Performance Concerns About Myrinet Switches And Fabrics

Ludmila Cherkasova, Vadim Kotov, Tomas Rokicki
Performance Analysis Project
Concurrent Computing Department

Memo HPL–CCD–95–7

This document contains a brief description of some features of the Myrinet switches that may have a significant negative impact on overall performance.

1 Overall Architecture

In this section, the overall architecture of the Myrinet switch is described at a very high level.

A Myrinet switch is essentially a packaged wormhole $n \times n$ crossbar switch, with the inputs and outputs joined into a single bidirectional ports.

The switch is 'wormhole' because it has very little buffering and no intrinsic segmentation and reassembly of smaller packets. This has several effects. First, for messages longer than a few bytes, normally the head of the message is being ejected from an output port while the tail of the message is still being injected into an input port—only a small portion of a message can 'live' inside the switch at a given time. Also, if the head of the message is 'blocked' for some reason, such as contention for a subsequent port or slow response at the destination, the tail is also blocked; all port resources in use at the time of the blockage continue to be used until the blockage is removed. This is in comparison with packet-based networks, where if an output port is busy, the remainder of a packet is absorbed into a single switching element, freeing the input port for further work, usually before the output port is free.

The crossbar is constructed in a systolic manner from crosspoints. Each crosspoint has two inputs and two outputs; there is a single 'row' input and output, and a single 'column' input and output. Inputs to the systolic array arrive at the row inputs, proceed through the crosspoints along a row until they reach the appropriate column, at which point they proceed along the column crosspoints until they reach the appropriate column output.

The systolic array of an $n \times n$ crossbar has n rows and n columns internally. Each row and each column of the entire array has a single input and output. Thus, the number of inputs at the edge of the array is $2n$, and the number of outputs is also $2n$.

In a single switch environment, only the row inputs and the column outputs are used; the row outputs and column inputs are unconnected. This provides what Myricom calls 'perfect non-blocking' performance; an input port will only be blocked if the message currently being sent on that port requires a destination that is being used by some other message.

Ideally, the unused column inputs and row outputs could be used to cascade four $n \times n$ crossbars into a $2n \times 2n$ crossbar. Indeed, this may be possible at the multi-chip module level. Unfortunately, because of the flow control required to implement wormhole transfer, pairs consisting of a single row input and single column output are 'combined' into a bidirectional flow-controlled external 'link', and the unused column inputs and row outputs are simply not available outside of the switch. Thus, 'simple' systolic cascading of the switches is not possible.

2 Blocking

Because the switch provides essentially no buffering, it suffers from traditional head-of-line blocking. If the traffic pattern is such that all input ports address a single output port, then each input port will take its turn and throughput will be $1/n$ at each input port. If this is the long-term traffic pattern, no switch architecture can do better since the output port is fully saturated.

On the other hand, if the traffic pattern is such that input ports and output ports are randomly distributed, then a switch architecture with significant buffering can attain very close to 100% aggregate utilization. The buffers inside the switch can absorb transient traffic irregularities and smooth out the traffic for each output port, and thus long-term both input and output ports will all be fully saturated.

In practice, as the amount of buffering increases, the attainable aggregate utilization also increases. As the burstiness of the traffic pattern increases, the aggregate utilization decreases. Typically, the amount of buffering is chosen so that typical aggregate utilization under a fairly random workload can get fairly close to 100%.

In the Myrinet switch with essentially no buffering, even under random traffic patterns with very little burstiness the overall switch performance drops dramatically, to a maximum of about 50% of the aggregate throughput. Essentially, if an output port is requested by more than one input port, at least one of those input ports must block, and therefore not do useful work. This also blocks all messages behind that message in the network interface queue.

Two additional features can make this problem much worse. The first is the unbounded and unrestricted length of the messages in the Myrinet switch. As the variance of message length increases, the aggregate throughput degradation due to head of line blocking increases. The increase can be quite dramatic. We were able to construct realistic bimodal workloads, with uniformly random source and destination selection, for which the maximum aggregate throughput was less than 14% of port capacity.

