A PROPORTIONAL BANDWIDTH ALLOCATION SCHEME FOR MYRINET CLUSTERS

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

Abhishek Gulati, B.E.

* * * * *

The Ohio State University

2001

Master's Examination Committee:

Prof. Dhabaleswar K. Panda, Adviser

Prof. Ponnuswamy Sadayappan

Approved by

Adviser

Department of Computer and Information Science

ABSTRACT

Simultaneous advances in processor, network and protocol technologies have made clusters of workstations attractive vehicles for high performance computing. However, the requirements of emerging applications, and the need for efficient resource utilization have necessitated Quality of Service (QoS) mechanisms in clusters. The approaches to QoS in the wide-area networking context are not suitable for clusters because of the high overheads. Also, these approaches do not address the issue of contention between flows at the end-nodes.

In this thesis, we present a scheme for proportional bandwidth allocation in Myrinet clusters that simultaneously handles network and end-node contention. This scheme relies on the use of NIC-based "rate control" in conjunction with the global knowledge of traffic patterns in the cluster. Our approach is particularly attractive since it does not require hardware modifications, and can hence work with commodity systems. Experimental results for our implementation over the Myrinet/GM system demonstrate the efficacy and the efficiency of our scheme. Other advantages include the ease of application development allowed by its integration with MPI, and support for "best-effort" traffic.

Dedicated to my parents and my sister

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. D. K. Panda for his support and encouragement through the course of my graduate studies. I appreciate the time and effort he invested in guiding me and steering my research.

I am grateful to Prof. P. Sadayappan for his advice and insightful comments, and for agreeing to serve on my Master's examination committee.

I am thankful to Dr. Pete Wyckoff for his involvement in this work.

Thanks are also due to the student members of the Network-Based Computing Laboratory, particulalry Darius Buntinas for his willingness to help at all times.

This thesis would not have been possible without the financial support I received from various sources. I am indebted to the graduate school for awarding me the University Fellowship in the first year of my M.S., and to Prof. Panda for supporting me as a research associate in the following year.

Finally, I would like to thank all my friends who made my stay at OSU enjoyable.

VITA

January 27, 1977	Born - Ghaziabad, India.
1998	B.E., Computer Engineering, Netaji Subhas Institute of Technology, Delhi, India.
Winter 2001 - Spring 2001	Graduate Research Associate, The Ohio State University.

FIELDS OF STUDY

Major Field: Computer and Information Science

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Network-based computing systems are becoming increasingly popular for high-performance and high-throughput computing [1]. As illustrated in Figure 1.1, these systems consist of clusters of workstations connected by System, Local, or Wide Area Networks. Their cost-effectiveness and incremental scalability make them attractive alternatives to traditional supercomputers.

Figure 1.2 illustrates the software and hardware layers of a network-based computing system. Hardware components include processors, and high-performance interconnection networks (Myrinet [2], HIPPI [4], SCRAMNet [3] etc.). These networks feature peak bandwidths in the Gbps range, and one-way latencies as low as a few microseconds. Processor speeds too have been doubling every eighteen months, and the cost-effectiveness of 2-way and 4-way symmetric multiprocessors (SMPs) has made clusters of SMPs very popular.

The communication software layer seeks to deliver most of the raw network performance to higher layers. Past research in this area has largely focused on minimizing the software overhead associated with traditional network protocols like TCP/IP. The so-called "User-Level Network Protocols" (AM [8], FM [25], U-Net [9], VIA [29] and GM [14] etc.) do so by leveraging "smart" Network Interface Cards (NICs),

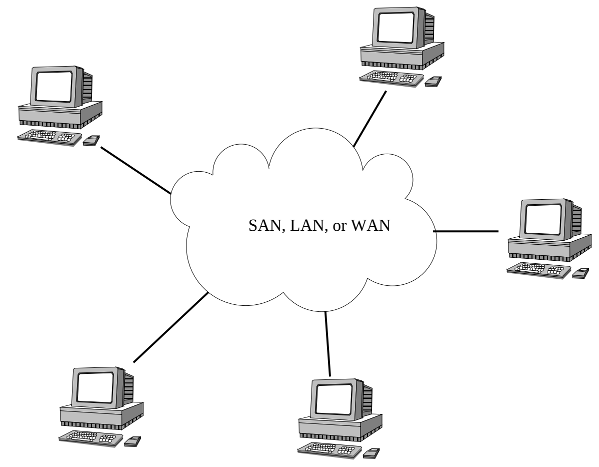

Figure 1.1: A Network-Based Computing environment.

taking operating system involvement out of the critical path of communication and eliminating extra message copies.

Another vital component of network-based computing systems is the programming environment layer, which enables portable applications to be developed easily. The responsibilities of this layer are:

- *Communication services*: It presents the raw communication services provided by the underlying layer in the form of covenient programming models. Popular models are *message-passing* (MPI [22] etc.), and *shared-memory* (TreadMarks [17] etc.)
- *Resource management*: It manages resources such as processors and networks.
- *Auxiliary services:* It provides support for start-up, authentication etc.

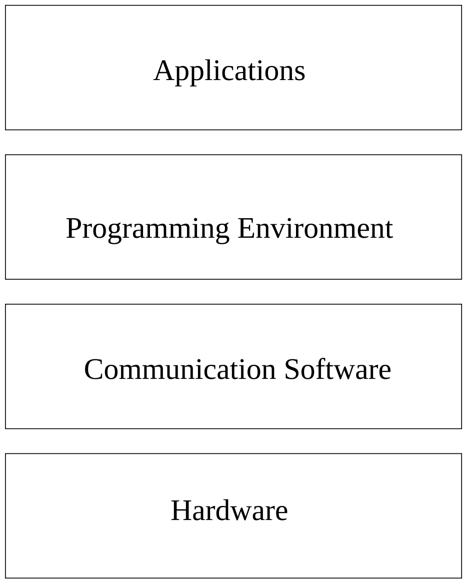

Figure 1.2: Components of Network-Based Computing Systems.

1.1 Cluster Usage Trends and their Implications

Simultaneous advancements in all these layers have triggered the widespread deployment of clusters for compute-intensive applications. However, so far, clusters are largely used in the single-application space-shared mode. This is wasteful, particularly if the availability of multiple processors per SMP node is not taken advantage of. Efficient resource utilization stipulates simultaneous use by multiple applications.

Further, in addition to conventional scientific computing applications, clusters are increasingly being used for latency and bandwidth sensitive applications like interactive simulations, scientific visualization and video-conferencing.

The implication of these two trends is that network traffic in clusters is composed of independent, possibly non-cooperating, communication flows, and, additionally, these flows are characterized by a variety of service requirements.

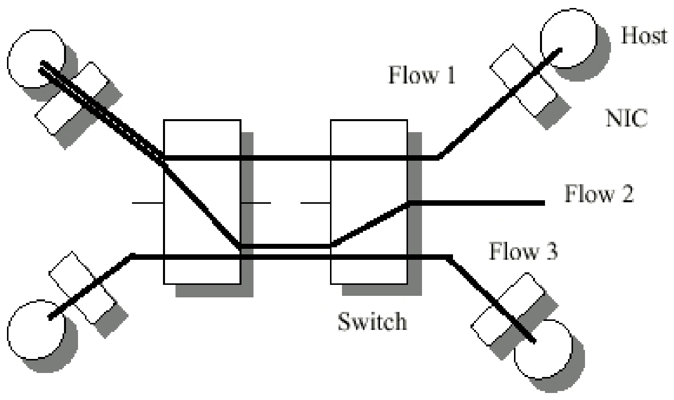

Figure 1.3: End-node and network contention between communication flows.

Coexisting communication flows contend not only in the network, but also for shared resources at end nodes. These shared resources include processors, I/O busses, Network Interface Cards (NICs), their buffers and DMA engines. Figure 1.3 shows a cluster of four nodes, connected by two cascaded switches. Flows 1 and 2 contend at an end-node, while flows 2 and 3 contend for the same output port on the first switch. Disorderly contention may affect performance in ways that may not be tolerated by latency or bandwidth-sensitive applications. In the above example, if flow 2 has, say, stringent bandwidth requirements, then contention with either of flows 1 and 3 will hinder meeting these requirements. Predictable performance benefits not only such latency or bandwidth-sensitive applications, but also best-effort parallel applications [21].

This motivates the need for Quality of Service (QoS) features [13] in next-generation clusters, which are not provided by any of the above-mentioned networks and protocols. Specifically, we require support for service differentiation, which entails providing different service levels (in terms of bandwidth, delay or delay jitter) to different communication flows. QoS issues have been dealt with in other contexts such as ATM networks. However, the solutions rely on switches to provide sophisticated packet scheduling disciplines [31]. The complexity and high buffer requirements of these schemes are unacceptable in clusters. Moreover, the issue of contention between multiple flows at an end host has not been raised in prior work.

1.2 Problem Statement and our Approach

The above discussion demonstrates the need for a solution for service differentiation in clusters that:

- handles multiple communication flows contending for *both* network and end-node resources,
- does not introduce much overhead, i.e., does not suffer from much performance degradation, and does not have significantly higher buffer requirements, and,
- can readily be used by higher layers, enabling applications to take advantage of the scheme.

In this thesis, we propose one such solution for Myrinet clusters. We focus on "bandwidth" as the QoS metric, and provide features for allocating bandwidths to different communication flows in the desired proportion. Our *proportional bandwidth allocation* scheme delegates the responsibility of imposing the packet scheduling discipline to source node NICs instead of switches. NIC-based "Rate control" is used in conjunction with a bandwidth resource manager. The manager uses the global knowledge of traffic patterns in the cluster to provide proportional bandwidth allocation to contending flows. The advantages of our scheme are:

- It can handle flows contending for both end-node and network resources.

- It does not require any hardware modifications, and can hence work with commodity systems. Switches do not require additional (per-flow) buffering, and are not required to detect non-conforming packets.

