A Performance Comparison of TCP/IP and MPI on FDDI, Fast Ethernet, and Ethernet

Saurab Nog and David Kotz

Department of Computer Science
Dartmouth College
Hanover, NH 03755-3510

saurab@cs.dartmouth.edu, dfk@cs.dartmouth.edu

Dartmouth Technical Report PCS-TR95-273
November 22, 1995
Revised January 8, 1996

Abstract

Communication is a very important factor affecting distributed applications. Getting a close handle on network performance (both bandwidth and latency) is thus crucial to understanding overall application performance. We benchmarked some of the metrics of network performance using two sets of experiments, namely roundtrip and datahose. The tests were designed to measure a combination of network latency, bandwidth, and contention. We repeated the tests for two protocols (TCP/IP and MPI) and three networks (100 Mbit FDDI (Fiber Distributed Data Interface), 100 Mbit Fast Ethernet, and 10 Mbit Ethernet). The performance results provided interesting insights into the behaviour of these networks under different load conditions and the software overheads associated with an MPI implementation (MPICH). This document presents details about the experiments, their results, and our analysis of the performance.

Keywords - Network Performance, FDDI, FastEthernet, Ethernet, MPI.

Revised - January 8, 1996 to emphasize our use of a particular MPI implementation, MPICH.

1 Introduction

This document presents benchmarking tests that measured and compared performance of TCP/IP and MPI (Message Passing Interface) [3] implementations on different networks and their results.

We chose these protocols (TCP/IP and MPI) because

- TCP/IP:
  - Greater flexibility and user control
  - Higher performance
  - Wide usage and support
- MPI:
  - Programming ease
  - A widely accepted standard for distributed computing applications.

Since our MPI implementation, MPICH, runs on top of P4 [2] which in turn uses TCP/IP, a performance comparison of TCP/IP and MPI gives a good estimate of the overheads and advantages associated with using the higher level abstraction of MPI as opposed to TCP/IP sockets.

We also compared three different networks to understand the inherent hardware and software limitations of our setup. We repeated the same set of experiments on

- 100 Mbps FDDI (Fiber Distributed Data Interface) token ring
- 100 Mbps Fast Ethernet
- 10 Mbps Ethernet

In most cases, we cross-checked our results with results from the publicly available "ttcp" package. The results seem to be in very close agreement, including the presence of spikes and other anomalies.

2 Setup Details

We ran two different tests ("roundtrip" and "datahose") on all six combinations of protocol (TCP/IP and MPI) and network (FDDI, Fast Ethernet, Ethernet). Although our experiments were not specific to any implementation of the hardware or software standards, the results were necessarily dependent on the specific implementations. Other implementations may have different performance.

2.1 Hardware & Software

The FDDI and Ethernet experiments used 8 RS6000/250s running AIX 3.2.5 at 66MHz. The FDDI was an isolated network and the Ethernet experiments were run when the Ethernet was lightly loaded (usually overnight).

The Fast Ethernet tests used 6 Pentium/100 based PCs with 32 MB RAM, running FreeBSD 2.1 on an isolated network.

Both the RS6000s and PCs had MPICH version 1.0.10 (released July 31, 1995) running on top of P4.

2.2 Environment

All the tests, except those involving Ethernet, were conducted in an isolated mode. For the Ethernet experiments we did not disconnect ourselves from the rest of the world, so we were careful to choose quiet times of the day for testing. This minimized the risk of interference from external traffic.

We repeated the tests several times (at least 5), running a few thousand iterations of each test every time. We chose the best values (max for bandwidths and min for time) as our final result. Thus our numbers approximate best-case performance results.

All TCP/IP tests used the TCP_NODELAY option (except the Fast Ethernet datahose).[1] Normally, the TCP/IP protocol delays small packet transmission in the hope of gaining advantage by coalescing a large number of small packets. This delay results in artificially high latency times for small packets. We used TCP_NODELAY to overcome this behaviour and eliminate the wait period. We also set both the sending and receiving side kernel buffers at 64K bytes (maximum allowed on the RS6000s, FreeBSD Pentiums allow up to 128K). The increased buffers were prompted by the observation [1] that kernel buffers are the bottlenecks for most network operations.

For MPI, we used the default configuration in all cases.

2.3 Performance Measures

In these tests we were interested in the following measures of network performance:

- Total Bandwidth: the sum of the individual bandwidths of concurrent processes. It signifies the effective bandwidth usage for the set of all participating hosts.
- Bandwidth/Pair of processes: the average bandwidth each individual pair of processes was able to use.
- Time/iteration: the average time it takes to complete one iteration of roundtrip or datahose.

[1] FreeBSD often crashed when flooded with small messages.

3 Roundtrip

The roundtrip test is designed to measure the following parameters

- network latency
- network contention overhead
- synchronization overheads

All machines were connected to the same network but were paired for communication purposes. Each host talks to its designated partner only. There is a master-slave relationship between the two process in each pair. The master initiates the communication and then measures the time interval it takes for the slave to receive and send the message back. This completes one roundtrip iteration. Our test does many roundtrip iterations in a burst to determine the average time/iteration and the network bandwidth achieved. Many iterations are necessary to overcome clock granularity, cold cache effects, and accuracy problems.

All processes synchronize before switching to the next message size. This serves two major purposes:

1. Each message size is timed separately, restricting timing and other errors to the phase in which they occured.
2. Synchronization prevents processes from running ahead or lagging behind the group.

Roundtrip is a store and forward test. Each slave process receives the complete message from its master and then sends it back. Since at any given time only 1 process in a pair is writing data to the network, there is no contention between the master and slave processes of a pair.

We ran roundtrip on 1, 2, 3 and 4 pairs (up to 3 pairs on FreeBSD) of processors simultaneously to determine how contention influences network performance. Results for various combinations of network and protocol are presented in the following graphs.

The pseudo-code for roundtrip is as follows:

```
for (all interesting message sizes ) {

    set message_size
    synchronize (all master and slave processes)

    if (master){
         /* I am the master */

         start_timing;          /* Initialize timer */

        /* do a large number of iterations */
        for(iteration_count=0;iteration_count<MAX_ITERATIONS;iteration_count++)
          {
            /* Initiate communication with slave */
            write(slave,message,message_size)

            /* Wait for reply from slave */
            read (slave,buffer,message_size)
          }

        stop_timing;           /* Stop timer */
    } else {
        /* I am the slave */

        /* do a large number of iterations */
        for(iteration_count=0;iteration_count<MAX_ITERATIONS;iteration_count++)
          {
            /* Wait for the master to initiate communication */
            read (master,buffer,message_size)

            /* Reply back to the master with the same message */
            write(master,buffer,message_size)
          }
   }        /* End if-else */
}           /* End for */
```