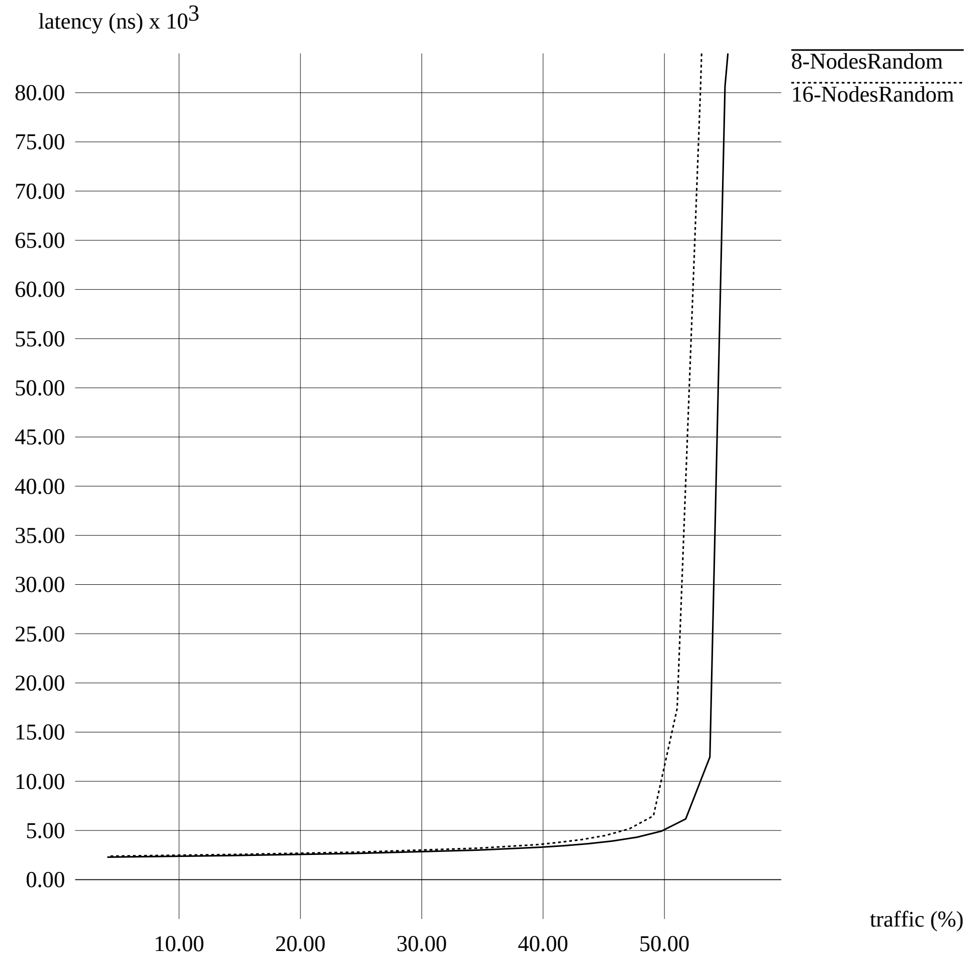

Figure 1: Myrinet switch performance under random uniform traffic

We built a simulation model of the Myrinet switch which works at about the level of detail discussed above. We ran this model over a set of different traffic patterns and workloads to evaluate the basic switch performance.

Figure 1 shows switch performance (the latency and throughput) under random uniform traffic for 8 and 16 terminal nodes. Each message is 140 bytes long.

The maximal attainable throughput for a switch with 8 nodes and 16 nodes is 52% and 55%, respectively. The latency curve is very flat and varies from 2000 ns (for a traffic density of 5%) to 6000 ns (for a traffic density of 50%)

We also performed a number of simulations using bursty workloads. These workloads are defined primarily by a message length distribution. Rather than considering many different message length distributions, we consider only bimodal distributions consisting of short messages and long messages. We define short messages to be from 140 bytes to 660 bytes long, and we give each length equal probability.

