# DESIGNING EFFICIENT INTER-CLUSTER COMMUNICATION LAYER FOR DISTRIBUTED COMPUTING

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Vijay Kota, B.Tech, PGDM

* * * * *

The Ohio State University
2001

Master's Examination Committee:

Dr. Dhabaleswar K. Panda, Adviser

Dr. P. Sadayappan

Approved by

_____

Adviser

Department of Computer and Information Science

## ABSTRACT

User-level network interface protocols such as GM, FM, VIA have become increasingly popular to achieve low latency in cluster computing. However, communication with most of these protocols is restricted to a System Area Network (SAN) and the wide area interconnectivity of clusters remains under-explored. In spite of demonstrably superior performance due to features like zero-copy and OS bypassing, these protocols haven't been deployed in a wide-area context. Distributed programming models such as CORBA, Legion, I-WAY, Legion etc. still rely on traditional WAN protocols like TCP despite its inherent overheads. In this thesis, we explore the design space for inter-cluster communication models based on existing SAN protocols and identify several design issues that need to addressed such as end-to-end reliability, message fragmentation, protocol conversion, routing policy, addressing issues and support for multi-point connections.

We design, develop and implement Inter-Cluster GM (ICGM) - an experimental deployment of the GM messaging system in an inter-cluster environment. In particular, we describe how nodes lying in separate clusters could exchange messages using gateway nodes. Such an environment promises potential for applications written on top of GM to be directly ported to geographically distributed clusters

without any additional middleware layer. This will allow applications to take advantage of high performance intra-cluster communication as well as harnessing the computing power from distributed clusters.

Extensive performance evalution of ICGM vis-à-vis the corresponding sockets implementation has been performed. The ICGM implementation has been shown to deliver latency benefits of around 45 us for message sizes upto 1K using Fast/Gigabit Ethernet in the wide-area. Over Gigabit Ethernet, ICGM delivers a peak bandwidth of 23 Mbytes/s – a 15% improvement over TCP. Our experiments with MPICH and Gigabit Ethernet have demonstrated latency and bandwidth benefits of 120 us and 11 Mbytes/s respectively.

**DEDICATION**

**Dedicated to my family**

## ACKNOWLEDGMENTS

I wish to thank my adviser, Dr. Dhabaleswar Panda, for his support and guidance during the course of my research. I am also grateful to Dr. P. Sadayappan and Dr. Pete Wycoff of the OSC for helping me in ironing out a lot of technical difficulties.

I also wish to thank Darius Buntinas, Abhishek Gulati, Jiuxing Liu and Igor Grobman from the NOWLab for their help on issues concerning GM and TCP.

I am indebted to my friends Ajay Joshi, Mandar Joshi, Prashant Nikam, Sidharth Kapileshwar, Ramesh Jagannathan, Praveen Holenarsipur, Nagasuresh Reddy, Prakash Krishnamurthy and Nikhil Chandhok for their support during the course of my studies at OSU.

## VITA

August 31, 1973 …… Born – Kotipalli, INDIA

1994 …… B.Tech, Electronics & Communication Engg.,
I.I.T, Madras, INDIA

1995 – 1997…… P.G.D.M, I.I.M, Lucknow, INDIA

FIELDS OF STUDY

Major Field: Computer and Information Science

## TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Goal

In this work we propose an inter-cluster communication scheme based on existing System Area Network (SAN) protocols. The objective is to provide low-latency and high-bandwidth communication across clusters using wide-area interconnects thus allowing truly distributed computing over the wide-area. We modify a user-level protocol to provide inter-cluster communication facilities in a transparent manner in order to avoid recoding of existing applications that wish to exploit this feature.

## 1.2 Motivation

Developments in high-speed network technologies and low-overhead communication protocols have led to increasing acceptance of networks of workstations (or simply clusters) in many high-performance distributed computing environments. Cluster computing is increasingly becoming a viable alternative to massively parallel processor architectures (MPPs) with the availability of gigabit-per-second networking technologies such as Gigabit Ethernet [1] and Myrinet [2]. The inherent overheads of

general-purpose wide-area protocols such as TCP/IP [3] have encouraged research in development of user-level networking protocols such as FM [4], GM [5] and VIA [6]. However, distributed computing over the wide-area has not received an equal amount of attention. Most of the work on cluster computing has been limited to communication within a single cluster of workstations. Thus, application programmers have not been able to harness the computing power of geographically distributed clusters. Notable efforts in truly distributed computing include distributed programming models such as CORBA, Legion [7] and other general remote method invocation systems. Other wide-area computing scenarios have been tried out using the I-WAY [8] software environment and the Globus meta-computing toolkit [9].

Our work tries to explore issues involved in adding wide-area capabilities to existing user-level network protocols. In general, applications that require frequent but not intense communication would benefit from our work. Secondly, distributed computing on clusters is not limited to geographically separated clusters. A single organization may have multiple clusters sharing a physical facility for administrative reasons. They may have to be separated due to heterogeneous hardware or the administration might want to separate them on the basis of cost centers. Physical space constraints may force the use of multiple rooms too distant to use short-haul network cables. In this scenario, using TCP-based multi-protocol communication systems such as PBS or Condor would involve additional overhead financially – providing wide-area connections at all the computing nodes – and administratively – assigning an IP address to each computing node. Moreover, an organization cannot

use private IP spaces for security or convenience. This is another area where we believe the work described in this thesis could have a significant impact.

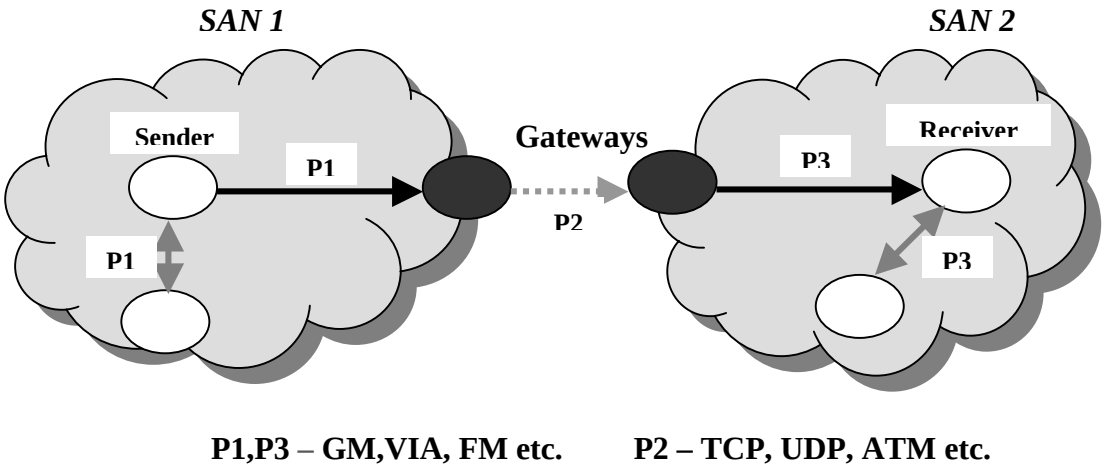



Figure 1.1 An example of inter-cluster communication using SAN protocols

As shown in Figure 1.1, we accomplish communication across clusters through the use of dedicated "gateway" nodes – with hardware for connecting to the cluster as well as the wide-area. In the above figure, P1 and P3 are SAN protocols that have wide-area features e.g. modified versions of GM, VIA etc. P2 is a traditional WAN protocol such as TCP and is used by the gateways for forwarding inter-cluster traffic.

## 1.3 Outline of Thesis

The rest of this thesis is organized as follows: Chapter 2 discusses the design issues involved in developing inter-cluster communication protocols. Chapter 3 provides some background information on the GM message passing system. The specifics of the ICGM implementation are discussed at length in Chapter 4. In an effort to quantify the contribution made by ICGM, several performance evaluations have been conducted and the results of these experiments are presented in Chapter 5. We discuss related work on inter-cluster communications in Chapter 6 and highlight how our work differs from these efforts. Chapter 7 summarizes our experiences with this project and identifies areas for future work related to ICGM.

# CHAPTER 2

## ISSUES IN INTER-CLUSTER COMMUNICATIONS

Several design issues need to be addressed when extending communications to include nodes outside the cluster. This can be attributed to the fact that different protocols are used within a cluster and over the WAN. The following discussion of design issues will assume the inter-cluster communication scheme shown in Figure 2.1. The figure shows two System Area Networks (SANs) which are connected by a non-SAN technology. Designated nodes, henceforth referred to as “gateway” nodes form the end-points of this inter-cluster connection. For convenience, we shall use some protocol aliases for the rest of this chapter. P1 refers to the user-level protocol that is used for communication within the sender’s cluster. P2 refers to the protocol used over the WAN connection. P3 (which could be same as P1) refers to the SAN protocol used within the receiver’s cluster. P1 and P3 are low-latency, high-bandwidth user-level protocols like GM, VIA etc. on top of high-performance inter-connects like Myrinet. P2 is a traditional WAN protocol like TCP/IP or UDP/IP which can be used over a variety of networking technologies like Fast Ethernet, ATM, Gigabit Ethernet etc. The gateway software needs to be carefully designed

after taking into consideration the various issues that might arise due to such differences in protocols. Another factor that complicates the gateway software design is the various link speeds within and between clusters. A detailed discussion of these design issues follows in the sections below. We also propose design alternatives for prospective implementers.

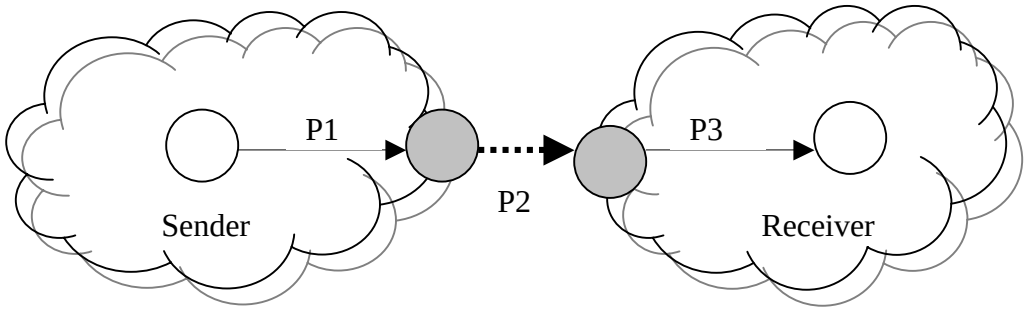



Figure 2.1. Clusters communicating using gateways : P1 and P3 represent protocols used within the clusters. P2 denotes a protocol used over the wide area. The gray circles represent gateway nodes.