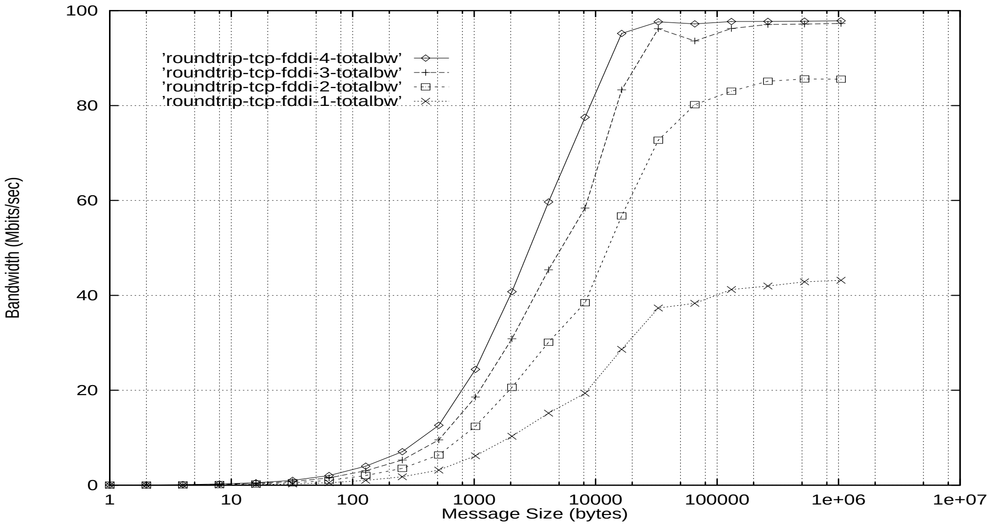

Figure 1: Roundtrip : TCP/IP : FDDI : Total Bandwidth

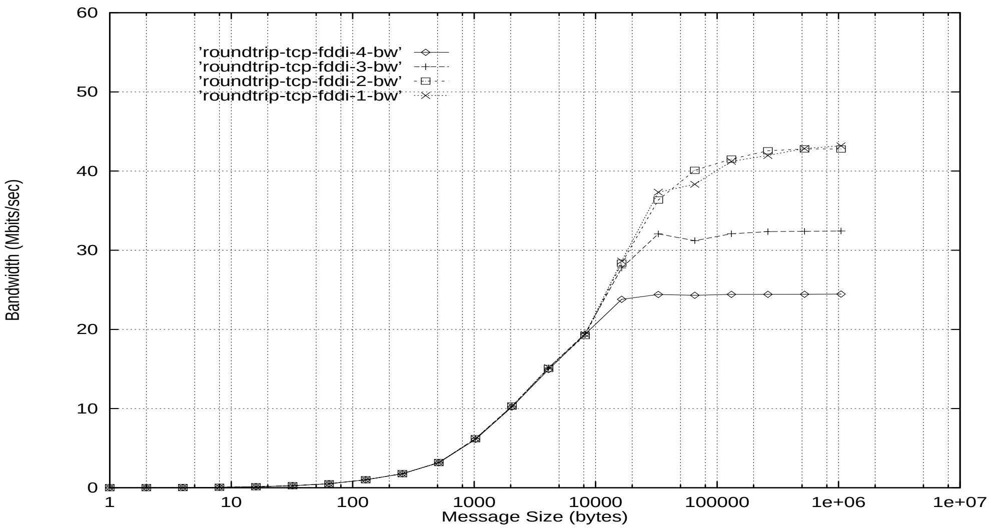

Figure 2: Roundtrip : TCP/IP : FDDI : Bandwidth per Pair of Processes (Master-Slave)

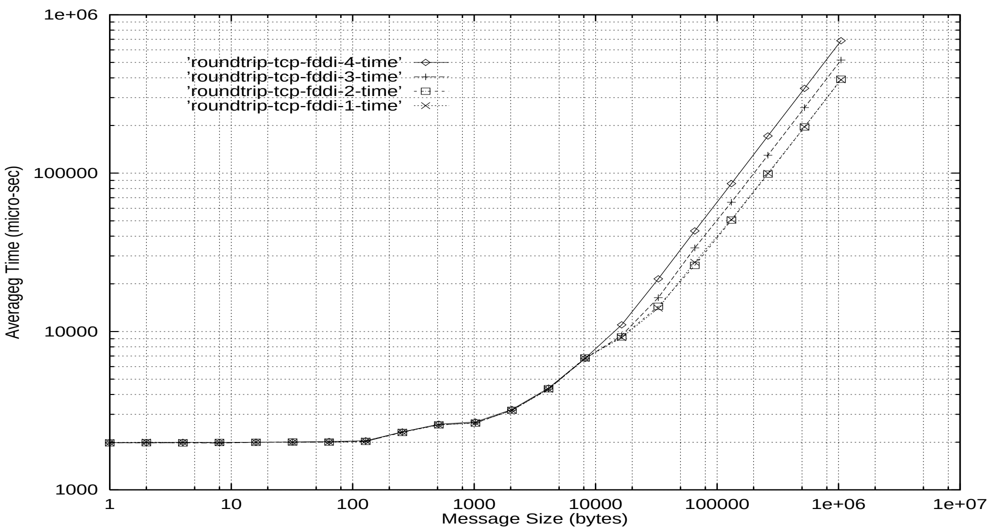

Figure 3: Roundtrip : TCP/IP : FDDI : Time per Iteration

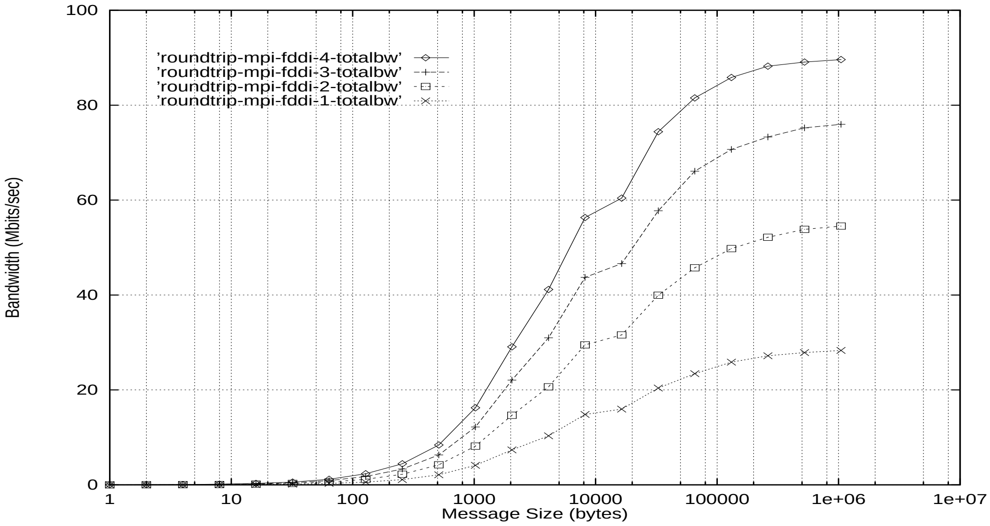

Figure 4: Roundtrip : MPI : FDDI : Total Bandwidth

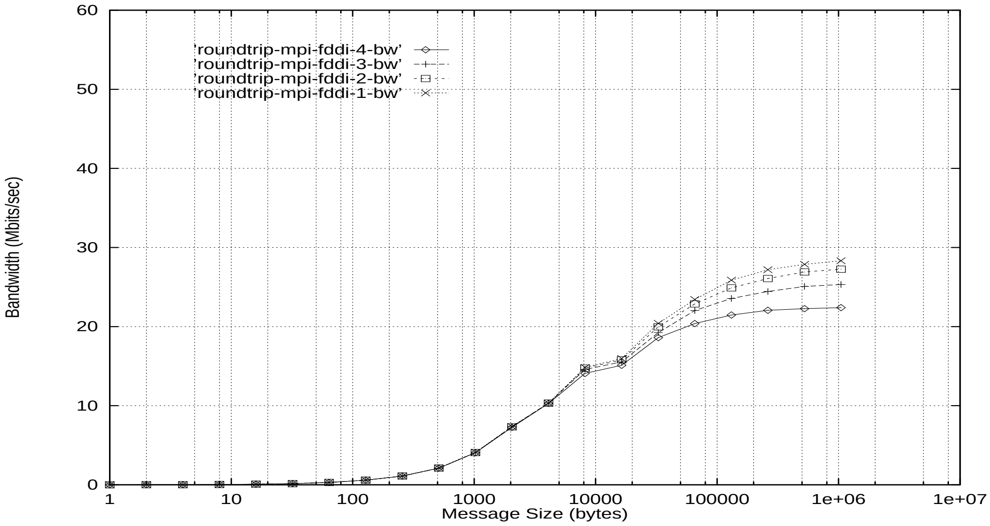

Figure 5: Roundtrip : MPI : FDDI : Bandwidth per Pair of Processes (Master-Slave)