- As demonstrated by experimental results, proportional bandwidth allocation is provided without much overhead.

- Finally, support for "best-effort" traffic, and extension to the programming environment layer (MPI), greatly simplifies application development.

It must be noted that the scope of our work is limited to bandwidth allocation. We have not considered delay, delay jitter, or other QoS metrics. Further, we do not address the issue of CPU contention, or contention with non-network traffic on the I/O bus. In environments characterized by more that one compute-intensive job per processor, or many I/O-intensive jobs, our scheme needs to be used in conjunction with CPU and I/O bus reservation schemes to ensure end-to-end bandwidth guarantees. Finally, we have implemented our scheme only for the wormhole-routed[24] Myrinet cluster, though extensions to other networks is certainly feasible.

1.3 Thesis Organization

The rest of this thesis is organized in the following manner. Chapter 2 presents background and related work. In Chapter 3, we introduce our approach and describe it in detail. In Chapter 4, we outline design and implementation issues. Experimental results are reported in Chapter 5. Conclusions and future work are presented in Chapter 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we cover background material and discuss related work. The notion of “communication flow” is made more concrete, and the operation of user-level network protocols and wormhole-routed networks is described. We also illustrate how communications flows are serviced in existing systems. Finally, we discuss related work on QoS issues in network-based computing systems.

2.1 Communication Flows

A "communication flow" is a message stream with a well-defined source and a well-defined sink. The communication end-points (sources and sinks) are logical; there may be many such endpoints on the same network node. Additionally, many communication flows may be multiplexed over the same physical link. Each end-point may be the source or sink of many communication flows. Examples of communications end-points are Virtual Interfaces (VIs) in VIA, Queue Pairs (QPs) in the Infiniband Architecture [15], or TCP/IP sockets. Figure 2.1 further highlights these concepts. The logical diagram represents three network nodes, N1, N2, and N3; four communication end-points, A, B, C, and D; and four communication flows, (A,B), (A,C), (C,D), and (D,C).

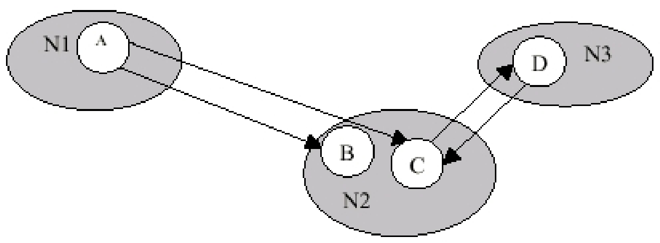

Figure 2.1: Logical representation of communication flows.

2.2 User-Level Network Protocols

User-level network protocols allow message sends and receives without operating system involvement, while still providing protected access to the network interface. Figure 2.2 illustrates typical operations assocaited with a user-level network protocol. Operating system involvement is required initially, for registration of DMAable memory and setting up channels (VIA Virtual interfaces, GM ports or FM contexts, etc.). Channels are constructs that, among other things, allow user applications to inform the NIC firmware about new send or receive requests. This is typically done by posting descriptors describing the required operation. A send descriptor specifies the location and the size of the send buffer, as well as the destination. A receive descriptor describes the location of the buffer where the received message is to be placed.

The NIC firmware obtains descriptors by polling memory-mapped descriptor queues (as in GM), or through a doorbell mechanism (as in VIA). Fetched send descriptors are serviced by DMAing data from registered memory regions into the NIC buffer and then dispatching packets to the network. On receiving packets from the network,

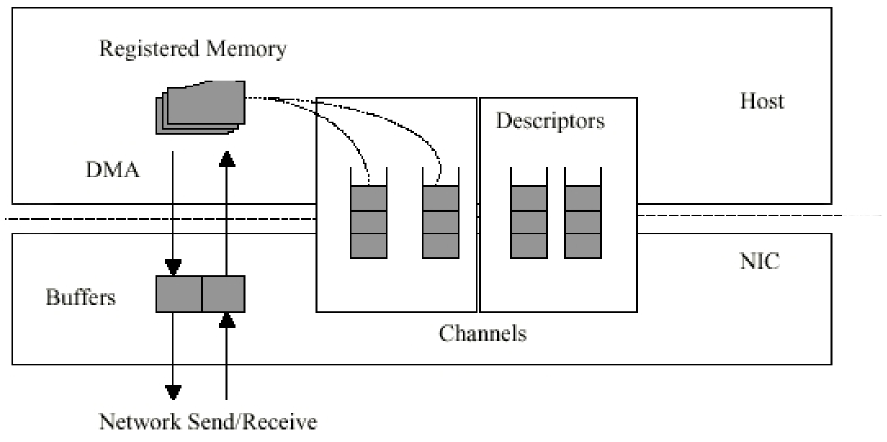

Figure 2.2: Typical operations associated with a user-level network protocol.

the NIC DMAs the data into the registered memory buffers indicated by receive descriptors.

In existing user-level network protocols, send descriptors from various flows are typically serviced in a round-robin fashion. This is inherently unfair, as it favors communication flows with longer message sizes. A fairer variation is supported by some user-level network protocols. Here, a send descriptor is not serviced all at once, but in multiple passes. Packet dispatches from various flows are interleaved to allow uniform usage of NIC resources. However, in both cases, the scheduling discipline does not take into account the service requirements of various communication flows.

The impacts of the two scheduling disciplines are illustrated in Figure 2.3. Two flows, A and B, are shown. A's message size is thrice the supported MTU (Maximum Transmission Unit), while B only has single-packet messages. In case a), with round-robin servicing of send descriptors, A uses NIC resources thrice as often as B.

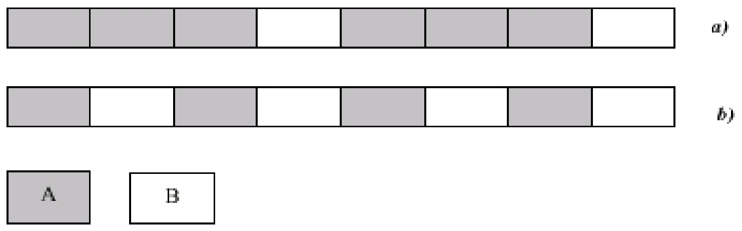

Figure 2.3: Impacts of two different packet scheduling disciplines in a user-level network protocol.

In case b), with fair interleaved packet scheduling, NIC resource usage is uniform, independent of message sizes.

2.3 Wormhole-Routed Networks

In wormhole-routed networks [24], packets are comprised of small units called flits. The key distinction from store-and-forward networks is that a flit is forwarded to the output port (if it is free) as soon as possible, without waiting for the entire packet to arrive at the input port. This allows lower latencies and better network utilization.

Most wormhole-routed switches service packets destined for the same output port in a round-robin fashion. A communication flow desiring high bandwidth may still suffer if its packets have to traverse longer paths to the destination than packets from other flows. This is so because the former may have to wait on more intermediate switches than the latter.

This is illustrated in Figure 2.4, which shows four flows, A, B, C, and D. Flows B, C, and D contend for a port on one switch, while flows A and B contend for a port on the other switch. Although both switches service the flows in round-robin fashion, flow B suffers because it has to traverse two switches on the path to its destination.

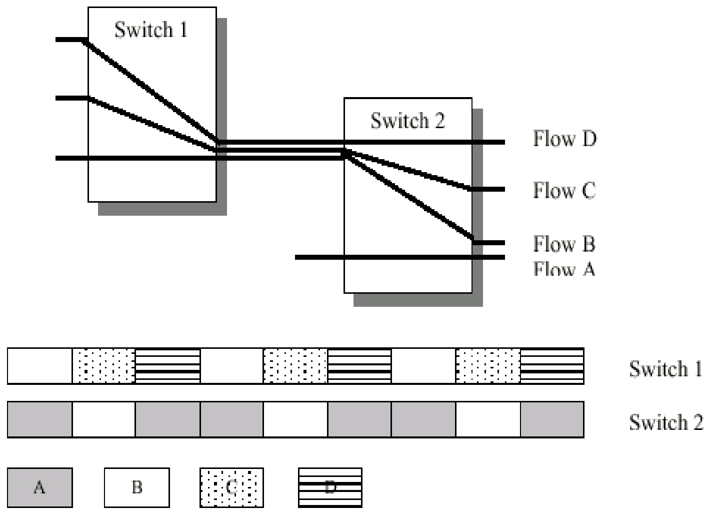

Figure 2.4: Effect of contention in wormhole-routed networks.

2.4 Related Work

In recent years, a few other studies have examined service differentiation and related QoS issues in clusters environments.

In [30] and [7], the authors propose entirely new hardware or hardware/software infrastructures with QoS features. They outline design choices and describe the components of the proposed architectures.

Other approaches ([16], [18], [23]) recommend modifications to existing hardware, particularly switches, to incorporate low-cost packet scheduling and queueing algorithms. In [12], Gerla et al describe three schemes for providing QoS in wormhole-routed networks. However, these too require modifications to switches or extra NICs on hosts.

Theoretical work in this area includes two admission control schemes ([27] and [5]) for meeting delay requirements in wormhole-routed networks. In [20], the authors present simulation results of their scheme for best-effort bandwidth reservation by manipulation of packet sizes for various communication flows.

Few studies have examined QoS issues in the context of User-level Network Protocols. FM-QoS [6] is an extension to FM that provides predictable performance by devising conflict-free communication schedules. Feedback-based synchronization [28] is used to maintain the notion of global time, required in order to meet these schedules.

Our work differs from most prior work on QoS in cluster environments in two important respects. Firstly, we simultaneously address the shortcomings of existing networks and protocols (discussed earlier in this chapter) in dealing with the issues of network and end-node contention. Secondly, our scheme does not require modifications to existing hardware. Our scheme is purely implemented in software, and firmware on the NIC.