## 2.1 End-to-end Reliability

Most user-level network interface protocols provide the user application with reliable, ordered delivery of messages using acknowledgements (ACKs and NACKs) and timeout mechanisms. Protocols like VIA leave this as an option for the user to exercise. Any inter-cluster communication scheme should preserve the original semantics of the user level protocol. Otherwise, the application would have to distinguish between traffic within the cluster and outside the cluster. This would mean loss of transparency to the user application. There are two alternatives for implementing the acknowledgement scheme:

***Piece-wise acknowledgments*** : In this scheme, no changes are made to the current acknowledgment mechanisms used by P1 and P3 for intra-cluster communications. This implies that as soon as the sender's gateway receives a message from the sender, the sender is acknowledged. Thus the sender assumes that the packet has reached the final destination. As long as P2 provides reliable, ordered delivery between the gateways, this poses no problem in terms of retransmission of packets or in-order delivery. Again, reliable and ordered delivery over the final hop between the receiver's gateway and the receiver is ensured by P3. This process is diagrammatically shown in Figure 2.2. The sequence numbers indicate the temporal ordering of the messages and acknowledgments.

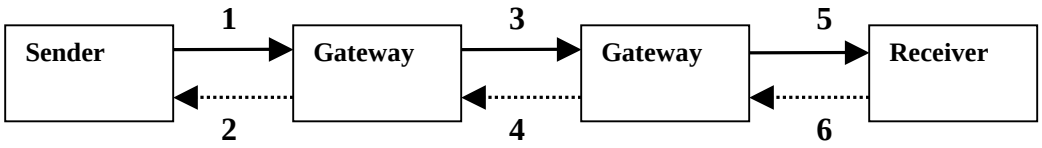



Figure 2.2 : Piece-wise acks: The numbers indicate the sequence of operations. Data exchanges are shown by solid lines and dotted lines represent acks.

***Chained acknowledgements*** : Per this scheme, the acknowledgement mechanism of P1 is modified such that the sender's gateway does not immediately acknowledge a message received from the sender. Instead, it awaits a special message from the receiver's gateway stating that the receiver has actually received the message. When this message is received by forwarder process at the gateway, it is passed down to the protocol layer of P1 which then sends out an acknowledgement to the sender. Figure 2.3 depicts this and the solid lines, dashed lines and sequence numbers have the same meaning as before.

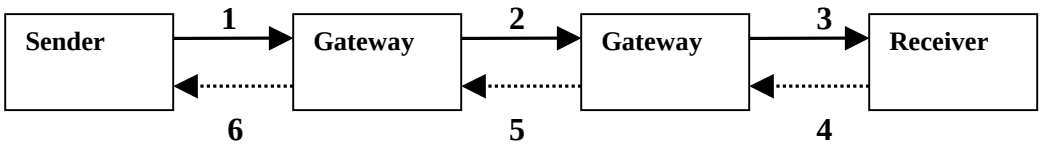



Figure 2.3 : Chained acks

Ideally, the latter scheme of chained acknowledgements seems a more appropriate design choice because it preserves the semantics of reliable transmission. A sender is acknowledged only after the message has reached the final destination. However, there is a price attached to this scheme. Many user-level protocols use “registered” memory for send and receive operations to ensure DMA transfers from the user space to NIC memory without kernel intervention and swap outs. Since there is a limit on the amount of memory that can be registered, these buffers can be reused only after acknowledgements are received. Thus, using chained acknowledgements delays buffer reuse and might cause blocking in the send/receive processes due to lack of available registered buffers. The chained acknowledgement scheme also requires significant changes to the protocol software.

While the piece-wise acknowledgement scheme does not have the above drawbacks, it does suffer from the disadvantage of not being able to notify a sender about failures on subsequent hops. In practice, however, most distributed programs (including

MPICH/GM) do not check for NACKs. A NACK causes an abort of the entire parallel process and is considered sufficiently rare and fatal. So, this semantic change would affect only those few programs that do handle node or link failures gracefully. A compromise between the two schemes would be to make software changes to support either scheme and subsequently allow the user to choose the appropriate scheme while configuring the protocol software.

### 2.2 Message Fragmentation and Reassembly

Packet-based protocols typically split up large messages into more manageable chunks using the Message Transfer Unit (MTU). Large messages are fragmented at the sender side while recording the appropriate information in the packet header. The protocol stack at the receiver is responsible for reassembling these fragments before passing it on to the application process. Unless proper care is exercised while designing the gateway software, mixing various protocols could cause loss of important header information. One option is to retain the fragmentation and reassembly scheme used by P1 which means that the fragments are reassembled at the sender gateway before they are forwarded by the gateway process. But this would be wasteful since the gateway has to wait for all the fragments that belong to a large message before it can be forwarded. This approach is sub-optimal given that each fragment has sufficient information to initiate the forwarding process. A better alternative is to pipeline the computation and the communication. The protocol software at the gateway could be modified not to reassemble the fragments, but instead pass them on the gateway software at the upper layer. The reassembly of the fragments is thus postponed till all of them are received at the final destination.

## 2.3 Protocol Conversion

Using several protocols on various hops can complicate matters if some of these are packet-based and the others are stream-based. Consider a scenario where P1 and P3 are packet-based and P2 is a stream-based protocol like TCP. P1 would encode all the source-destination information in the packet header but P2 does not respect packet boundaries. Thus, P2 might send only a portion of a large packet if its send buffers cannot accommodate the entire packet. Alternatively, it could try optimizing bandwidth by transmitting several small packets in a single data stream – potentially mixing up disparate data flows. In this case, the gateway at the receiving end needs to have some means of distinguishing between the various packets. Even if all the protocols use packets, the implementor still needs to take care of the differences between the packet formats of each protocol.

## 2.4 Routing Policy

To support inter-cluster communication, the routing policy used by P1 needs to be modified. Many SAN protocols store point-to-point routes and rely on the NIC to make routing decisions. One alternative for routing in such a scenario would be to store the routes for all possible source-destination pairs and leave the routing modules of the protocol intact. However, this is very expensive because the routing tables are usually maintained in expensive SRAM on the NIC and maintaining point-to-point routing information for even a few medium-sized clusters would easily exhaust this memory. Moreover, this is not a scalable design choice. With increasing number of nodes and/or clusters, the routing tables would soon become unwieldy. The increased size would also have an adverse effect on the memory requirements and latency. An

alternative is to modify the routing module of the protocol software to make forwarding decisions depending on the target node. Thus, P1 would need modifications to distinguish between local and remote nodes and identify the gateway required for the communication. The challenge is to design a routing policy that is scalable without having an adverse effect on the performance of intra-cluster communications.

### 2.5 Addressing

Since the routing policy depends on the manner in which addresses are assigned to the hosts, the addressing scheme has to be chosen carefully to avoid conflicts. This may not be very easy – especially if the clusters involved in the communication are under different administrative domains. One alternative is to use a global address space under the control of a single authority and requiring each participating cluster to register with this entity to get a range of unique node ids. Another option is to let each cluster have its own addressing scheme and add a proxy mechanism to the gateway software. Virtual addresses could be used for nodes outside the cluster and the gateways would take care of translating these to the actual addresses used on the remote cluster. This also needs development of a protocol between the gateways to agree on the virtual addresses to be used.

### 2.6 Multi-point Connections

Some user-level protocols assume point-to-point connections between communicating nodes. Depending on how the protocol software sets up the data structures, a connection might be identified by the physical end-points instead of the source-destination pair. In such cases, introducing the gateway in the communications

path would complicate issues. For instance, a node that needs to communicate to more than one remote node might need to use the same gateway for all the data flows. As a result, the implementer has to introduce some mechanism for the gateway to be able to distinguish between the multiple data flows across the same connection. This would necessitate changes to the data structures – for instance, to maintain state information per logical connection (identified by sender-receiver) rather than a physical connection.

## 2.7 Conclusion

As discussed in the preceding subsections, the usage of heterogeneous protocols in a wide-area context presents several challenges for the implementor. In Chapter 4, we revisit these issues and provide a detailed description of the design choices made in our implementation. Where applicable, we also discuss the motivation to choose one design alternative over another.

# CHAPTER 3

# OVERVIEW OF GM

GM is a message-based communication system over Myrinet which is a gigabit-per-second interconnect technology increasingly deployed in many clusters [5]. Like many user-level network protocols, GM's design objectives include low CPU overhead, portability, low latency and high bandwidth. In achieving these objectives, GM takes advantage of the Myrinet Network Interface Card (NIC). The Myrinet NIC is "intelligent" in the sense that it has on-board SRAM and a processor (called LANai) which executes a monitor program called the Myrinet Control Program (MCP). The MCP is loaded into the NIC memory by the driver (bundled with GM) and the MCP then handles all communications over the Myrinet interface thus bypassing the operating system and the host CPU.

## 3.1 Message-passing in GM

GM provides reliable, ordered delivery between communication endpoints with two levels of priority. The communication endpoint is called a *port* and is associated with a host *node*. All communications are "connectionless" and the sender builds a message alongwith the receiver's node id and port number. GM maintains reliable

connections between each pair of hosts in the network and multiplexes the traffic between ports over these connections. Figure 3.1 shows the resulting reliable logical connections – the dotted lines – between peer processes as well as processes belonging to different hosts. Sends and receives in GM are regulated by implicit tokens which represent space allocated to the client in various internal GM queues.

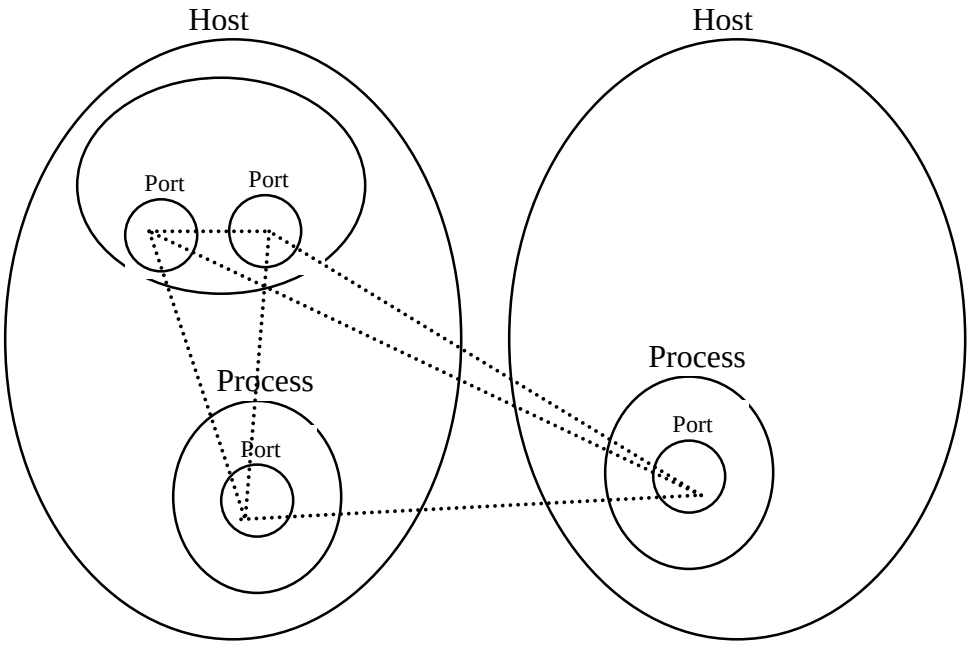



Figure 3.1 : End-to-end communications in GM. Ports represent the end-points and the dotted lines represent logical connections

### 3.2 Sending messages

A user process that wishes to send a message needs to issue a gm_send() primitive. This results in a send descriptor being written to a send queue maintained in LANai memory. Figure 3.2 illustrates this process. Among other fields, the send descriptor contains the destination node, destination port and a pointer to the message buffer. The send state machine in the MCP polls the send queue for outgoing messages. On finding a pending send descriptor, the MCP constructs a GM “packet” and initiates a DMA to transfer the data to be sent. The sender process needs to ensure that the pages containing the data are not swapped out in the midst of a DMA by “pinning” the memory via a gm_register_memory() primitive. The sender is also responsible for not reusing the data buffer before the send is complete. The sender can optionally specify a completion handler for each send. Since all sends are regulated by tokens, it is the responsibility of the sender process to ensure the availability of a token before attempting a send. The send completion handler helps the sender in keeping track of the send tokens and recycling registered buffers.