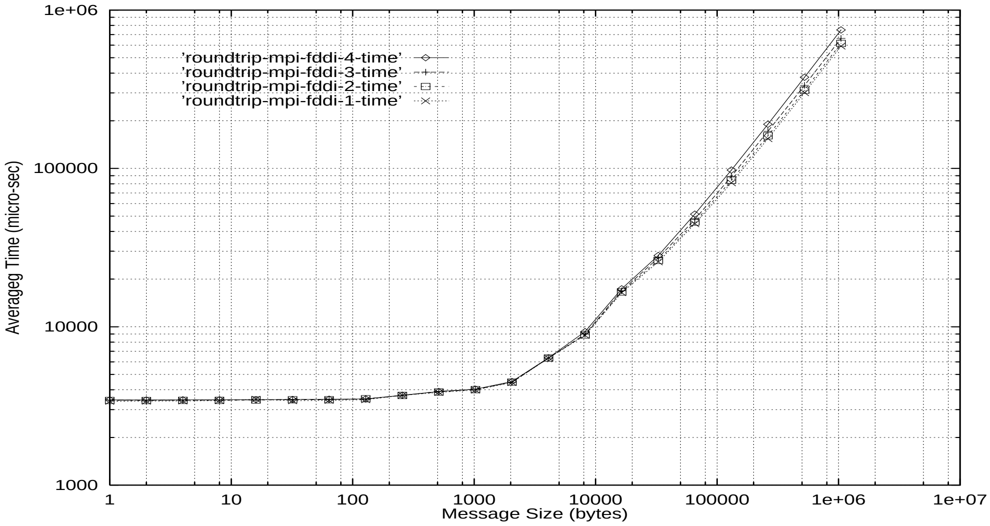

Figure 6: Roundtrip : MPI : FDDI : Time per Iteration

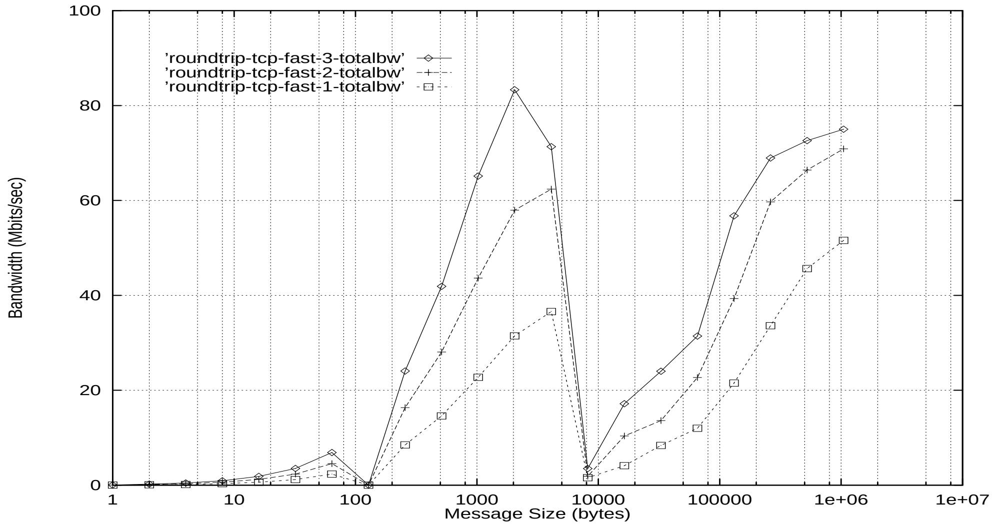

Figure 7: Roundtrip : TCP/IP : Fast Ethernet : Total Bandwidth

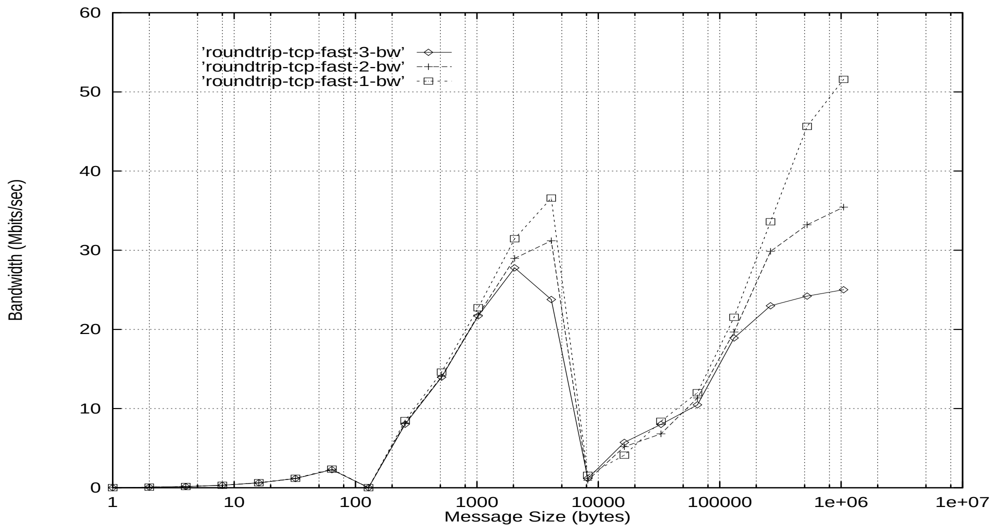

Figure 8: Roundtrip : TCP/IP : Fast Ethernet : Bandwidth per Pair of Processes (Master-Slave)

