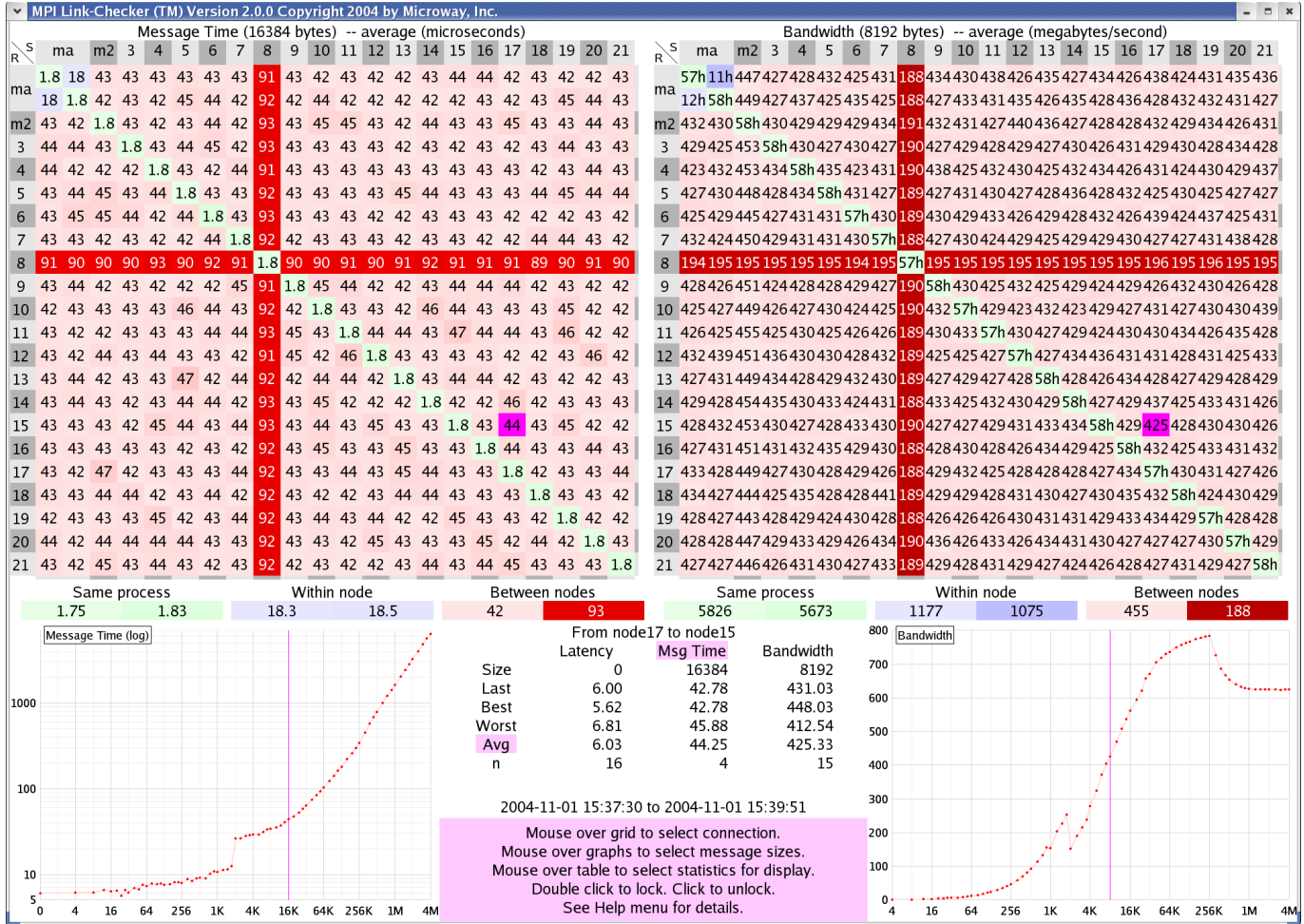


# MPI Link-Checker™ 2.1.0

## Application Note

Copyright 2007 by Microway Inc.

Microway's MPI Link-Checker is a software product which exercises an MPI cluster and uses the data collected to find problems with the system it is running on. It can detect both subtle and gross problems with everything in the MPI path: the OS, BIOS, device drivers, MPI implementation, motherboard, PCI bus, NICs, cables and switches. It does this by statistically comparing the latency and bandwidth of all the nodes in the system.