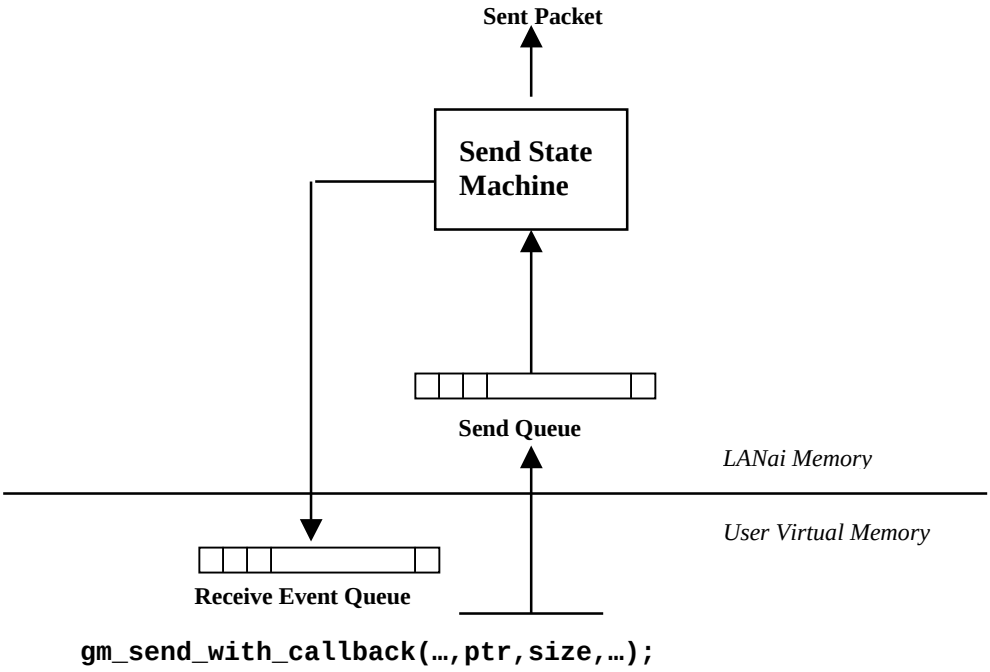



Figure 3.2 : Steps involved in sending messages with GM

### 3.3 Receiving messages

Receiving messages in GM is again regulated by receive tokens. A receive token represents a buffer in user space where the MCP can DMA a message received from the network. The list of receive tokens is maintained in LANai memory and stores the size and priority of expected messages. It should be noted that *a priori* information about all possible combinations of size and priorities is assumed. As in the case of sending, the receive buffers should be registered as well to allow uninterrupted DMA

of message data from LANai buffers to user memory. Figure 3.3 shows the sequence of events during message receipt.

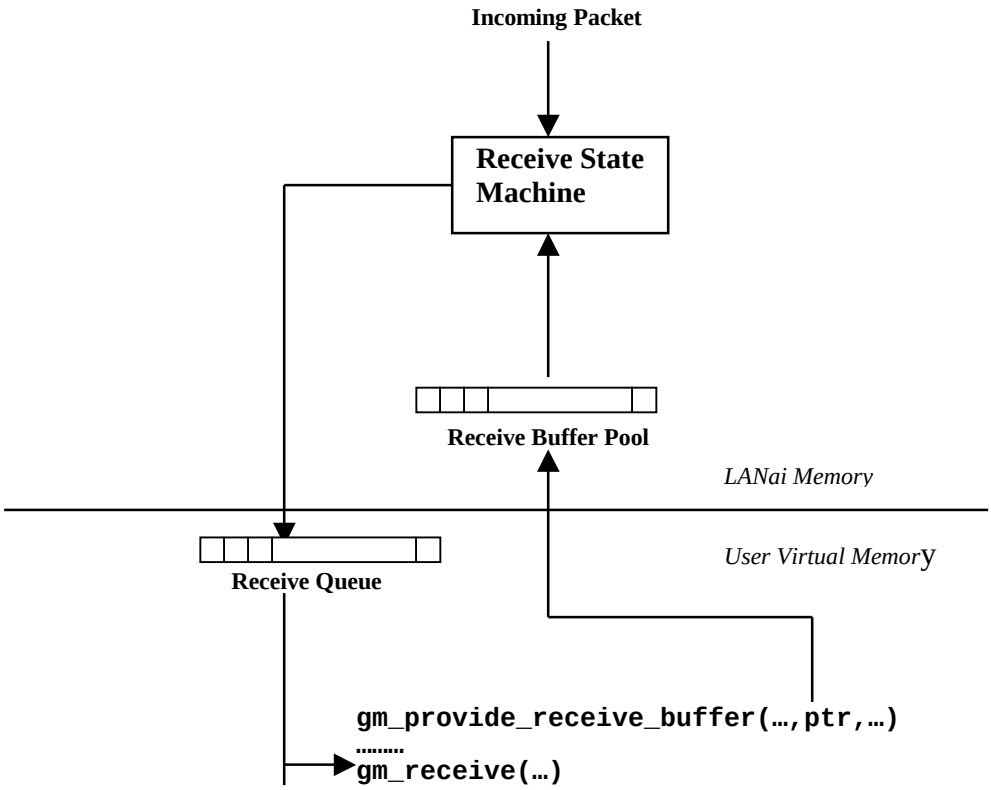



Figure 3.3 : Steps involved in receiving messages in GM

Upon receiving a message from the network, the MCP checks if a matching receive token is available. If so, it initiates a DMA of message data and creates a receive

descriptor – essentially a data structure with sender information and pointers to message data – which is also DMAed to a receive queue in user space. Finally, the MCP queues an acknowledgment to be sent to the sender. If a matching receive token was not found, the MCP discards the message and queues a negative acknowledgment to the sender to indicate failure of receipt. The receiver process polls for events in its receive queue to check for availability of data.

# CHAPTER 4

# IMPLEMENTATION OF ICGM

While GM satisfies the original design objectives of low-latency and high-bandwidth over Myrinet, it also requires each host to be physically connected to every other host it wants to communicate with. As a result, developers interested in truly distributed computing are forced to either recode the applications written in GM to use traditional WAN protocols or use additional layers of software to dynamically choose from a variety of available protocols. Our work on ICGM attempts to address this issue by extending the scope of the GM communication model to allow nodes residing in different clusters to communicate.

## 4.1 Basic Concept

In our implementation, we modify the MCP to make forwarding decisions based on the target node. Each cluster has dedicated nodes called “gateway” nodes which are connected to gateways of other clusters using standard WAN interconnects like ATM, Fast Ethernet etc. All inter-cluster traffic is routed through these gateways which have daemon processes running on them to take care of forwarding. The gateway software

at the application level closely interacts with changes at the MCP level to achieve forwarding over the wide area.

### 4.2 A Detailed Example of ICGM usage

The following example illustrates the role played by ICGM in the critical path of communication. Figure 4.1 depicts a typical inter-cluster communication scenario.

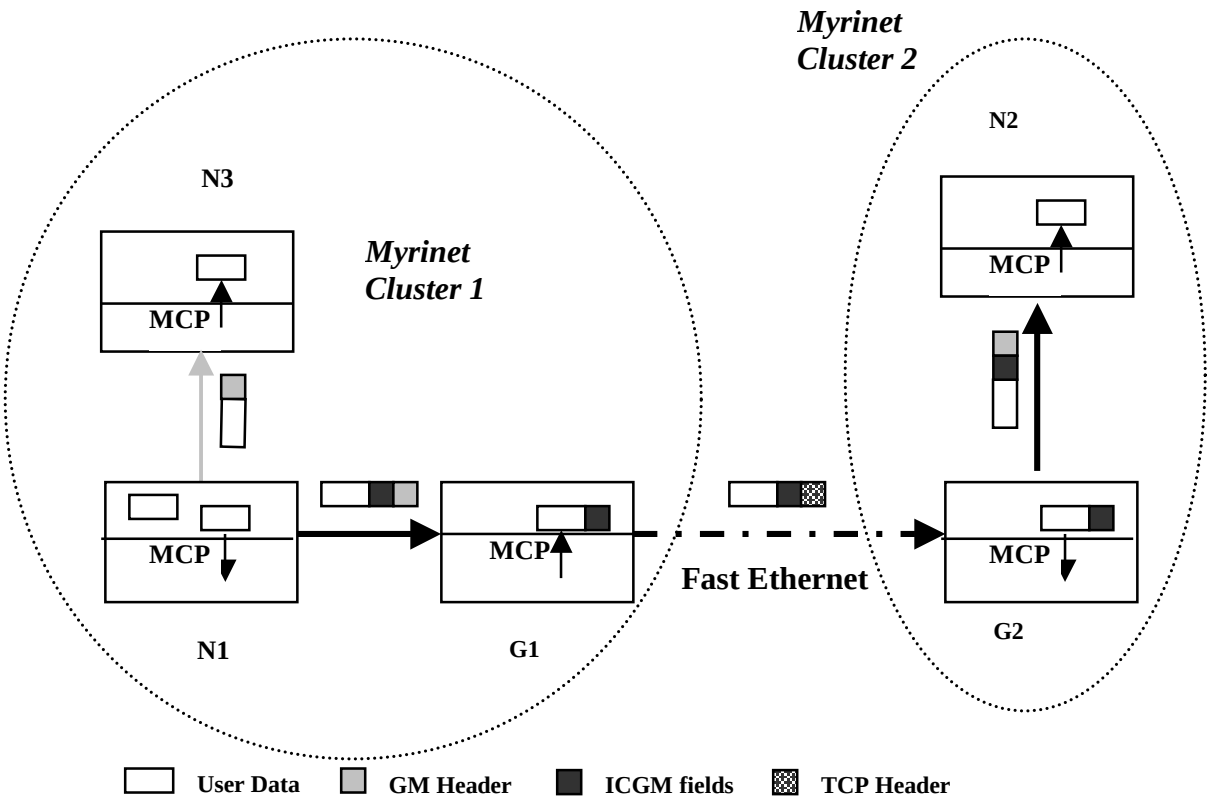



Figure 4.1 : Inter-cluster and Intra-cluster communication with ICGM

In the above figure, sender N1 attempts to send messages to receiver N2 that lies in a different cluster and receiver N3 which lies in the same cluster. Clusters 1 and 2 use ICGM over Myrinet internally and are interconnected by a Fast Ethernet link. The gateway nodes for each cluster are represented by G1 and G2, respectively. In the case of intra-cluster communication between N1 and N3 – shown in the figure by a gray arrow - ICGM decides that the destination lies in the same cluster. It then constructs a GM packet and prepends a source route to it before sending it out on the Myrinet link. The case of inter-cluster communication between N1 and N2 – represented by black arrows - is more interesting. A message sent from N1 to N2 undergoes the following additional processing:

