**Mapping Heracles Communication Network: Data and Analysis**

Nathan Justice Robert Fitzgerald Matthew Kalan

Department of Computer Science and Engineering University of Colorado Denver, Graduate Students

Our project code is available at:

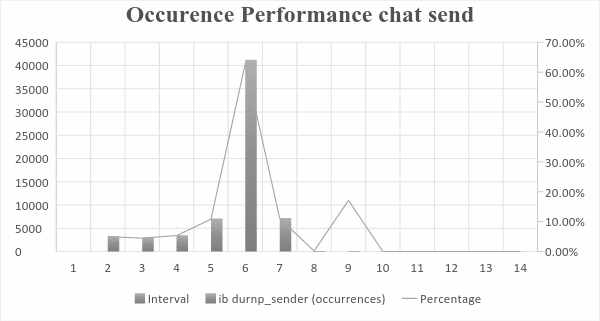
<https://github.com/robfitzgerald/csci5593-project>

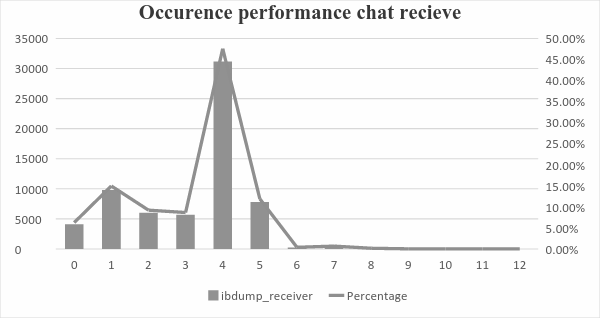
**1 RDMA Send and Receive**

This test was designed to analyze inter-data packet delay or occurrences, monitor network performance, and observe flow and congestion control. It also analyzed the switch buffer management. The test was done using ibdump and sniffer.pcap. The ibdump function is limited by the host ram capacity, the loss of data packets due to interference, and the size of data packets. Sniffer.pcap only works for SDRAM (8 Gbits/s) data rate and records 2GB file capacity.

**1.1 Data Capture**

The test transferred data on a SDR link with no congestion, with 872 bytes payload (0.8k MTU). This was done four times. The graph below shows ibdump\_send raw data and ibdump\_recieve, including their transfer intervals and the percentages of the occurrence.





The expected inter-packet interval from the same source was four microseconds. The network had optimal performance at the initial send node and at node six on both send and receive. However, as the network became busy, there were delays and interrupts, which became incidents or occurrences. For example, during the message passing and receiving stages from node three to node four, the delay recorded was approximately 47% of the transfer time. The more the occurrences, the greater the percentage time to accomplish a transfer as can be seen from the graph.

**1.2 Flow control (FC) mechanism**

The sniffer.pcap observed that a sender is sending approximately 128 Megabytes of data to a receiver, using RDMA\_WRITE using 2k MTU at 65536 data packets. Each packet containing at least 2074 bytes. Each packet occupies 33 FC blocks. It should be noted that the IB sender will not send data packets unless it knows for sure that the other side of the physical link has enough buffer to hold the data.

Flow Control Packets (FCPs) are used to report the available buffer space. The FCTBS (flow control table) stores the number of bytes that the table can handle. At a point, the FCTBS on the last node was 2262. On the previous node, it was 1404. Since the number of blocks per FC in the message is 33FC, to determine the number of packets sent from the previous node to the current node divide the (2262 – 1404)/33 = 26 data packets sent from last node to current node.