CHAPTER 3

PROPORTIONAL BANDWIDTH ALLOCATION SCHEME

In this chapter, we describe our scheme for proportional bandwidth allocation in Myrinet clusters. The overall architecture is introduced first, followed by detailed descriptions of its various components.

3.1 Overall Architecture

As mentioned in Chapter 1, our scheme for proportional bandwidth allocation in Myrinet clusters delegates the responsibility of imposing the packet scheduling discipline to source node NICs instead of switches. NIC-based "Rate control" is used in conjunction with a bandwidth resource manager. The manager uses the global knowledge of traffic patterns in the cluster to provide proportional bandwidth allocation to contending flows.

Figure 3.1 shows where the various components of our scheme appear in the four-tiered architecture introduced in the first chapter. Note that no modifications are made to the lower-most layer (hardware). Detailed descriptions of these components appear below.

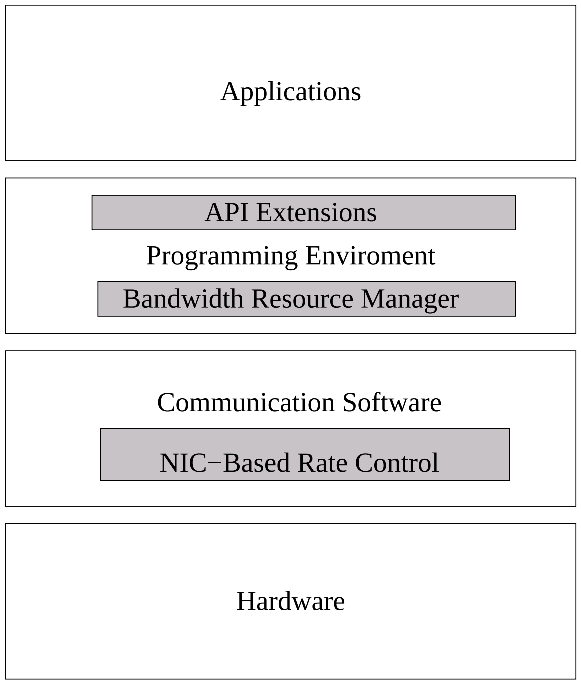

Figure 3.1: Components of the Proportional Bandwidth Allocation scheme.

3.2 NIC-Based Rate Control

"Rate control" refers to the act of imposing a specific discipline on the rate at which data is fed to the network interface and injected into the network. This has previously been used in other contexts. For example, in the Infiniband Architecture, "Injection rate control" seeks to prevent ports with high-speed links from overrunning ports with slower links. Paced TCP [19] uses rate control in high delay-bandwidth networks to allow the sender to reach its maximum window size during slow-start.

In this work, we propose the use of rate control to proportionally allocate bandwidth between various flows that contend for NIC and network resources. The rate at which packets are injected to the network interface affects how shared resources are used (and hence the bandwidth achieved) by the corresponding flow.

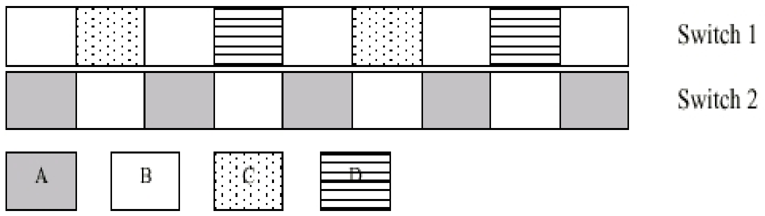

Figure 3.2: Easing network contention through rate control.

Consider two flows sourced at the same end-node. By regulating these flows by different amounts, we can affect the usage of the PCI bus, the NIC-to-host DMA engine, the NIC buffer, and the host-to-network DMA engine by the two flows, as desired.

Rate control has a similar effect on flows contending in the network. A flow can achieve the desired bandwidth if the rates of the interfering flows are suitably manipulated. For the example given in Section 2.3, if flows C and D are regulated by decreasing their packet rates to half of that for flow B, then the latter does not suffer despite having to traverse more switches (Figure 3.2). In [5], Chen et al show how rate control considerably improves network fairness and increases the probability of meeting message deadlines for real-time communication.

3.2.1 NIC-Based vs. Host-Based Implementation

There are two clear alternatives as to where rate control can be implemented: at the host or at the NIC. While a host-based implementation is simpler, it is not preferable due to two reasons. Firstly, in most user-level protocols, the host deals with messages, not packets. Messages can very large, which makes host-level rate control much coarser. More importantly, since message sends bypass the operating system,

```
At any time $t$,
Let $j$ be the communication flow such that $ndt_j = min\ (ndt_1,\ ndt_2...ndt_m)$
If $ndt_j <= t$
        Dispatch packet from flow $j$.
        $ndt_j = ndt_j + idt_j$.
```

Figure 3.3: The Rate Control Algorithm.

there is no way to ensure that communication flows are actually regulated. On the other hand, with a NIC-level implementation, all sends will have to go through the NIC firmware with rate-control features. NIC firmware is downloaded to the NIC by OS components and can hence be trusted.

3.2.2 Rate Control Algorithm

Our rate control algorithm associates each flow with a parameter called the *Inter-packet Dispatch Time* (IDT). It refers to the minimum time interval between consecutive injections of packets from a communication flow to the network interface.

At any given time, let $f_1,\ f_2...f_m$ be m communication flows sourced at a particular node. Let $(idt_1,\ idt_2...idt_m)$ be the corresponding IDT values. We also define a set of *Next Dispatch Time* (NDT) values $(ndt_1,\ ndt_2...ndt_m)$, which specify the absolute time before which a packet from a given flow should not be dispatched. The NDT values are initialized to the current time when communication starts. Then, we can use the algorithm given in Figure 3.3 to implement packet scheduling at the network interface.

The algorithm works as follows. Assuming that there are always outstanding send requests, the communication system can service them at most once in every T time units. This T corresponds to the peak achievable bandwidth of an uncontended flow,

t	0	1	2	3	4	5	6	7	8	9	10	11
ndt_A	0	2	2	4	4	6	6	8	8	10	10	12
ndt_B	0	0	3	3	6	6	6	6	9	9	12	12
Packet dispatched from flow:	A	B	A	B	A	-	A	B	A	B	A	-

Table 3.1: Working of the Rate Control Algorithm

B_{max}. Under the constraint that all IDT values are larger than T, the rates at which packets from these flows are dispatched will be in the ratio of $\frac{1}{idt_1}$:$\frac{1}{idt_2}$:$\frac{1}{idt_3}$: ......$\frac{1}{idt_m}$.

Table 3.1 illustrates the algorithm with a concrete example. Here, flows A and B have IDTs of 2 and 3 (normalized with respect to T), respectively. The NDT values for the two flows are initialized to zero. At $t = 0$, since both ndt_A and ndt_B are equal, a packet from flow A is arbitrarily chosen to be dispatched, and ndt_A is updated to 2. Now, at $t = 1$, ndt_B is less than ndt_A (and also less than t), so a packet from flow B is dispatched, while simultaneously incrementing ndt_B by the corresponding IDT value (3). The other values in the table are obtained similarly. We observe that the ratio in which packets from the two flows are dispatched is 3 : 2 or $\frac{1}{2}$: $\frac{1}{3}$ as expected. Note, also, that there are some instances, such as $t = 5$, when both ndt_B and ndt_A are larger than t, hence no packet is dispatched.

3.3 Bandwidth Resource Manager

The bandwidth resource manager is a logical entity with the following responsibilities:

- To handle bandwidth allocation or deallocations requests.

- To inform the rate control agents of changes in IDT values for various flows arising from the above-mentioned allocations and deallocations.

Note that the resource manager may actually be implemented as a collection of many entities, working as peers or in a master-slave fashion.

3.3.1 Call Admission Criteria

The criteria for admitting a bandwidth allocation request are as follows:

1. If the request is granted, then the cumulative bandwidth requirement for all flows originating or terminating at the *source node* should not exceed its peak capacity.

2. If the request is granted, then the cumulative bandwidth requirement for all flows using the *switch output ports traversed by this flow* should not exceed the corresponding outgoing link capacities.

3. Finally, if the request is granted, then the cumulative bandwidth requirement for all flows originating or terminating at the *destination node* should not exceed its peak capacity.

If the request can be granted according to the above criteria, then its IDT value is chosen as follows:

$idt = T * (\frac{B_{max}}{B_{req}})$,

where B_{req} is the requested bandwidth, and,

B_{max} is the maximum achievable bandwidth of a flow originating at the source node.

3.3.2 Support for "Best-Effort" Traffic

A desirable feature of the bandwidth resource manager is to support "best-effort" traffic, i.e., flows with unspecified bandwidth requirements. The objective of the resource manager is to maximize network usage by these flows, while not disturbing allocations for other "premium" flows. Note that this requires that the IDT values for best-effort flows be dynamically updated with changes in the allocation status.

3.4 API Extensions

API extensions are required to enable applications to request allocations or deallocations. To facilitate portability, this must be done in a standard-compliant fashion. As described in the following chapter, we have integrated our scheme with MPI. This allows applications to take advantage of the QoS features of the underlying layers, without requiring them to written directly over these layers.

3.5 Assumptions

In previous sections, we have shown that our scheme is capable of providing proportional bandwidth allocation to flows contending at end-nodes and in the network. It must be noted, however, that our scheme relies on certain assumptions. Below, we outline these assumptions and indicate why they are required.

- All traffic in the cluster is internal, with well-defined sources and sinks. External traffic, not subject to rate control, has the potential to disrupt cluster traffic.
- All nodes in the cluster belong to the same administrative domain and can be trusted. Since rate control at the source node is done by trusted software

components, such as NIC firmware, there is no need for policing at the switches. If this assumption were to be violated, switch modifications would be required for enforcement of packet rates.