**Figure 1**  
**MPI Link Checker Screen Shot of 22 Node Cluster: Opteron-InfiniBand**

This release can be used in a number of different modes, including real time and off line. While it can detect a whole host of issues, it does not pinpoint the precise cause of the problem, just the nodes which have problems and the impact of the problem on latency and bandwidth. Often, problems like bad cables do not prevent a link from functioning, just from functioning at full speed. In a large cluster running a parallel application which has serial dependencies, a single bad cable can bring the entire cluster to its knees. The technique we originally used at Microway to validate MPI clusters ran a suite of MPI applications and checked the accuracy of the results.



This technique did not catch the low level problems that MPI Link-Checker does. The real problem with simply running benchmarks to validate a cluster's performance is that performance is sensitive to the software components which frequently change. In addition, many parallel benchmarks are not sensitive to the individual hardware components in a system, just the worst node. A single bad cable does not prevent a cluster from running, but it can dramatically impact the overall speed of the cluster. So, when benchmarks under perform, you can not tell whether you have a bad node, or possibly a system wide problem or a poorly implemented algorithm! There are times when bad cables or NICs can introduce data integrity problems. This rare case does get picked up running a benchmark like the NAS suite. MPI Link-Checker also detects this rare case and points it out by flashing a yellow background on the bad node. For most MPI clusters, the things that MPI Link-Checker uncovers can be very revealing about the operation of specific nodes. It can often find problems that are very subtle or that would have been impossible to find without a tool which combines data collection techniques with a sophisticated visual front end. A good example of a subtle problem can be seen in the latency and bandwidth plots in Figure 1, which exhibit a pair of problems discussed below. Understanding these issues makes it possible to adapt algorithms to the hardware and MPI implementation, being used.

The software can be run in a real time mode, displaying cluster data as it is collected, or in an off line mode collecting data for days or weeks and presenting that data to the graphics front end on an as-needed basis. When left in data collection mode, it can detect hard to find intermittent problems, as the bad results get collected along with the good, but only show up when the statistics are set to find the worst results. The data collected in off line mode can quickly add up to hundreds of megabytes, and takes longer to analyze. The analysis front end statistically examines the data and presents it in a multi-dimensional format, making it possible to examine the information in a rational manner. Often, a problem will show up for specific packet sizes and can affect either latency or bandwidth, but not necessarily both. It can also appear in reception or transmission or both. It's also possible for problems to fade in and out, with things like operating temperature or people disturbing cables that are not properly connected or bad.

Figure 1 is a screen shot of a 22 node Opteron InfiniBand cluster. The two pinkish-red matrices which dominate the figure are referred to as grids. They contain snapshots of the latency and bandwidth data collected for particular packet sizes using the statistical summation technique chosen by the user. The plots below display the same data as a function of packet size, for an individual node. Moving the cursor across the screen, changes the node being plotted. Examining the last, best, average and worst cases, makes it possible to pinpoint things like problems that are intermittent. To change the packet sizes being displayed on the grids, simply move the vertical line in either plot with the mouse and then double click. The left and right hand side packet sizes are not linked to each other. The algorithms used to collect the information are not identical, and as a result, the information displayed in the left and right hand grids and plots, are not 100% correlated. By latency, we mean the time required for a zero byte payload message to go back and forth between two nodes – this is the so called “ping pong” test. The left side can display either the latency or the packet transfer time. When we measure bandwidth, we use an algorithm which floods the system with many packets at the same time, in an attempt to saturate the fabric. We had to “back off” on the bandwidth paradigm we initially used to avoid breaking one particular MPI release, demonstrating that the product can be used to partially verify MPI operation. What that also means is that the product in its current form does not present the absolute highest bandwidth achievable by a cluster.

The dark red cross that appears on both of the grids in Figure 1 indicates that a problem exists in this cluster in both latency and bandwidth for node 8. The left hand cross tells us explicitly that for a packet size of 16 Kbytes, node 8 has a problem with packet transfer times. For all of the good nodes outside of the cross, the average transmission time is 44 microseconds. For node 8, these times grow to around 90 microseconds. Looking up the column of the cross, we can read off

the transmission times between node 8 and the other nodes in the system. These times vary between 91 and 93 microseconds. Looking across the row that forms the cross, we can read off the receive times, which vary from 90 to 93. Similarly, looking at the right hand grid, we discover that for packet sizes of 8 Kbytes node 8 only achieves bandwidths of 190 MB/sec, while the other nodes in the cluster typically average 430 MB/sec! The packet size used for all the nodes in the grid can be read off of the heading at the top of the grids along with the time or bandwidth units and the statistical method employed, which in this case is average. The background colors chosen for the grid are summarized in the legend beneath each grid.

