Locabus: A Kernel to Kernel Communication Channel for Cluster Computing

Paul Werstein, Mark Pethick, and Zhiyi Huang

Department of Computer Science
University of Otago
Dunedin, New Zealand
wersteinⓐcs.otago.ac.nz, mpethickⓐcs.otago.ac.nz, hzyⓐcs.otago.ac.nz

Abstract. This paper proposes a kernel to kernel communication system for use in cluster computers. It is implemented directly on the Ethernet data link layer. This allows use of Ethernet's inherent broadcast capability. This system is implemented and performance tests are run. The results show significant improvement in broadcast performance.

1 Introduction

Many of the operations of cluster computers and the parallel processes running on them rely on some message passing system. The underlying network protocol for such message passing is usually TCP/IP or UDP/IP. The goal of this research is to develop a high performance, reliable, lightweight communications channel for use in cluster computers. This channel makes use of Ethernet's broadcast capability as well as providing point-to-point communications.

In the cluster environment, we can make optimizations over TCP/IP since there is no need for routing. Ethernet switches are commonplace, and we can assume a highly reliable network where packet loss is rare.

The proposed communications channel is called Locabus. It provides a reliable, connectionless, datagram-based service for broadcast traffic within a cluster computer environment. It also provides for point-to-point communications. It is implemented as a kernel module for the Linux kernel.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 give the details of the Locabus implementation. A preliminary set of performance results are given in Section 4. Finally, we give conclusions and directions for further research in Section 5.

2 Related Work

Many reliable broadcast and multicast protocols have been proposed for wide area networks. Examples of broadcast protocols include [1, 2]. Examples of multicast protocols are [3–8]. In contrast, this research concentrates on broadcast in a cluster of computers. Such an environment allows us to take advantage of the characteristics of the environment including rather reliable transmission, short delays, and lack of need for routing decisions.

Lane, Daniels, and Yuan in [9] study a cluster environment, but their studies are based on implementing multicast using the UDP interface. In this research, our approach is implemented directly on the Ethernet interface rather than using UDP/IP. This approach contrasts with [10] which implements MPI by modifying the Ethernet device driver.

Four types of protocols can be used to assure reliability of broadcast or multicast transmissions: sender-initiated, receiver-initiated, tree-based, or ring-based protocols [11]. In the sender-initiated approach, a message is broadcast and each receiver unicasts an ACK to the sender. This leads to the ACK implosion problem with a large number of nodes. In the receiver-initiated approach, receivers inform the sender when they detect an error or missing packet. Quite often a polling technique is used to determine periodically that receivers are alive and have received all messages. In a tree-based protocol, nodes are divided into groups with each group having a leader. The group leader is responsible for reliable delivery within the group and informing the sender of errors. In a ring-based protocol, receivers take turns being the token site. The token site is responsible for sending an ACK to the sender. Receivers send NAKs to the token site when they detect an error. Since the token site rotates among the receivers, the sender can determine the status of the receivers.

3 Locabus

In designing the Locabus communications system, certain assumptions are made to take advantage of the cluster environment. These assumptions include:

– The underlying network is Ethernet. This allows use of its broadcast capabilities. The network is unreliable, but the error rate is very low.
– An Ethernet switch is used so collisions are rare.
– All nodes in the cluster are on a private network with a gateway to any public network. The gateway takes care of authentication of any outside communications. Thus there is no need for authentication or encryption between nodes. There also is no need for routing.
– The underlying operating system is homogenous. In this case, Linux is used. It is not necessary that the machines be homogenous in capabilities or processor type.

To minimise administration, Locabus employs a distributed, self-discovery technique. As nodes are booted in the cluster, they periodically broadcast a discovery message for up to one minute. The existing nodes reply with a list of the nodes they know about. Thus each node learns about the presence of all other nodes.

Timestamps are used to detect reconnects. A reconnect causes other nodes to reset the information they maintain about the reconnected node. This action provides for an orderly recovery from a node which fails and recovers. A probe function is provided to deal with missing nodes.

The Locabus packet header is shown in Table 1. The *type* field specifies the type of packet. Currently implemented types include: data, ACK, NAK, connection establishment. The *proto* field specifies the protocol for the packet. The currently implemented protocols are described in Section 3.2. The *node_to* and *node_from* fields specify the source and destination for the packet. This allows for 65,535 nodes in a cluster. 0xFFFF is the address used for broadcast. Currently, the lower 16 bits of the IP address are used for these fields since they are unique in the cluster and require no additional administration.

