# Incorporating Quality-of-Service in the Virtual Interface Architecture *

Shailabh Nagar

IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY

Chun Liu† Gokul B. Kandiraju†
Anand Sivasubramaniam† N. Gautam‡
†Department of Computer Science & Eng.
‡Department of Industrial & Manufacturing Eng.
The Pennsylvania State University
University Park, PA

## Abstract

*User-level Networking (ULN) is gaining rapid acceptance in the commercial world with Virtual Interface Architecture (VIA), and Infiniband more recently, being touted as the interface of choice to diverse peripherals. There is an important issue with respect to ULNs that has been largely unaddressed. This is the issue of accommodating multiple communication channels (scalability) and being able to sustain the Quality-of-Service (QoS) requirements of each of these channels. Removing the operating system from the critical path makes the scalability and QoS issue all that much harder since the supervisory role to prevent interference across channels has to be performed by the network interface. This paper argues for a different approach to structuring the processing of operations (called PVIA) on the network interface than what is widely used. This newer approach makes it easier to incorporate QoS features, which is referred to as QoSVIA. All these mechanisms have been implemented on an actual Myrinet cluster. The results from the implementation, together with detailed simulations, illustrate the potential of QoSVIA being able to accommodate the QoS needs of a larger number of channels than the mechanisms in use today.*

## 1. Introduction

A driving force behind the advocation of clusters has been the lower latencies achieved by a reduction in the overheads of conventional networking stacks and the avoidance of crossing protection boundaries [17, 11, 14, 12, 15, 9, 10]. Hardware [7] and software [3, 4, 5, 1, 2] implementations of VIA have been or are currently being developed, together with preliminary studies examining the performance of this platform.

The previous studies and the published performance results of ULNs have only looked at latency and bandwidth of a single communication channel [6, 4]. Only a recent study [13] has looked at the issue from the scalability viewpoint. In this paper, QoS is a broad term that is used to describe the desirable performance criteria for a given communication channel. This could include (a) low latency (*Best-Effort*), (b) high bandwidth/throughput (*BE*), (c) (deterministic or statistical) guaranteed latency (bound on the interval between injection at one node and ejection at another), and (d) (deterministic or statistical) guaranteed bandwidth (over a period of time) which could be *CBR* (Constant Bit Rate) or *VBR* (Variable Bit Rate) traffic. In this paper, we limit ourselves to Best-Effort (BE), CBR and VBR channels, with the term QoS channel being broadly used to refer to the latter two.

Developing QoS-aware communication for a cluster requires a substantial design effort at several levels of the system architecture: network, network interface and operating system scheduler.

For the purposes of this discussion, we assume that the network is already QoS-aware, i.e. one of the other techniques is used to ensure that when different channels across nodes pump in messages, the network would deliver them to the corresponding destinations as specified by the negotiated QoS parameters. We also do not consider the issue of host CPU scheduling in this paper, and assume that the processes can inject/eject messages as needed.

This paper examines the issues in designing a ULN such as VIA on an off-the-shelf interface (Myrinet), and first shows that the optimizations for a single channel which several current implementations use (which we refer to as SVIA), may not be very efficient with the presence of multiple channels. A firmware design (called PVIA) that attempts to increase the concurrency of activities on the NIC is shown to give better performance in such cases. Subsequently, the PVIA design is augmented with a rate-based scheduler (QoSVIA) to meet the bandwidth requirements of CBR and VBR channels. PVIA and QoSVIA have been implemented and evaluated on a Pentium (running Linux)

* This research has been supported in part by NSF Career Award MIPS-9701475, NSF grants DMI-0075572, CCR-9988164, CCR-9900701, CCR-0097998, EIA-0103583 and equipment grants EIA-9818327, EIA-9617315.

cluster connected by Myrinet, and compared with a publicly distributed VIA implementation [4] that is representative of a SVIA design.

The rest of this paper is organized as follows. The next section gives a quick background on cluster network interfaces (Myrinet in particular) and ULN (with respect to VIA). The design and implementation of the different firmware designs is discussed in Section 3. Experimental evaluation of this system is conducted in Section 4. Finally, Section 5 summarizes the contributions of this paper and outlines directions for future work.

## 2. Overview of ULN and NIC Operations

We describe the ULN and NIC operations by discussing the implementation of VIA.