- The underlying network uses deterministic source routing, which means that the path taken by packets from a given flow is fixed. This simplifies the call admission criteria.
- The underlying network features programmable NICs (for implementing rate control, without native support for the same), and provides fair access to switch ports.

It must be noted that most of these assumptions are generally valid in modern clusters with Myrinet interconnection.

CHAPTER 4

IMPLEMENTATION IN MYRINET/GM

In this chapter, we describe the implementation of the rate control feature and the bandwidth resource manager over Myrinet/GM. Overviews of Myrinet and GM are provided first. We also show how our scheme is integrated with the Message-Passing Interface (MPI).

4.1 Myrinet and GM Overviews

Myrinet is a switched gigabit-per-second local area network technology. The switches are crossbar and use wormhole switching. A Myrinet NIC connects to the I/O bus and features a programmable processor, LANai, and three DMA engines (one for host-NIC transfer, the other two for sending and receiving data to/from the network respectively). The NIC also contains a small, but fast SRAM, which holds the firmware and is also used to stage data that goes into and out of the network.

GM is a message-based communication system for Myrinet featuring low CPU overhead, low latency and high bandwidth. Reliable delivery is provided between end-points called ports. Two levels of priority are supported, and message order is preserved for messages of the same priority between the same sending and receiving ports. A combination of a port and a priority is given a special name: "sub-port".

Messages must be sent from and received into DMAable memory. Also, sends and receives are token-regulated; a client (i.e. user application) must have send and receive tokens for a port in order to perform the corresponding operations.

The software components of GM include a library, a driver and the NIC firmware called the Myrinet Control Program (MCP). The library links with the client and provides the Application Programming Interface (API). It includes functions for opening ports, allocating DMAable memory, sending and receiving messages etc. The former operations are carried out with the help of the driver, but no driver involvement is required for message sends and receives. These are carried out through direct interaction between the client and the MCP as described below.

The MCP consists of four state machines: SDMA, RDMA, SEND and RECV. When a client wishes to send a message, a send descriptor describing the message is written to a memory-mapped location in the NIC. The SDMA state machine detects the descriptor, builds a corresponding send token and inserts the token into an appropriate send queue. Further, it polls send queues as described below, prepares packets and DMAs data into the transmit buffer. The SEND state machine transfers packets from the transmit buffer to the network. This is illustrated in Figure 4.1.

As shown in Figure 4.2, queues in GM are hierarchically arranged . At the highest level is a circular connection queue, which has an entry for each remote node with which communication is active. Each connection entry points to a circular queue of sub-ports that are communicating over that connection. Lastly each sub-port entry points to a queue of send tokens. GM defines two kinds of send tokens; those that are "sendable", and those that are not. The former category refers to tokens that have not been completely serviced. The latter category includes those send tokens that have

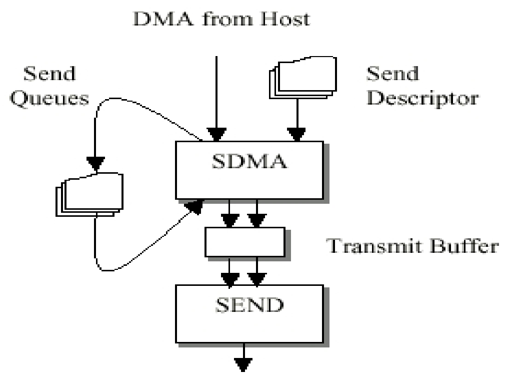

Figure 4.1: GM MCP operation for message sends.

been completely serviced, but are waiting to be acknowledged before being removed from the queue. At any time, one connection entry and a corresponding sub-port entry are designated as the "head" entries. After one packet dispatch corresponding to the first “sendable” token from the head sub-port, the sub-port queue for the head connection is rotated and so is the connection queue. This allows fair network access as described in Section 2.2.

The RECV and RDMA state machines similarly cooperate for message receives.

4.2 Rate Control Implementation

We now look at the modifications made to various components of GM for the implementation of the rate control feature.

4.2.1 GM Library

The GM library has been extended to allow the bandwidth resource manager to specify the IDT values for various communication flows. A flow is defined as the

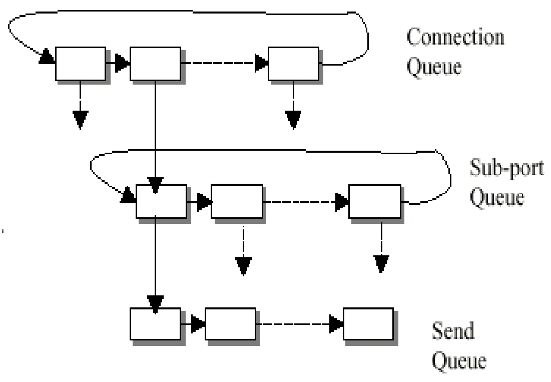

Figure 4.2: Queue organization in GM.

ordered pair (*source port*, *target port*, *priority*). This ensures that GM message order semantics are not violated, as rate control does not require out-of-order sends on packets belonging to the same flow. A limitation of the current implementation is that it allows a port to communicate with only one port on each remote machine.

The unit in which IDTs are specified is the least count of the Real Time Clock (RTC) available on the LANai processor.

4.2.2 MCP SDMA State Machine

The MCP SDMA state machine implements the rate control algorithm introduced in 3.2.2. The implementation can be divided into two logical parts:

Flow management: The SDMA state machines maintains the IDT, NDT, and validity information for all flows sourced at that node. The state machine marks flows as valid or invalid, depending upon whether or not there are send tokens for the flow at that moment. As described above, the IDT values are supplied by the bandwidth resource manager. The NDT value is initialized to zero when a flow is initiated. When a send token for that flow is enqueued, the NDT value is updated in the following manner:

```
If the send token is the first "sendable" one in the queue
        $NDT = max\ (NDT,\ RTC)$.
```

Packet scheduling: The SDMA state machine determines the flow from which the next packet is to be dispatched in the following manner:

```
Find the minimum NDT from all valid flows. Let us call it $NDT_{min}$.
If $NDT_{min} >= RTC$
        Service the first "sendable" token (if any) for the corresponding flow.
        $NDT + IDT$.
```

4.3 Bandwidth Resource Manager Implementation

The bandwidth resource manager is implemented as a set of manager processes, one on each node in the cluster. One of these processes is designated as the "master" manager. The GM Library has been extended to allow applications to request allocation or deallocation of bandwidth. These new functions, which are blocking in nature, result in requests being issued to the the local manager. The local manager forwards these requests to the "master" manager, which uses the admission criteria described in Section 3.3.1, to decide if the request can or cannot be accepted. The requesting manager is informed of the decision. If the request has be accepted, all relevant managers are apprised of the updates in IDT values.

To support "best-effort" flows, the surplus bandwidth at each node and switch port is fairly set aside for all such flows incident on the node or the port. The bandwidth allocated to a flow is the *minimum* of these values for all nodes and ports traversed by it. Note that this simple arbitration is sub-optimal in terms of overall network utilization.

4.4 Integration with MPI

While applications can directly be written over GM, ease of programming and portability aspects are simplified if the programmer is presented with a familiar programming model, such as the Message-Passing Interface (MPI). In this section we give a brief overview of MPI, its implementation over GM, and the modifications we made to this implementation.

4.4.1 MPI Overview

MPI [22] is a message passing library which offers a host of point-to-point and collective interprocess communication functions to a set of single threaded processes executing in parallel. All communication is performed within the confines of a communicator. A communicator is a process group, with each process having a unique rank in the group, ranging from 0 to N-1, where N is the number of processes in the communicator.

MPICH [11] is a freely available, portable implementation of MPI. The mechanism for achieving portability is a specification called the *Abstract Device Interface (ADI).* All MPI functions are implemented in terms of this ADI, while the ADI layer itself can use message-passing functions native to the underlying system. MPICH-GM is an implementation of MPI for Myrinet clusters that uses GM as the underlying message-passing system.

4.4.2 MPICH-GM Modifications

In order to take advantage of the QoS features provided by our version of GM in a standard-compliant fashion, we use the "Attribute" mechanism provided by MPI. MPI provides functions to set and get the values of attributes for a communicator. An attribute is identified by an integer, *keyval*, and its value may be of any arbitrary type (*void ** in C). When used with special *keyvals* for QoS attributes, the "set attribute" call is mapped to the GM function for requesting bandwidth allocation or deallocation. Process ranks are converted to appropriate GM node and port identifiers. Also, since the MPI layer incurs additional overhead, bandwidths achieved at the MPI level are lower than raw GM bandwidths. Therefore, the requested bandwidth is mapped the corresponding raw GM bandwidth. The "get attribute" call is used similarly for determining if the previously made request was accepted or denied.

Note that the use of the attribute mechanism for this purpose has previously made in [26].

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, we discuss our experimental results. We describe the test environment, and present results that demonstrate the effectiveness of our scheme in providing proportional bandwidth allocation, without adding much overhead.

5.1 Test Environment

We evaluated our implementation on a cluster of machines with 300 MHz Pentium II processors, 128 MB RAM, running RedHat Linux 6.0 with kernel version 2.2.5. These machines were connected by a 16-port Myrinet switch and had LANai 4.3 NICs with 33 MHz processors.

The experiments we conducted involved measuring the bandwidths and latencies achieved for different message sizes or under different communication flow patterns. Round-trip latency was measured using the standard ping-pong test. Bandwidth measurements involved pumping a large number of messages, getting a small-sized acknowledgement from the receiver, and dividing the total volume of data transferred by the total elapsed time.