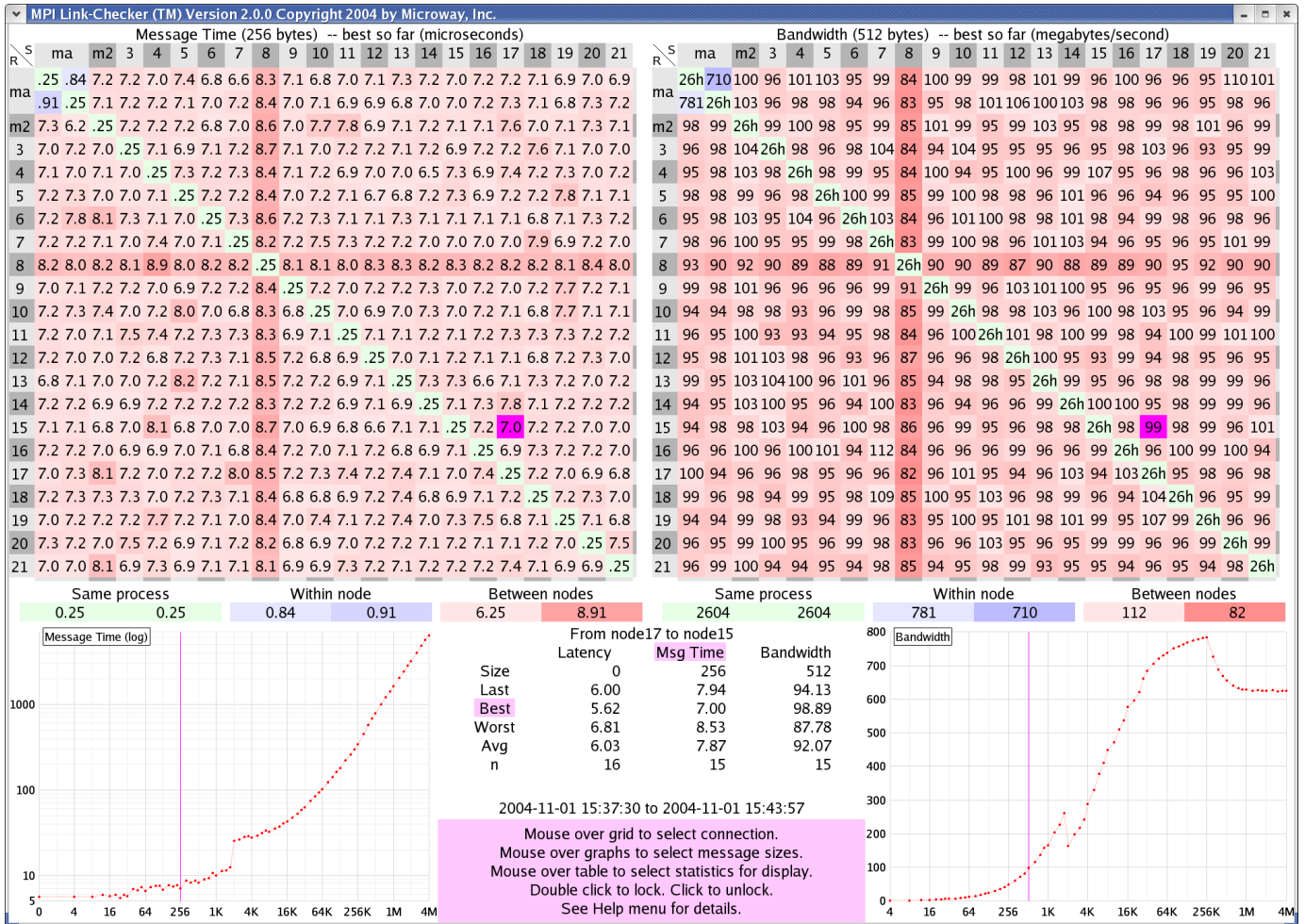
