A Reliable Transmission Protocol for Myrinet

Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy
Institut für Programmstrukturen und Datenorganisation*,
Fakultät für Informatik, Universität Karlsruhe,
Am Fasanengarten 5, D-76128 Karlsruhe, Germany

In: Proceedings of the 2nd Workshop on Cluster-Computing, pages 135 -- 144, Karlsruhe, Germany, March 25 - 27, 1999

Abstract

This work presents a low-level communication protocol for Myrinet, which offers reliable data transmission at network interface level. The protocol is used within the ParaStation2 system, a high-performance cluster for parallel computing. Although most projects using Myrinet assume the hardware to be reliable, there is strong evidence that this assumption does not hold and reliable data transmission has to be ensured using an appropriate protocol. ParaStation2 exploits Myrinet's programmable network interface (NI) to implement link level flow control based on an ACK/NACK mechanism with timeout and retransmission.

We describe the design and implementation of ParaStation2's transmission protocol and we evaluate it performance by comparing it to two similar protocols offering reliable data transmission, namely AM-II from Berkeley [CMC97] and VMMC-II from Princeton [DBL+97].

1 Introduction

The reliable transmission protocol was developed as low-level communication layer for the ParaStation2 system, a high-performance cluster using off-the-shelf workstations and PCs. ParaStation2 is the successor of ParaStation [WBT96, War98], now using Myrinet [BCF+95] instead of the classic ParaStation as communication hardware. As the ParaStation software relies on the reliable data transmission of the ParaStation hardware, the design consideration for ParaStation2 focus on a flow control protocol to ensure reliable data transmission at network interface level, which is different to most other projects using Myrinet.

There are several approaches which use Myrinet as hardware interconnect to build parallel systems: Active Messages and the Berkeley NOW cluster, especially Active Messages-II [CMC97], Illinois Fast Messages (FM) [PLC95], the basic interface for parallelism from the University of Lyon (BIP) [PT97], the link-level flow control protocol (LFC) [BRB98a] from the distributed ASCI supercomputer, PM [TOH+98] from the Real World Computing Partnership in Japan, the virtual memory mapped communication VMMC and VMMC-II [DBL+97] from Princeton University, Hamlyn [BJM+96], the user-level network interface U-Net [BBVvE95], and Trapeze [YCGL97].

*Now: Scarasoft AG, Mühlfelder Straße 10, 82211 Herrsching, Germany. Email: {warschko, blum}@scarasoft.com

The major difference between these projects and ParaStation2 is twofold. First, ParaStation2 focuses on a variety of standardized programming interfaces, such as UNIX sockets (TCP/IP), MPI, PVM, and Java sockets and RMI with a reasonable performance at each level rather than a single purpose, nonstandard, proprietary interface which squeezes most out of the hardware for a specific application.

The second difference is due to reliability assumptions of the Myrinet hardware (see figure 1).

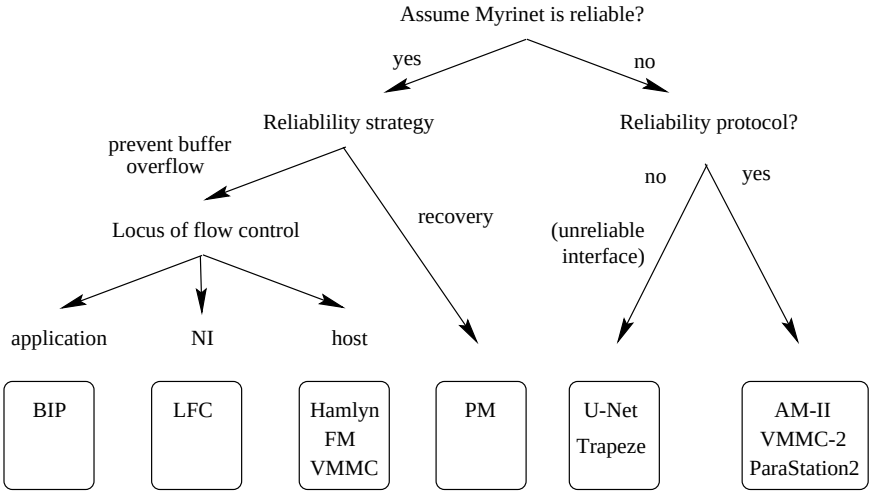

Figure 1: Myrinet and Reliability (from [BRB98b])