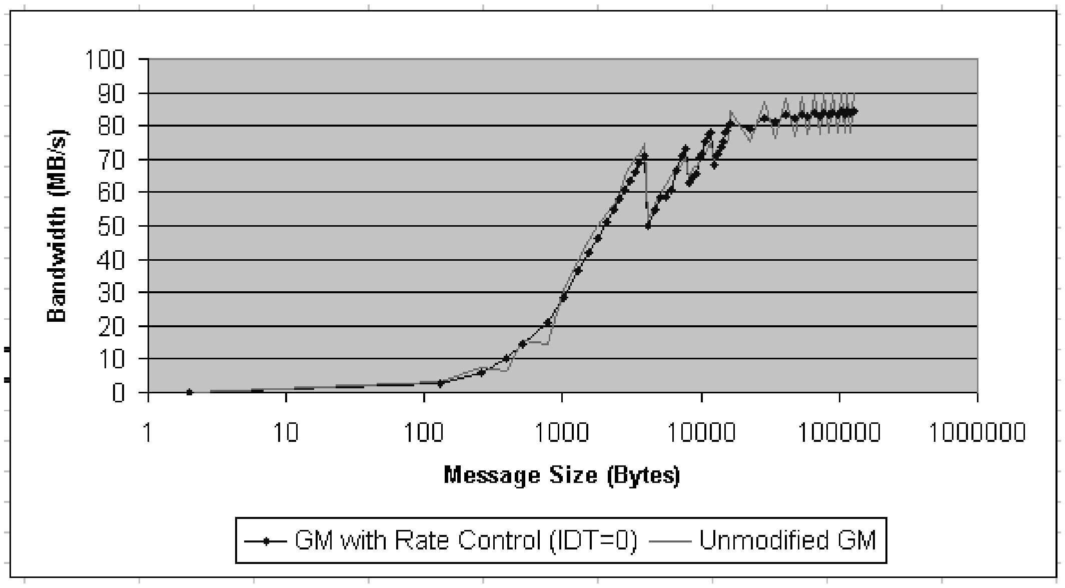

Figure 5.1: Effect of the Rate Control feature on bandwidth.

5.2 Overhead of the Rate Control Mechanism

We determined the overhead of the rate control mechanism by comparing the performance of unmodified GM with the modified version. Figure 5.2 demonstrates that the rate control feature incurs an overhead of less than 1% in terms of latency, averaged over a range of message sizes. Similarly, Figure 5.1 shows that the average bandwidth loss for a range of message sizes is less than 1%, while the worst-case bandwidth loss for any message size in the same range is 4.5%. Note that for some message sizes, the modified version of GM appears to perform better than the unmodified version. This is because of the difference in the message segmentation schemes in the two versions. While the original version of GM attempts to segment multi-packet messages into equal sized chunks, we have disabled that feature for our version. In our case, all packets (other than the last one) for a message are MTU-sized.

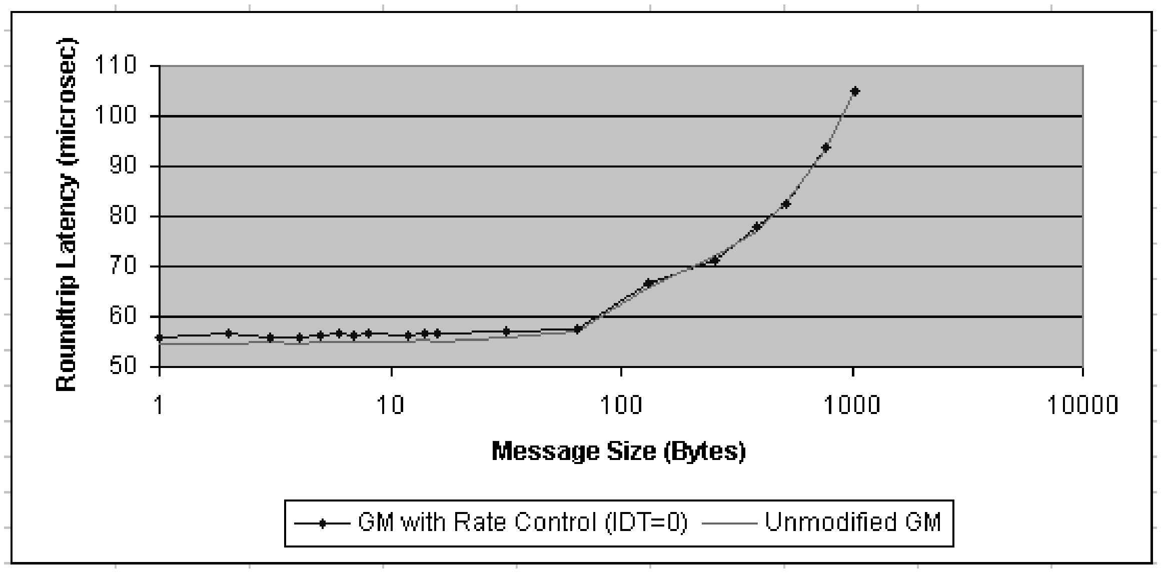

Figure 5.2: Effect of the Rate Control feature on round-trip latency.

5.3 Determining Parameter Values

In order to map the bandwidth requirement of a flow to the corresponding IDT, the bandwidth manager uses the values of the parameters T and B_{max}. To determine these values, we varied the IDT for a flow with 64KB-sized messages and measured the achieved bandwidth. B_{max} corresponds to the bandwidth achieved for IDT=0 (unregulated flow). Further, since packets are dispatched once every T time units, setting an IDT value less than T will not affect the achieved bandwidth.

Using the above reasoning, we determined the value of T to be 96 LANai 4.3 RTC units. As shown in Figure 5.3, for values of IDT less than 96, the achieved bandwidth is equal to B_{max}. These values have been used for the remainder of our experiments.

In order to decide whether a new flow can be admitted, the bandwidth manager must also be aware of the maximum rate at which packets can be dispatched from

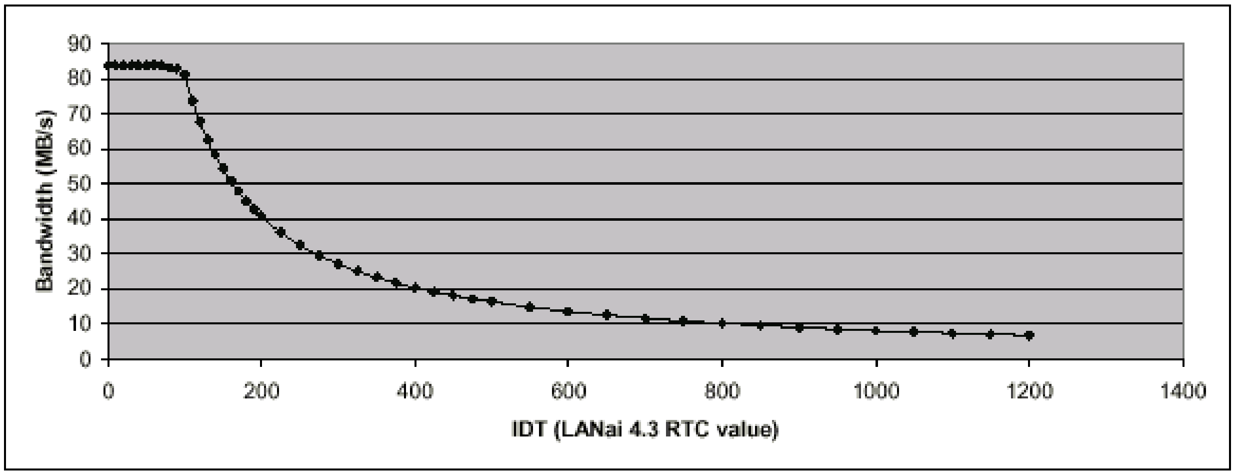

Figure 5.3: Effect of IDT on the achieved bandwidth.

a node, for different values of the incoming packet rate. We used a three node configuration to determine these values. We established flow 1 from node A to node B, and varied its IDT during the experiment. For each of these IDTs, we calculated the maximum bandwidth possible for a second flow, flow 2, from node B to node C that *does not* affect the first flow. In other words, flow 1 achieves almost the same bandwidth for a given IDT as it would have, had flow 2 not existed. These maximum bandwidths can simply be read off the graph, which is plotted in Figure 5.4. Note that the sum of the bandwidths for the two flows is not constant. This is because the amount of processing required for incoming messages is less than that for outgoing messages. When packets are arriving at a much higher rate than the rate at which they are being dispatched, the NIC is capable of doing more work per unit time, leading to higher net throughput.

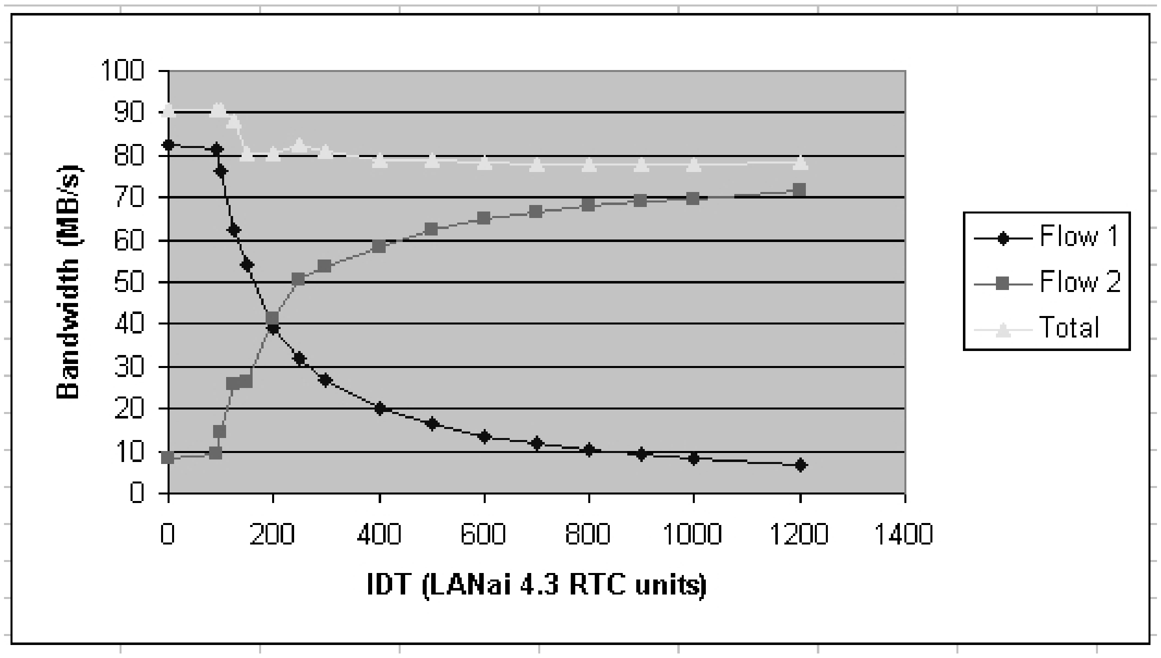

Figure 5.4: Maximum bandwidth of an outgoing flow for various incoming packet rates.

