falls behind the performance of MPI over InfiniBand using Open MPI 1.1.3.

3.2 Benchmarks of the 1:n Communication Pattern

The 1:n communication pattern, described in Section 2.4, was used to compare the flow-control implementation of the specialized ESP [24] protocol with the traditional TCP/IP implementation. The results are shown in Fig. 6. This shows that the ESP implementation achieves a reasonably higher bandwidth and lower latency than TCP/IP for this particular communication pattern.

3.3 Benchmarks of the LogGP Communication Pattern

The LogGP communication pattern implements the LogGP parameter measurement method presented in [19]. The left diagram in the following figures shows the message size dependent overhead of the communication protocol and the right part shows the network transmission graph T. This graph can be used to

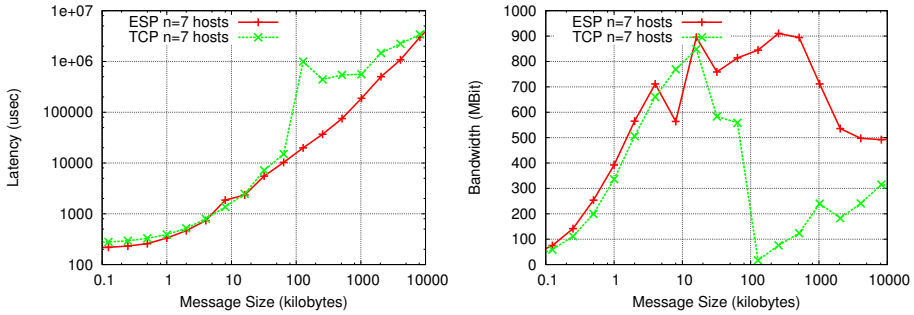

Fig. 6. Comparison of the Latency (left) and Bandwidth (right) of a congested 1:8 communication for TCP and ESP

derive the two parameters g (value at size=1 byte) and G (slope of the graph) and is discussed in detail in [19]. It results from different $PRTT$ measurements, and is calculated as $T(s) = (PRTT(n, 0, s) - PRTT(1, 0, s))/(n - 1)$ where we chose $n = 16$ and s represents the message size.

Fig. 7 compares the InfiniBand IP over IB implementation with Open MPI 1.1/OFED. We see that the overhead is as high as the network transmission time. This is due to our use of blocking communication to benchmark this results. Two different protocol regions for the MPI implementation and the pattern can be seen ($g = 1.82, G = 0.0012$ for $s < 12kiB$ and $g = 19.75, G = 0.0016$ for $s > 12kiB$). The IPoIB implementation results in $g = 7.79, G = 0.0061$.

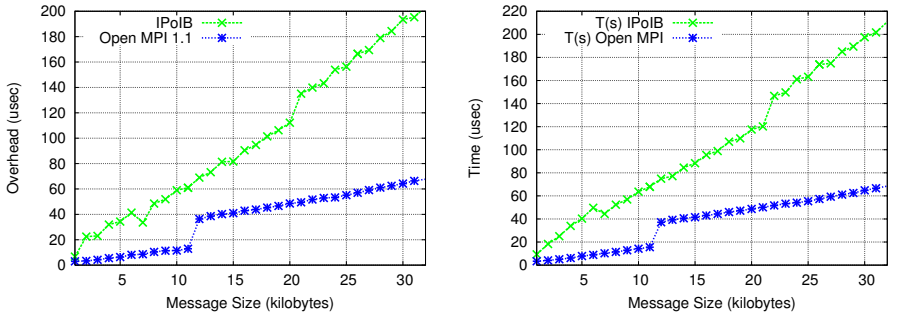

Fig. 7. LogGP graphics for different InfiniBand based transport protocols

4 Conclusions and Future Work

We introduced the Netgauge network performance analysis tool with support for many different network communication protocols. The modular structure and clear interface definition allows for an easy addition of new network modules to support additional protocols and benchmarks. Netgauge offers an extensible set of different simple and complex network communication patterns. The large number of protocol implementations enables a comparison between

different transport protocols and different transport layers. Different benchmark examples and other tests demonstrated the broad usability of the Netgauge tool.

We are planning to add new communication modules for different networks and to design more complex communication patterns.

Acknowledgments

Pro Siobhan. The authors want to thank Matthias Treydte, Sebastian Rinke, René Oertel, Stefan Wild, Robert Kullmann, Marek Mosch and Sebastian Kratzert for their contributions to the Netgauge project and Laura Hopkins for editorial comments. This work was partially supported by a grant from the Saxon Ministry of Science and the Fine Arts, a grant from the Lilly Endowment and a gift the Silicon Valley Community Foundation on behalf of the Cisco Collaborative Research Initiative.