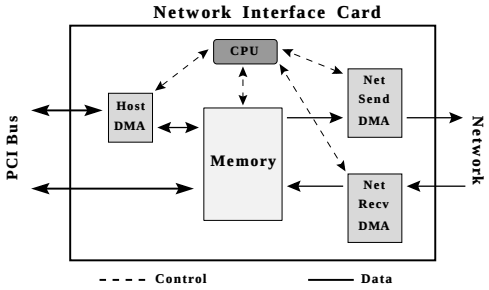



**Figure 1. Myrinet NIC**

In ULNs, the operating system (device driver) is used only to set up channels of communication (when protection checks are done). Subsequently, user processes can directly interface to the network. This has been made possible due to innovations in network interface cards (NIC), such as Myrinet, which are able to monitor regions of memory for messages to become available, and directly stream them out onto the network without being explicitly told to do so by the operating system running on the host CPU. Similarly, an incoming message is examined by the NIC, and directly transferred to the corresponding application receive buffers in memory (even if that process is not currently scheduled on the host CPU). Figure 1 shows the network interface card for Myrinet. This card contains a processor called LANai, a DMA engine (referred to as *HDMA* in this paper) which is used to transfer data between host memory and card buffer (SRAM), a DMA engine (referred to as *NSDMA*) to transfer the data from SRAM out onto the network, and a DMA engine (referred to as *NRDMA*) to transfer data on to the SRAM from the network.

The following actions are important considerations when implementing a ULN:

- How does an application process notify the NIC about a message that it wants to send, or how does the NIC detect the presence of an outgoing message? Similarly, how does it inform the NIC about a message that it wants to receive?
- How and when should the NIC bring down the message from the host memory down onto its local buffer? Subsequently, when should this message be sent out on the network?
- How and when should the NIC pick up an incoming message from the network? Subsequently, when should this is transferred up to the host memory?

The hardware features on the Myrinet which help us implement these actions are discussed next. In this paper, we are not examining the protection operations of VIA and limit ourselves to the Send and Recv operations.

### 2.1. Doorbells

Event (send or receive) notification to the NIC by application processes is done in VIA by a mechanism called *doorbells*. Each VIA channel has two sets of doorbells, one each for send and receive. When an application wants to send or receive a message, it creates a header for it (called a *descriptor*), makes this accessible to the NIC, and then rings a doorbell for that channel. The newer interface (LANai 7) provides a hardware queuing mechanism whereby (doorbell) writes to different channels can be merged (by hardware) into a single queue in FCFS order, which can then be polled by the LANai [13] to save the polling cost. But this can lead to additional overheads if the implementation needs to service messages in a different order (based on QoS requirements).

### 2.2. HDMA Operation

Once a doorbell (send or receive) is detected, the LANai first needs to get hold of the descriptor before it can get to the corresponding data buffer. Since the NIC SRAM is a precious resource, some of the VIA implementations [4] keep the descriptors on the host memory, and this needs to be DMA-ed (using HDMA) down to the card SRAM. The LANai then examines the descriptor before it DMAs (again using HDMA) down (for a send, with the reverse direction used for a receive) the data.

The newer interface (LANai 7) offers a hardware queuing mechanism for DMA operations as well, wherein the LANai can simply insert a DMA request in a queue, and can in its own time come back and check the status for completion.

### 2.3. NSDMA and NRDMA operation

These DMAs on the Myrinet NIC do not provide hardware queuing features. If the LANai has request for the NSDMA when it is busy, then it has to maintain its own queue (in software), and come back periodically to check for availability.

## 3. Firmware Design

The key determinant to the performance and efficacy of a ULN mechanism is the firmware running on the LANai (called the LANai Control Program or LCP), which ties all these actions together and coordinates the different concurrent activities in a seamless manner to maximize the throughput and responsiveness. For guaranteed service, the scheduling and coordination of NIC activities becomes even

more critical. At the same time, the LCP has to be fairly efficient since the LANai is relatively slow compared to the host CPU.

As described in the previous section, the LCP needs to perform the following operations:

- Poll doorbell queue(s) for a send/receive notification.
- Transfer the descriptor associated with the doorbell from the host memory onto the SRAM using HDMA.
- Transfer the data associated with a send descriptor from host memory to SRAM using HDMA.
- Transfer the packet onto the network using NSDMA.
- Pick up packet from network using NRDMA.
- Transfer data from SRAM to the host memory using HDMA.
- Transfer completion information (of send/receive) to host memory using HDMA.

The speed at which these operations are performed, and any overlap/pipelining of operations to maximize concurrency are important for not just the latency/bandwidth of a single channel, but also the scalability (and perhaps meeting the guaranteed performance) of the ULN as a whole. The LANai itself is the only entity that can initiate these operations and three such ways of structuring it are discussed below.

### 3.1. Sequential VIA (SVIA)

A simple way of structuring the LCP is to sequentialize the entire sequence of operations itemized above. The resulting sequence of operations from the time the application posts a send, till the completion information is propagated back to the host is shown pictorially in Figure 2. A similar sequence can be performed for the receive consequently. We refer to such an implementation as SVIA.

The advantage with this approach is that it is optimized to bring down the latency of a single message on a channel from the time its doorbell is detected to the time it is sent out on the network. There is very little time lost between the completion of one operation of the message to the initiation of the next. As a result, several current VIA implementations [4] use this approach. At best, there is a slight overlap in [4] where the NRDMA is polled for incoming messages when the LANai is waiting for a HDMA transfer to be complete. The drawback with SVIA is that it is not optimized to handle (a) multiple messages (or potentially a longer message) on a channel, or (b) messages on multiple (send or receive) channels. One way of optimizing the former (multiple messages or a longer message on a channel) is to overlap the functioning of the HDMA and NSDMA operations (for a send), so that both can go on concurrently. This is the approach that has been successfully used in Trapeze [18] to maximize the bandwidth of a single channel.