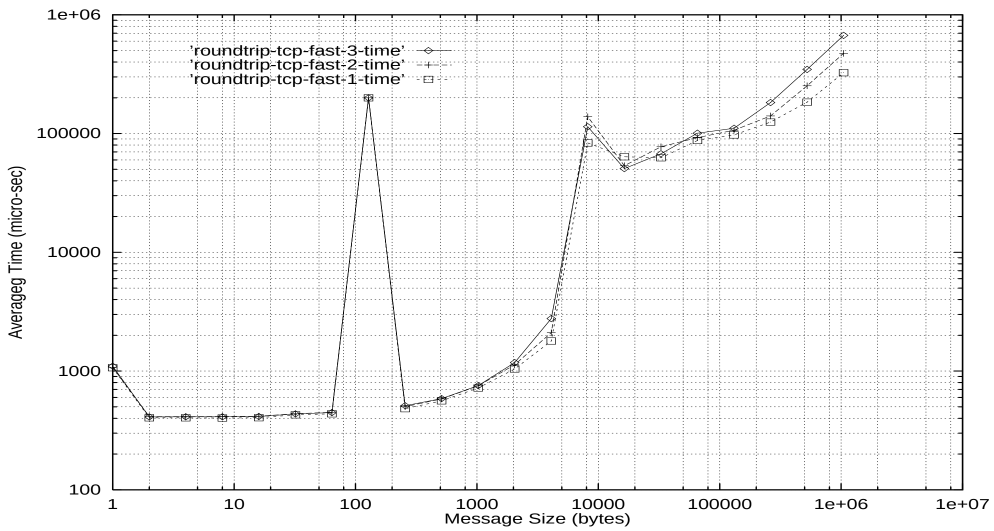

Figure 9: Roundtrip : TCP/IP : Fast Ethernet : Time per Iteration

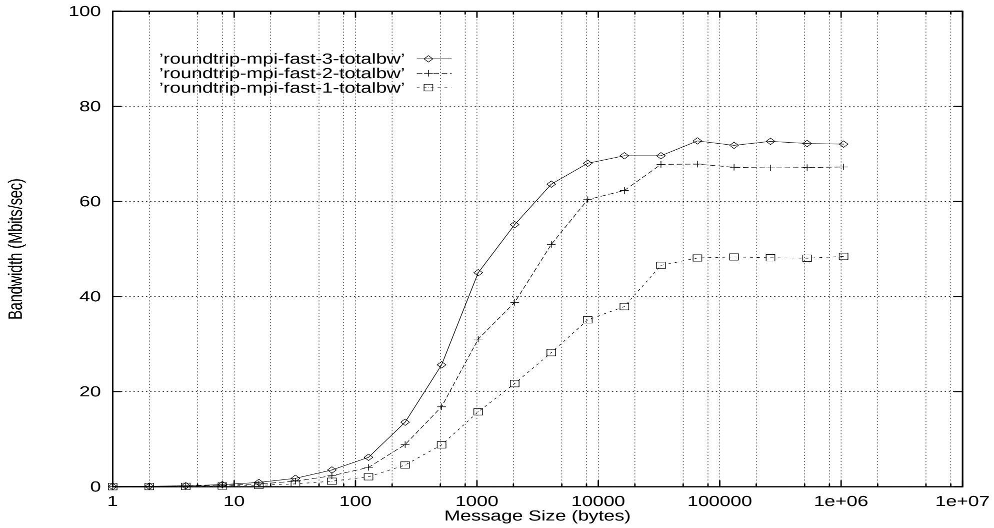

Figure 10: Roundtrip : MPI : Fast Ethernet : Total Bandwidth

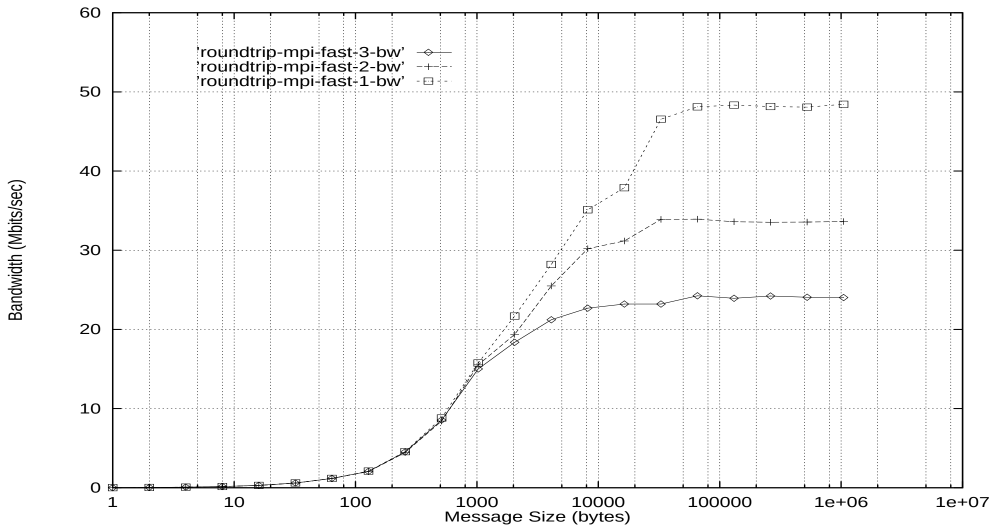

Figure 11: Roundtrip : MPI : Fast Ethernet : Bandwidth per Pair of Processes (Master-Slave)

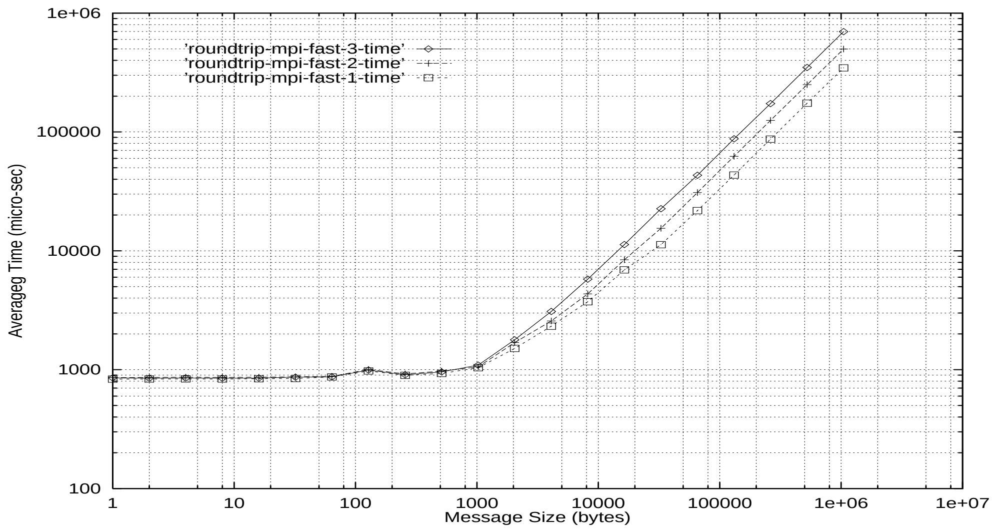

Figure 12: Roundtrip : MPI : Fast Ethernet : Time per Iteration

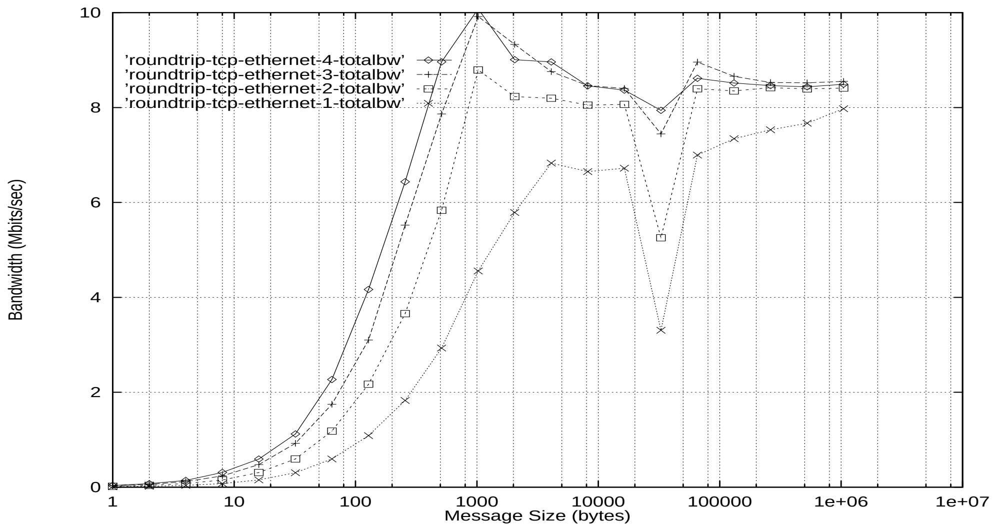

Figure 13: Roundtrip : TCP/IP : Ethernet : Total Bandwidth

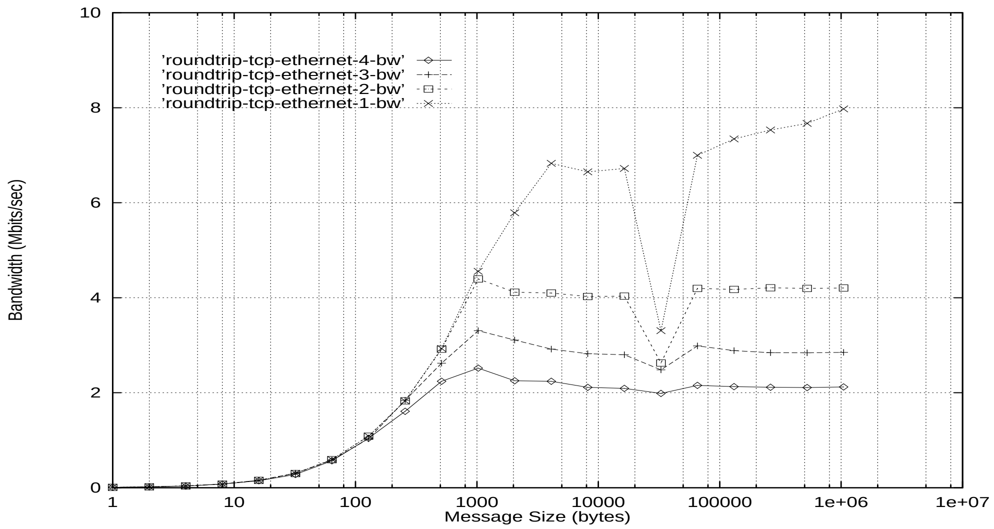

Figure 14: Roundtrip : TCP/IP : Ethernet : Bandwidth per Pair of Processes (Master-Slave)