Most approaches assume Myrinet to be reliable or pass the unreliability on to the application layer. Only AM-II, VMMC-2 and ParaStation2 accept the unreliability of Myrinet and provide mechanisms to ensure reliable data transmission. The reason why most projects assume Myrinet to be reliable is mainly due to the rather low error rate at hardware level. In contrast to that, we've observed that the Myrinet link-level flow control mechanism seems to fail by overwriting or dropping complete packets under certain circumstances. The only way to detect this behaviour is to count packets or to use sequence numbers within packets, because the hardware neither blocks the transmission nor signals any error. Furthermore, the hardware does not distinguish between data and control packets while dropping one of these. Thus, a simple flow control protocol to prevent buffer overflow assuming that control packets will be delivered reliably is not sufficient to ensure reliable transmission. Although AM-II [CMC97] and VMMC2 [DBL+97] do not explicitly state problems with the Myrinet hardware, they introduced a protocol to ensure reliable communication as they switched from AM to AM-II or VMMC to VMMC2 respectively. The same holds for ParaStation2 which also started using strict back-blocking until serious problems arose.

2 Myrinet and Reliability

Although many systems assume Myrinet to be reliable, there are four sources of unreliability within a Myrinet system:

1. Message corruption at link level.

2. Loss of packets due to insufficient buffer space.
3. Myrinet's timeout mechanism to detect 'dead' links.
4. Failure of Myrinet's hardware flow control mechanism.

Item 1: Myrinet has a very low error rate, but the risk of a packet getting corrupted still exists. The hardware is able to detect this kind of failure by using Myrinet's build in CRC checksum. Handling a CRC error is different: either it's considered as 'fatal event' (because it's unlikely to happen), or a higher-level protocol takes care and arranges a retransmission.

Item 2: Even if the network is fully reliable, the software protocol may still drop packets due to insufficient buffer space. In fact, this is the most common cause of packet loss, because each protocol needs communication buffers on both the host and the NI, and both are scarce resources. Handling this kind of error can be accomplished in one of two ways: either *recover* from buffer overflow by retransmitting dropped packets, or *prevent* buffer overflow by using a credit scheme to stop the sender before the overflow happens.

Item 3: If a Myrinet component fails to consume an incoming packet from the network for the duration of a predefined timeout period[1] the packet is discarded. This mechanism is used to detect 'dead' links and components to prevent the network from stalling. But the hardware is unable to distinguish between a 'dead' link and a link that is busy because of heavy traffic in the network or because of a back-blocking situation. Thus, the timeout mechanism may be invoked even if all components are fully operational. Although it is possible to detect this kind of failure (the NRES bit in the control register is asserted), it is hard to recover unless the protocol is prepared to handle any kind of packet loss (data and control packets).

Item 4: Myrinet provides a hardware based flow control mechanism at link level, but this mechanism only seem to work properly during one data transmission and not between multiple consecutive transmissions of packets. So it is possible to send any number of packets to a receiver which is unwilling or unable to accept incoming packets. This situation may be detected by the timeout mechanism above (see item 3), but if the receiver stalls less than the predefined timeout period, there is a high probability that one of the pending packets gets corrupted or is even lost without any hardware supplied notification. In this situation it is up to the transmission protocol to detect the failure (e.g. using sequence numbers) and to initiate an appropriate response to ensure a reliable data transmission.

Many Myrinet protocols (left side of figure 1) only consider buffer management (item 2) as necessary to provide sufficient quality of service for their programming interfaces. CRC errors (item 1) are unlikely to happen and treated as 'fatal events', network resets (item 3) are turned off, and packet losses as described within item 4 are not detected and therefore not handled at all. If this happens, it's up to the user to detect and react accordingly. This situation is unsatisfactory for the ParaStation2 system, and thus we started to implement a reliable transmission protocol which is able to handle any of the four unreliability sources mentioned above.