### 3.2. Parallel VIA (PVIA)

One could view the LCP as a finite state machine moving from one state to another based on NIC events (doorbell posting, DMA completion etc). Such a view without any overheads for event detection and state transitioning, would represent an ideal implementation that maximizes the concurrency (keeps hardware resources as busy as possible at all times) of activities on the NIC. Our PVIA (since it tries to increase the concurrency of activities on the card) tries to achieve this view. Implementation of such a view requires event notification and state maintenance mechanisms.

**Event Notification:** While one could have chosen an interrupt based mechanism to notify the LANai of event occurrences (such as DMA completion), we have instead opted for a polling mechanism to detect these events in PVIA due to a couple of reasons. First, not all events can interrupt the LANai (e.g. doorbell posting), and a polling based mechanism is necessary for these in any case. Second, the overhead of an interrupt mechanism are high, especially with a relatively slow processor such as the LANai. The downside of a polling mechanism is that there could be a gap between the event occurrence and when the LANai responds to it (depending on how busy the LANai is).

**State Maintenance:** The LCP needs to keep track of the state of each NIC device (if it is busy or not, and what request is currently being serviced). In addition, to avoid busy waiting for a device, it needs to queue up any additional requests. The NIC state is thus a tuple containing the request being serviced by each hardware device (HDMA, NSDMA, and NRDMA), and a queue of waiting requests for each of these devices.

The LCP for PVIA thus sequentially goes through each hardware device (doorbell queue, HDMA, NSDMA, NRDMA), checking each for an event to be processed (doorbell posted, HDMA descriptor/data transfer complete, NSDMA transfer complete, incoming message has arrived), and if so it processes this event - either activates the next hardware activity in the pipeline to process this event, or simply inserts it into a hardware (HDMA) or software (NSDMA) queue - and proceeds to the next hardware device in the sequence (does not have to wait for that device to complete).

### 3.3. QoS-aware VIA (QoSVIA)

With the LANai having to cater to several channels, it becomes extremely important as to how the channel activities need to be serviced (queued) to meet the performance criteria. While FCFS may not be a bad choice for a system with just Best-Effort traffic (that is used in PVIA), that may not be the case for a system with mixed traffic types. In QoSVIA, we take the above-mentioned PVIA implementation and augment it with some scheduling algorithms to determine channel priorities. VirtualClock [20] is one well-known rate-based scheduling algorithm that can be used here to regulate the allocation of resources to different channels based on bandwidth requirements. The idea of this algorithm is to emulate a Time Division Multiplexing (TDM) system, and its adaptation to QoSVIA is outlined below.

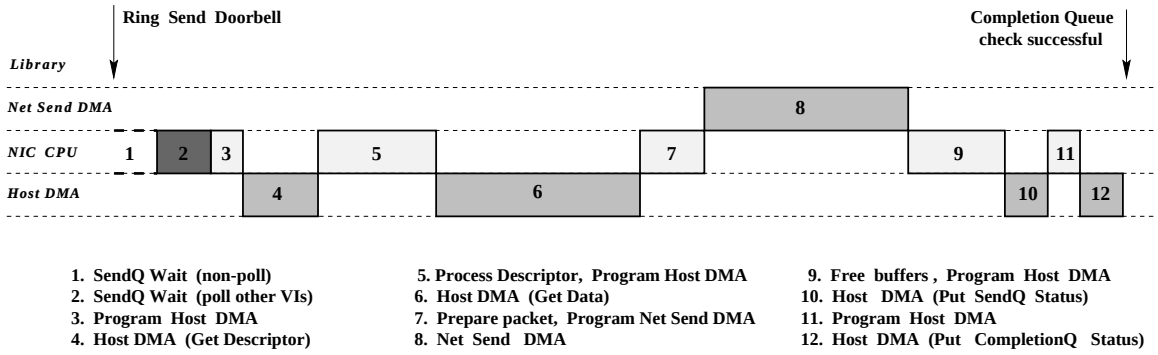



**Figure 2. Sequence of Actions in Sending a Message**

When a VI channel is opened (this is a driver call), the application also specifies the desirable bandwidth if it needs to use it for CBR/VBR traffic (in the VipCreateVI call). The driver then needs to communicate with both its NIC as well as the destination NIC to find out whether they can handle the load (without degrading service of other channels). If they cannot, then an error is returned to the application, which can then try to renegotiate. In our current implementation, we have not yet implemented this admission control mechanism (incorporating this is part of our future work), and assume that the NICs are able to handle the load.