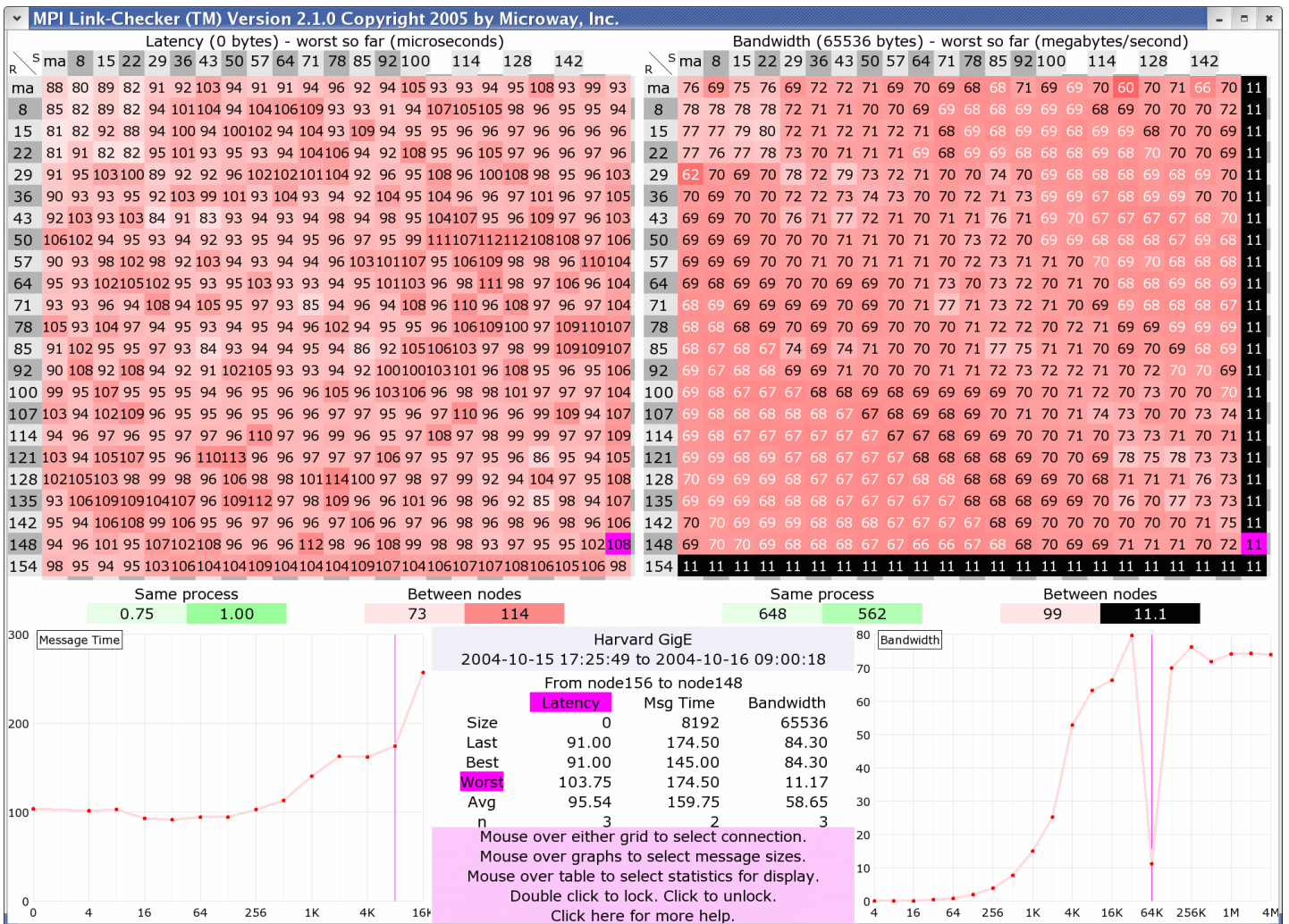