**1.3 Switch Buffer Management**

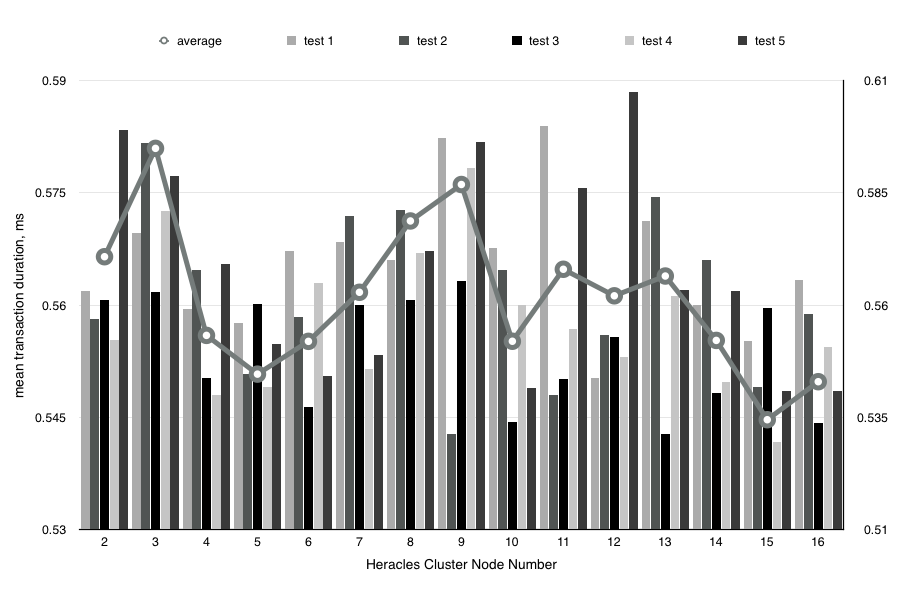
The switch used is MLNX SX6018. The sniffer was used to determine the switch buffer size. When the port on the connectX3 showed green light, it signifies the buffer is empty. When the lights start changing it signified that the buffers in the switch are filling up. When the light changed to full yellow, it means the transfer is complete and the buffer is full. This transfer rates happens as coded by the programmer and the availability of storage space in the host connect. The average buffer transfer rate is four nanoseconds. The input VL of the switch buffer space was around 32 kilobytes, with configuration of 4 VLS, i.e. 4\*32kB = 128 kilobytes for each input port.

**2 Baseline Tests**

This test was designed to find a minimum cost for using Open MPI on Heracles. For each combination of nodes, we transacted 1000 messages per test. The transaction times were averaged to get 120 average transaction times for the 16 choose 2 combinations. This test was performed 5 times: 4/26 at 10:44pm and 10:48pm; 5/1 at 11:20am, 11:25am, and 11:30am.

**2.1 Test Results**

We found that the average messaging time for all nodes communicating with all other nodes was within a range of 0.05 ms. Nothing about these results suggests a distinct time difference for communication on specific nodes. If the performance of nodes 3, 8, and 9 seems heavier, it would still likely take a number of additional samples to say anything convincing about their deviation.

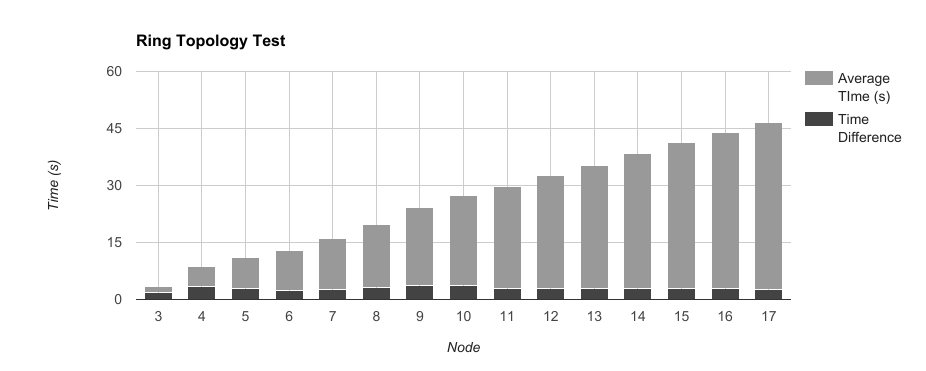


**3 Topology Tests**

Our hypothesis was that the heracles network was a complete topology, where every compute node had a direct communication line to every other compute node. One portion of our implementation was dedicated to proving this programmatically. To accomplish this we designed three messaging patterns to simulate various logical topologies. We decided to pattern a ring topology, a star topology, and a complete topology.