3 Design and Implementation

This section explains the basic ideas of our reliable transmission protocol, how it works and how it is implemented.

[1] between 62.5 ms and 4 sec.

3.1 Basic Idea

The basic idea of our reliable transmission protocol is that every data packet has to be acknowledged using a technique called *positive acknowledgement with retransmission*. This technique is well known and used as basic principle within the TCP protocol [Com91, Tan89]. The sender keeps a record of each packet it sends in one of its transmission buffers and waits for an acknowledgement before it releases the buffer. The sender also starts a timer when it sends a packet and retransmits the packets if the timer expires before an acknowledgement arrives. To achieve better performance our protocol uses multiple buffers and allows multiple outstanding ACK's, a technique known as *sliding windows* [Com91, Tan89]. The current implementation of our protocol uses 8-bit sequence numbers and a window size of 8 packets. In case of a corrupted or lost packet (or ACK) our protocol implements a *go back N* behaviour, retransmitting the first unacknowledged and all subsequent packets, rather than using a *selective retransmission* strategy. In contrast to the TCP protocol, the ParaStation2 protocol also uses negative acknowledgements (NACK). In case of insufficient buffer space (see item 2 in section 2) the receiver sends a NACK back to the sender to prevent further message transmission. As usual, the NACK stops transmission of further packets and triggers the retransmission of the rejected packet(s).

3.2 Basic operation

Figure 2 shows the basic operation during message transmission of the ParaStation2 protocol. The basic protocol has four independent parts: (a) the interaction between the sending application and the sender network interface (NI), (b) the interaction between the sending and the receiving NI, (c) the interaction between the receiving NI and the receiving host, and (d) the interaction between the receiving application and the host.

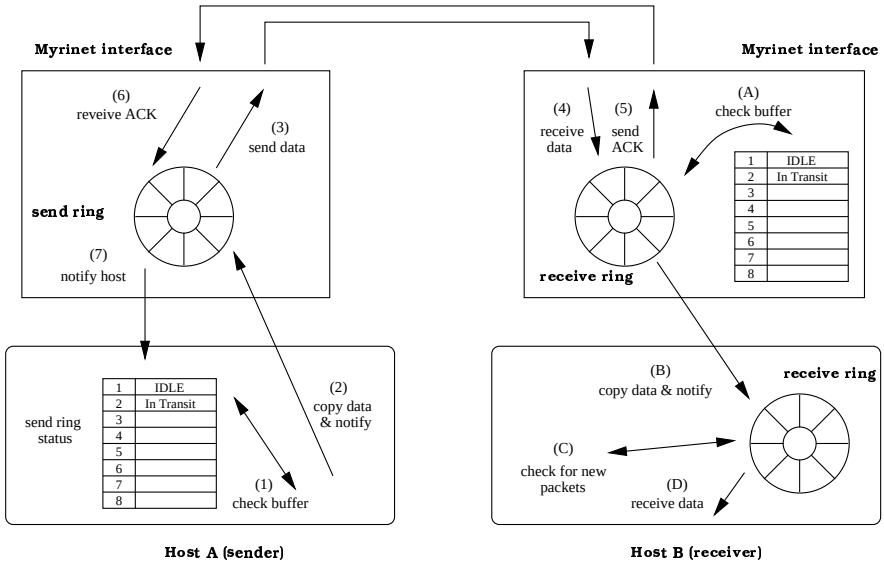

Figure 2: Data transmission in ParaStation2

First, the sender checks if there is a free send buffer (step 1). This is accomplished by a simple table lookup in the host memory, which reflects the status of the buffers of the send

ring located in the fast SRAM of the network interface (Myrinet adapter). If there is buffer space available, the sender copies (step 2) the data to a free slot of the circular send buffer located in the network interface (NI) using programmed I/O. Afterwards the NI is notified (a descriptor is written) that the used slot in the send ring is ready for transmission and the buffer in host memory is marked as in transit. A detailed description of the buffer handling is given in section 3.3. In step (3), the NI sends the data to the network using its DMA engines.