- The MCP on N1 decides that N2 lies on a different cluster and hence needs forwarding. It constructs an ICGM packet – which uses a new packet type for demultiplexing and contains additional header fields used in subsequent hops. It then source routes it to the gateway required to reach N2 (G1 in this case).
- The MCP on G1 receives the ICGM packet and DMAs the data to the buffer allocated by the forwarder process on G1.
- The forwarder process on G1 polls for incoming packets using the gm_receive() primitive. When it finds a packet that has been received, it forwards it across the non-Myrinet link to the gateway responsible for the receiver's cluster. Our current implementation establishes a TCP connection between the two gateways and uses socket calls to read/write data.

- The forwarder process on G2 polls for incoming data over the non-Myrinet link. Upon receiving some data, it issues a gm_send() primitive to deliver the message to the actual destination.
- The MCP on G2 constructs an ICGM packet and source routes it to N2.
- The MCP on N2 receives the packet. It uses the fields in the ICGM packet header to identify the original sender and constructs an appropriate receive descriptor before the message data and the receive descriptor are DMAd to the receiving process' memory.

### 4.3 ICGM Design Choices

The most important consideration while designing ICGM was an appropriate protocol over the wide-area. Since GM provides reliable ordered delivery, we felt that the changes to ICGM could be simplified by using a reliable protocol like TCP between the gateways. Our implementation takes advantage of the reliable transport layer of TCP and enforces end-to-end reliability through the piece-wise acknowledgement scheme thus optimizing the reuse of registered buffers. Message fragmentation in ICGM is handled using delayed reassembly of fragments as described earlier. Fields were added to the ICGM packet header to identify fragments and the MCP was modified to reassemble these fragments only at the final destination. The entire process is transparent to the user processes while allowing pipelining. For routing, we made minor changes to the routing module to implement forwarding. Every GM cluster has a "mapper" node that dynamically keeps track of the routes within the cluster and shares this information with the other nodes in the cluster. Thus ICGM did not require any extra code to make the nodes aware of the cluster topology. Assuming

a simple addressing scheme like contiguous address spaces (eg: Nodes 1 thru 64 in cluster 1; nodes 65 thru 100 in cluster 2 etc.) , the forwarding logic can be coded with a few instructions without significantly increasing the latency on the critical path. The current version of ICGM uses hardcoded gateway ids and future versions should allow more flexible routing.

The ICGM implementation involved modifications to GM at the MCP level as well as development of gateway software which runs on the gateway nodes. While the MCP is executed by the LANai processor, the gateway software runs like any other application program making use of the send and receive primitives provided by the GM API and the sockets library.

### 4.4 Data Structure Changes

Among the most important changes to the GM data structures is the modification of the GM packet header format. ICGM relies on a new type of GM packet – henceforth referred to as the ICGM packet. This is very similar to the original GM packet except that we use a new value for the packet type field in the header to distinguish it from the GM packets carrying intra-cluster data. The ICGM packet header also contains additional fields to aid in demultiplexing. Figure 4.2 shows the structure of the ICGM packet header. The fields in gray represent the fields that are not present in GM packets. These fields are available to the gateway software to make routing decisions.

<table>
<tr><td colspan="2">Packet type</td><td colspan="2">Packet subtype</td></tr>
<tr><td colspan="2">Target Node Id</td><td colspan="2">Sender Node Id</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="2">Length</td><td>Target subport Id</td><td>Sender subport Id</td></tr>
<tr><td colspan="4">Header checksum (optional)</td></tr>
<tr><td colspan="2">IP checksum (optional)</td><td colspan="2">Reserved (optional)</td></tr>
<tr><td colspan="2">Source Node Id</td><td colspan="2">Destination Node Id</td></tr>
<tr><td>Source subport Id</td><td>Destination subport Id</td><td colspan="2">Length</td></tr>
<tr><td colspan="2">Packet subtype</td><td colspan="2">Unused</td></tr>
</table>

Figure 4.2: ICGM Packet Header format

The packet type is used by GM to identify valid GM packets. The packet subtype is used to distinguish between various packets such as data packets, ACKs, NACKs, etc. In the case of data packets, this field also stores the fragmentation information when a large message is split up into smaller packets. The node id fields are used to uniquely identify the hosts while the subport field is used to differentiate between disparate simultaneous connections on a single host. The sequence number is used in implementing GM's "go back N" protocol for reliable transmission. The length and checksum fields are self-explanatory.

The sender and target fields correspond to the physically-connected end-points in the current hop of the communication whereas the source and destination fields correspond to the original sender of the information and the final destination of the message. For instance, in Figure 4.1, the first hop from N1 to G1 would have sender and source values set to N1; target value set to G1 and destination value set to N2. Similarly, when the ICGM packet is on its final hop from G2 to N2, the sender value is set to G2; the target and destination values are set to N2 and the source value stays at N1. The length field is repeated so that the receiver gateway can reconstruct a GM packet from an incoming TCP byte stream. It should be noted that the fields that are a part of the original GM packet header format are stripped off before delivery to the application layer. An alternative to making fairly complex changes to the MCP to modify this behaviour, is to include the relevant information alongwith the packet data. The packet subtype field is required since fragments are not reassembled at the gateway but are instead forwarded to the destination which then uses this field to reassemble the message.

Another data structure that was changed for ICGM was the send token. The send token is used to store all the information passed by the user by invoking a gm_send() primitive and this information is used for retransmissions if any. We added the destination information to the send token as shown in Figure 4.3. The bold fields in the figure represent our additions to the original structure.

```
typedef union gm_send_token {
      .
      .
      struct gm_st_reliable {
            gm_send_token_lp_t  next;
            GM_SEND_TOKEN_TYPE_8 (type);
            gm_s8_t  size;
            gm_u16_t  target_subport_id;
            gm_u32_t  send_len;
            gm_subport_lp_t  subport;
            gm_up_t  orig_ptr;
            gm_up_t  send_ptr;
            gm_u16_t  dest_node_id;
            gm_u16_t  dest_subport_id;
      } reliable;
      .
      .
}
gm_send_token_t;
```

Figure 4.3 : Changes to the GM send token structure are shown in bold

All the changes described in this subsection were made in the file *include/gm_types.h* of the GM software distribution.

### 4.5 MCP Software Changes

All send and receive logic in the GM MCP is governed by four state machines – SEND, SDMA, RECV and RDMA. The respective source files are *mcp/gm_send.h*,

*mcp/gm_sdma.h*, *mcp/gm_recv.h* and *mcp/gm_rdma.h*. The ICGM implementation required changes to the following modules:

***gm_sdma.h***

- Modify the handler that polls for sends. Upon finding a send event, the handler checks if forwarding is needed and if so, populates the destination fields in the send event structure
- Change the SHORTCUT macro as well as send and resend logic in the SDMA handler to construct ICGM packets when inter-cluster communication is needed

***gm_rdma.h***

- Manipulate packet subtype at gateways to bypass reassembly of fragments
- Modify the RDMA handler to offset the DMAs by the overhead introduced by the ICGM packet type
- Use the extra ICGM fields at the final destination to populate the receive event structure (used by the application program) to reflect the identity of the original sender and not the receiver's gateway

***gm_recv.h***

- Add ICGM packet type to the list of valid packet types expected by the MCP

### 4.6 Gateway software outline

The gateway process runs in user virtual memory and is not very different from a typical GM client application. The only difference is that it also uses the Sockets API to handle communication over the non-Myrinet link. Figure 4.4 lists the pseudocode for the gateway software.

```
while ( true )   // Run forever
  e = gm_receive() // Poll for packets on Myrinet
  if (e->type = GM_RECV_EVENT) // Got an ICGM packet
    Read destination information from first few bytes
    Choose appropriate gateway
    write(e->buffer, e->len) on corresponding socket
    gm_provide_receive_buffer (e->buffer)  // Allow buffer reuse
  endif

  /* # of TCP connections = # of non-Myrinet links */

  for each TCP connection
    tcplen = read(tcpbuffer) // Non-blocking read
    if(tcplen > 0)// some data has been received
      entire_buffer_processed = false
      while (entire_buffer_processed = false)
        Read length from header fields
        if(length <= remaining portion of tcpbuffer)
          /* Header fields have destination information */
          gm_send(dest_id,dest_port,length)
        else // wait for the entire packet
          save the partial GM packet
          entire_buffer_processed = true
        endif
        Advance pointer in tcpbuffer // Chk for other ICGM packets
        Set entire_buffer_processed if done
      endwhile
    endif
  endfor
endwhile
```

Figure 4.4 : Pseudocode of software running on the gateway nodes

It should be noted that the GM receive() calls and the Socket read() calls are non-blocking to allow the gateway to process data from other interfaces. From the structure of the gateway code, one can observe that all operations are serialized. Ideally, the gateway should be running as multiple threads with one thread per interface over which data is expected. In this case, we could have used blocking receive primitives. However, current versions of GM do not allow a thread-safe programming model and hence the gateway process runs in a monolithic fashion. Despite this, we are able to obtain reasonable performance as can be gathered from the experimental results shown in the next chapter.