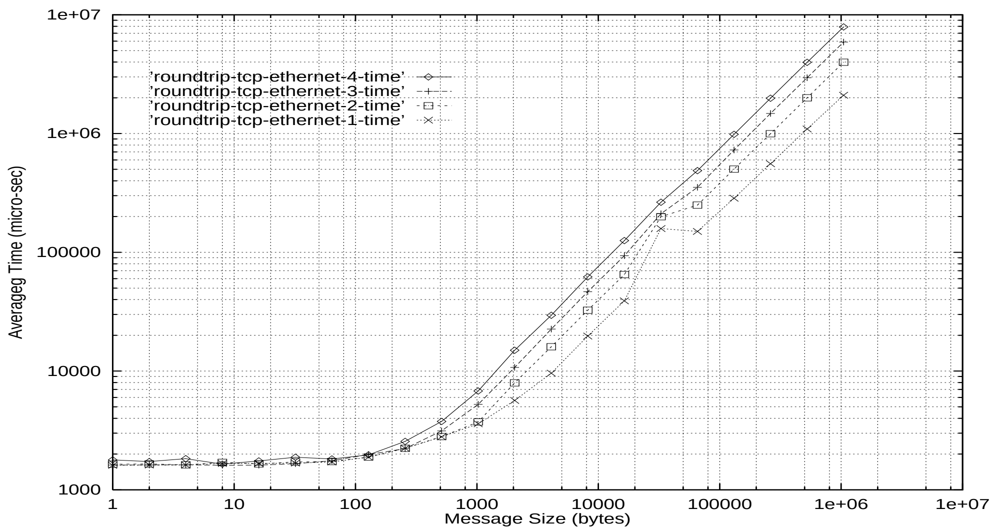

Figure 15: Roundtrip : TCP/IP : Ethernet : Time per Iteration

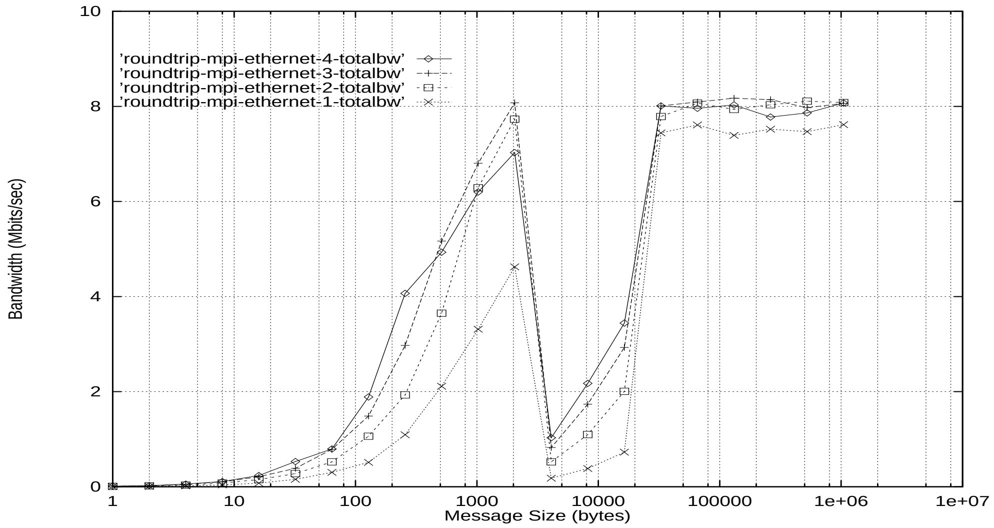

Figure 16: Roundtrip : MPI : Ethernet : Total Bandwidth

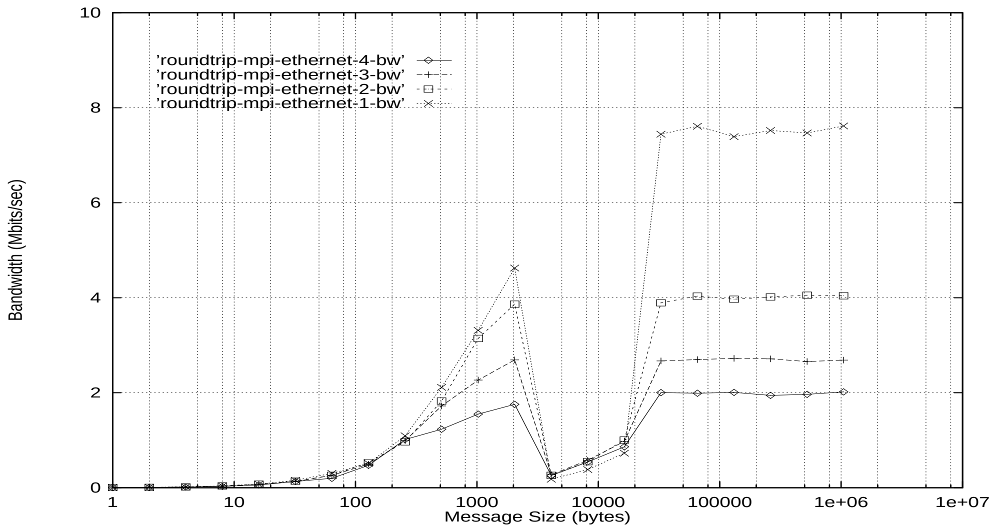

Figure 17: Roundtrip : MPI : Ethernet : Bandwidth per Pair of Processes (Master-Slave)

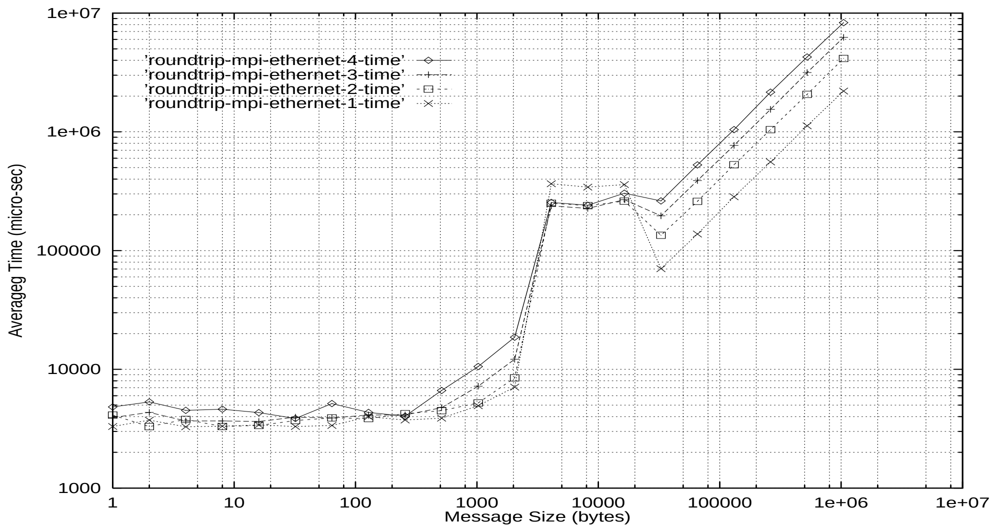

Figure 18: Roundtrip : MPI : Ethernet : Time per Iteration

4 DataHose

Datahose is designed to measure the raw bandwidth of the network. The results are a combination of the following factors:

- ability of a process to pump data onto the network and the network's capability to deliver it.
- network contention overhead.

Unlike roundtrip, datahose enforces no synchronization between the master and slave processes of a pair. The master process keeps writing data to the network and the slave reading from it without caring about each other's state. Packet flow control for datahose is thus handled by TCP. The time it takes for the slave to receive all data is the time for one datahose iteration. Many datahose iterations are done to get an accurate count.

All datahose process pairs synchronize before changing message sizes. Synchronization is done to prevent older messages from "spilling over" to the next stage. This step is the only time that a master and slave explicitly synchronize.

Datahose floods the network with messages. Since there is no flow control inherent in the test (except for that provided by TCP/IP), datahose with the TCP_NODELAY option crashed the FreeBSD Pentium machines repeatedly. This effect forced us to un-set TCP_NODELAY for the Fast Ethernet tests. TCP/IP is thus able to coalesce many small sized packets before sending them over, reducing the average time per packet. However, for large messages ($>$ 1 K) TCP_NODELAY has a minimal impact on performance.

Datahose was run on 1, 2, 3 and 4 pairs (up to 3 pairs on FreeBSD) of processors simultaneously to determine how the presence of other processes on the network influences bandwidth/process. Results for various combinations of network and protocol are presented in the following graphs.