5.4 Proportional Bandwidth Allocation Using Rate Control

In this section, we discuss our results that show how rate control achieves proportional bandwidth allocation for a variety of communication flow patterns.

5.4.1 Single Sender with Multiple Receivers

In these sets of experiments, we had a number of flows sourced at the same network node, but destined for different network nodes. As a result of our single crossbar switch topology, these flows do not contend in the network, but do experience end-node contention. Figure 5.5 clearly demonstrates that by manipulating the IDT values for two flows (with 64KB messages), we can proportionally allocate bandwidth between them. Note that the bandwidth is being shared in the inverse ratios of the

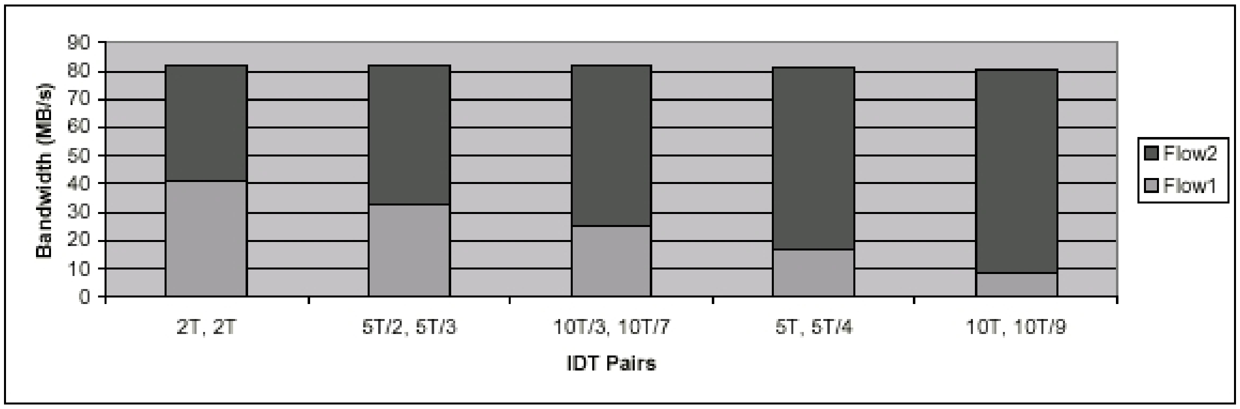

Figure 5.5: Proportional bandwidth allocation for two flows sourced at the same node.

IDT values. For example, when the IDT pair ($2T$, $2T$) is used, both flows achieve bandwidths of 40.91 MB/s each; thus the bandwidth is allocated in the ratio of $\frac{1}{2T}$:$\frac{1}{2T}$, or 1 : 1. For the IDT pair ($\frac{10T}{3}$, $\frac{10T}{7}$), the flows achieve bandwidths of 24.96 MB/s and 56.88 MB/s, which are again in the ratio of $\frac{3}{10T}$:$\frac{7}{10T}$, or 3 : 7. The same is also true for three flows sourced at the same node (Figure 5.6). For the IDT triple ($2T$, $4T$, $4T$), the achieved bandwidths of (39.81, 20.5, 20.5) MB/s are in the ratio of $\frac{1}{2T}$:$\frac{1}{4T}$:$\frac{1}{4T}$, or 2 : 1 : 1.

5.4.2 Single Receiver with Multiple Senders

In this experiment, we had two flows sourced at different nodes, directed to the same destination. These flows experience network contention (for the switch port leading to the receiving NIC). Figure 5.7 shows that our scheme still divides the bandwidth for the two flows proportionally, in the inverse ratio of the IDT values. For the IDT pair ($10T$, $\frac{10T}{9}$), the achieved bandwidths of (8.52, 75.91) MB/s are in the ratio of $\frac{1}{10T}$:$\frac{9}{10T}$, or 1 : 9.

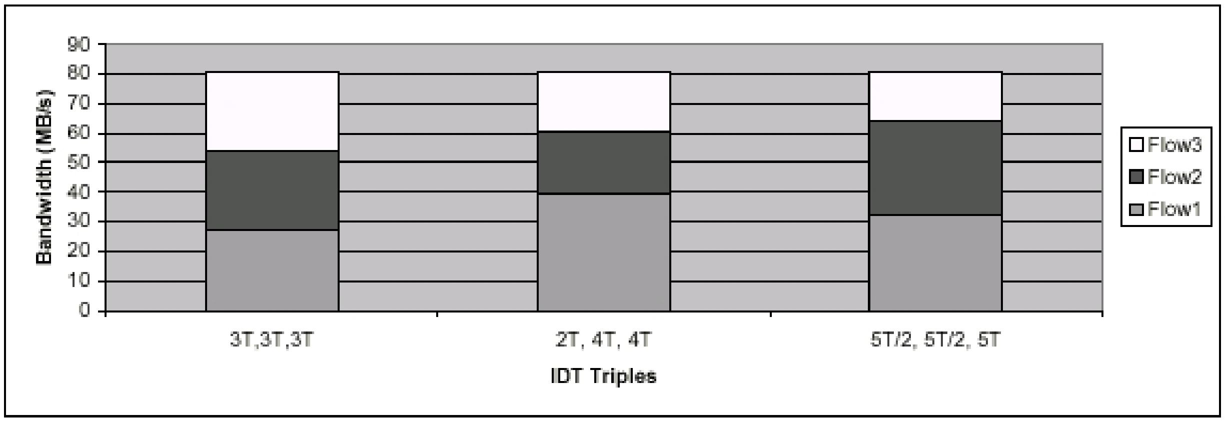

Figure 5.6: Proportional bandwidth allocation for three flows sourced at the same node.

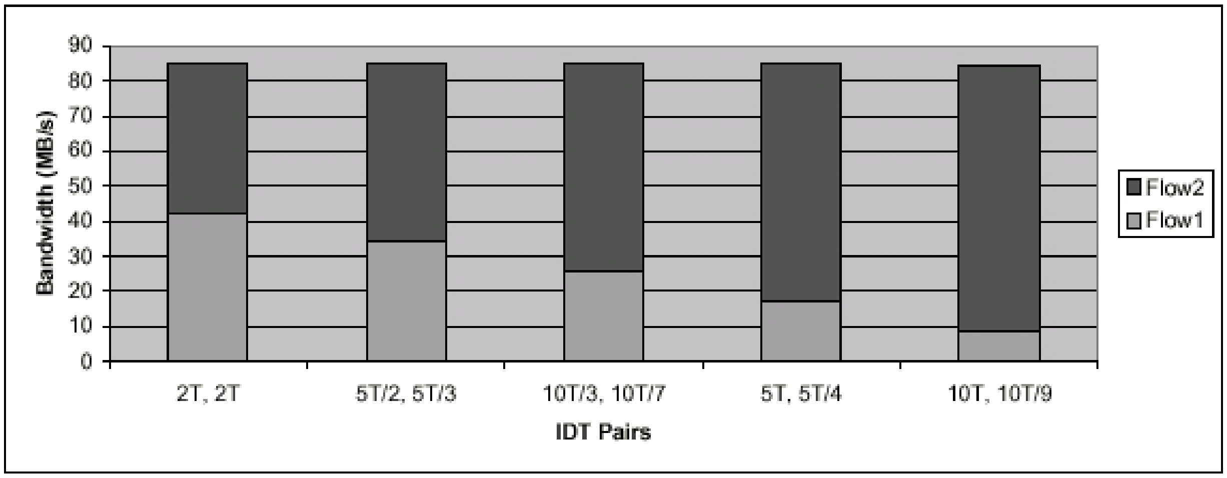

Figure 5.7: Proportional bandwidth allocation for two flows destined for the same node.

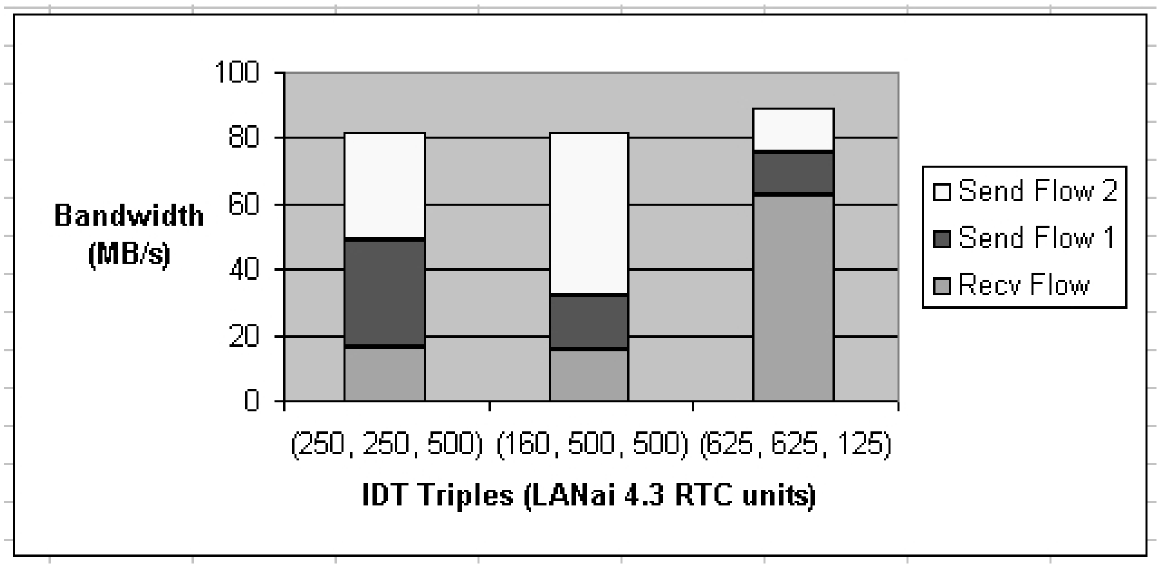

Figure 5.8: Bandwidth sharing between three flows: two sourced at a node, and the third destined for that node.