Figure 2

The values along the diagonals of the grids are the transmission times within each processor or node, which are a strong function of how the particular MPI implements its shared memory paradigm. The master node, ma, is shown in the upper left corner. For it, we show the transmissions times between two processes on one CPU and the times between two processes on the same motherboard, but on different CPU's. Whether or not this information appears is a function of how many processors we tell MPI it has to work with. If we select more, the diagonal will expand, but the performance will go down. Since our goal is to explore problems with the hardware and not the MPI implementation, we recommend not over populating the processors with MPI tasks. Looking at the legends at the bottom of the grids, we can quickly determine that the transmission time within the CPU for a 16K packet is 1.8 microseconds while that between CPU's on the same motherboard is 18 microseconds. Similarly, the bandwidth for 8K packets

between processes on a single CPU is 5.6 GB/sec and between processors on the same node it goes down to 1.1 GB/sec. Using this resource of MPI Link Checker makes it easy to compare the MPI performance of four and eight way processor systems to singles and duals. Both of the plots below the grids are for node 17 transmitting to node 15. This can be seen by examining the location of the purple boxes in the lower right quadrant of both grids. The X axis of the grids specifies the transmitter, while the Y axis specifies the receiver. Looking at the transfer time plot, we discover that a zero byte packet takes 6 microseconds, which is the latency of this InfiniBand combination. We also see a kink at 2 Kbytes, which is caused by the transition from Eager to Rendezvous protocols within MPI. Looking at the right hand side, we can see the impact of this jump in latency along with a drop off above 256 Kbytes. The latter is an artifact of the InfiniBand driver being used. The two plots always refer to the same connection.

Figure 2 shows the same cluster running with smaller packet sizes (256 bytes on the latency side and 512 bytes on the bandwidth side). Note that the crosses are much less visible as the difference



**Figure 3**

in both latency and bandwidth is now only about 15%. Note, that instead of looking at average times, we are now looking at the best times. Being able to vary the statistics being used to look for problems turns out to be a very important feature of the product. Using the tool to find

problems with an intermittent cluster, there are times when the problem we are looking for might occur only 5% of the time, or possibly only after a certain temperature gets hit. What makes it possible to find these kind of problems is the fact that we can look for a worst case value, that is hidden amongst rather good average values. When the worst case value only crops up on just a single, or small group of nodes, that tells us we have found an intermittent problem.

### **Advanced Features**

MPI Link-Checker can be used to measure the transfer times between processes running on the same node and the same processor! In Figure 2, we arranged the tasks, so that the master node had extra processes running on it. The data we collected for the master node appears in the upper left hand part of the grid. From this we can deduce that the transfer time for 256-byte messages between MPI processes running on the same processor is 250 ns while the transfer time for processes on the same motherboard, 840 ns. The bandwidths are also displayed and peak at 2.6 GB/sec. These results are very dependent on the MPI implementation being used.

In addition to the latency and bandwidth plots and off line data collection features, Version 2.0.0 offers a number of features which can be used to speed up data analysis. To reduce display times, we have made it possible to reduce the number of nodes being presented. Rather than looking at all the data, we make it possible to divide the cluster up into groups of nodes and then examine the data for each group against the others. This also makes it possible to examine the cluster using the knowledge of the cluster's topology, which can affect things like latency. If we create these groups taking advantage of this knowledge, the grids that result will produce uniform grid patterns. Problem nodes then show up clearly, because they end up breaking the region with crosses. The new release contains a GUI which gets used to define node domains and then use the stored definitions to drive the MPI data collection and display processes. If you examine the left hand "latency side" of Figure 4, which shows the transfer times between nodes in a 158 node cluster, you will note that there are regions of good and bad connectivity. The bad regions contain transfer times in which extra hops (transversing a complex switch) had to be made between nodes. If this cluster had been properly broken up into local domains of equal latency using our group facility, these features would not show up on this grid. Once a cluster's domains have been properly defined, the information can be used in the future to pin point subtle problems. ,