The pseudo-code for datahose is as follows :

```
for (all interesting message sizes ) {

    set message_size
    synchronize (all master and slave processes)

    if (master){
         /* I am the master */

        /* do a large number of iterations */
        for(iteration_count=0;iteration_count<MAX_ITERATIONS;iteration_count++)
          {
            /* Just keep writing to the network */
            write(slave,message,message_size)
          }
    } else {
        /* I am the slave */

        start_timing;          /* Initialize timer */

        /* do a large number of iterations */
        for(iteration_count=0;iteration_count<MAX_ITERATIONS;iteration_count++)
          {
            /* Keep reading from the network whatever the master process has written */
            read (master,buffer,message_size)
          }

        stop_timing;           /* Stop timer */
   }        /* End if-else */
}           /* End for */
```

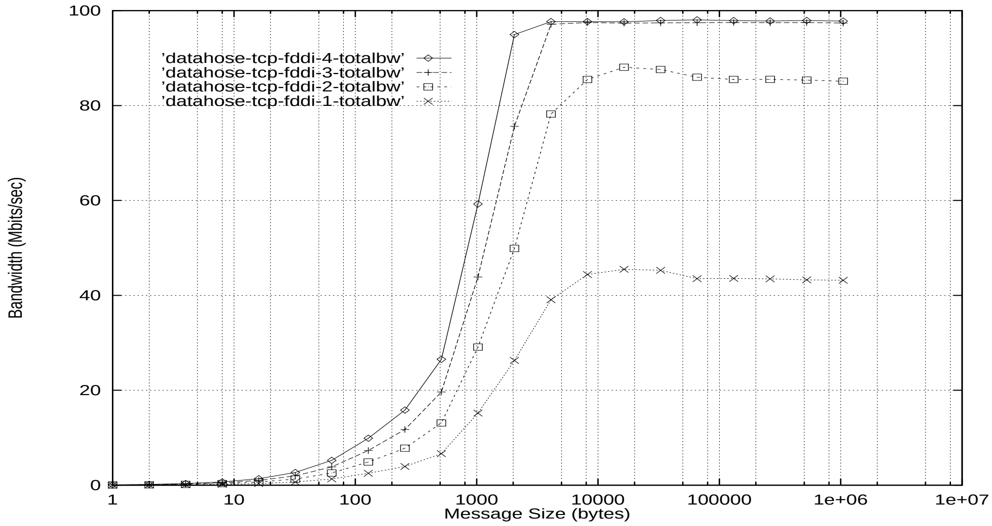

Figure 19: Datahose : TCP/IP : FDDI : Total Bandwidth

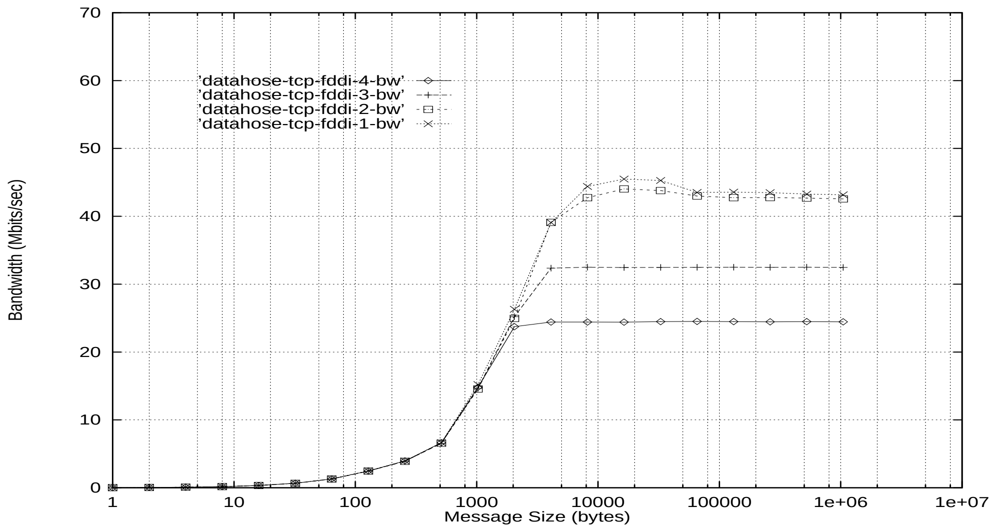

Figure 20: Datahose : TCP/IP : FDDI : Bandwidth per Pair of Processes (Master-Slave)

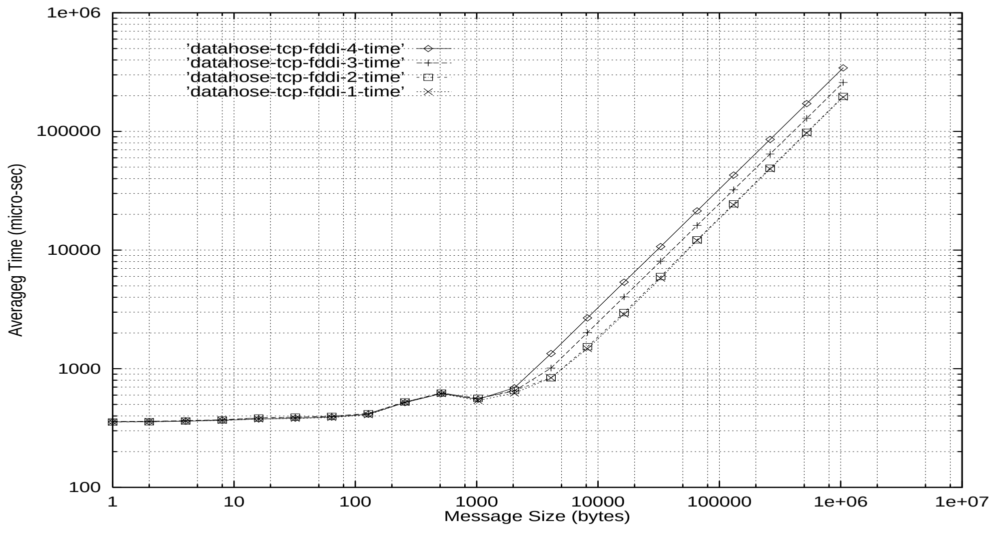

Figure 21: Datahose : TCP/IP : FDDI : Time per Iteration

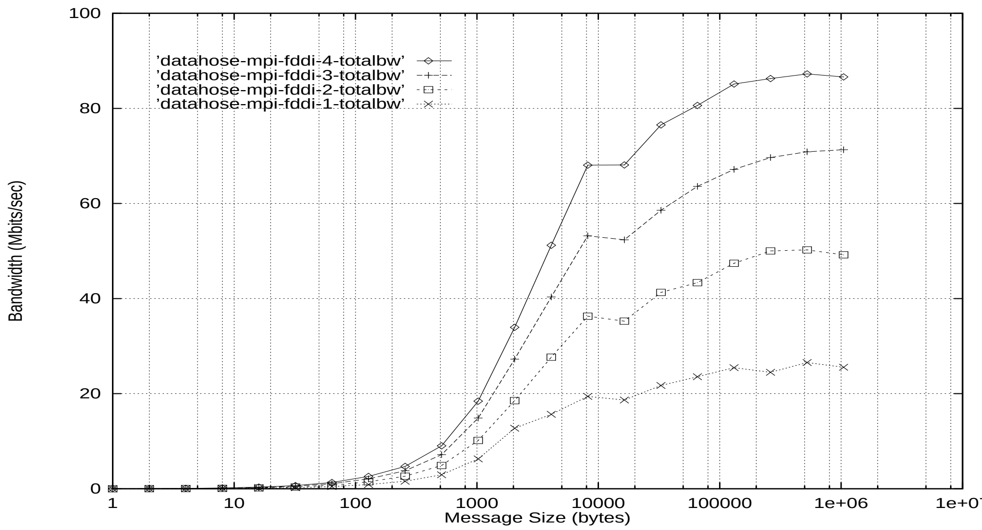

Figure 22: Datahose : MPI : FDDI : Total Bandwidth

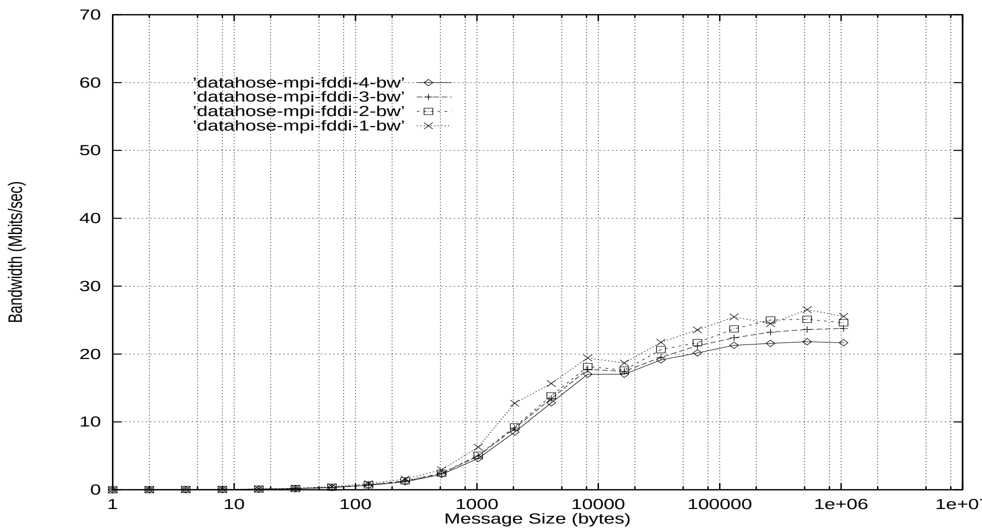

Figure 23: Datahose : MPI : FDDI : Bandwidth per Pair of Processes (Master-Slave)