5.4.3 Receiving and Sending at the Same Node

In this experiment with three communication flows, packets from one flow arrive at a network node at some rate. The maximum bandwidth achievable by an outgoing flow, for a given incoming packet rate (as determined from Figure 5.4) is shared by two flows sourced at that node, in the inverse ratio of the IDT values (Figure 5.8). For the IDT triple (250, 250, 500), the incoming flow achieves a bandwidth of 16.37 MB/s corresponding to IDT=500. As can be read from Figure 5.4, the maximum outgoing bandwidth achievable for this incoming rate is approximately 65.0 MB/s. This value is shared equally by the two send flows, i.e., in the ratio of $\frac{1}{250}$:$\frac{1}{250}$.

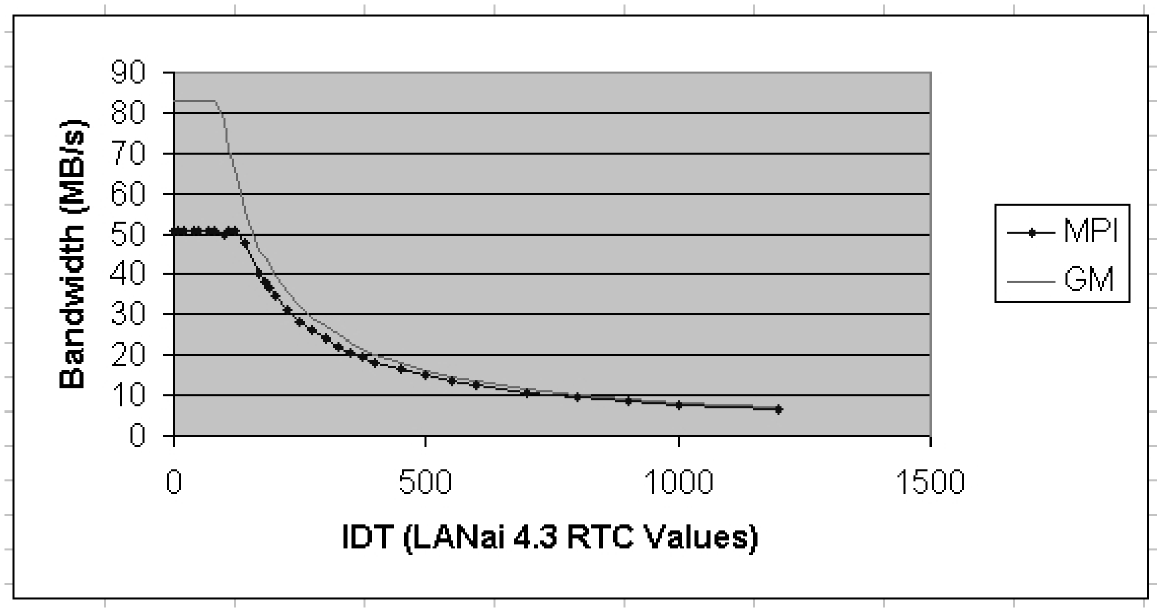

Figure 5.9: Effect of IDT on the bandwidth achieved at the MPI level.

5.5 MPI-Level Evaluation

While all the above experiments were conducted at the GM level, for the following experiments, our bandwidth measurement application made use of MPI communication primitives.

5.5.1 Effect of the IDT on the Achieved Bandwidth

Figure 5.9 shows the effect of varying the IDT on the bandwidth achieved by a communication flow at the MPI level. For comparison, the GM level bandwidth is also shown. The results clearly demonstrate the overhead of the MPI layer: even for IDT values less than 140, the communication flow achieves bandwidths in the range of 50 MB/s, though the underlying GM layer is capable of sending at much higher rates.

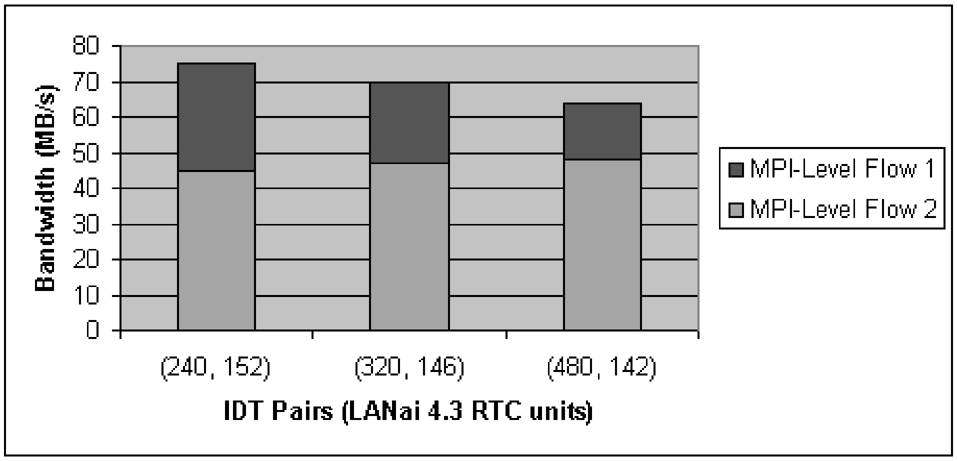

Figure 5.10: Proportional bandwidth allocation between two flows sourced at the same node (MPI-level).

5.5.2 Proportional Bandwidth Allocation between MPI-Level Flows

Figure 5.10 shows the MPI-level bandwidth achieved by two flows sourced at the same node, but destined for different nodes. Here again, the achieved bandwidths are in the inverse ratio of the IDT values. Note that for MPI-level flows, we do not use IDT values less than 140, since the overhead introduced by the MPI layer prevents flows with such values from reaching a potentially higher bandwidth.

5.6 Best-Effort Communication Flows

In these experiments, we had best-effort communication flows co-existing with premium communication flows in the cluster. We sought to observe the variations in the bandwidths allocated to best-effort flows, as premium flows were initiated and terminated. We used *packet dispatch frequency* as a measure of the achieved bandwidth. For this purpose, we collected the LANai RTC values at packet send events in the MCP, and post-processed the collected data.

In the first experiment, we started with one best-effort communication flow that was allowed to send out data at peak rate. At later points in time, we started premium flow 1 (with a bandwidth requirement of 40 MB/s), and then premium flow 2 (with a bandwidth requirement of 20 MB/s) on the same node. These flows were active for some time, before terminating. Figure 5.11 shows the packet dispatch frequencies for the three flows during these different stages, with zero values indicating that the flow is not active (Note that for clarity, the graph has been perturbed a bit; near-zero values should be interpreted as zeros). As can be seen, the packet rate of the best-effort flow is adjusted according to the current allocation status of the premium flows. When premium flow 1 is active, the packet dispatch frequency of the best-effort flow is around 4.7 packets every thousand ticks of the LANai 4.3 RTC, which corresponds to an approximate bandwidth of 38 MB/s. When premium flow 2 is also active, this value drops to approximately 18 MB/s.

Figure 5.12 shows similar results for the case with two best-effort flows, and one premium flow on the same node. Note that the bandwidth not allocated to the premium flow is shared equally by the two best-effort flows. For example, before the premium flow is started, the two best-effort flows each get around 39 MB/s, which is approximately half the peak bandwidth. When the premium flow (with a 20 MB/s bandwidth requirement) is active, the two best-effort flows share the surplus 58 MB/s equally. As in the previous case, an uncontended best-effort flow is allowed to send packets at peak rate.

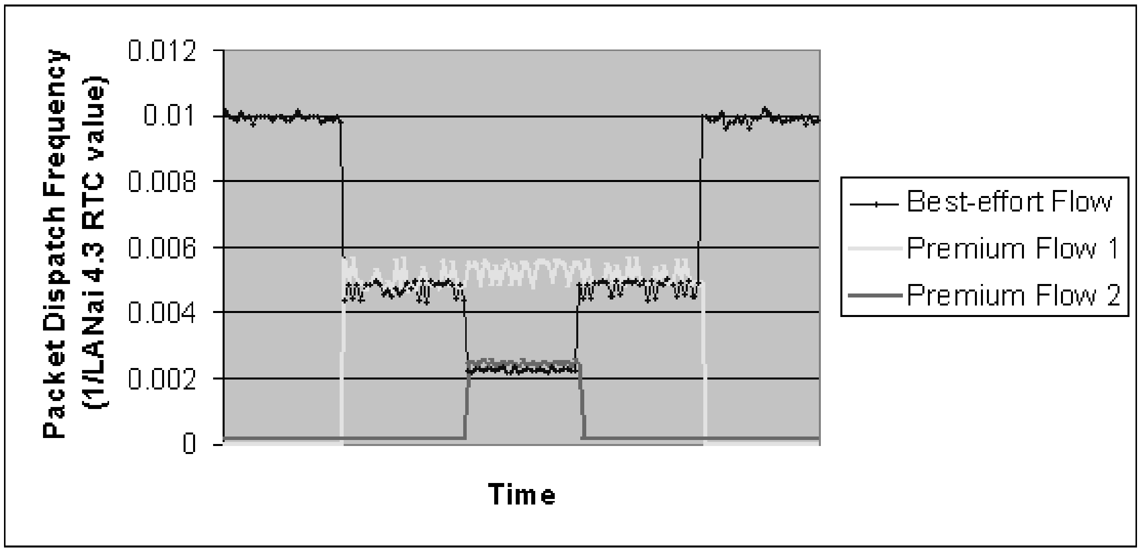

Figure 5.11: Packet dispatch frequencies: two premium flows and one best-effort flow.