Table 1. Locabus packet

Field	Type	Description
type	__u8	Packet type
proto	__u8	Application protocol identifier
node_to	__u16	Recipient node (0xFFFF = broadcast)
node_from	__u16	Sending node
seq_num	__u16	Packet Sequence Number
ack_num	__u16	Ack num if ack packet
len	__u16	Packet length including header
window	__u16	Current credit window of sender
csum	__u16	16 bit checksum of data + header

Seq_num is the packet sequence number. Locabus numbers packets, not bytes as in TCP/IP. *Ack_num* is an acknowledgment number if the packet is performing acknowledgment. Piggyback ACKs are used when possible. The total length of the packet including the header is specified in the *len* field. *Window* is used to implement a TCP/IP-style credit system for the number of unacknowledged packets allowed. Finally, *csum* is a 16 bit checksum of the data and header. The data follows the header. The data size can be up to 1484 bytes to maintain compatibility with the underlying Ethernet network.

The Locabus system uses a 3-way handshake to establish connections. The disconnect is done by a 2-way handshake. The kernel maintains 160 bytes of state data per connection in contrast to 360 bytes required for TCP.

3.1 Flow Control

When doing point-to-point or unicast communication, a go-back-N flow control scheme is employed. The window size is 1024 with an acknowledgment time of 0.2 seconds. Locabus uses 2 levels of cache for socket buffers. The first level is a set of per node buffers with the hardware headers precomputed. The second level is a set of general buffers.

For broadcast communication, Locabus uses a NAK based protocol with polling and random acknowledgment. This design prevents the ACK implosion problem associated with ACK based broadcast and is simpler to implement than a tree based or ring based protocol. It also reduces the workload on the receivers.

3.2 Implementation Details

Figure 1 shows the overall design of the system. The basic Locabus system is implemented as a kernel module. This is shown in the figure as the *mod_locabus* box.

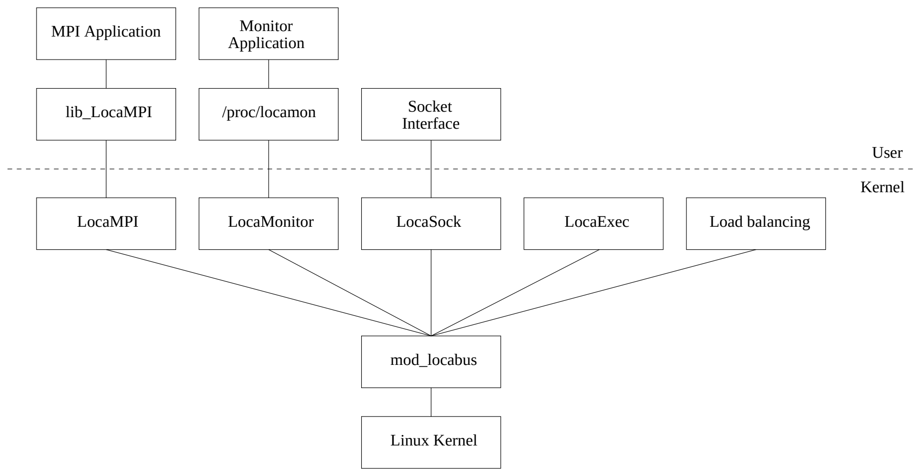

Fig. 1. Design of Locabus

Currently two protocols are implemented using Locabus. Those protocols are locamonitor, and locasock. They are shown on the figure just below the dashed line dividing kernel space from user space. The boxes for *LocaExec* and *load balancing* represent research in process. Work is ongoing on a complete implementation of MPI [12] called *LocaMPI*. Currently only the broadcast and send/receive functions are implemented.

LocaSock implements a socket interface using Locabus as the communications channel instead of TCP/IP or UDP/IP. LocaSock provides a reliable datagram service with a maximum datagram size of 64KB. Existing UDP socket code can be used simply by changing the domain from *AF_INET* to *AF_LOCABUS*. The LocaSock header is shown in Table 2.

Sport and *dport* are the source and destination port numbers respectively. *Ident* is a fragmentation identifier and *frag* is the fragment number. *Nfrags* is a count of the total number of fragments in the packet.