The driver translates this bandwidth information into a period, called `Vtick`, which essentially specifies the ideal inter-arrival/service time between successive packets for a pre-determined packet size. A smaller `Vtick` specifies a higher bandwidth requirement. For Best-Effort channels, `Vtick` is set to infinity. The `Vtick` for the channel is then conveyed to the LCP, which keeps this information together with the other channel specific data. In addition, the LCP maintains another variable, `auxVC` for each channel, and calculates this value when it notices a doorbell for that channel ($i$) as follows:

$$auxVC_i \leftarrow \max(\text{current time}, auxVC_i)$$

$$auxVC_i \leftarrow auxVC_i + Vtick_i$$

timestamp the doorbell with $auxVC_i$

The VirtualClock algorithm specifies that packets be served in increasing timestamp order to allocate bandwidth for resources proportional to their specified requirements.

Once the time stamp is determined, the next issue is to figure out how to allocate the resources based on these timestamps. The resources that are specifically of concern are the HDMA and the NSDMA. NRDMA brings in messages in FIFO order and the LCP has no control over this ordering - it can only pull out these messages and perhaps put them in timestamp order in the HDMA queue to be eventually transferred to host memory.

When the doorbell on a channel is posted, it is timestamped using the VirtualClock algorithm, and the corresponding descriptor is inserted in the hardware HDMA queue in timestamp order. Similarly, the software NSDMA queue is maintained in time stamp order. While one could hypothesize that maintaining the NSDMA queue in FCFS order would suffice since a time-stamp based ordering has already been imposed in the previous stage of the pipeline, it could so happen that a doorbell for a message with a more critical deadline (higher priority) arrives later than a message currently being serviced (or already serviced) by the HDMA. Keeping the NSDMA queue also in timestamp order helps us address some of those problems (though not completely).

We would like to point out that HDMA (hardware) queue maintenance in timestamp order introduces an additional complication. With two entities (LANai and HDMA) manipulating the queue, there is the fear of race conditions. This is not really a problem if insertions are always done at the tail, as is the case in normal SVIA/PVIA, since the HDMA will not miss seeing any request. However, insertion in timestamp order (possibly in the middle of the queue) can create problems, and we do not have the luxury of creating critical sections. In our implementation, we address this problem by tolerating slight inaccuracies (inserting in a slightly different position) in cases where the position for insertion is very close to the head (where the HDMA is operating) of the queue. Some of these inaccuracies may get remedied in the next pipeline stage - the software NSDMA queue that is also maintained in time-stamp order.

The basic difference between PVIA and QoSVIA is only in the time-stamping of events using the VirtualClock algorithm and the possible insertions into the middle of the HDMA/NSDMA queues (PVIA inserts only at the tail). There is a cost that is paid in performing these operations and our experimental evaluations will compare the benefit gains with the incurred costs.

Finally, it is well understood [19] that a rate-based service discipline, such as VirtualClock, can potentially starve best-effort channels (as long as there is demand from a CBR/VBR channel). There are alternate scheduling mechanisms, such as hierarchical schedulers [8], that one could use to avoid some of these problems.

## 4. Performance Results

PVIA and QoSVIA have been implemented and evaluated on both an experimental platform as well as using simulations, and compared with SVIA. For SVIA, we use a publicly distributed version [4] which is representative of several other implementations optimized for low latency on a channel. The evaluation platform consists of a Pentium/Linux cluster connected by a Myrinet switch. Each node of the cluster has dual 400 MHz Pentium Pros with

256MB memory and a 66 MHz 64 bit PCI bus.

The metrics of concern are the (a) *jitter fraction* (denoted as $\theta$) for the QoS (CBR or VBR) channels, and (b) *1-way latency* for messages on BE channels. Jitter fraction is defined as follows. Let us say a QoS channel has negotiated a certain bandwidth (e.g. an inter-arrival time for a specified message size) with the underlying system. When messages are pumped into this channel (we are assuming that the workload generator can pump in at the negotiated rate in the very least if not larger, based on our assumption that the native OS scheduler is QoS aware as stated in Section 1), the application expects the messages to be separated by at most the negotiated inter-arrival time at the receiver end. Else, it incurs a jitter for that message, with a *jitter duration* equal to the difference between the inter-receipt time of the messages and the negotiated inter-arrival time. Jitter fraction is a normalized version of this metric, where the mean jitter duration (over all the messages) is divided by the negotiated inter-arrival time. While we have obtained the mean jitter duration and the number of messages incurring jitters in the experiment, the results shown below are specifically for $\theta$ which we feel is more useful from the application's viewpoint (for example, a 1 ms jitter duration on a 1 MByte/sec channel is not as significant as a 1 ms jitter duration on a 10 MByte/sec channel, and $\theta$ captures this difference). The number (or fraction) of packets incurring jitters is also an issue we consider in some cases.

### 4.1. Results from Experimental Platform

**Raw Performance of VIA Implementations** Before we examine the QoS capabilities of the three VIA implementations, we first examine the raw performance (that others have typically optimized for), to see how much overhead is added by PVIA and QoSVIA on BE channels (without any QoS traffic). We investigate this issue by first giving the breakup of the 1-way latency (generated by using a simple ping-pong experiment on 1 channel between two machines) for SVIA, PVIA and QoSVIA.