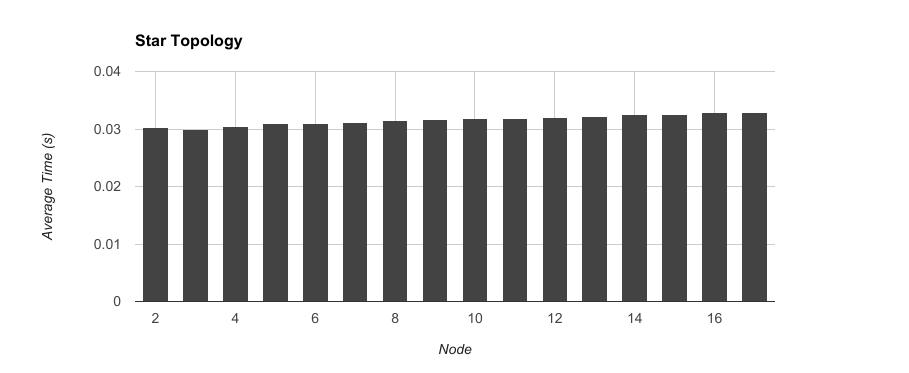
**3.1 Ring Topology**

Below is a graphical representation of the results of a ring topology test.

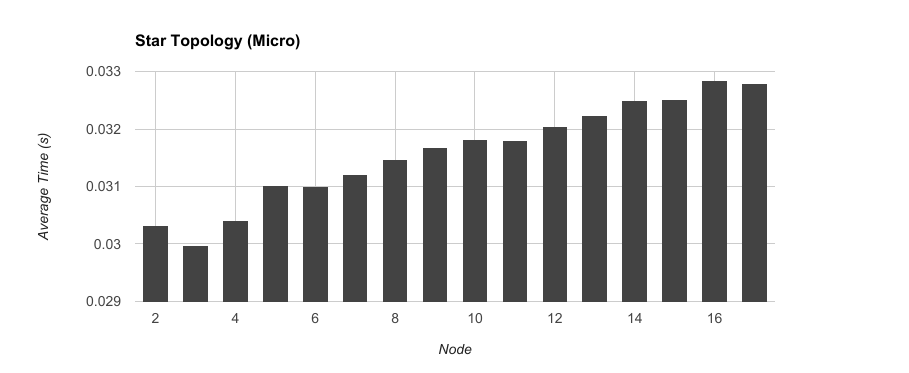
 Each light gray bar represents the average message transaction time of a 1000 messages received by an individual node. As you can see these bars incrementally increase in size for each node. This can be explained by the fact that in our ring test a node does not send a message until it has received all of its messages. This delay creates a staggering effect between each node. Despite taking an average this delay is still present because each node has a long wait time on their initial message, and increasing the number of messages passed increases the amount of time each process must wait. Therefore we measure the difference between a node’s average message time and the node’s predecessor, this provides the average message time without the initial delay (dark gray bar). When analyzing the difference in average message time, it is clear the nodes differ very little and therefore meet the criteria for a ring topology, this however does not prevent the network from being a complete topology as a ring topology is a subset of the complete topology.

**3.2 Star Topology**

The star topology was measured by considering a node to be the center of the star, have it communicate back and forth with every other node, and then repeat the process for each node. If this test were performed on a star topology we would see one node with significantly smaller average message time relative to the other node’s average message times. As we can see in the graph below there is no definitive center node in this network. This proves that the topology is not a star, but it also does not prove that the topology is a ring or a completely connected network.