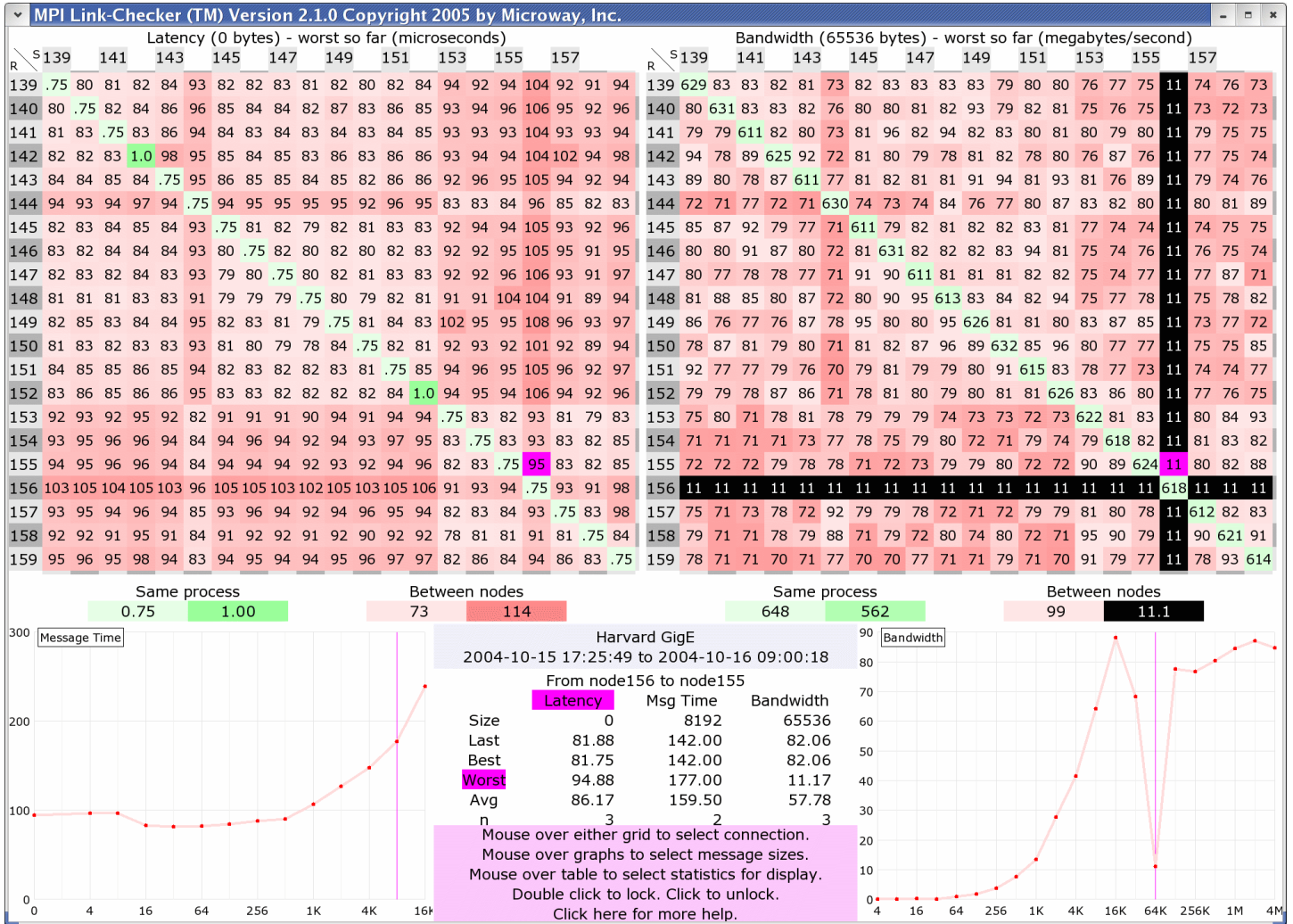
Even if you are willing to wait for the program to analyze a densely packed grid, you will still have another problem, and that is that our tool does not have the resolution to properly display a complete grid when the cluster being analyzed has more than 40 nodes. While crosses will still show up for bad nodes, the details are no longer legible and are not written to the screen. To solve the display time and screen resolution problem we added a drill down feature, which makes it possible to overload the grid and then drill down until the desired information becomes legible.

Figure 3 is a screen shot of a 158 node cluster that clearly does not fit onto our standard 32x32 grid. When you specify worst case statistics, the standard cross we are used to seeing now pops out of the data onto the screen, simply because each cell in the grid now contains the worst value of all those nodes contained by the cell. In this case, the bad node is 156, and it appears as a dark backwards "L" centered on the lower right corner. Looking at the y axis labels suggests that the problem node falls between nodes 154 and 158. If the user wants to cheat, he doesn't even have to drill down into the grid to figure out which node is having problems. Just passing the cursor over any of the grid cells on the bandwidth side which are black and appear to be operating at 11 MB/sec (instead of the 67..80 typical of worst case cells) results in the legend area identifying the guilty party. In Figure 3, the purple cell where the cursor sat when this GIF was shot, results in the legend clearly identifying that node 156 transmitting to node 148 is the offender that is running at 11 MB/sec. To examine the region around 156 in greater detail all that one needs to do

is drill down using the mouse (twice) to end up with a grid that displays the details for the region surrounding node 156. The result of drilling down is shown in Figure 4.