References

1. Pallas GmbH: Pallas MPI Benchmarks - PMB, Part MPI-1. Technical report (2000)
2. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: towards a realistic model of parallel computation. In: Principles Practice of Parallel Programming. (1993) 1–12
3. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: LogGP: Incorporating Long Messages into the LogP Model. Journal of Parallel and Distributed Computing **44**(1) (1995) 71–79
4. Kielmann, T., Bal, H.E., Verstoep, K.: Fast Measurement of LogP Parameters for Message Passing Platforms. In: IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, London, UK, Springer-Verlag (2000) 1176–1183
5. Culler, D., Liu, L.T., Martin, R.P., Yoshikawa, C.: LogP Performance Assessment of Fast Network Interfaces. IEEE Micro (February 1996)
6. Iannello, G., Lauria, M., Mercolino, S.: Logp performance characterization of fast messages atop myrinet (1998)
7. Bell, C., Bonachea, D., Cote, Y., Duell, J., Hargrove, P., Husbands, P., Iancu, C., Welcome, M., Yelick, K.: An Evaluation of Current High-Performance Networks. In: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, IEEE Computer Society (2003) 28.1
8. Iancu, C., Husbands, P., Hargrove, P.: Hunting the overlap. In: PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT'05), Washington, DC, USA, IEEE Computer Society (2005) 279–290
9. III, J.W., Bova, S.: Where's the Overlap? - An Analysis of Popular MPI Implementations (1999)
10. Lawry, W., Wilson, C., Maccabe, A.B., Brightwell, R.: Comb: A portable benchmark suite for assessing mpi overlap. In: 2002 IEEE International Conference on Cluster Computing (CLUSTER 2002), 23-26 September 2002, Chicago, IL, USA, IEEE Computer Society (2002) 472–475
11. Turner, D., Chen, X.: Protocol-dependent message-passing performance on linux clusters. In: CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing, Washington, DC, USA, IEEE Computer Society (2002) 187

12. Turner, D., Oline, A., Chen, X., Benjegerdes, T.: Integrating new capabilities into netpipe. In Dongarra, J., Laforenza, D., Orlando, S., eds.: Recent Advances in Parallel Virtual Machine and Message Passing Interface,10th European PVM/MPI Users' Group Meeting, Venice, Italy, September 29 - October 2, 2003, Proceedings. Volume 2840 of Lecture Notes in Computer Science., Springer (2003) 37–44
13. Pakin, S.: Reproducible network benchmarks with conceptual. In Danelutto, M., Vanneschi, M., Laforenza, D., eds.: Euro-Par 2004 Parallel Processing, 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Proceedings. Volume 3149 of Lecture Notes in Computer Science., Springer (2004) 64–71
14. Pakin, S.: Conceptual: a network correctness and performance testing language. In: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, IEEE (April 2004) 79–89
15. Chandrasekaran, B., Wyckoff, P., Panda, D.K.: Miba: A micro-benchmark suite for evaluating infiniband architecture implementations. Computer Performance Evaluation / TOOLS (2003) 29–46
16. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (September 2004)
17. Intel Corporation: Intel Application Notes - Using the RDTSC Instruction for Performance Monitoring. Technical report, Intel (1997)
18. Squyres, J.M., Lumsdaine, A.: A Component Architecture for LAM/MPI. In: Proceedings, 10th European PVM/MPI Users' Group Meeting. Number 2840 in Lecture Notes in Computer Science, Venice, Italy, Springer-Verlag (September / October 2003) 379–387
19. Hoefler, T., Lichei, A., Rehm, W.: Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. (03 2007)
20. Postel, J.: Transmission Control Protocol. RFC 793 (Standard) (September 1981) Updated by RFC 3168.
21. Postel, J.: User Datagram Protocol. RFC 768 (Standard) (August 1980)
22. Postel, J.: Internet Protocol. RFC 791 (Standard) (September 1981) Updated by RFC 1349.
23. Reinhardt, M.: Optimizing Point-to-Point Ethernet Cluster Communication. Master's thesis, TU-Chemnitz (2006)
24. Hoefler, T., Reinhardt, M., Mietke, F., Mehlan, T., Rehm, W.: Low Overhead Ethernet Communication for Open MPI on Linux Clusters. **CSR-06**(06) (7 2006)
25. Myricom, Inc.: The GM Message Passing System. Myricom, Inc. (2000)
26. The InfiniBand Trade Association: Infiniband Architecture Specification Volume 1, Release 1.2. InfiniBand Trade Association. (2004)
27. Nieplocha, J., Carpenter, B.: ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. Lecture Notes in Computer Science **1586** (1999) 533pp
28. Gropp, W., Lusk, E., Thakur, R.: Using MPI–2 Advanced Features of the Message–Passing Interface. MIT Press (1999)
29. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface. Technical Report, University of Tennessee, Knoxville (1997)
30. IEEE standard: IEEE standard for Scalable Coherent Interface. (1993)