We define our workloads by the percentage of short and long messages in the workload; this is the primary variable defining the workloads. For instance, in Figure 2 the performance results of a Myrinet switch with 8 and 16 nodes are shown for two workload with 10% and 80% long messages. In both workloads, short messages vary randomly from 140 bytes to 660 bytes, and all long messages are 3220 bytes long.

For both of these workloads Myrinet switch performance is reasonably good. However, as the variance of message length increases, the aggregate throughput degradation due to head of line blocking also increases. Figure 3 shows the Myrinet switch performance with 16 nodes for seven different types of bursty workloads called here "burst 1" to "burst 7". All of these workloads have 99% short messages and 1% long messages. Short messages here vary from 100 to 700 bytes in length, and the length of long messages depends on the workload. The "burst 1" workload has 3,000,000 byte long messages.
The "burst 2" workload has 2,000,000 byte long messages.
The "burst 3" workload has 1,000,000 byte long messages.
The "burst 4" workload has 500,000 byte long messages.
The "burst 5" workload has 200,000 byte long messages.
The "burst 6" workload has 100,000 byte long messages.
The "burst 7" workload has 50,000 byte long messages.

For all of these workloads, switch throughput is very low. This is easily explained. Imagine that a particular output port is busy transferring a long message. The average message length is much less than this long message, due to the high traffic variance. As each input port transfers several messages, they are likely to encounter a message that requires that particular output port, at which point they block. By the time the original long message has been successfully transferred, a substantial fraction of the input ports might be blocked on that particular output port. The key idea is that if the long messages are substantially longer than the average message, the long messages tend to 'clog' required output ports, which clog more and more input ports. If the majority of the bytes are long messages (which is typical), this happens more and more frequently.

Figure 4 shows the percentage of bytes belonging to short messages for the different workloads.

This problem can be largely solved by breaking up long messages into shorter ones (usually within a factor of two or three of the average message length, to keep performance reasonably