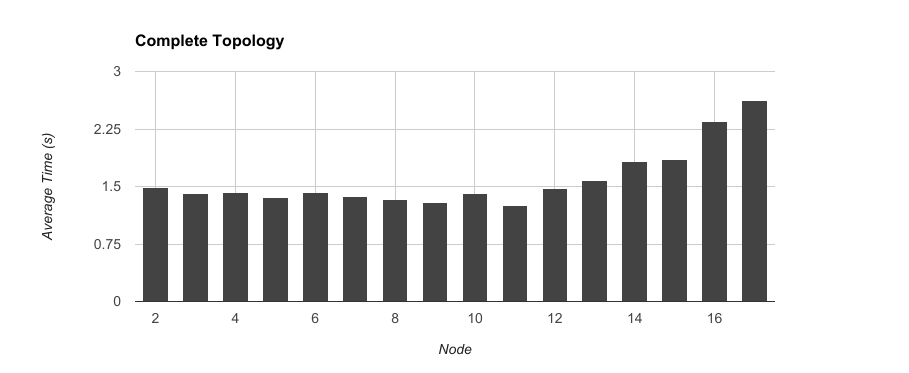


When we analyze this data at a microscale we can see that despite the overall trend of average message time remaining the same, we can see that the later that a process runs in the queue the average message time is typically slower. While the scale is miniscule, it is important to note that there appears to be degradation in the message passing process as time goes on. If we had designed the topology tests to also run in reverse this pattern could be proven to be relative to the order of processes run, not which node the process ran on.



**3.3 Complete Topology**

The complete topology test has all nodes attempt to simultaneously send messages to all other nodes. This is similar to the star topology test, but the processes do not uniformly assume a center node. This difference creates more traffic during the iteration which results in a higher slowdown in later messages. This could be proven to be associated with the process’s place in the queue instead of the node that ran the process, if the test was designed to also run in reverse order.