Table 1 gives the time when the different send/receive operations. Finally, the 1-way latencies (halving the round-trip time) are also given. It should be noted that one may expect the 1-way latency to reflect the sum of the Status Done on the Send and Receive sides. While this is roughly the case, it should be noted that measurement of these NIC activities are not really measured using round-trip messages, and use a different clock on the NIC (while the round-trip latency is measured at the host by the application). Experimental variations from one run to the next also account for such discrepancies. Still the breakup, gives a good indication of the overhead of different operations.

It is quite apparent that the restructuring of the firmware in PVIA and QoSVIA add to the overheads of these operations (state and queue maintenance) compared to SVIA, which is to be expected. However, these overheads are relatively independent of message size (and only dependent on the efficiency of the LCP). As a result, the differences between these implementations become less important as we move to larger message sizes, even with a ping-pong experiment which does not allow any scope for parallelism/pipelining in the NIC operations. Further, the overheads for QoSVIA compared to PVIA, due to time-stamping and priority queue orderings, are not very significant (the priority queue is not really stressed in this case since the ping-pong experiment keeps at most one message in the queue at any time).

| Msg. Size | SVIA | PVIA | QoSVIA |
|---|---|---|---|
| 4 | 8 | 26 | 32 |
| 1K | 14 | 28 | 34 |
| 4K | 48 | 36 | 46 |
| 16K | 152 | 86 | |

**Table 2. Average 1-way Message Latency in microsecs using Ping-Bulk experiment**

To study a more realistic situation where there is scope for parallelism, we run a ping-bulk experiment between two machines (sender pumps in several messages and waits for several messages, with the receiver doing the reverse) using two uni-directional (one for send and another for receive) channels between two machines. It should be noted that the window size (number of messages to send before switching directions) can have an effect on the LCP performance. Small window sizes would limit the scope of parallelism, and large window sizes may create problems since VIA does not mandate flow control. By varying the window size, we have found the point that provides the lowest average 1-way latency per message in each experiment, and the corresponding results are shown in Table 2 as a function of the message size for the three LCP designs.

The results in this table confirm our arguments about the necessity for a parallel/pipelined firmware with larger loads on a channel. The overheads of the optimizations are overshadowed by the performance enhancements at higher loads. For instance, both PVIA and QoSVIA outperform SVIA after message sizes of 4K. With small messages (1K or less), the overheads of the optimized LCP are not able to overlap with the diminished operation costs. With larger messages, the benefits of the overlap materialize. Though not explicitly shown here, we can expect similar benefits from PVIA/QoSVIA with multiple channels (even if the load on a channel does not increase), since there is literally no change in the behavior of the firmware with increase in BE channels (the doorbells on different channels are merged by the hardware into a single queue).

**Performance with CBR Channels (QoS)** In the next experiment, we evaluate the three LCPs using CBR traffic. The experiment is composed as follows. We employ one sender machine that uses three classes of QoS channels (requiring 1 MByte/sec, 2 MBytes/sec and 3 MBytes/sec respectively) to send out messages to three receiver machines (each receiver receives one class of channels). The load is increased by increasing the number of channels in each class as 1,2,3,4 and 5 channels, giving overall injected loads of 6, 12, 18, 24 and 30 MBytes/sec respectively. Carefully designed workload generators were used to inject messages in the different channels at the specified rates, and the time difference between successive receipt of messages at the receiver is used to measure jitter duration and $\theta$. The resulting jitter fraction ($\theta$) with the three LCPs is shown in Figure 3. We can observe that SVIA has slightly higher $\theta$ (around 10% in many cases) than the other two schemes. Between PVIA and QoSVIA, there is very marginal differences (the

| NIC Operation | SVIA | | | | PVIA | | | | QoSVIA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 1K | 4K | 16K | 4 | 1K | 4K | 16K | 4 | 1K | 4K | 16K |
| Send | | | | | | | | | | | | |
| HDMA Prog. (Desc.) | 3 | 3 | 3 | 3 | 7 | 7 | 7 | 7 | 9 | 9 | 9 | 9 |
| Desc. Down | 6 | 6 | 6 | 6 | 11 | 10 | 11 | 10 | 13 | 13 | 13 | 13 |
| HDMA Prog. (Data) | 9 | 9 | 9 | 11 | 19 | 19 | 23 | 34 | 24 | 24 | 30 | 49 |
| Data Down | 11 | 18 | 43 | 141 | 23 | 28 | 52 | 148 | 27 | 33 | 57 | 152 |
| NSDMA Done | 12 | 25 | 69 | 244 | 28 | 39 | 82 | 254 | 32 | 43 | 86 | 261 |
| HDMA Prog. (Status) | 13 | 27 | 71 | 245 | 33 | 45 | 87 | 260 | 37 | 49 | 91 | 266 |
| Status Done | 14 | 28 | 72 | 247 | 38 | 49 | 92 | 264 | 41 | 53 | 95 | 270 |
| Receive | | | | | | | | | | | | |
| HDMA Prog. (Desc.) | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Desc Down | 5 | 5 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| HDMA Prog. (Data) | 8 | 8 | 9 | 11 | 10 | 10 | 14 | 24 | 12 | 12 | 17 | 35 |
| Data Up | 10 | 17 | 42 | 138 | 14 | 19 | 42 | 138 | 15 | 21 | 44 | 139 |
| HDMA Prog. (Status) | 11 | 18 | 43 | 140 | 19 | 24 | 47 | 143 | 20 | 25 | 48 | 143 |
| Status Done | 12 | 20 | 44 | 141 | 23 | 28 | 51 | 146 | 24 | 29 | 52 | 147 |
| 1-way Latency | 30 | 53 | 121 | 385 | 69 | 90 | 153 | 423 | 73 | 90 | 157 | 426 |