LocaMonitor is a distributed cluster monitoring protocol. It also is implemented as a kernel module. Each node which runs LocaMonitor periodically broadcasts statistics such as CPU utilisation, memory usage, and disk and network statistics. These statistics are collected by all nodes which maintain the

Table 2. LocaSock header

Field	Type	Description
sport	__u16	Source Port
dport	__u16	Destination Port
ident	__u16	Fragmentation Identifier
frag	__u8	fragment number
nfrags	__u8	Number of fragments for packet

data in the */proc* filesystem. A separate display application reads the */proc* files and displays the data in an humanly readable form. The data will be used in the future for load balancing.

4 Performance Results

To determine the efficiency of the Locabus implementation, we conduct a benchmark of communication time. It compares the time to send data over the standard TCP/IP interface with the time for Locabus.

For this test, we implement MPI_Send(), MPI_Recv(), and MPI_Bcast() using Locabus as the underlying communications system. These functions are the message sending, receiving, and broadcasting functions of MPI, respectively. We compare these functions to the same functions that exist in the LAM implementation of MPI. This removes any computational costs associated with a particular application.

The first benchmark tests the send/receive performance. It compares the time to send/receive a varying number of 1KB messages between two nodes. The results are shown in Figure 2. As expected, Locabus exhibits only a slight performance increase owing to having not to use the IP layer software. The performance increases slightly with increasing number of messages due to the smaller overhead and smaller packet header size.

The second benchmark compares the performance of the broadcast performance for 8192 1KB messages and 65,536 1KB messages with a varying number of nodes. The results are shown in Figures 3 and 4, respectively. In this benchmark, a root node broadcasts a series of 1KB messages, and the receivers send a single acknowledgment.

In both broadcast tests, LocaMPI outperforms standard MPI. The time for LocaMPI is almost constant with a slight increase in time as the number of nodes increases. This is due to the increasing number of ACKs required. For LAM/TCP is increase is slightly worse than $O(\log_2 N)$. (The $\log_2 N$ line is shown on the figures for reference.) This result is due to the need to decompose the broadcast into a series of unicast messages.

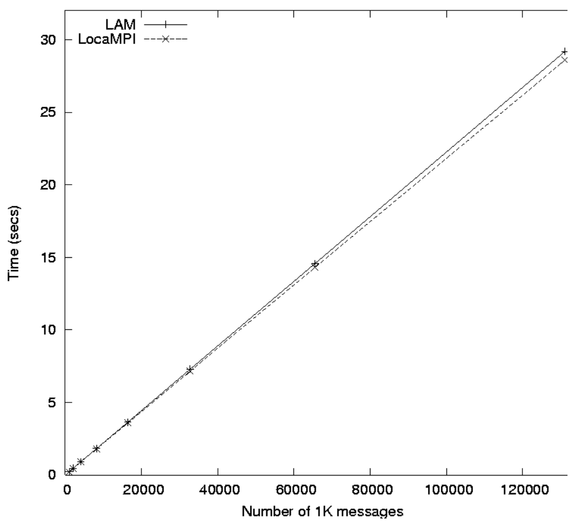

Fig. 2. Send/receive benchmark

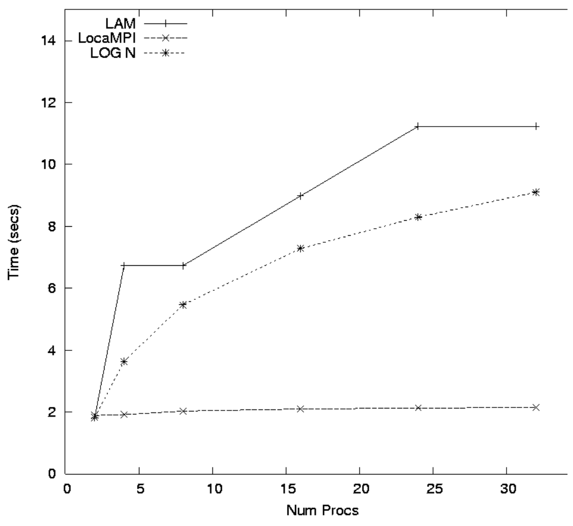

Fig. 3. Broadcast benchmark (8192 1KB messages)