When the NI receives a packet (step 4) it stores the packet in a free slot of the receive ring using its receive DMA engine. The flow control protocol ensures that there is at least one free slot in the receive ring to store the incoming packet. Once the packet is received completely and if there's another free slot in the receive ring, the flow control protocol acknowledges the received packet (step 5). The flow control mechanism is discussed in section 3.4. As soon as the sender receives the ACK (step 6), it releases the slot in the send ring and the host is notified (step 7) to update the status of the send ring.

In the receiving NI the process of reading data from the network is completely decoupled from the transmission of data to the host memory. When a complete packet has been received from the network, the NI checks for a free receive buffer in the host memory (step A). If there is no buffer space available, the packet will stay in the NI until a host buffer becomes available. Otherwise the NI copies the data into host memory using DMA and notifies the host about the reception of a new packet by writing a packet descriptor (step B). Concurrently, the application software checks (polls) for new packets (step C) and eventually, after a packet descriptor has been written in step (B), the data is copied to application memory (step D).

Obviously, the data transmission phases in the basic protocol (step 2, 3, 4, and B) can be pipelined between consecutive packets. The ring buffers in the NI (sender and receiver) are used to decouple the NI from the host processor. At the sender, the host is able to copy packets to NI as long as there is buffer space available although the NI itself might be waiting for acknowledgements. The NI uses a transmission window to allow a certain amount of outstanding acknowledgements which must not necessarily equal the size of the send ring. At the receiver the NI receive ring is used to temporarily store packets if the host in not able to process the incoming packets fast enough.

3.3 Buffer handling

Each buffer or slot in one of the send or receive rings can be in one of the following states:

`IDLE`: The buffer is empty and can be used to store a packet (up to 4096 byte).

`INTRANSIT`: This buffer is currently involved in a send or receive operation, which has been started but which is still active.

`READY`: This buffer is ready for further operation either a send to the receiver NI (if it's a send buffer) or a transfer to host memory (if it's a receive buffer).

`RETRANSMIT`: This buffer is marked for retransmission, because of a negative acknowledgement or a timeout (send buffer only).

Figure 3 shows the state transition diagrams for both send and receive buffers in the network interface.
At the sender the NI waits until a send buffer becomes `READY`, which is accomplished by the host after it has copied the data and the packet descriptor to the NI (step 2 in figure 2). After

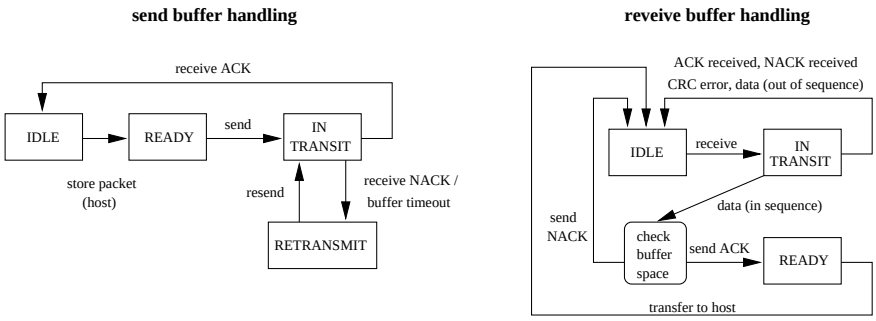

Figure 3: Buffer handling in sender and receiver

the buffer becomes `READY` the NI starts a send operation (network DMA) and marks the buffer `INTRANSIT`. When an acknowledgement (ACK) for this buffer arrives (step 6 in figure 2), the buffer is released (step 7) and marked `IDLE`. If a negative acknowledgement (NACK) arrives or the ACK does not arrive in time (or gets lost) the buffer is marked for retransmission (`RETRANSMIT`). The next time the NI tries to send a packet it sees the `RETRANSMIT` buffer and resends this buffer, changing the state to `INTRANSIT` again. This `RETRANSMIT` – `INTRANSIT` cycle may happen several times until an ACK arrives and the buffer is marked `IDLE`.