# CHAPTER 5

# PERFORMANCE EVALUATION

During the ICGM implementation, numerous tests were run to check for protocol correctness and to ensure that there is no deviation from the semantics of the original GM protocol. However, correctness is not sufficient to allow researchers to exploit the wide-area computing capabilities of ICGM. It is desirable that ICGM offers better performance in comparison to the existing wide-area models of inter-cluster communication. To this end, several experiments were conducted to quantify how ICGM fares relative to the current practices in distributed computing. Currently, applications that perform distributed computing over the wide-area communicate use sockets over TCP at the transport layer and thus it was a natural choice for a reference benchmark. In the rest of this chapter, we describe the results of our experiments and try to weigh the pros and cons of ICGM. In particular, we ran test GM applications to measure the round-trip latency and the bandwidth offered by ICGM and compared these with the corresponding figures for test applications written using sockets. Moreover, we also had test applications written using MPICH [10]. MPICH is

quickly becoming the de-facto platform for development of high-performance distributed applications. This means that an application developer is more likely to use the MPICH API in the development process than coding with the low-level GM or sockets primitives. Hence, it is our opinion that it would be worthwhile to measure exactly how much of the benefits – if any – at the ICGM layer are passed on to the higher layers of MPICH and applications written using MPICH. For this purpose, we used test applications written using MPICH-GM as well as MPICH on sockets and compared the performance. We also ran the NAS parallel benchmarks [23] on the two versions of MPICH for comparative evaluation. Finally, we checked if the ICGM logic added too much overhead for the case of intra-cluster communication or not.

### 5.1 Experimental Testbed and Setup

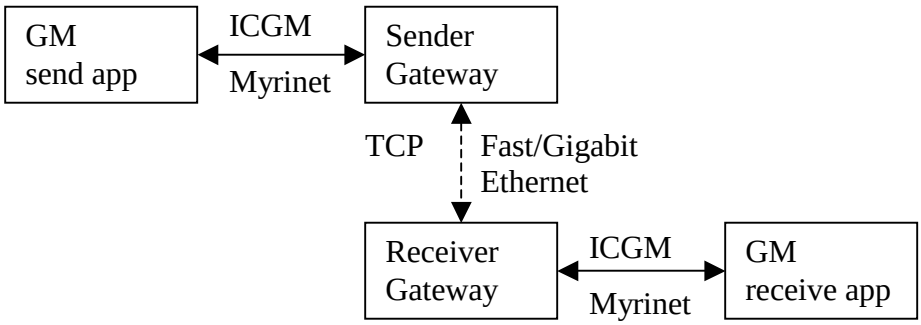
The results presented here were obtained using a cluster of PCs. All the PCs are 300 MHz Pentium II processor nodes and have 100 MHz system bus. Each node has 128 MB of SDRAM, 16 KB of L1 data cache, 16 KB of L2 instruction cache and 512 KB of L2 data cache. Each node has a 33 MHz/32-bit PCI bus and runs the Linux 2.2.5-15 operating system. All the nodes had Myrinet NICs running LANai version 7. Every node has a Fast Ethernet card and a Gigabit Ethernet card. On the software front, we used GM 1.1.3, MPICH-GM 1.2..3 and NPB 2.3 [23] for our testing purposes. MPICH-GM 1.2..3 ships with a default switch-point – the message size at which it crosses over from an “eager” protocol to “rendezvous” protocol – of 16K. Our initial tests with GM and ICGM showed a drop in MPICH bandwidth at 16K due to this switch. This was affecting our analysis of effects – if any - of ICGM and/or gateway software overheads for message sizes greater than 16K. Hence, we modified

the above cutoff to 32K. For the “p4” device used in sockets communications, we also increased the TCP send and receiver socket buffer sizes to 4 GB to make optimal use of the higher bandwidths provided by Gigabit Ethernet.

The above cluster was used to simulate two clusters by logically grouping the PCs into different clusters by assigning a unique block of node ids to each “cluster”. One PC from each cluster was then used as a gateway node and the gateway software was installed on these. The sender and receiver nodes communicate with their respective gateways using ICGM over Myrinet while the gateways communicate with each other using socket calls over the Ethernet interfaces. Thus, each data path consists of 3 hops – the first and the last over Myrinet and the middle one over non-Myrinet. For a fair comparison, all IP traffic from the sender node to receiver node was also forced through three hops so that TCP applications on either the sender or receiver node also go through two hops of Myrinet and one hop of non-Myrinet in between. To achieve this, IP forwarding was turned on at the gateway nodes and the routing tables at the sender and receiver were updated with static routes passing through the gateway.

Figure 5.1 shows the set up for the experiments. It is our belief that this represents a fairly common inter-cluster configuration - namely two Myrinet clusters connected by Fast Ethernet/ Gigabit Ethernet links.

***ICGM Experimental Setup***




***Sockets Experimental Setup***

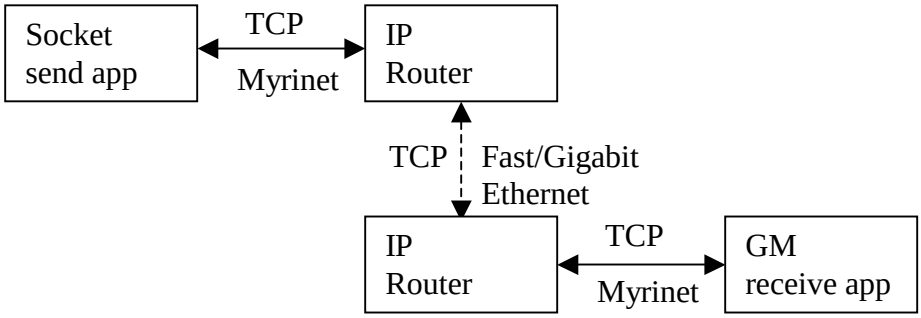



Figure 5.1 : Experimental setup for comparative performance evaluation

## 5.2 Latency Results

In the latency experiments, we determine message latency to be one half of the measured round-trip time taken by a packet from the sender to the receiver. The test application on the sender node starts a timer; sends a message to a receiver application running on the receiver and awaits a reply. Upon receipt of this message,

the receiver replies using the same message. When this reply reaches the sender, it stops the timer and repeats this process a number of times and finally averages the time taken over the number of runs. The latency is then determined to be one-half of this average. This “pingpong” test is then repeated for varying message sizes (from 64 bytes to 1K in increments of 64 bytes). The pingpong tests for latency were run using both 100 Mbps Fast Ethernet and 1 Gbps Gigabit Ethernet on the second hop.

### 5.2.1 Latency Results over Fast Ethernet

In Figure 5.2, we compare the latency results for the ICGM pingpong application and the TCP pingpong application. From the figure, it can be seen that ICGM has lower

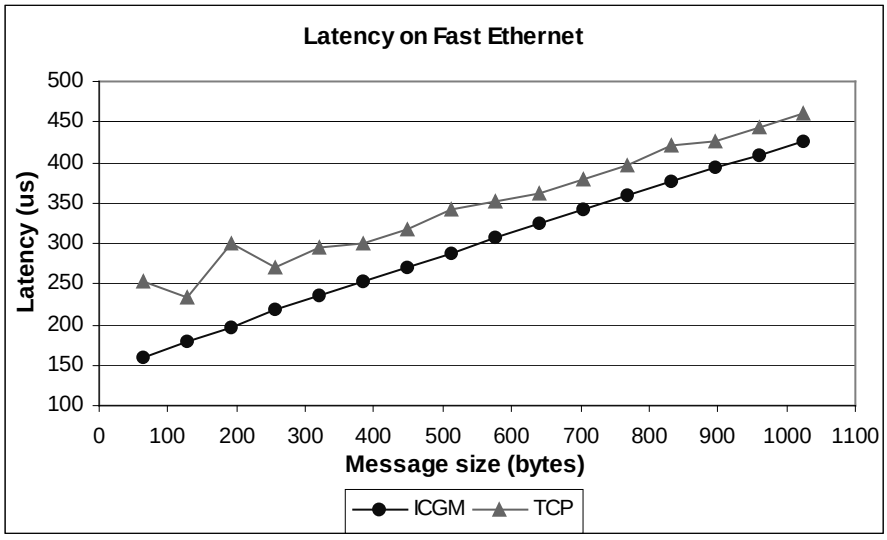



Figure 5.2 : Comparison of latency over Fast Ethernet

latency than TCP for message sizes upto 1K. ICGM latency is about 45 microseconds lower than the TCP latency for the same message size.

### 5.2.2 Latency Results over Gigabit Ethernet

Performance of the pingpong applications using Gigabit Ethernet as the cluster interconnect are shown in Figure 5.3. ICGM again performs better than TCP though the latency difference is slightly lower (40 microseconds on the average) as compared to the Fast Ethernet case.

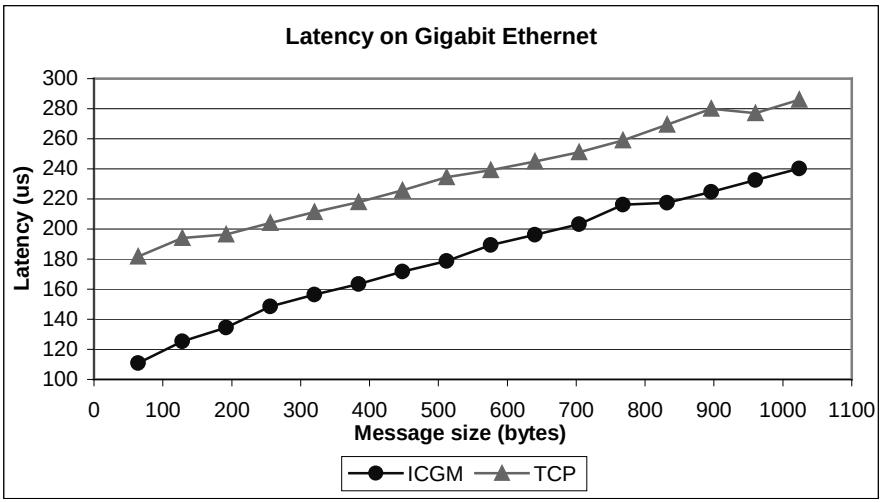



Figure 5.3 : Comparison of latency on Gigabit Ethernet

### 5.2.3 MPICH latency over Gigabit Ethernet

The performance of MPICH over ICGM relative to MPICH over sockets was compared using Gigabit Ethernet on the middle hop. Figure 5.4 demonstrates that MPICH over ICGM shows significant improvement in latency over the sockets implementation of MPICH. MPICH-ICGM reduces latency by about 120 microseconds.