**Table 1. Break-up of time expended in different operations during message transfer for different message sizes. Times are in microsecs, and indicate the time when that operation is completed since the send doorbell is rung for the send side, and receive doorbell is rung for the receive side. Receive descriptors are pre-posted.**

latter is a little better though the difference is hardly noticeable in the graph). The differences between PVIA and QoSVIA are a little more noticeable when one observes the percentage of packets (messages) that incur jitters in Figure 4 for the same experiment (results are shown for each of the classes). QoSVIA has fewer packets incurring jitters, especially at higher loads suggesting that it is more conducive to meeting the QoS requirements of channels.

We would like to point out that this experiment has not really been able to stress the system enough to bring out differences more prominently. As was mentioned, the VIA implementations (all three, including the one in the public domain [4] that is used here), do not have flow control capabilities. Consequently, we are not able to pump in messages at a faster rate (as this experiment suggests, the differences are likely to be felt at higher loads, or else just the raw bandwidth availability is enough to meet the QoS demands by any implementation). This is also the reason why the absolute values of $\theta$ in Figure 3 are very low. Even at these relatively small loads, some differences between the LCP implementations can be felt. Higher level messaging layers built on top of these LCPs may, on the other hand, be able to subject the system to higher loads to experience the differences. The differences between the schemes at higher loads are evaluated next using a simulator.

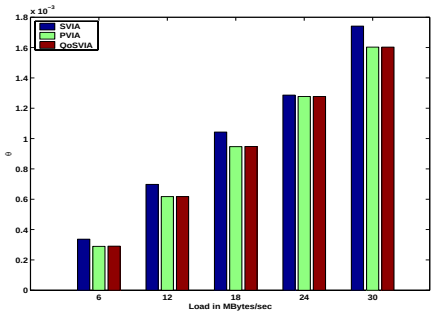
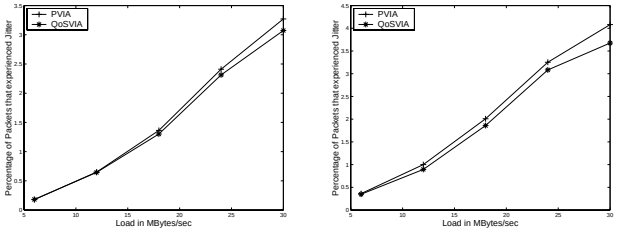



**Figure 3. Experimental Results Using 1 sender and 3 receivers with three classes (1, 2 and 3 MBytes/s) of QoS channels, with each receiver receiving one class. The $\theta$ shown is for the 2 MBytes/s class.**

## 4.2. Simulation Results

With the experimental set up discussed above, we are somewhat limited by how much we can stress the system, and by the accuracy of the workload generator in sending/receiving packets at the exact times (due to the vagaries of the system). This is where simulation comes in useful, and we have developed a detailed simulator modeling the different hardware components and the three firmware designs, using the breakup of time spent in the different stages from the measurement on the platform given earlier. The simulation results not only augment the experimental observations, but also serve to explore alternate hardware designs/enhancements (such as hardware queuing for the NSDMA) which is not possible on the current experimental platform. It should be noted that the simulator uses infinite sized buffers/queues, and thus gives an indication of what is possible with each scheme, since our goal is to see how the schemes compare at high loads.

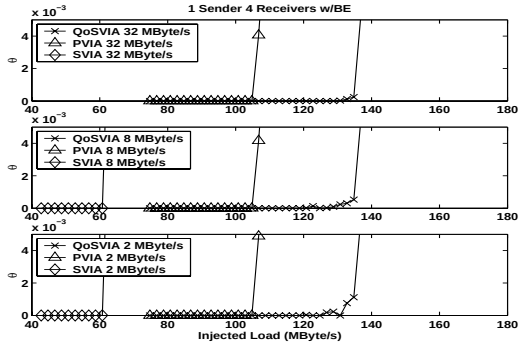
**Results for Higher Loads** The first set of simulation results shown in Figure 5 (a) shows the observed $\theta$ for three classes of QoS CBR channels (2, 8 and 32 MBytes/sec) on a 1-sender 4-receiver system as a function of the injected load for the three firmware designs. The injected load is varied by increasing the number of QoS channels, with the bandwidth on any channel remaining the same. In addition to QoS channels, there is also one BE channel that injects messages with a mean rate of 33 MBytes/sec. All QoS channels obey the negotiated bandwidth allocation, and inject messages (all of size 4 KB) as per this rate. *All the graphs are*