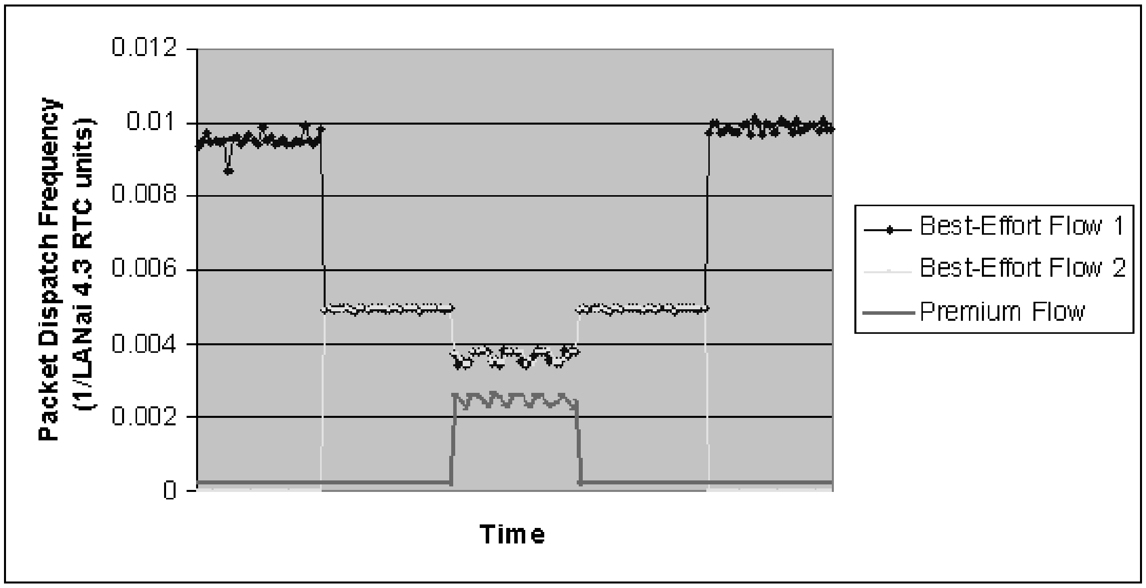

Figure 5.12: Packet dispatch frequencies: one premium flow and two best-effort flows.

5.7 Summary of Results

In this chapter, we demonstrated how our scheme can provide proportional bandwidth allocation for a variety of communication flow patterns, both at the GM level, and at the MPI level. We also showed that our scheme does not incur much overhead. On an average, increases in the latency and bandwidth losses are both minimal. Finally, we demonstrated how our scheme can support best-effort flows by dynamically adjusting IDT values according to the allocation status of premium flows.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed a proportional bandwidth allocation scheme for Myrinet clusters. The scheme uses rate control at source nodes, in conjunction with the global knowledge of cluster traffic patterns. The implementation of NIC-based rate control, and the bandwidth resource manager for the Myrinet/GM system was presented. We also showed how the scheme can be integrated with the Message-Passing Interface. Experimental results demonstrate that we can handle both end-node and network contention, without adding significant overhead. Further, no hardware modifications are necessary, allowing the use of commodity components.

As future work, we would like to develop schemes for other QoS metrics, such as latency, delay jitter etc. We could also like to investigate how other programming abstractions, such as the shared-memory model, can take advantage of a communication flow-centric QoS scheme.

Another trend in network-based computing is the use of computing grids for the deployment of geographically distributed applications. End-to-end QoS in these environments requires co-reservation of heterogeneous resources (networks, CPUs, I/O units), across multiple domains. The Globus Architecture for Reservation and Allocation (GARA) [10] addresses these issues and we intend to integrate our scheme with GARA.

BIBLIOGRAPHY

[1] T. Anderson, D. Culler, D. Patterson. A Case for Networks of Workstations (NOW). IEEE Micro:pages 54-56, Feb 1995.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kualwik, C. L. Seitz, J. N. Seizovic, Wen-King Su. Myrinet-a gigabit-per-second local-area network. IEEE Micro, 15(1):29-36, February 1995.

[3] T. Bohman. Shared Memory Computing Architectures for Real-Time Simulation - Simplicity and Elegance, Technical Report, Systran Corporation, 1994.

[4] I. Chlamtac, A. Ganz, M. G. Kienzle, A HIPPI Interconnection System. IEEE Transactions on Computers, Vol. 42 No. 2:138-150, Feb 1993.

[5] B. Chen, H. Li, W. Zhao. Meeting Delay Requirements in Computer Networks with Wormhole Routing. In Proceedings of the IEEE International Conference on Distributed Computing, Hong Kong, May 1996.

[6] K. Connelly, A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In Proceedings of Supercomputing Conference, San Jose, CA, November 1997.

[7] H. Eberle, E. Oertli. Switcherland. A QoS communication architecture for workstation clusters. In Proceedings of ACM ISCA '98, Barcelona, Spain, June 1998.

[8] T. von Eicken, D. Culler, S.C. Goldstein, K.E. Schauser. Active messages: a mechanism for integrated communication and computation. In Proceedings of the 19th International Symposium on Computer Architecture:256–266, May 1992.

[9] T. von Eicken, A. Basu, V. Buch, W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In Proceedings of the 15th ACM Symposium on Operating System Principles, December 1995.

[10] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. International Workshop on Quality of Service, 1999.

[11] W. Gropp, E. Lusk, N. Doss, A. Skjellum. A High Performance, Portable Implementation of the MPI message Passing Interface Standard. In Parallel Computing, 22(6), 1999, pages 789-828.

[12] M. Gerla, B. Kannan, B. Kwan, P. Palnati, S. Walton, E. Leonardi, F. Neri. Quality of service support in high-speed, wormhole routing networks. In International Conference on Network Protocols, October 1996.

[13] R. Guerin, H. Schulzrinne. Network Quality of Service. In The Grid: Blueprint for a Future Computing Infrastructure, pages 479-503, Morgan Kaufmann Publishers, 1999.

[14] The GM Message Passing System. Documentation available at http://www.myri.com/scs/index.html.

[15] The Infiniband Architecture Specification, Volumes 1 and 2, Release 1.0, available at http://www.infinibandta.org.

[16] S. S. Kanhere, A. B. Parekh, H.Sethu, Fair and efficient packet scheduling in wormhole networks. In the Proceedings of the 14th International Parallel and Distributed Processing Symposium, 2000.

[17] P. Keleher, A. L. Cox, S. Dwarkadas, W. Zwaemepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In Proceedings of the 1994 Winter Usenix Conference, Jan 1994.

[18] J. H. Kim, A. Chien. Rotating combined queuing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks. In Proceedings of the International. Symposium on Computer Architecture: 226-236, May 1996.

[19] J. Kulik, R. Coulter, D. Rockwell, C. Partridge. Paced TCP for High Delay-Bandwidth Networks. In Fourth IEEE International Workshop on Satellite-Based Information Systems, Rio de Janeiro, Brazil, December 8th, 1999.

[20] B. Kwan, P. Hu, N. Bambos, L. Kleinrock, J. Touch, H. Xu. Segmentation-based Bandwidth Reservation in Wormhole Routing Networks: A Performance Study. 1996. Submitted for publication.

[21] R. Mraz. Reducing the variance of point-to-point transfers for parallel real-time programs. IEEE Parallel and Distributed Technology: 20-31, Winter 1994.

[22] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, March 1994.

[23] M. W. Mutka. Using Rate Monotonic Scheduling Technology to Support Real-Time Communications in Wormhole Networks. In the Proceedings of the Second Workshop on Distributed and Parallel Real-Time Systems: 194-199, April, 1994.

[24] L. M. Ni, P. K. McKinley. A survey of wormhole routing techniques in direct networks. IEEE Computer, vol. 26, pages 62-76, Feb. 1993.

[25] S. Pakin, M. Lauria A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In Proceedings of Supercomputing '95, December 1995.

[26] A. J. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, B. Toonen, MPICH-GQ: Quality-of-Service for Message Passing Programs. In Proceedings of Supercomputing '00, November 2000.

[27] S. Sundaresan, R. Bettati. Distributed Connection Management for Real-Time Communication over Wormhole-Routed Networks. Technical Report 96021, Department of Computer Science, Texas A&M University, December 1996.

[28] H. J. Song, A. Chien. Feedback-Based Synchronization for QoS Traffic in Cluster Computing. In Proceedings of the 6th International Conference on Parallel Interconnects, 1998.

[29] Virtual Interface Architecture Specification, Version 1.0. Available at http://www.viarch.org.

[30] R. West, R. Krishnamurthy, W. Norton, K. Schwan, S. Yalamanchili, M. Rosu, S. Chandra. Quic: A quality of service network interface layer for communication in NOWs. In Proceedings of the Heterogeneous Computing Workshop, in conjunction with IPPS/SPDP, San Juan, Puerto Rico, April 1999.

[31] L. Zhang, Virtual Clock: A new traffic control algorithm for packet switching networks. In Proceedings of ACM SIGCOMM'90:pages 19-20, Philadelphia, PA, September 1990.