**Figure 4**

Both of the grids now reveal important features of the cluster that heretofore were impossible to

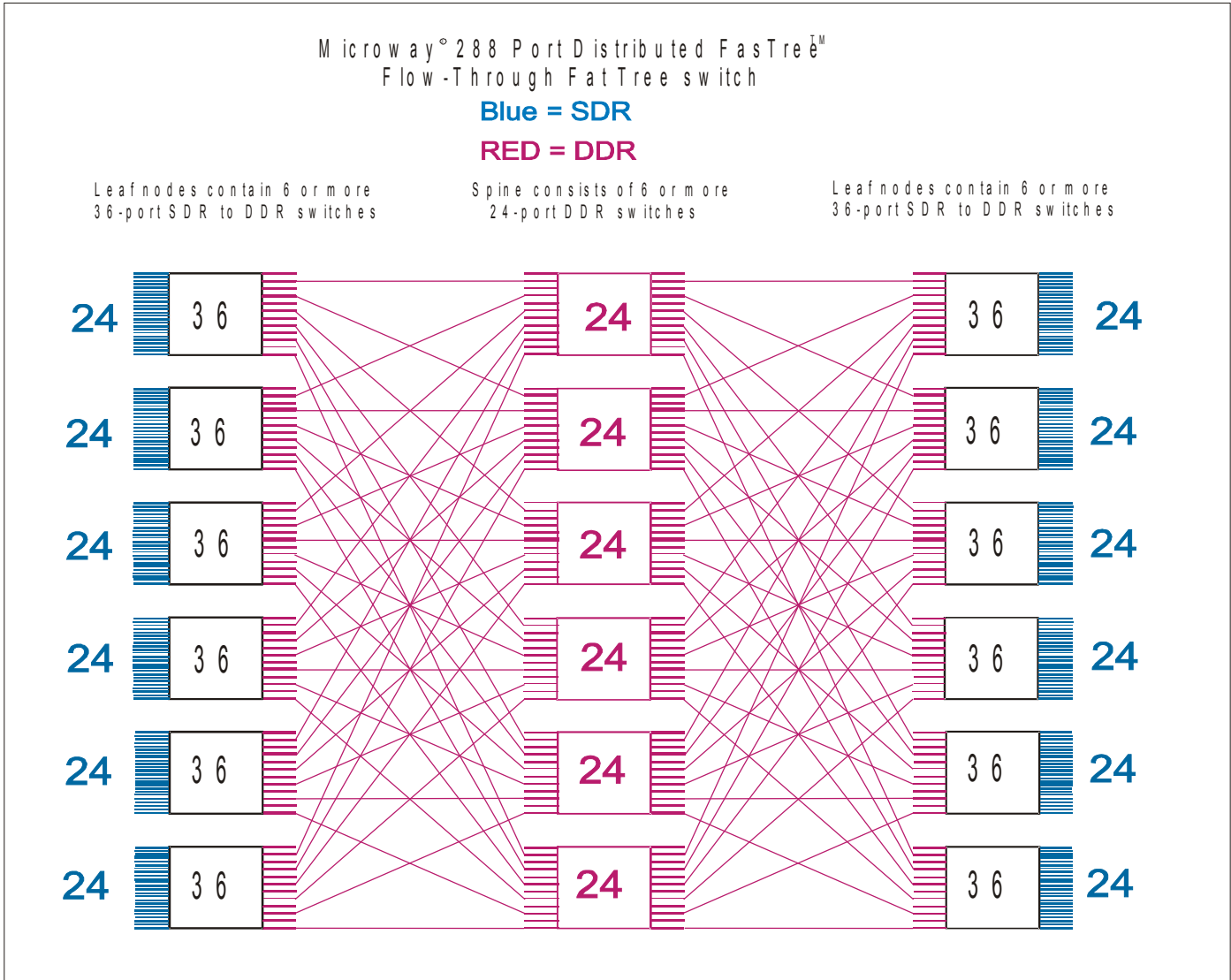


observe with any other tool. First, the right hand bandwidth grid clearly displays the problem with node 156. Second, the bandwidth vs packet size plot for node 156 transmitting to node 155 shows a spike for packet sizes of 64K. This is actually an artifact, and typical of what one finds for intermittent problems within a cluster. In this case, a keen eye reviewing the data as it was coming in from the MPI collector figured out that there was a problem with node 156, and it's cable was reseated before the data being displayed was fully collected. On the left side, which was measuring transfer times, the worst case data clearly shows the problem with node 156, but it also suggests that some sort of problem exists with node 144. And, while 144 also shows problems on the bandwidth side grid, the bandwidth side does not reveal what the real problem turns out to be. Looking at the transfer times for node 144 reveals that it communicates with nodes 153..159 at the same rates that these nodes communicate with each other. What that translates into, is the fact that 144 is a member of the nodes in the region 153..159. All that it takes to fragment one of these plots, is to hook up the leaf of a complex switch to nodes that do not fall in numeric order. If the nodes had been defined in advance (or the nodes hooked up to the

switch in a rational manner) as part of a group, then the program would have lumped all of the nodes within their groups, and the display that resulted would be much more uniform. There are other domains which the latency grid points out. For example, nodes 145..152 appear to be members of a group of 8 nodes. Examining the diagonal for the other nodes in the system, will reveal other groups in the system, along with the topology of the switch used to hook all of the nodes together.

**Features for Large Clusters - FastCheck™**

The next version, Version 3.0.0, will include new features under development for very large clusters. These will simplify searching for bad nodes and will also make it possible to display the



**Figure 5 – Microway 288 Port InfiniBand FasTree Fabric**

information on screens much faster. The data collection process we use collects information that is proportional to the  $2 \cdot N^2 \cdot P$ , where N is the size of the cluster and P is the number of packet sizes probed (typically 85). For a 40x40 cluster, this requires us to run 272,000 fast executing probe routines. For a cluster that is 1,000 x 1,000, this jumps to 170 million probes. And, it turns out that some of these probes take a while to converge on accurate results, so the data collection