At the receiver the buffer handling is quite similar (see figure 3). As soon as the NI sees an incoming packet it starts a receive DMA operation and the state of the associated buffer changes from `IDLE` to `INTRANSIT` (see step 4 in figure 2). Assuming that the received packet contains user data, is not corrupted, and has a valid sequence number[2] the NI checks for another free buffer in the receive ring. If there is another free buffer it sends an ACK back to the sender and the buffer is marked `READY`. Otherwise a NACK is sent, the packet discarded and the buffer released immediately (marked `IDLE`). The check for a second free buffer in the receive ring ensures that there is at least one free buffer to receive incoming packets anytime, because any packet eating up the last buffer will be discarded. When the received packet contains protocol data (ACK or NACK), the NI processes the packet and releases the buffer. In case of an error (CRC) the buffer is marked `IDLE` immediately without further processing. If the received data packet does not have a valid sequence number, the packed is discarded and the sender is notified by sending a NACK back. Thus the receiver refuses to accept out-of-sequence data and waits until the sender will resend the missing packet.

3.4 Flow control protocol

ParaStation2 uses a flow control protocol with a fixed sized transmission window and 8 bit sequence numbers (related to individual sender/receiver pairs), where each packet has to be acknowledged (either with a positive or a negative acknowledgement) in combination with a timeout and retransmission mechanism in case that an acknowledgement gets lost or does not arrive within a certain amount of time. The protocols assumes the hardware to be `unreliable` and is able to track any number of corrupted or lost packets (containing either user data or protocol information). Table 1 gives an overview of possible cases within the protocol, an explanation of each case as well as the resulting action initiated.
When a data packet is received, the NI compares the sequence number of the packet with

[2]For a discussion of the ACK/NACK protocol see section 3.4.

packet type	sequence check	explanation	resulting action
DATA	$<$	lost ACK	resend ACK
	$=$	ok	check buffer space (see fig 3)
	$>$	lost data	ignore & send NACK
ACK	$<$	duplicate ACK	ignore packet
	$=$	ok	release buffer
	$>$	previous ACK lost	ignore packet
NACK	none		mark buffer for retransmission
CRC	none	error detected	ignore packet

Table 1: Packet processing within receiver

the assumed sequence number for the sending node. If the numbers are equal, the received packet is the one that is expected and the NI continues with its regular operation. A received sequence number smaller than expected indicates a duplicated data packet caused by a lost or late ACK. Thus the correct action to take is to resend the ACK, because the sender expects one. Is the received sequence number larger than expected, the packet with the correct sequence number has been corrupted (CRC) or lost. As the protocol does not have a selective retransmission mechanism the packet is simply discarded and the sender is notified with a negative acknowledgement (NACK). Thus, this packet will be retransmitted later either because the sender got the NACK, or because of a timeout. As the missing packet also causes a timeout at the sending side, the packets will eventually arrive in the correct order.

On the reception of an ACK packet, the NI also checks the sequence number and if it is ok it continues processing and releases the acknowledged buffer. If the received sequence number is smaller than assumed, we've received a duplicated ACK because the sender ran into a transmission timeout before the correct ACK was received and the receiver has resent an ACK upon the arrival of an already acknowledged data packet[3]. The response in this case is simply to ignore the ACK. A received sequence number larger than what is expected indicates that the correct ACK has been corrupted or lost. Thus the action taken is to ignore the ACK, but the associated buffer is marked for retransmission to force the receiver to resend the ACK. The buffer associated with the assumed (and missing) ACK will timeout and be resent which also forces the receiver to resend the ACK.

A received NACK packet does not need sequence checking; the associated buffer is marked for retransmission as long as it is in the `INTRANSIT` state. Otherwise the NACK is ignored (the buffer is in `RETANSMIT` state anyway). In case of an CRC error the packet is dropped immediately and no further action is initiated, because the protocol is unable to detect errors in the protocol header.