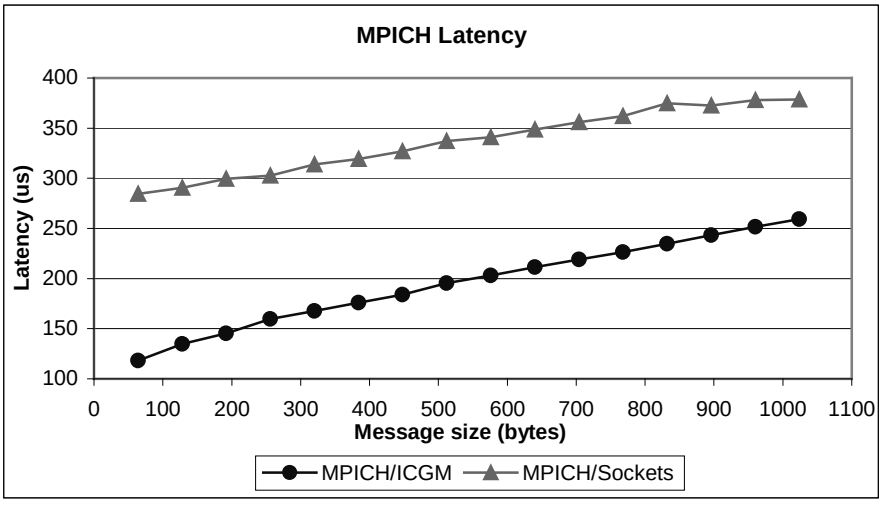



Figure 5.4 : Comparison of MPICH latency

## 5.3 Bandwidth Results

The bandwidth applications measure the portion of raw bandwidth that is available at the application level. In our test application for bandwidth, the sender starts a timer and pumps several messages for a given message size and waits for an acknowledgment from the receiver. When the receiver's acknowledgment is received by the sender, the timer is stopped. The bandwidth is then measured using the message size and the duration between the start of data transmission and receipt of acknowledgment after adjusting for the time taken to send the acknowledgment itself - using the values obtained from the latency experiments. This test is conducted for messages ranging in size from 200 to 20000 bytes. The results of the bandwidth tests are presented below.

### 5.3.1 Bandwidth Results over Fast Ethernet

In Figure 5.5, we compare the performance of the ICGM bandwidth application against the TCP application using a Fast Ethernet interconnect. It can be observed that ICGM slightly outperforms TCP for most of the message sizes. Both ICGM and TCP peak at around 11 Mbytes/sec which is reasonable given the fact the raw bandwidth on Fast Ethernet is only 100 Mbps.

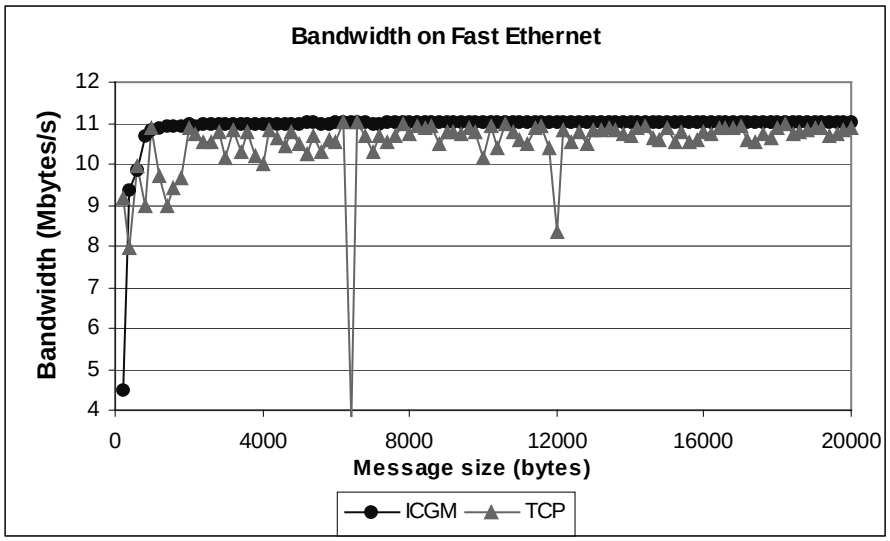



Figure 5.5 : Comparison of bandwidth on Fast Ethernet

### 5.3.2 Bandwidth Results over Gigabit Ethernet

The bandwidth measurements were also taken using Gigabit Ethernet at the datalink layer. Our belief was that this would help us identify the bottlenecks – if any – in the gateway software, other than those due to the inherent limitations of TCP. From Figure 5.6, we find that ICGM performs much better than TCP especially at large message sizes. ICGM delivers a peak bandwidth of around 23 Mbytes/sec while TCP peaks at slightly higher than 20 Mbytes/sec.

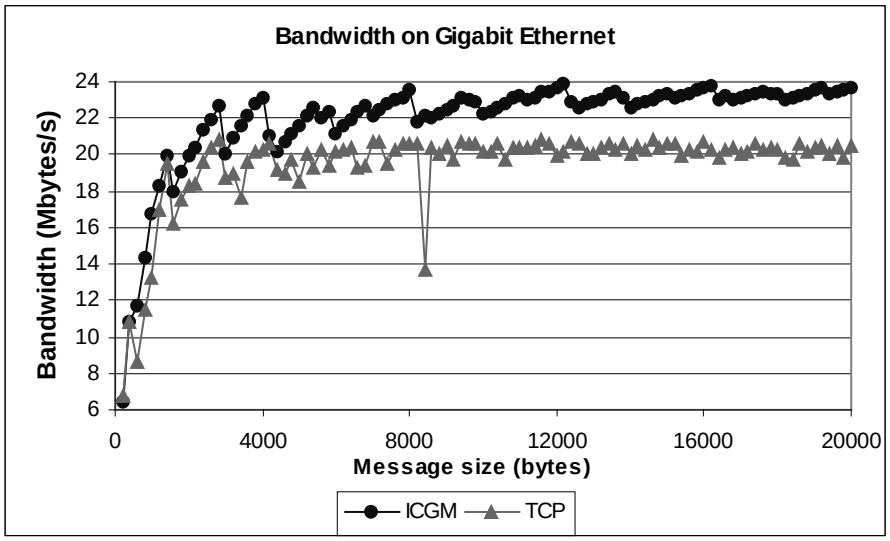



Figure 5.6 : Comparison of bandwidth on Gigabit Ethernet

### 5.3.3 MPICH bandwidth over Gigabit Ethernet

As in the latency experiments, we try to ensure that the benefits at the ICGM layer are reflected at the higher layers. Test bandwidth applications were written for MPICH over ICGM as well as the sockets implementation of MPICH. As shown in Figure 5.7, the MPICH implementation on ICGM again outperforms its counterpart over sockets. MPICH-ICGM delivers a peak bandwidth of about 23 Mbytes/sec against a peak bandwidth of 12 Mbytes/sec using the sockets implementation of MPICH.

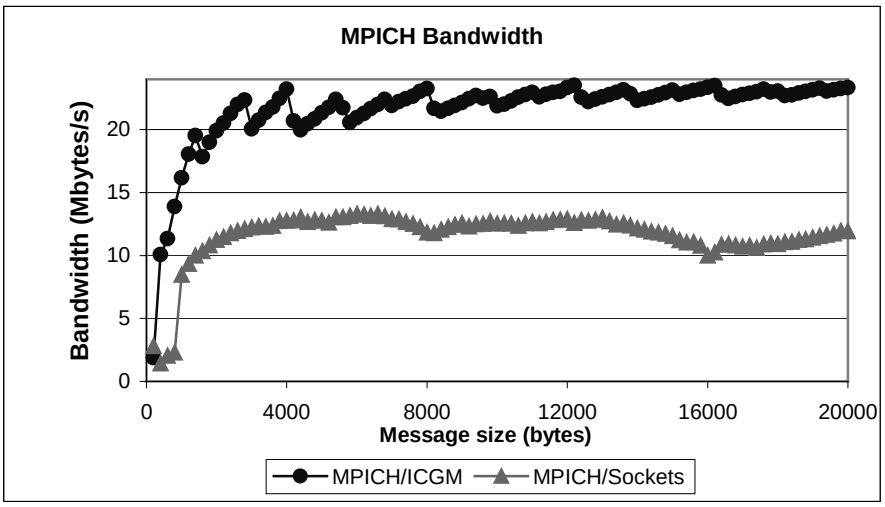



Figure 5.7 : Comparison of MPICH bandwidth

## 5.4 ICGM overhead

In the ICGM implementation we also need to ensure that the common case of intra-cluster communication does not suffer due to the changes at the MCP level. It is vital that ICGM does not add too much overhead to the critical path of message transmission when the message does not require any forwarding. To observe the effects of the MCP changes on intra-cluster communications, we compared the performance of test applications with the original GM layer and that obtained using the ICGM layer.

### 5.4.1 Latency for intra-cluster messages

Figure 5.8 shows the effect of ICGM on the latency for intra-cluster messages. As expected, there is a slight overhead associated with ICGM due to the additional checks. However, this difference is not very significant and is limited to 1-3 microseconds.

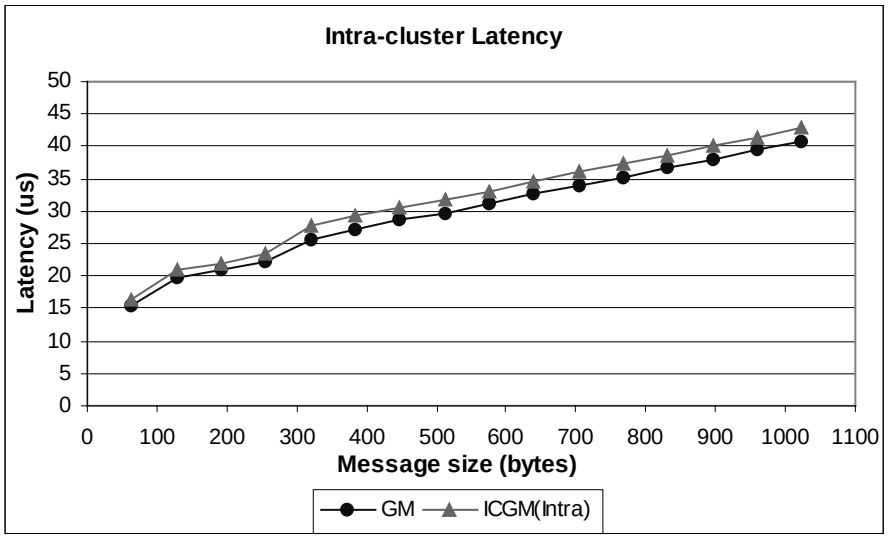



Figure 5.8 : Latency overhead

### 5.4.2 Bandwidth for intra-cluster messages

The bandwidth for messages within a cluster is measured for ICGM and GM. The results are shown in Figure 5.9. ICGM bandwidth shows more variability than GM for lower message sizes but delivers higher bandwidth as the message size increases. The reason for the variability can be attributed to changes in the GM fragmentation scheme. GM tries to split a large message into roughly equal chunks that would fit within the MTU of 4K. This explains the saw-toothed shape of the plots at roughly 4K intervals. At these points, additional overhead is incurred for the extra DMA for a new packet. In the ICGM implementation however, we modified the default fragmentation scheme by just splitting a message into chunks of 4K. The reason was