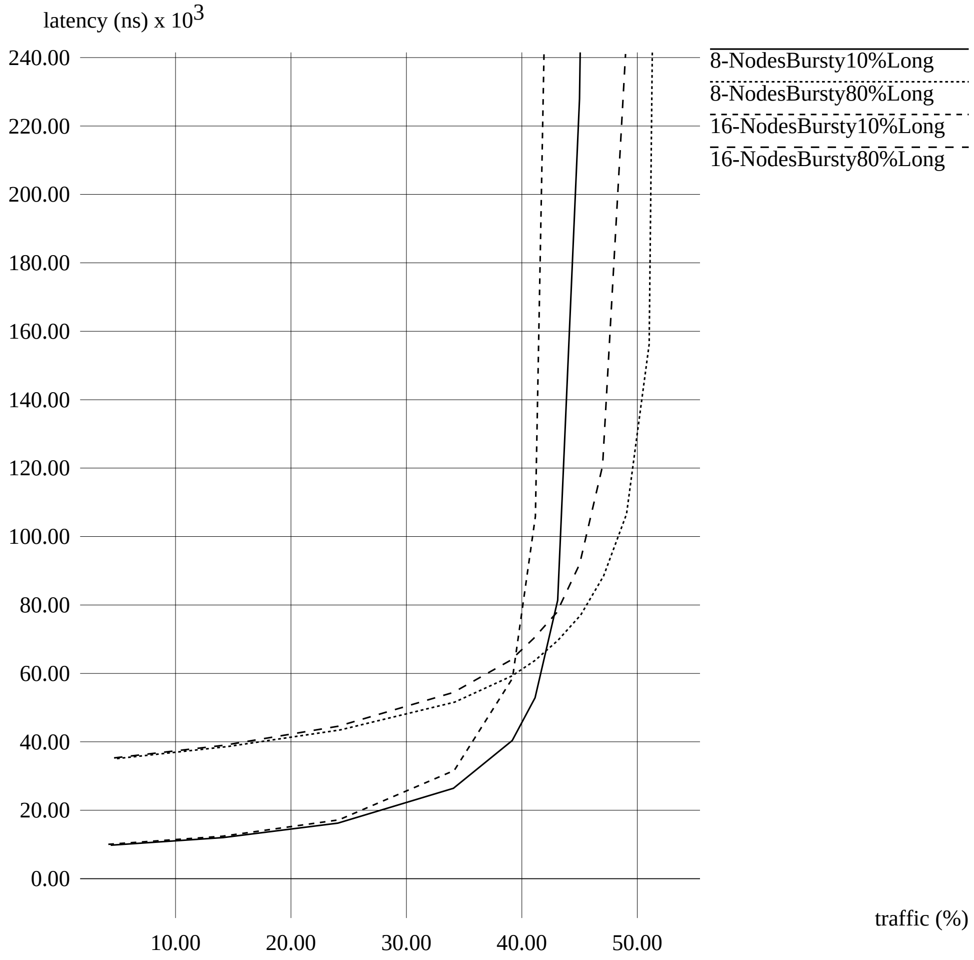

Figure 2: Myrinet switch performance under bursty workload with 10% land 80% long messages workload

high), and interspersing messages to several destinations in the injection stream. That is, if a long message is broken into 100 shorter messages, and there is other waiting traffic from that input port, the waiting traffic should be interspersed in the gaps between the long message segments.

It is important to note that the segmentation size for a wormhole network must usually be much smaller than the segmentation size for a buffered, packet-switched network, because in the latter short-term destination blockage does not block the input ports.

Thus, the advantages of 'unbounded message length' and 'no required segmentation and reassembly' claimed for wormhole networks (without virtual channels, which Myrinet lacks) are largely red herrings; in order to obtain reasonable performance, segmentation and reassembly must still be done, and usually at a finer grain than would be required for buffered networks.