The resulting protocol is able to handle any number of corrupted or lost packets containing either user data or protocol information, as long as the NI and the connection between the incorporated nodes is working. The protocol was developed to ensure reliability of data transmission at NI level, not to handle hardware failures in terms of fault tolerance. The protocol itself can be optimized in some cases (e.g. better handling of ACK's with a larger

[3]This case may sound strange, but we've seen this behaviour several times.

sequence number), but this is left to future implementations. In comparison to existing protocols, this protocol can roughly be classified as a variation of the TCP protocol (using NACK's and a fixed size transmission window).

4 Performance

Table 2 compares the performance of the ParaStation2 protocol to VMMC-2 and AM-II, which both use a reliable transmission protocol for Myrinet.

System	Latency [μs]	Throughput [MByte/s]
ParaStation2	14.5 – 18	65 – 90
VMMC-2	13.4	90
AM-II	21	31

Table 2: Performance comparison between reliable systems

The communication latency of ParaStation2 is between 14.5μs and 18μs (platform dependent, see table 3) and compares well to the 13.4μs of VMMC-2 (Intel/PCI/Linux platform) and the 21μs of AM-II (Sun Sparc/SBUS/Solaris platform). ParaStation2's 65 MByte/s to 90 MByte/s throughput is as high as the 90 MByte/s of VMMC-2 (using the same platform, see table 3), and two to three times as high as AM-II (31 MByte/s). The low performance for AM-II is caused by the Sparc/SBUS interface and not due to the AM-II transmission protocol.

In table 3, performance figures of all software layers in the ParaStation2 system are presented. The various levels presented are the hardware abstraction layer (HAL), which is the lowest layer of the hierarchy, the so called *ports* and TCP layers, which are build on top of the HAL, and standardized communication libraries such as MPI and PVM, which are optimized for ParaStation2 and build on top of the ports layer. Latency is calculated as round-trip/2 for a 4 byte ping-pong and throughput is measured using a pairwise exchange for large messages (up to 32K). N/2 denotes the packet size in bytes when half of the maximum throughput is reached. The performance data is given for three different host systems, namely a 350MHz Pentium II running Linux (2.0.35), a 500MHz and a 600MHz Alpha 21164 system running Digital Unix (4.0D).
The latency at HAL level of 14.5μs to 18μs is somewhat higher than for systems which do not ensure reliable data transmission such as LFC (11.9μs) or FM (13.2μs) [BRB98a]. This is because neither LFC nor FM copies the data it receives to the application and second, both LFC and FM incorrectly assume Myrinet to be reliable. The 90 Mbyte/s throughput of ParaStation2 for the Intel platform is between FM (up to 60 MByte/s), LFC (up to 70 MByte/s), PM (90 MByte/s), and BIP (up to 125 MByte/s) [ABD+98].

Switching from a single-programming environment (HAL) to multi-programming environments (upper layers) results in a slight performance degradation regarding latency as well as throughput. The reason for increasing latencies is due to locking overhead to ensure correct interaction between competitive applications. The decreased throughput is caused by additional buffering, a complex buffer management, and locking overhead.

System	Measurement	Programming interface HAL	Ports	TCP	MPI	PVM
Pentium II	Latency [μs]	14.5	18.7	20.2	25	30
350 MHz	Throughput [MByte/s]	90	78	76	73	58
	N/2 [Byte]	512	1000	1000	2000	2000
Alpha 21164	Latency [μs]	17.5	24	24	30	29
500 MHz	Throughput [MByte/s]	65	55	57	50	49
	N/2 [Byte]	512	500	500	1000	1000
Alpha 21164	Latency [μs]	18.0	24	25	27	32
600 MHz	Throughput [MByte/s]	75	65	71	62	57
	N/2 [Byte]	1024	1000	1000	2000	2000

Table 3: Basic performance parameters of ParaStation2

5 Conclusion and further work

In this paper we've presented the design of ParaStation2's low level protocol to ensure reliable data transmission at network interface level. The advantage of this approach was that we could reuse the ParaStation code with minor changes and getting the complete functionality of the ParaStation system (especially the variety of standardized and well-known interfaces) for free.