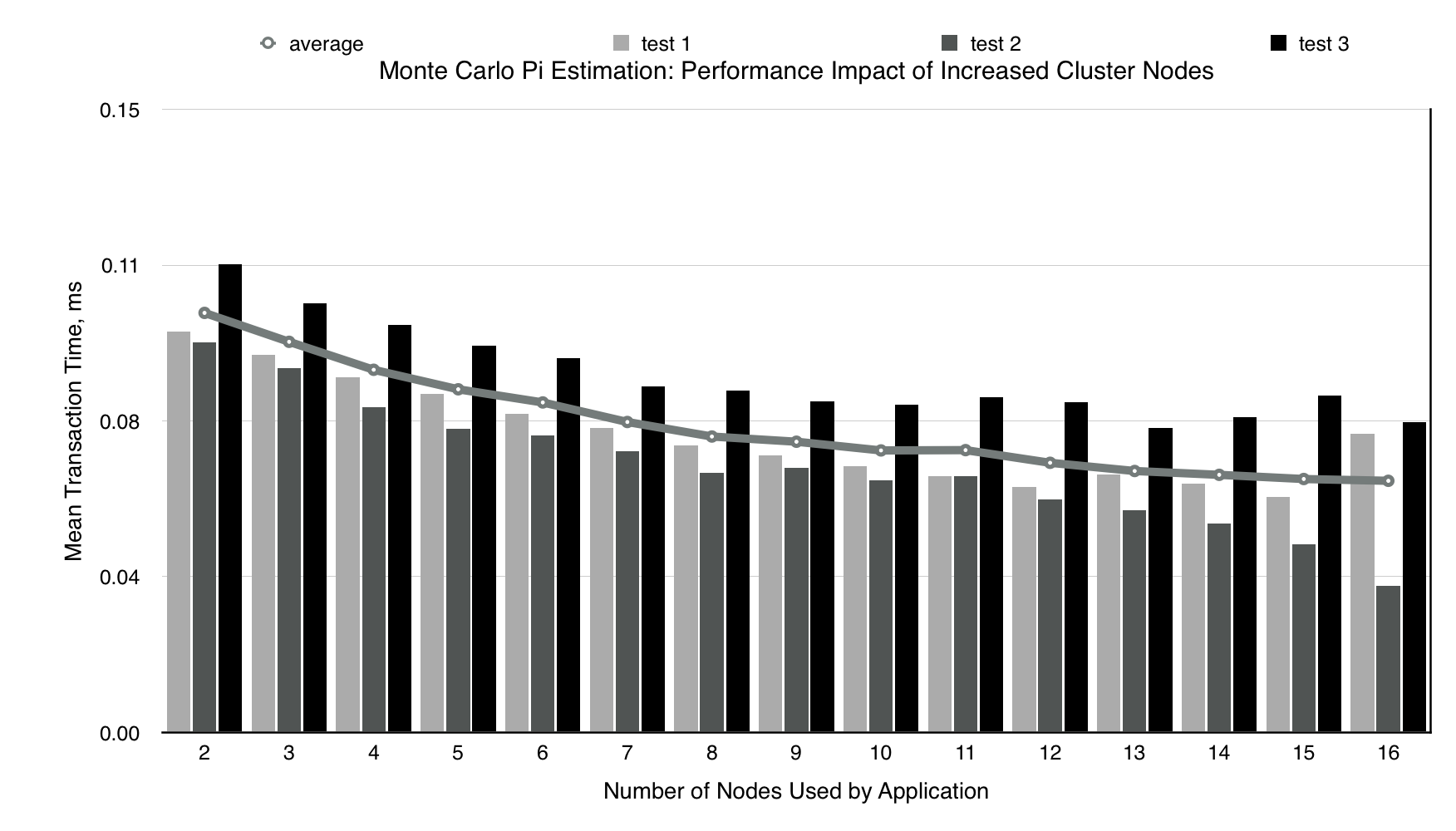
**4 Monte Carlo Pi Test**

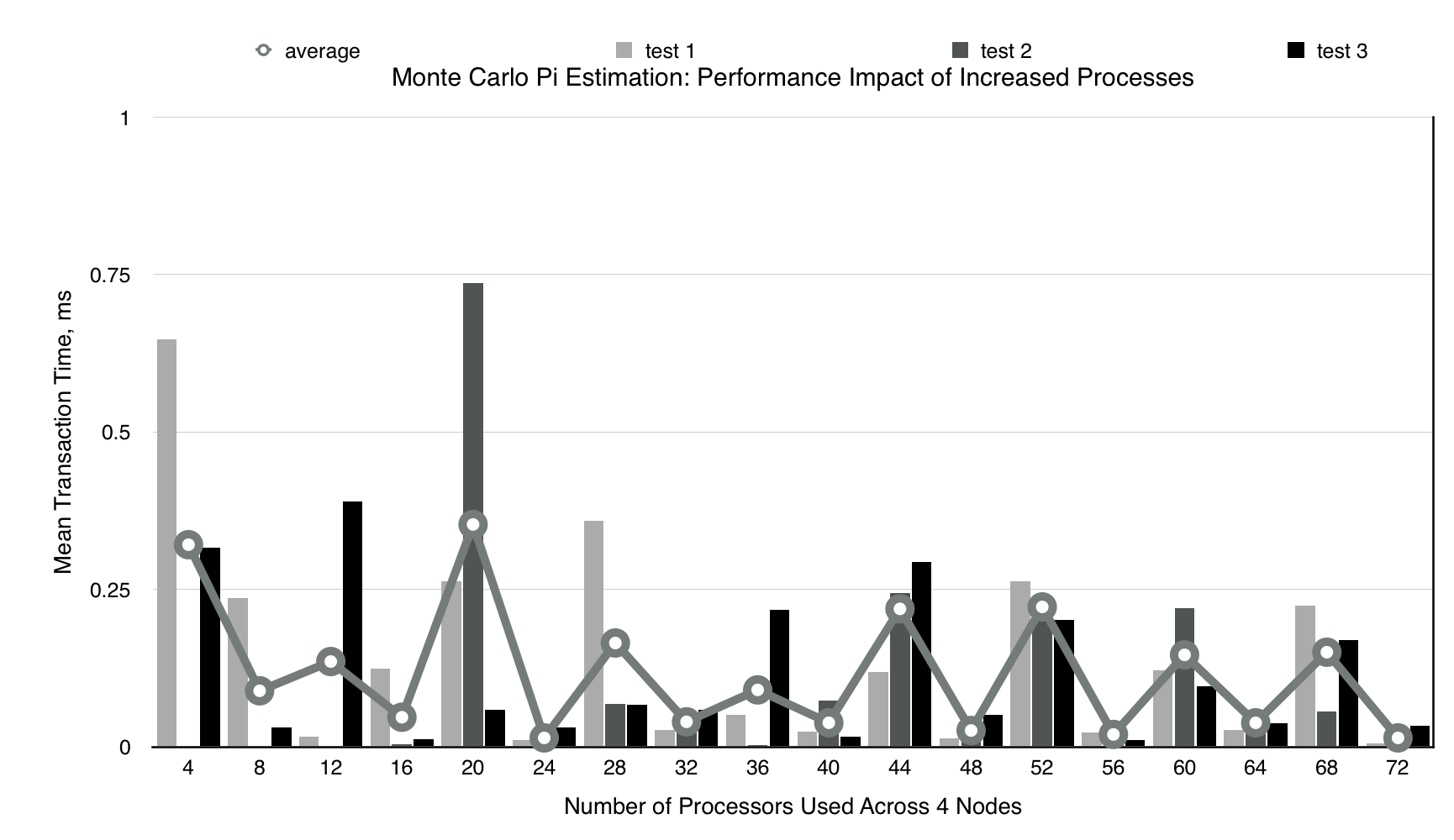
So far, our test applications have been performing little to no work beyond calculating the parameters of the next MPI message. For this test, we employed our MPI processes in the estimation of Pi by stochastic methods, namely the Monte Carlo method. This simple application produces random coordinates with vector values within the range [0,1]. For any values produced which fall within the first quadrant of the unit circle, we produce a “1” value; otherwise, a “0”. These values are summed, and the sum is returned to the master process, which knows the total number of random values produced. The operation will give us an estimate of the value of π.

We modified two MPI parameters in these tests: # of nodes, and # of processes. The result is two different sets of result data.