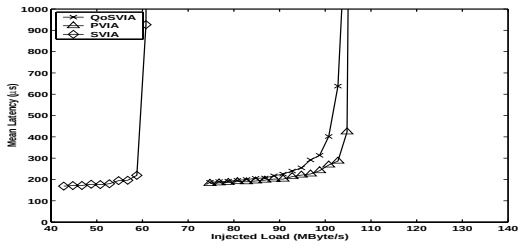

(a) 1 MBytes/s (b) 3 MBytes/s

**Figure 4. the percentage of packets experiencing jitters for PVIA and QoSVIA.**

*plotted as a function of total injected load and not as a function of load on that channel class.*




(a) $\theta$ for QoS Channels



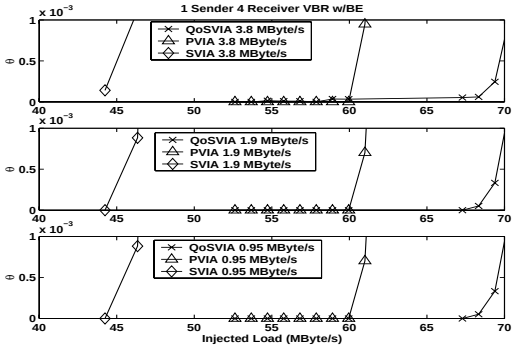

(b) Latency for BE Channel

**Figure 5. Simulation results for a 1-sender 4-receivers configuration with 3 QoS classes (2, 8 and 32 MBytes/s) and 1 BE channel (33 MBytes/s).**

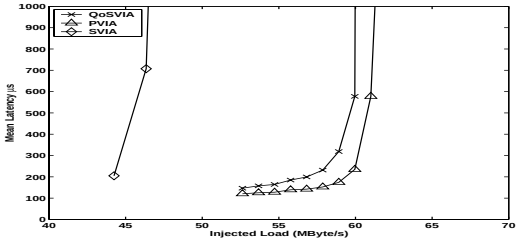
The first point to note is that the simulation results confirm the earlier experimental observation that at lower loads (less than 60 MBytes/sec), there is little difference between the schemes (the scale of the $\theta$ graph in the experimental results is much more amplified that what is shown here). However, at higher loads (after 62 MBytes/sec), we find that SVIA is not able to meet the demands of the QoS channels. On the other hand, PVIA is able to handle their demands until the load exceeds 100 MBytes/sec, and QoSVIA can go even as high as 130 MByte/sec injected load before resulting in any meaningful jitter. These results confirm the importance of a pipelined firmware such as PVIA for higher bandwidth, together with a QoS conscious service discipline. QoSVIA is able to reduce the jitter on each of the traffic classes compared to SVIA or PVIA.

Figure 5 (b) shows the latency of messages on the BE channel as a function of the total system load for the same experiment. It should be noted that the load in the BE channel itself remains the same (only the number of QoS channels is increased). This is one way of examining how the load on one class affects the performance of the other. We find that SVIA performance is much worse than the other two, saturating when the load exceeds 60 MBytes/sec. Between PVIA and QoSVIA, we find the former giving slightly lower latency than the latter. This is because PVIA does not differentiate between the traffic classes. On the other hand, QoSVIA gives higher priority to QoS channels, and thereby penalizing BE performance a little. Still, the differences between QoSVIA and PVIA for BE traffic are not that significant, and the benefits of QoSVIA for QoS classes can offset this slight deficiency.

**VBR QoS Channels** In addition to the experiments using CBR channels, we have also considered VBR traffic in our experiments. Specifically, we have used the MPEG-1 trace files from [16] to generate the workload. This experiment uses 1 sender and 4 receivers, with three classes of QoS channels (0.95, 1.9 and 3.8 MBytes/sec) and one BE channel (16.7 MBytes/s). Within each channel, we inject 20, 40 and 80 MPEG-1 streams (adhering to the specified data rates), and the jitter fraction for QoS channels and latency for BE channel are shown in Figure 6. As with CBR traffic, we find that PVIA and QoSVIA, are much better than SVIA, with QoSVIA able to sustain the highest load.