The evaluation of ParaStation2 shows that ParaStation2 compares well with other approaches in the cluster community using Myrinet. ParaStation2 is not the fastest systems in terms of pure latency and throughput, but in contrast to most other approaches it offers the reliable interface which is – in our experience – more important to the user than an ultra high-speed, but unreliable interface.

The future plans for ParaStation2 are to optimise the interface between the software and the Myrinet hardware to get even more performance out of the system. Second, ports to other platforms such as Sparc/Solaris and Alpha/Linux are on the way.

References

[ABD+98] Soichiro Araki, Angelos Bilas, Cezary Dubnicki, Jan Edler, Koichi Konishi, and James Philbin. User-space communication: A quantitative study. In ACM, editor, *SC'98: High Performance Networking and Computing: Proceedings of the 1998 ACM/IEEE SC98 Conference: Orange County Convention Center, Orlando, Florida, USA, November 7–13, 1998*. ACM Press and IEEE Computer Society Press, November 1998.

[BBVvE95] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-net: A user-level network interface for parallel and distributed computing. In *Proc. of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado*, December 3-6, 1995.

[BCF+95] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jarov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[BJM+96] G. Buzzard, D. Jacobson, M. MacKey, S. Marovich, and J. Wilkes. An Implementation of the Hamlyn Sender-Managed Interface Architecture. In *The 2nd USENIX Symp. on*

Operating Systems Design and Implementation, pages 245–259, Seattle, WA, October 1996.

[BRB98a] Raoul A. F. Bhoedjang, Tim Rühl, and Henri E. Bal. LFC: A Communication Substrate for Myrinet. In *Fourth Annual Conference of the Advanced School for Computing and Imaging*, Lommel, Belgium, June 1998.

[BRB98b] Raoul A. F. Bhoedjang, Tim Rühl, and Henri E. Bal. User-Level Network Interface Protocols. *IEEE Computer*, 31(11):52–60, November 1998.

[CMC97] B. Chung, A. Mainwaring, and D. Culler. Virtual Network Transport Protocols for Myrinet. In *Hot Interconnects'97*, Stanford, CA, April 1997.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP. Volume I: Principles, Protocols, and Architecture.* Prentice-Hall International Editions, 1991.

[DBL+97] C. Dubnicki, A. Bilas, K. Li, , and J. Philbin. Design and Implementation of Virtual Memory-Mapped Communication on Myrinet. In *11th Int. Parallel Processing Symposium*, pages 388–396, Geneva, Switzerland, April 1997.

[PLC95] Scott Pakin, Mario Lauria, and Andrew Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, California, December 3-8 1995.

[PT97] L. Prylli and B. Tourancheau. Protocol Design for High Performance Networking: A Myrinet Experience. Technical Report 97-22, LIP-ENS Lyon, July 1997.

[Tan89] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall International Editions, second edition, 1989.

[TOH+98] H. Tezuka, F. O'Carrol, A. Hori, , and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *12th International Parallel Processing Symposium*, pages 308–314, Orlando, Florida, Mar 30 - Apr 3, 1998.

[War98] Thomas M. Warschko. *Effiziente Kommunikation in Parallelrechnerarchitekturen.* PhD thesis, Universität Karlsruhe, Fakultät für Informatik, March 1998. In: VDI Fortschritt-Berichte, Reihe 10, Nr. 525, VDI-Verlag, ISBN: 3-18-352510-0.

[WBT96] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. The ParaStation Project: Using Workstations as Building Blocks for Parallel Computing. In *Proceedings of the International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'96), New Horizons*, pages 375–386, Sunnyvale, California, USA, August 9–11, 1996. Best paper award.

[YCGL97] K. Yocum, J. Chase, A. Gallatin, and A. Lebeck. Cut-Through Delivery in Trapeze: An Exercize in Low-Latency Messaging. In *The 6th Int. Symp. on High Performance Distributed Computing*, Portland, OR, August 1997.