**4.1 Test Results**

We expected that increasing nodes would have little effect, as adding one node incurs the same costs as adding the previous, in terms of process loading and core workloads. We expected that increasing processes might increase wait time for message passing, as more processes on a node would be competing for CPU time. In both cases, the overall trend was improvement, with increased nodes seeming to strictly improve transaction wait times. However, while increasing processes did result in an overall trend toward decreasing times, it seemed to be impacted by random performance losses, consistent with the hypothesis that the increased work on cores would add some unpredictable behavior.





**5 Traffic Test**

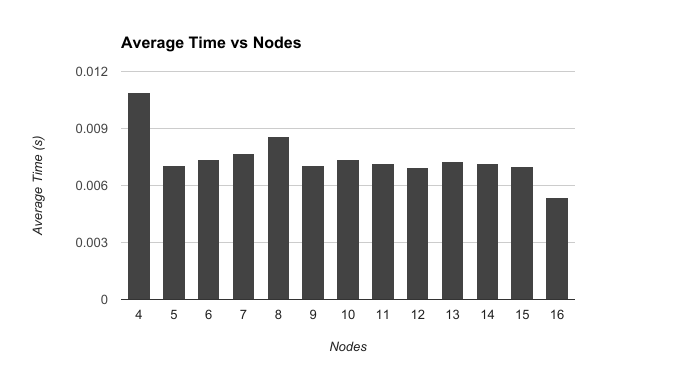
Our implementation also simulates traffics and measures the average round trip message time between 2 nodes. Using this test we studied if the number of nodes or the numbers of processes affected message time.