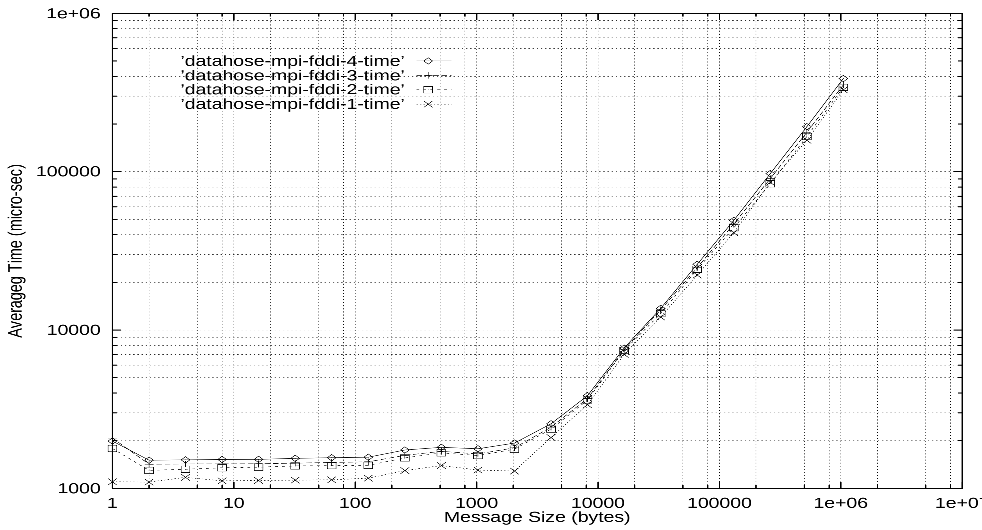

Figure 24: Datahose : MPI : FDDI : Time per Iteration

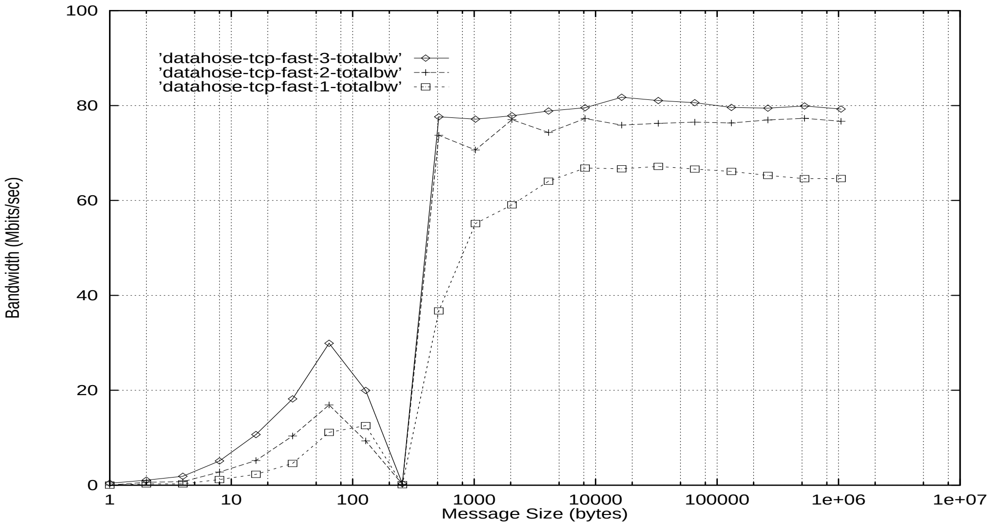

Figure 25: Datahose : TCP/IP : Fast Ethernet : Total Bandwidth

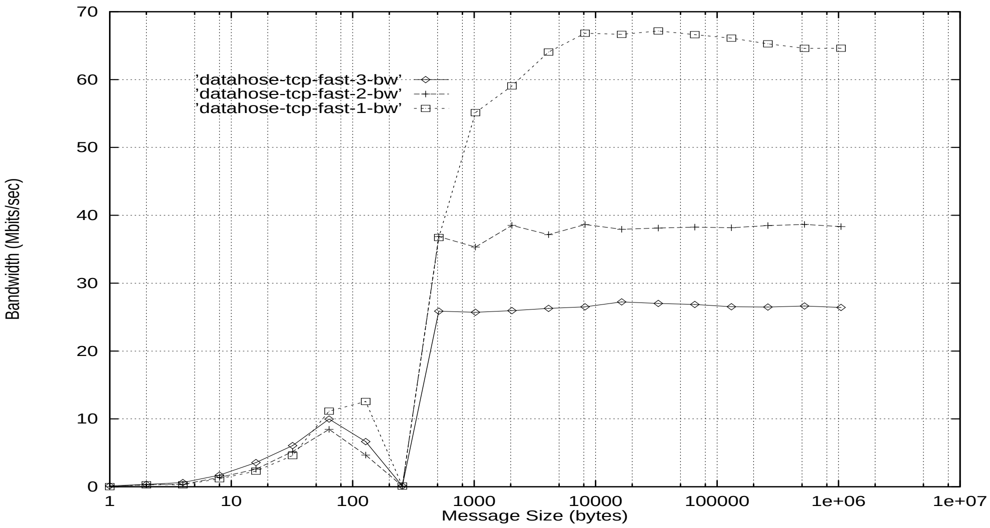

Figure 26: Datahose : TCP/IP : Fast Ethernet : Bandwidth per Pair of Processes (Master-Slave)