(a) $\theta$ for QoS channels




(b) Latency for BE channel

**Figure 6. Simulation results with MPEG-1 traces for the 1 sender 4 receivers configuration with 3 QoS classes (0.95, 1.9 and 3.8 MBytes/s) and 1 BE channel (16.7 MBytes/s)**

## 5. Concluding Remarks and Future Work

The primary contribution of this paper is in the design and implementation of firmware for the Myrinet NIC to provide both high bandwidth and QoS-aware communication. Most prior research in this area has looked at optimizing the latency and/or bandwidth of messages on a single communication channel. On the other hand, with the anticipated cluster usage, it is also important to explore the scalability question (how does performance scale as the number of channels is increased?). We have proposed and implemented an alternative structure for the firmware on the NIC (PVIA) that uses an event-driven mechanism to improve the degree of parallelism of activities on the NIC. PVIA has been shown to give better performance from the scalability viewpoint compared to a typical implementation optimized for a single channel (taken from the public domain [4]) experimentally.

This study has also extended PVIA (called QoSVIA) with a well-known rate-based service discipline [20] to handle several bandwidth sensitive channels. Incoming packets are time-stamped using this algorithm, and the requests are inserted into the queues maintained by PVIA for state maintenance in priority order. Both PVIA and QoSVIA have been implemented on an experimental platform, and evaluated using channels requiring periodic deadlines to be met.

It should be noted that one could very well use other scheduling strategies, including hierarchical scheduling mechanisms, within QoSVIA for time-stamping packets. Such techniques can limit the detriments to BE performance. Another important issue is on admission control mechanisms during channel setup which involves negotiation not only with the local NIC, but also across the network. In addition to this, we are also trying to implement higher level messaging layers (incorporating flow control) on top of our platform. This will be integrated with our other cluster efforts on host CPU scheduling and network improvements for QoS-aware features. Finally, we are trying to develop and port applications that can benefit from QoS features on a cluster for a complete system evaluation.

## References


[1] N. P. C. P. at Lawrence Berkeley National Laboratory). M-VIA: A High Performance Modular VIA for Linux. Available at `http://www.nersc.gov/research/FTG/via`.

[2] M. Bazikazemi, V. Moorthy, L. Herger, D. K. Panda, and B. Abali. Efficient Virtual Interface Architecture Support for IBM SP Switch-Connected NT Clusters. In *Proceedings of International Parallel and Distributed Processing Symposium*, May 2000.

[3] F. Berry, E. Deleganes, and A. M. Merritt. *The Virtual Interface Architecture Proof-of-Concept Performance Results*. Intel Corp.

[4] P. Buonadonna, A. Geweke, and D. E. Culler. An Implementation and Analysis of the Virtual Interface Architecture. In *Proceedings of Supercomputing '98*, November 1998.

[5] T. Corp. ServerNet and the VI Architecture. Available at `http://www.servernet.com/`.

[6] A. Gallatin, J. Chase, and K. Yocum. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. In *1999 USENIX Technical Conference (Freenix Track)*, June 1999.

[7] Giganet. cLAN Product Overview. Available at `http://www.giganet.com/products/overview.htm`.

[8] P. Goyal, X. Guo, and H. M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Proceedings of 2nd Symposium on Operating System Design and Implementation*, pages 107–122, October 1996.

[9] G. Henley, N. Doss, T. McMahon, and A. Skjellum. BDM: A Multiprotocol Myrinet Control Program and Host Application Programmer Interface. Technical Report MSSU-EIRS-ERC-97-3, ICDCRL, Dept. of Computer Science and HPCL, NSF Engineering Research Center for Computational Field Simulation, Mississippi State University, May 1997.

[10] M. Jacunski, V. Moorthy, P. Ware, M. Pillai, D. Panda, and P. Sadayappan. Low Latency Message-Passing for Reflective Memory Networks. In *Lecture Notes in Computer Science, 1602*, pages 211–224. Springer-Verlag, January 1999. From Proceedings of Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing (CANPC'99).

[11] S. S. Lumetta, A. M. Mainwaring, and D. E. Culler. Multi-Protocol Active Messages on a Cluster of SMP's. In *Proceedings of Supercomputing '97*, November 1997.

[12] S. Nagar, D. Seed, and A. Sivasubramaniam. Implementing Protected Multi-User Communication for Myrinet. In *Lecture Notes in Computer Science, 1362*, pages 30–44. Springer-Verlag, February 1998. From Proceedings of Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing (CANPC).

[13] S. Nagar, A. Sivasubramaniam, J. Rodriguez, and M. Yousif. Issues in Designing and Implementing a Scalable Virtual Interface Architecture. In *Proceedings of the 2000 International Conference on Parallel Processing*, pages 405–412, August 2000.

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

[15] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance. In *PC-NOW Workshop held in conjunction with IPPS/SPDP98, Orlando,USA*, 1998.

[16] O. Rose. http://nero.informatik.uni-wuerzburg.de/MPEG/.

[17] T. von Eicken, A. Basu, V. Buch, and 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.

[18] K. Yocum, J. Chase, A. Gallain, and A. R. Lebeck. Cut-Through Delivery in Trapeze: An Exercise in Low-Latency Messaging. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, August 1997.

[19] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. In *Proceedings of the conference on Communications architecture & protocols*, pages 113 – 121, 1991.

[20] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks. *ACM Trans. on Computer Systems*, 9(2):101–124, May 1991.