time can really add up! However, is it really necessary to collect the information between every node in a cluster and every other node? We think not. We have just added a feature to the product called FastCheck. FastCheck uses a more compact version of the algorithm we use in the full cluster test, but it assumes that if a node checks out ok (i.e. within 5% of the best node) that there is no reason to perform a full check. In the event that a node falls between the cracks, it does a more thorough test of that node and determines whether or not the problem was a glitch or a statistically meaningful event and whether or not it is in transmission or reception. It then provides details of the failure. FastCheck takes about 100 nodes per second and the time it takes to perform a test scales with the number of nodes. What the user does at this point is up to him, but a good starting point is to declare a subnet of nodes that are near each other on the same switch and run link checker with the group, to see what the characteristics of the problem really are. For example, in an InfiniBand connected cluster, if the problem only shows up in transmission and not reception, what that indicates is that the transmission path contains a faulty capacitor someplace in the line between the output and input nodes. If it appears for all nodes, what that means is the capacitor probably resides in the switch that the HCA is hooked up to.

Possible problems with the following nodes:

node2 sending 75 percent efficient with 3 slow connections out of 6  
node3 receiving 99 percent efficient with 2 slow connections out of 6  
node4 sending 77 percent efficient with 5 slow connections out of 6  
receiving 66 percent efficient with 6 slow connections out of 6

There are other more complicated problems that need to be solved for very large network fabrics. Figure 5 contains a 288 node fat tree InfiniBand fabric. Whether the switches are connected with cables or PCB's, problems can arise anywhere in the communication path. We have a feature that will be shortly released which uses our grouping facility to reduce the time it takes to define tests which do a complete search of a cluster connection space. Once these tests have been run and domains defined, cluster verification gets reduced to running specific tests at periodic intervals, and then either comparing the results with historic results or looking for "crosses" in grids. Included in future features, is an automatic search facility that finds system problems, after the graphical techniques discussed above have been used to understand and define the clusters interesting topological features. The basic idea behind these tests is to make it possible to more quickly find problems and run tests over night that quickly will confirm for the SA's running a cluster, that all of the nodes are running at full speed.