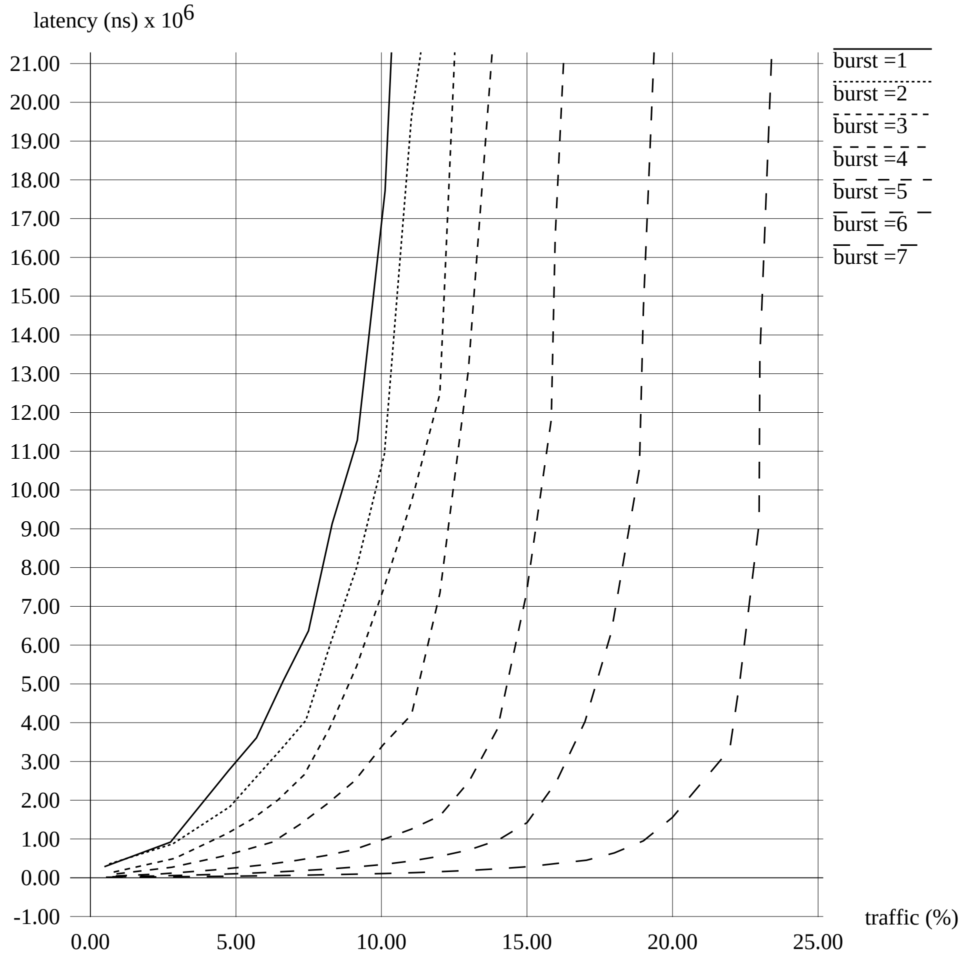

Figure 3: Myrinet switch performance under seven different bursty workloads.

Of course, the Myrinet switch has a fairly intelligent and powerful network interface design that can handle the segmentation and reassembly duties reasonably efficiently. A quantitative analysis must be performed to calculate the impact of the required segmentation and reassembly.

3 Cascading—Performance Hits

Another major concern is how these switches can be cascaded together to form larger 'fabrics'. Currently, an 8×8 switch is available; a 16×16 switch should be available during the summer. In order to connect more terminal nodes, the switches must be connected together by 'trunk links'.

The highest performance technique is to connect the switches together in a 'perfect non-blocking' arrangement, much like the crosspoints are inside of a single switch. As we have