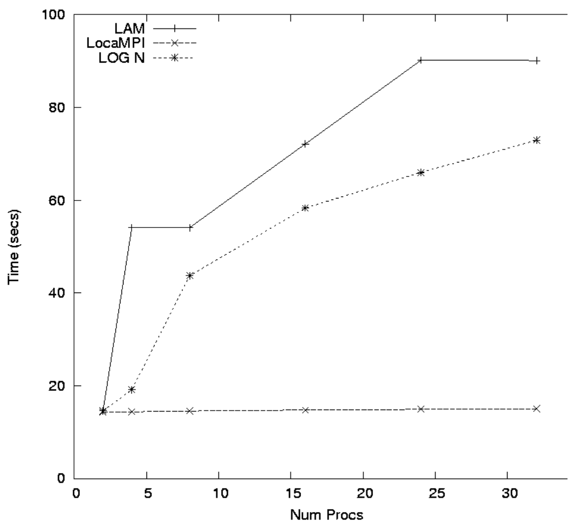

Fig. 4. Broadcast benchmark (65,536 1KB messages)

5 Conclusions

In this paper, we present a kernel to kernel communication system implemented directly onto the Ethernet data link layer. This implementation allows the use of Ethernet's broadcast capability. The implementation is called Locabus.

Over the Locabus software, a number of protocols are built. They include a socket interface (locasock) with is similar to Berkeley sockets. Locamonitor allows each node to know the status of all other nodes. It makes extensive use of Locabus's broadcast capability. A MPI-like interface for parallel programming, called LocaMPI, is partially implemented.

Our tests compare the LocaMPI's send/receive and broadcast performance to that of standard MPI. The broadcast benchmark tests show considerable performance compared to using MPI over TCP/IP.

We are continuing to explore ways to use Locabus in a cluster environment. The LocaMPI implementation is being developed so it has most of the functions contained in the MPI standard. A set of parallel applications will be benchmarked to determine their performance when using LocaMPI. Other areas currently under investigation include load balancing and remote process submission and monitoring.

References

1. Kaashoek, M., Tanenbaum, A., Hummel, S., Bal, H.: An efficient reliable broadcast protocol. ACM SIGOPS Operating Systems Review **23** (1989) 5–19
2. Melliar-Smith, P., Moser, L., Agrawala, V.: Broadcast protocols for distributed systems. IEEE Transactions on Parallel and Distributed Systems **1** (1990) 17–25
3. Barcellos, M., Ezhilchelvan, P.: An end-to-end reliable multicast protocol using polling for scalability. In: Proceedings of the Conference on Computer Communications (IEEE INFOCOM'98), San Francisco, California, USA (1998) 1180–1187
4. Crowcraft, J., Paliwoda, K.: A mulitcast transport protocol. ACM SIGCOMM Computer Communication Review **18** (1988) 247–256
5. Floyd, S., Jacobson, V., Liu, C.G., McCanne, S., Zhang, L.: A reliable multicast framework for light-weight sessions and application level framing. IEEE/ACM Transactions on Networking **5** (1997) 784–803
6. Holbrook, H., Singhal, S., Cheriton, D.: Log-based receiver-reliable multicast for distributed interactive simulation. ACM SIGCOMM Computer Communication Review **25** (1995) 328–341
7. McKinley, P., Rao, R., Wright, R.: H-RMC: A hybrid reliable multicast protocol for the Linux kernel. In: Proceedings of the IEEE/ACM SC-99 Conference, Portland, Oregon, USA (1999)
8. Talpade, R., Ammar, M.: Single connection emulation (SCE): An architecture for providing a reliable multicast transport service. In: Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS '95), Vancouver, Canada (1995) 144–151
9. Lane, R., Daniels, S., Yuan, X.: An empirical study of reliable multicast protocols over Ethernet-connected networks. In: Proceedings of the 2001 International Conference on Parallel Processing (ICPP'01), Valencia, Spain (2001) 553–560
10. Dougan, C.: KMPI: Kernel-level message passing interface. Technical report, Finite State Machine Labs (2003) http://hq.fsmlabs.com/∼cort/papers/kmpi/kmpi.pdf.
11. Levine, B., Garcia-Luna-Aceves, J.: A comparison of reliable multicast protocols. Multimedia Systems **6** (1998) 334–348
12. Gropp, W., Lusk, E., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing **22** (1996) 789–828