**5.1 Variable Number of Nodes**

To test variable number of nodes we ran traffic tests with a constant sixteen processes. We measure the roundtrip time of messages of two processes. For example the in test with 4 nodes, each node had four processes, and we measured the round trip message time of process 0 and process 15. Due to the number of processes being constant we are inadvertently also studying how the distribution of processes among nodes can affect the average runtime of messages.

|  |  |
| --- | --- |
| Nodes | Average Time |
| 4 | 0.0108833 |
| 5 | 0.007031582113 |
| 6 | 0.007348394888 |
| 7 | 0.007674592357 |
| 8 | 0.008573062262 |
| 9 | 0.007072244504 |
| 10 | 0.007372071658 |
| 11 | 0.007145640501 |
| 12 | 0.006952870521 |
| 13 | 0.007281743396 |
| 14 | 0.007166958132 |
| 15 | 0.006978253065 |
| 16 | 0.005373470437 |

The data above is graphed below. By analysing the data we can recognize that the a subtle trend of faster average message transit time exists when more nodes are introduced into the system. This is logical, as more nodes are used fewer processes must share the switch port with other processes. One could mistakenly assume that it is best to disperse processes to as many nodes as possible, but this does not account for the speed that one cluster may pass messages among its own cores.

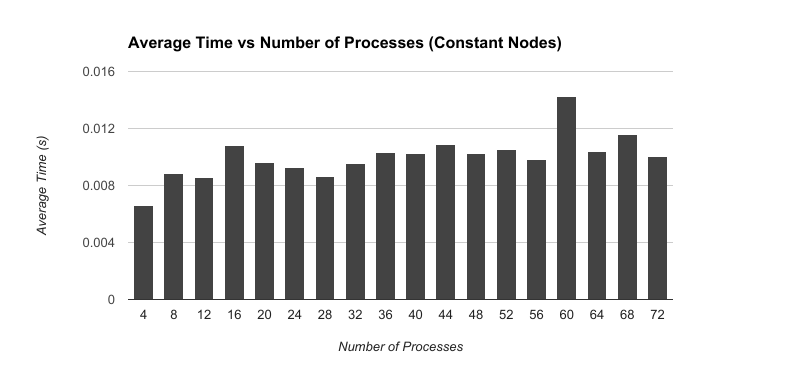


**5.2 Variable Number of Processes**

We measured the variable number of processes by running increasing multiples of four on the same four nodes. This test shows the changes in traffic on a system that with a limited number of nodes. The aggregated data from the test is presented below.

|  |  |
| --- | --- |
| Number of Processes | Average Time (s) |
| 4 | 0.006594248879 |
| 8 | 0.008833950185 |
| 12 | 0.008578091378 |
| 16 | 0.01079745 |
| 20 | 0.00961075 |
| 24 | 0.0092787 |
| 28 | 0.0086266 |
| 32 | 0.00951185 |
| 36 | 0.01035043333 |
| 40 | 0.01027735 |
| 44 | 0.01086755 |
| 48 | 0.0102252 |
| 52 | 0.0105133 |
| 56 | 0.00981255 |
| 60 | 0.01427495 |
| 64 | 0.0104 |
| 68 | 0.0115994 |
| 72 | 0.0100315 |

This graph reveals a fairly constant rate of average message transit time, that slightly favors faster times with fewer processes and slower times on the end of the spectrum with more processes.With each additional process the number of messages also increases because we assumed that each additional process would need additional messages to communicate with the other nodes. Overall there is very little change in the trend of increased processes.



All of our tests were unable to to account for other traffic or processes running on the network. These inconsistencies could generate vast differences in the subtle trends of this data.