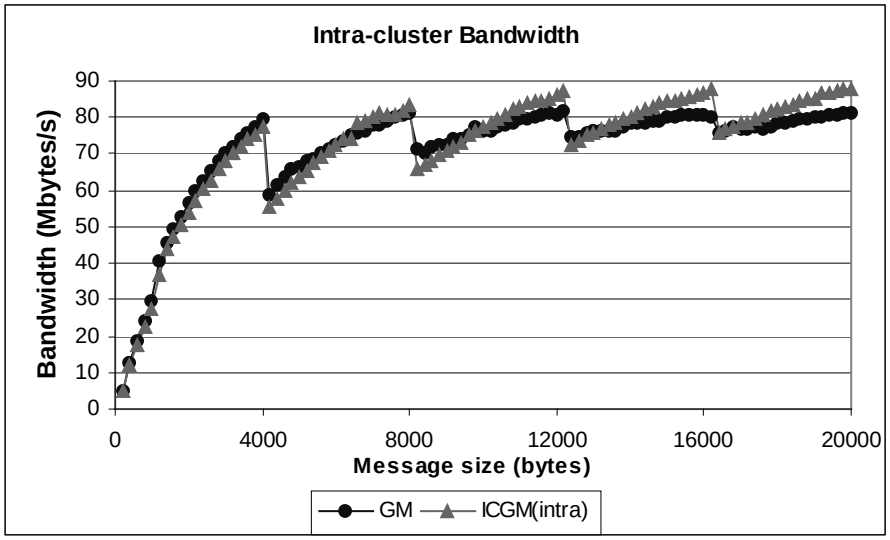



Figure 5.9 : ICGM bandwidth overhead

that our preliminary tests for the inter-cluster case showed better bandwidth with this scheme. Probably, a hybrid scheme would deliver smoother bandwidth for intra-cluster messages.

### 5.4.3 MPICH-GM vs MPICH-ICGM

MPICH performance is not adversely affected for intra-cluster traffic as can be seen from the Figure 5.10 and Figure 5.11. The latency is slightly higher in the ICGM case around 2 microseconds) while the bandwidth is almost the same for GM and ICGM.

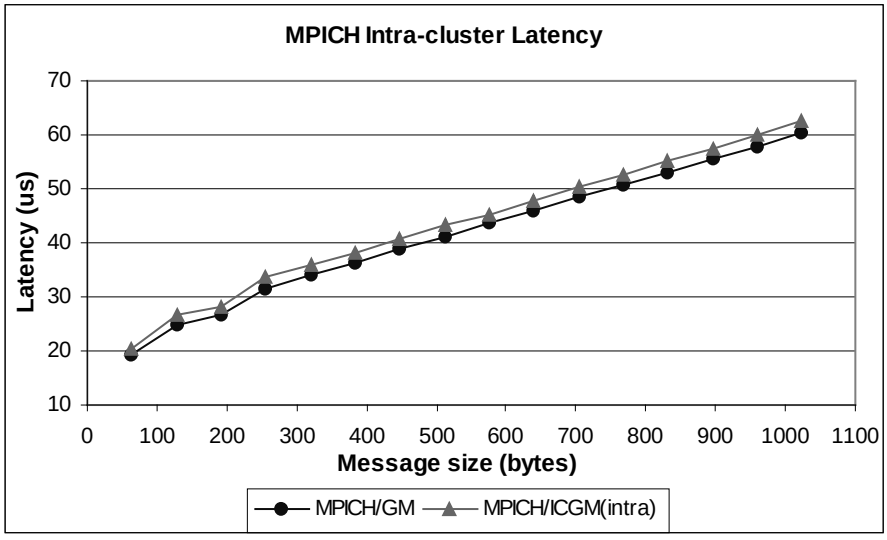



Figure 5.10 : MPICH latency overhead

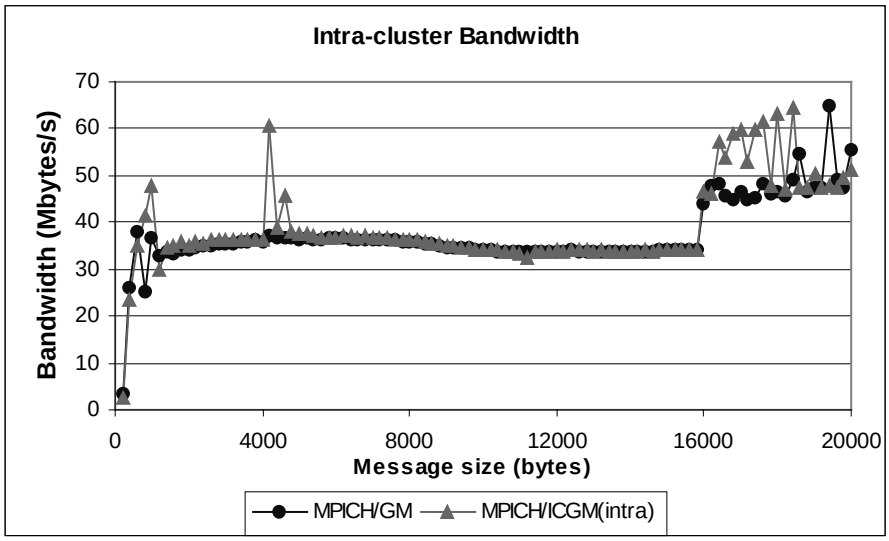



Figure 5.11 : MPICH bandwidth overhead

## 5.5 NAS Parallel Benchmarks

Applications from the NAS parallel benchmark (NPB) were used to test the performance of ICGM. The purpose of using the NPB suite was two-fold. Firstly, we wanted to test whether the benefits at the MPICH layer are passed on to the upper layers. Secondly, we wanted to test how the gateway software responds to multiple data flows. We ran the NPB suite using a 4-node configuration as well as an 8-node configuration. It should be noted that the 8-node figures for SP and BT are not available since these applications require the number of nodes to be a perfect square.

The experimental setups are the same as those for the MPICH tests – two Myrinet clusters interconnected by Gigabit Ethernet.

For the inter-cluster test case, we chose half the nodes from one cluster and the other half from the other cluster. The results of running the NPB applications are shown in Figure 5.12. The suffix indicates the number of nodes used for the run.

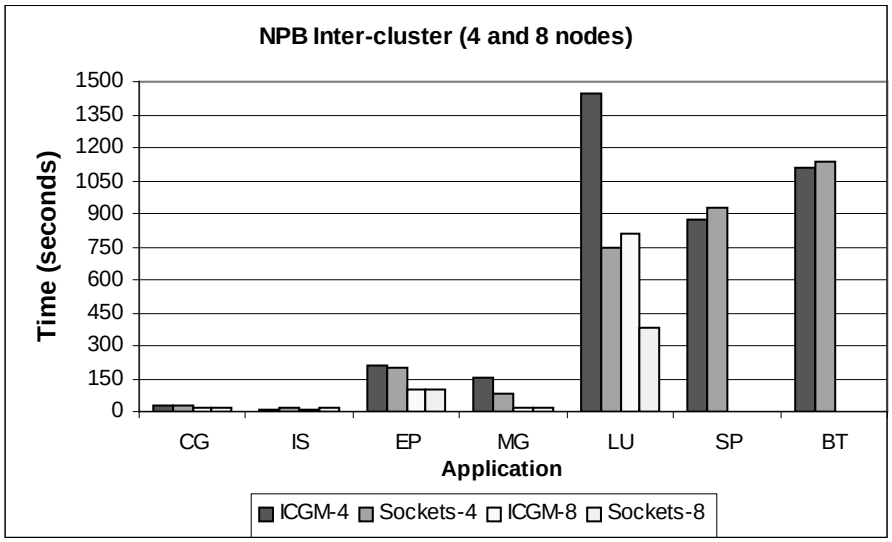



Figure 5.12 : Performance of NPB applications across the cluster using ICGM and Sockets implementations.

The results indicate that ICGM performs as well or better than the Sockets implementation for CG, IS, SP and BT. The percentage reduction in execution time with ICGM for the 4-node case is 8%, 28%, 6% and 2% for CG, IS, SP and BT respectively. In case of EP, there is a slight increase ( < 1%) in execution time using ICGM. When MG is run with a 4-node configuration, ICGM takes almost 85% longer than Sockets. However, when 8 nodes are used, ICGM reduces the execution time by 6%. For the LU application, Sockets performs consistently better than ICGM which takes almost twice as long to execute.

To analyse the reason for ICGM's poor performance in the case of MG and LU, we ran "intra-cluster" tests to study the behaviour of the NPB applications over unmodified GM. Thus, the intra-cluster results – shown in Figure 5.13 – compare the performance of GM vis-à-vis TCP. It should be noted that all traffic is inside the cluster and the source and destination are separated by a single Myrinet hop. GM performs better than TCP for CG, IS, SP and BT with percentage benefits of 14%, 43%, 7% and 2% with 4-nodes. The 8-node execution times for CG and IS are respectively 15% and 50% lower for GM. In the case of EP, there is a very slight difference ( < 1%) between the times obtained with the two implementations. MG over 4-nodes takes one-and-a-half times as long as TCP. However, MG over GM performs better with 8-nodes reducing the execution time by 12%. TCP significantly reduces the execution time for LU (by nearly 50%) for the 4-node as well as 8-node configurations.

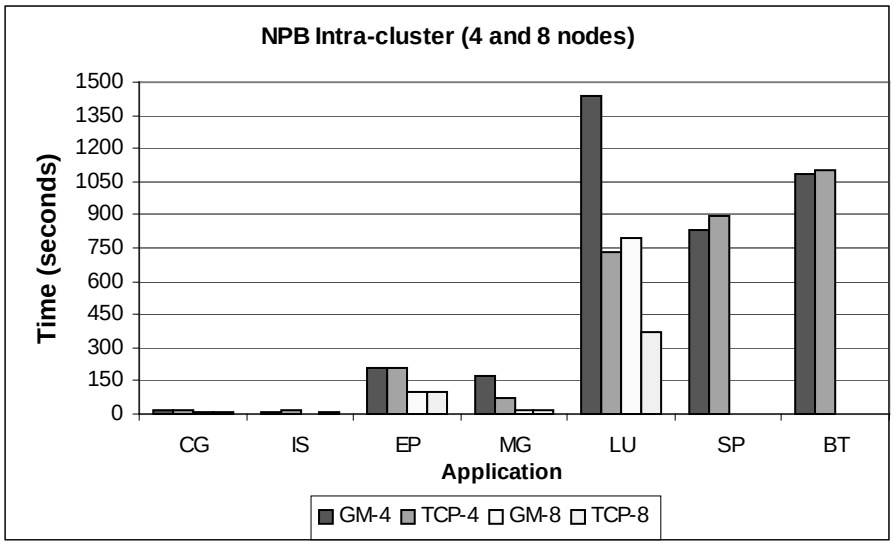



Figure 5.13 : Performance of NPB applications within a single cluster using GM and TCP

From the above results, we can conclude that for applications that do benefit from GM - CG, IS, EP, SP and BT- the ICGM implementation delivers equal or better performance than the equivalent Sockets implementation. We have also established that the poor performance with MG and LU is not related to the ICGM implementation.