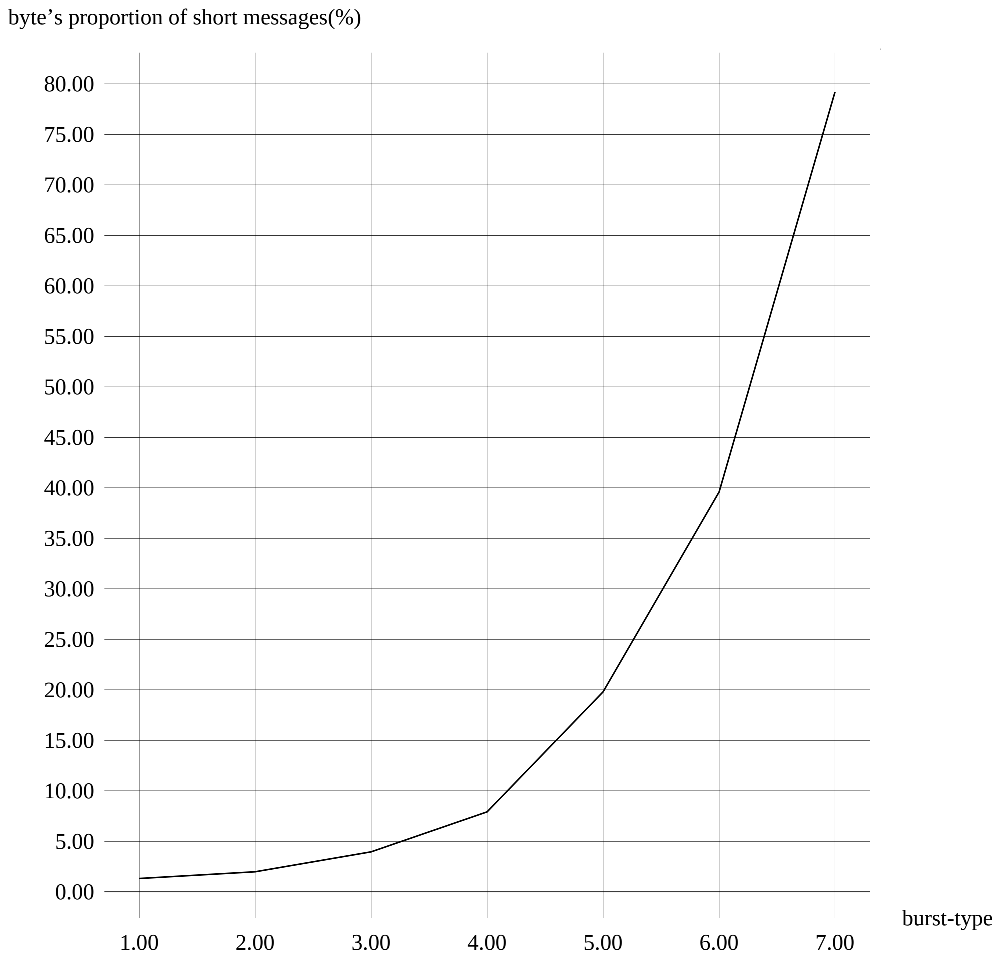

Figure 4: Proportion of bytes belonging to short messages against the long ones for seven bursty workloads

discussed, the necessary ports are not brought outside of the switch to make this easily possible. Essentially, to build a $2n \times 2n$ switch out of $n \times n$ switches, we will need quite a few more than simply four $n \times n$ switches.

The best found solution to cascade $n \times n$ switches into $m \times m$ nonblocking cascades required

$$\left(\frac{2m}{n}\right)^2 - \frac{2m}{n}$$

$n \times n$ switches. This is about a factor of four more switches than would be required to cascade them in a systolic fashion.

The following table shows, for a given ratio of fabric terminal ports to switch terminal ports, the number of individual switches required.

Scaleup	Switch count
2	12
3	30
4	56
5	90
6	132
8	240
12	552
16	992

Thus, to build a 64-node cluster out of 16×16 switches, we would need a scaleup of 4, requiring 56 individual switch chips.

For small values of scaleup, this is reasonable, even practical, as the multi-chip modules implementing the switches are probably fairly cheap. But beyond a few dozen, this technique can lead to significant packaging problems.

In addition, the latency resulting from a systolic array approach scales up with the size of the network by a factor of $2m/n$. Thus, 64-node cluster would be about eight times as slow as a single switch, since the average message would have to traverse eight switches.

If the required number of switches is too large, or the latency is too high, alternative cascading topologies can be used, such as rings, or stars, or meshes. Such topologies would make the blocking problem, already pretty severe for single switches, much, much worse as many separate traffic streams compete for trunk links. A single traffic stream occupying one trunk link could lead to blockage on almost every other trunk link in the system!

Imagine that one message is being sent over a trunk link, but it is blocked waiting for an output port to become free. Another message can then be blocked waiting for that trunk link to become free. If this message already occupies some second trunk link, that trunk link becomes blocked as well. Thus, the single output port that is being used is blocking all communication paths that require either that output port or either of the two trunk links. This blockage can and will spread throughout the network, reducing throughput to a dribble.

Using a complete topology (with each switch connected to every other switch) can reduce this problem, but this obviously severely limits the total size of the cascade.

4 Cascading—Deadlock

Indeed, such trunk blockage can deadlock the entire network, if the original message was blocked on a trunk link rather than an output port. Such a deadlock situation would probably arise fairly frequently, and the timeout exceptions would only allow the network to 'stutter'—have some small amount of throughput for very short periods of time between much longer timeout periods.

To solve this problem, deadlock-free network topologies such as trees can be used, but these topologies typically have very poor performance and significant performance limits. Careful routing (though a global routing assignment created by some central intelligence) can allow the use of a wider range of topologies, and this should help eliminate deadlock, but these solutions must be carefully investigated.

5 Cascading—Fairness

Finally, the local fairness in each switch leads to signficant global unfairness, whether the cascades are systolic or more conventional. The reason for this is that each trunk link is considered as a single port for fairness, while that trunk link might indeed represent traffic demands from many ports. The unfairness tends to be exponential in the required number of hops; nodes at a distance of one might see unfairness by a factor of ten for a particular output port, while nodes at a distance of three might see unfairness by a factor of a thousand.

Considering and solving the fairness problem also requires some careful analysis.