## 5.6 Conclusions

The experimental results discussed above indicate that ICGM delivers better performance than TCP in general in latency as well as bandwidth. For raw

applications written over ICGM or Sockets, ICGM reduces the latency by around 45 us on Fast and Gigabit Ethernet. The delivered bandwidth is also higher with ICGM especially over Gigabit Ethernet where we realize a benefit of about 3 Mbytes/s. Test applications written over MPICH and using Gigabit Ethernet in the wide-area also benefit in terms of latency – around 120 us lower with ICGM – and bandwidth – a gain of almost 11 Mbytes/s. We also demonstrate that the common case of intra-cluster communication does not incur a performance penalty due to the forwarding logic incorporated in ICGM. Comparative tests with the NPB suite also show that ICGM is able to pass on the performance gains from using GM to the application layer. Depending on the application and number of computing nodes, ICGM delivers 2%-50% reduction in execution time.

# CHAPTER 6

# RELATED WORK

In this chapter, we discuss related efforts in the domain of inter-cluster communication models that rely on existing user-level protocols. For each approach, we highlight the conceptual differences relative to our approach. We also indicate , where applicable, similarities in the approaches. A description of the various approaches is presented below.

## 6.1 Virtual Machine Interface

Virtual Machine Interface (VMI) is a high level messaging library developed by members of NCSA's cluster computing group. A recent effort at NCSA [12] attempts to address the issue of inter-cluster communication through the design of a gateway protocol called the Exterior Gateway Protocol. A project was undertaken to augment the VMI [13] library to support communications across clusters. VMI is intended to support multiple underlying communication protocols (shared memory, Sockets etc.) while providing a uniform API to application programmers. The VIA-based Exterior Gateway Protocol allows nodes belonging to different clusters to communicate by

interfacing with a gateway interconnecting the two clusters. The project also involves development of a load balancing strategy when multiple gateways can service a particular connection. The gateway in this case is a multi-homed host lying on both the clusters. Thus this approach is significantly different from our approach wherein gateway nodes lying on separate clusters communicate with each other using a traditional WAN protocol like TCP/IP. Moreover, the ICGM implementation does not require programmers to rewrite their code using a VMI middleware.

### 6.2 The PacketWay Specification

In 1997, there were attempts to develop an inter-networking protocol specifically for System Area Networks (SANs) and high-performance LANs. The PacketWay protocol [14], [15] is an open family of specifications intended to inter-network high-performance computing clusters. The inter-cluster communication model presented is very similar to the one we have implemented i.e. a dedicated node on each SAN is responsible for the communication (called a “router” in PacketWay) and the routers can be interconnected using a non-SAN technology. PacketWay is much more general than ICGM in the sense that the communication endpoints can be physical entities (a processor, a smart memory board etc.) or logical entities (e.g. a group of cooperating processes). Apart from the traditional IP-like forwarding [24], PacketWay allows for source routing affording high-speed communications. The Secure PacketWay specification also provides for secure communications over untrusted networks. Though the Packetway specification seems to be well thought-out and promises significant benefits to distributed computing, to the best of our

knowledge, there are no actual implementations available for current generation networks and user-level protocols.

### 6.3 MPICH/Madeleine

A slightly different approach to supporting heterogeneous communications has been adopted at ENS Lyon, France [16]. Instead of attacking the problem at the protocol level, this project aims at handling heterogeneity at the higher layer of MPICH. MPICH is a very popular implementation of MPI and in this project, MPICH was modified to support multi-protocol features. This implementation of MPICH is based on the Madeleine [17] communication library. A new device has been added to MPICH that can handle various underlying protocols – currently supported ones are TCP, SISCI and BIP. However, one limitation with the current implementation is the inability to forward packets across heterogeneous networks, i.e., all the communicating nodes have to be connected pair-wise which implies that each host needs to have appropriate hardware (an Ethernet card for instance) and an IP address. This is a significant difference as compared to our work wherein we try to use WAN interconnects to provide transparent inter-cluster access and avoid the expense of requiring a WAN connection on every node.

### 6.4 MPI/Pro

MPI Software Technology (MSTI) has developed a scalable, robust MPI 1.2 implementation called MPI/Pro [18] which is capable of handling multiple devices simultaneously. The MPI/Pro library allows parallel applications to run jobs across multiple clusters. Nodes within a cluster communicate using high-performance user-level protocols and inter-cluster communication is supported by using TCP/IP. This

approach is similar to the MPICH/Madeleine implementation described earlier. Again, each node needs to have an IP address and a network interface that supports IP. Thus, unlike ICGM, MPI/Pro cannot support private IP spaces for security or administrative convenience.

# CHAPTER 7

# CONCLUSIONS AND FUTURE WORK

## 7.1 Summary

In this thesis, we have discussed the motivation for the development of wide-area capable versions of user-level network protocols. We discussed the various design issues involved in the development of such protocols. In particular, we described our experiences with the development of ICGM – a version of Myricom's GM messaging system with inter-cluster communication capabilities.

In earlier chapters, we discussed the implementation details of ICGM. We have also performed comparative evaluations of ICGM against Sockets-based implementations. We could demonstrate experimentally that at the expense of slight overhead for intra-cluster communications, ICGM is able to outperform Sockets in latency as well as bandwidth. For message sizes of 1024 bytes and less, ICGM saves about 45 us using either Fast Ethernet or Gigabit Ethernet links between the clusters. In the bandwidth tests, both ICGM and Sockets deliver near-peak bandwidths of 11 Mbytes/sec on Fast Ethernet and on Gigabit Ethernet, ICGM delivers around 3 Mbytes/sec more than the equivalent Sockets implementation. Our experiments with MPICH show marked

improvements using ICGM. MPICH/ICGM offers latency benefits of about 120 us and exceeds MPICH/Sockets by around 11 Mbytes/sec in bandwidth. Experiments with the NAS parallel benchmarks indicate that applications like CG, IS, SP and BT that show better performance with GM can take advantage of lower execution times with ICGM as well. Performance benefits with ICGM range from 2%-50% depending on the application and the number of processing nodes.

## 7.2 Future Work

One of the areas in the current implementation of ICGM that can be improved is routing. Currently, the gateway node ids have been hardcoded and thus the gateway nodes are fixed at compile-time. However, this approach is not very flexible and the MCP should be changed to extract the information from a configuration file instead.

The performance evaluation described in earlier chapters compared the performance of ICGM to that of TCP. This is not very realistic given that application programmers are more likely to use higher-level APIs rather than coding in GM or Sockets. An actual inter-cluster scenario currently would have the application program sitting atop a layer of MPICH which in turn runs over a middleware that dynamically uses either the Myrinet device or the sockets device. Thus, a more accurate evaluation would be to compare the performance of ICGM with a software suite such as MPI/Pro. This would also give an insight into overheads associated with middleware layers.

In Section 2.1, we indicated that users should be able to configure ICGM with either the piecewise acknowledgement scheme or the chained acknowledgement scheme. The current implementation of ICGM uses only piecewise acks and it might be worthwhile to provide this option.

## BIBLIOGRAPHY


[1] R. Scheifert. *Gigabit Ethernet*. Addison-Wesley, 1998.

[2] N.J. Boden, D. Cohen et al. Myrinet: A Gigabit-per-Second Local Area Network *IEEE Micro*, Feb 1995.

[3] A. Barak, I. Gilderman and I. Metrik. Performance of the communication layers of TCP/IP with the Myrinet gigabit LAN. *Computer Communications*, Vol. 22, 1999.

[4] Scott Pakin, Vijay Karamcheti, and Andrew A. Chien. Fast Messages: Efficient, portable communication for workstation clusters and MPPs. *IEEE Concurrency*, April-June 1997.

[5] *Generic Messages Documentation.* http://www.myri.com/GM/doc/gm_toc.html

[6] *Virtual Interface Architecture Specification*. http://www.viarch.org

[7] *The Legion Project*. http://legion.virginia.edu

[8] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, 1996.

[9] *The Globus Project.* http://www-fp.globus.org

[10] W. Gropp, E. Lusk,and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT press, 1995

[11] *TreadMarks Overview.* http://www.cs.rice.edu/~willy/TreadMarks/overview.html

[12] Sudha Krishnamurthy. *Design of a Gateway Protocol using VMI for Inter-Cluster Communication* http://www.ncsa.uiuc.edu//General/CC/ntcluster/VMI/gateway.pdf

[13] Avneesh Pant, Sudha Krishnamurthy, Rob Pennington, Mike Showerman and Qian Liu. *VMI: An Efficient Messaging Library for Heterogeneous Cluster Communication.* http://www.ncsa.uiuc.edu//General/CC/ntcluster/VMI/hpdc.pdf

[14] D. Cohen, C. Lund, T. Skjellum, T. McMahon, and R. George. *Proposed specification for the end-to-end packetway protocol*. IETF draft, *1997*

[15] *PacketWay Documentation* http://www.erc.msstate.edu/research/labs/hpcl/packetway/index.html

[16] O. Aumage, G. Mercier and R. Namyst. *MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks.* http://www.ens-lyon.fr/~mercierg/ressources/ipdps_2k1.ps.gz

[17] O. Aumage, L. Bouge and R. Namyst. A Portable and Adaptative Multi-Protocol Communication Library for Multithreaded Runtime Systems. *Parallel and Distributed Processing. Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP '00)*

[18] *MPI/Pro for Linux.* http://www.mpi-softtech.com/product/mpi_pro_linux/default.asp

[19] Raoul A.F. Bhoedjang, Tim Ruhl and Henri E. Bal. User-Level Network Interface Protocols. *Computer*, November 1998

[20] I. Foster, J. Geisler, C. Kesselman and S. Tuecke. Managing Multiple Communication Methods in High-Performance Networked Computing Systems *Journal of Parallel and Distributed Computing*, Vol. 40, Jan. 1997

[21] W.R. Stevens. *TCP/IP Illustrated, The Protocols*. Addison-Wesley, Reading, MA, 1994

[22] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, 1998

[23] *The NAS parallel benchmarks.* http://www.nas.nasa.gov/Research/Reports/Techreports/1994/HTML/npbspec.html

[24] *Introduction to TCP/IP.* http://www.microsoft.com/windows2000/techinfo/reskit/samplechapters/cnbb/cnbb_tcp_zqku.asp