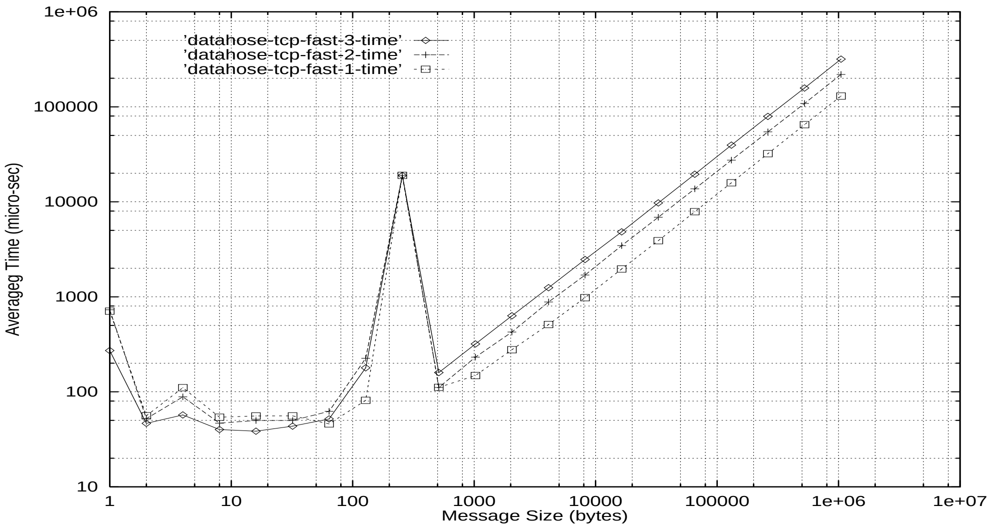

Figure 27: Datahose : TCP/IP : Fast Ethernet : Time per Iteration

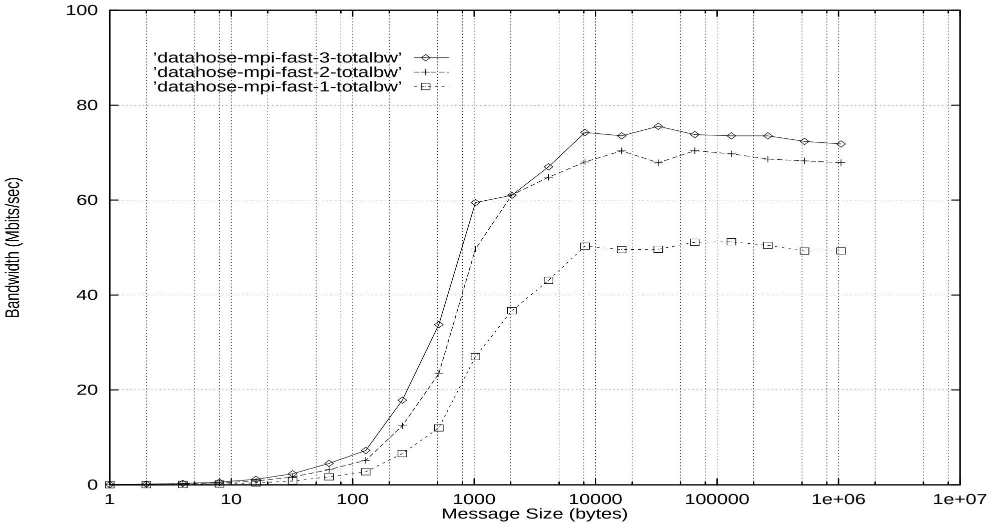

Figure 28: Datahose : MPI : Fast Ethernet : Total Bandwidth

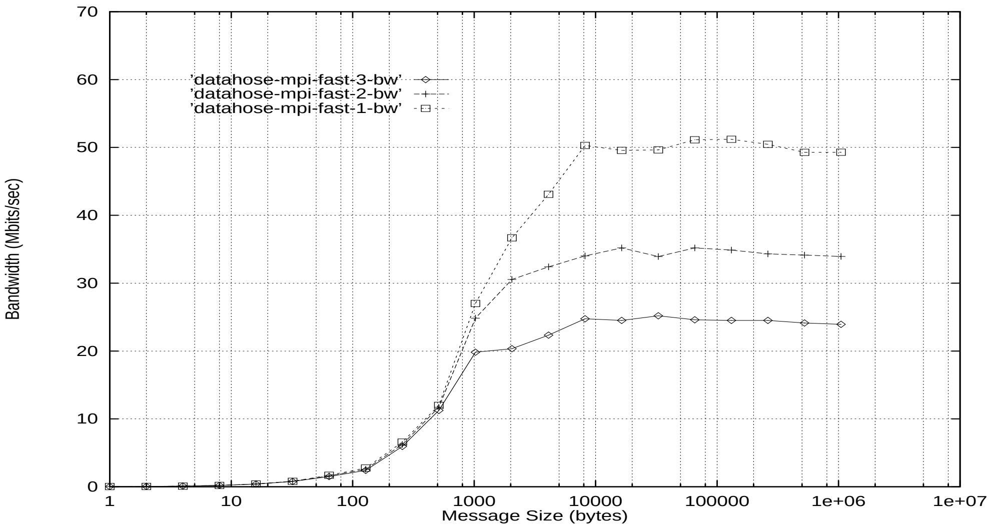

Figure 29: Datahose : MPI : Fast Ethernet : Bandwidth per Pair of Processes (Master-Slave)

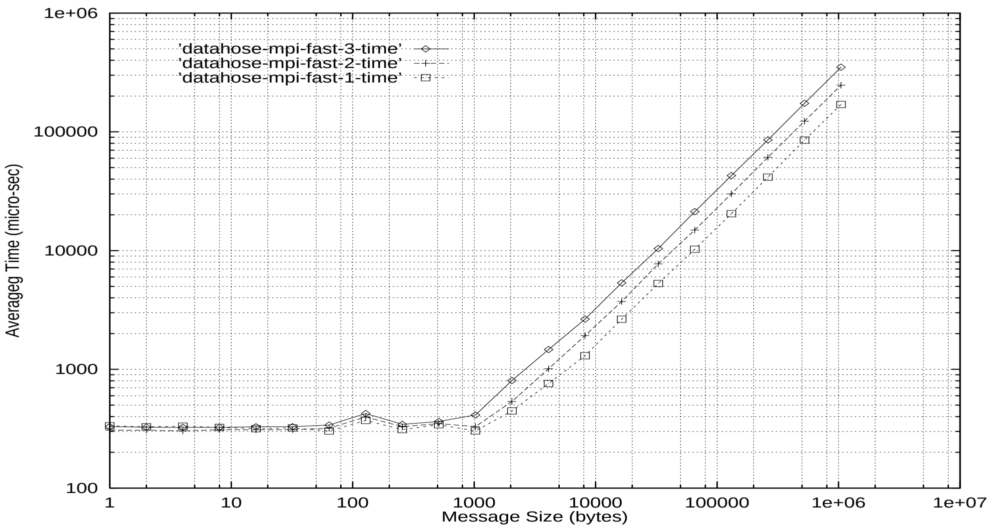

Figure 30: Datahose : MPI : Fast Ethernet : Time per Iteration

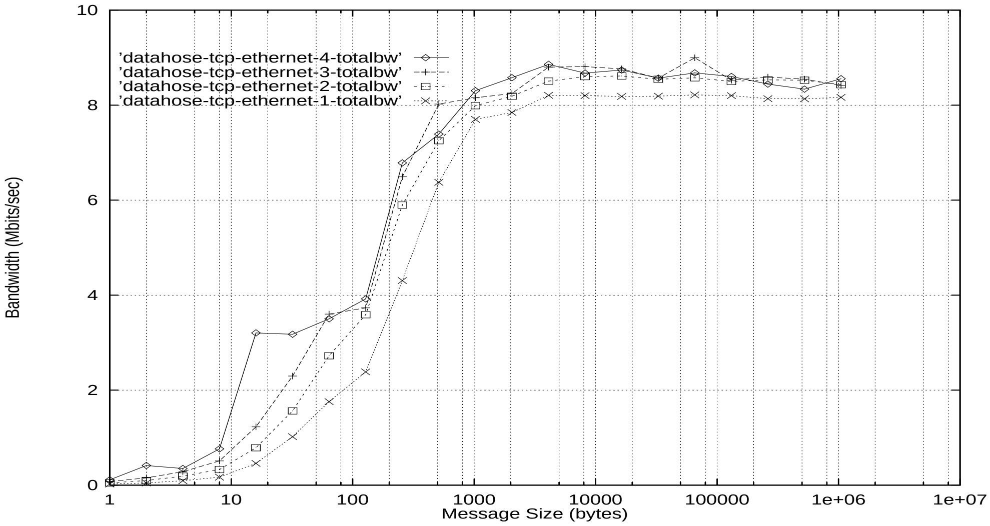

Figure 31: Datahose : TCP/IP : Ethernet : Total Bandwidth

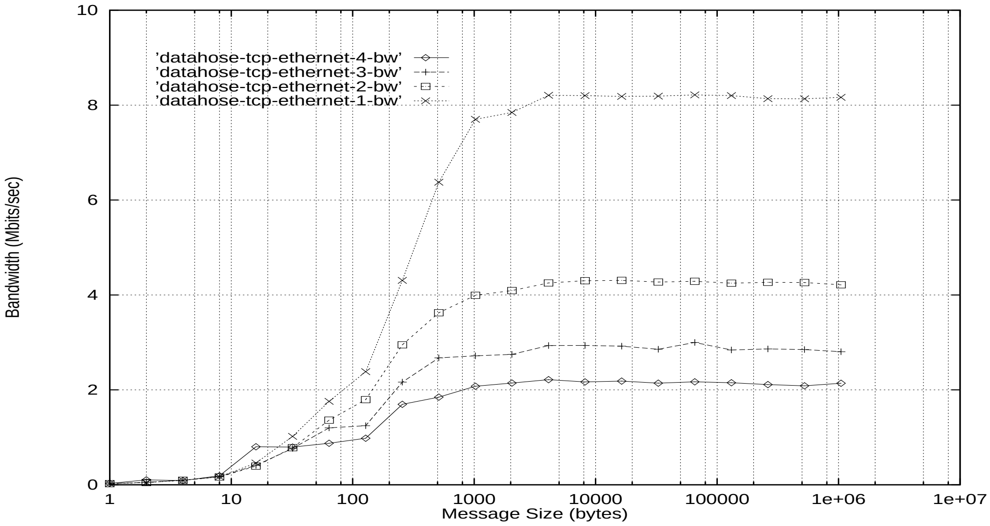

Figure 32: Datahose : TCP/IP : Ethernet : Bandwidth per Pair of Processes (Master-Slave)

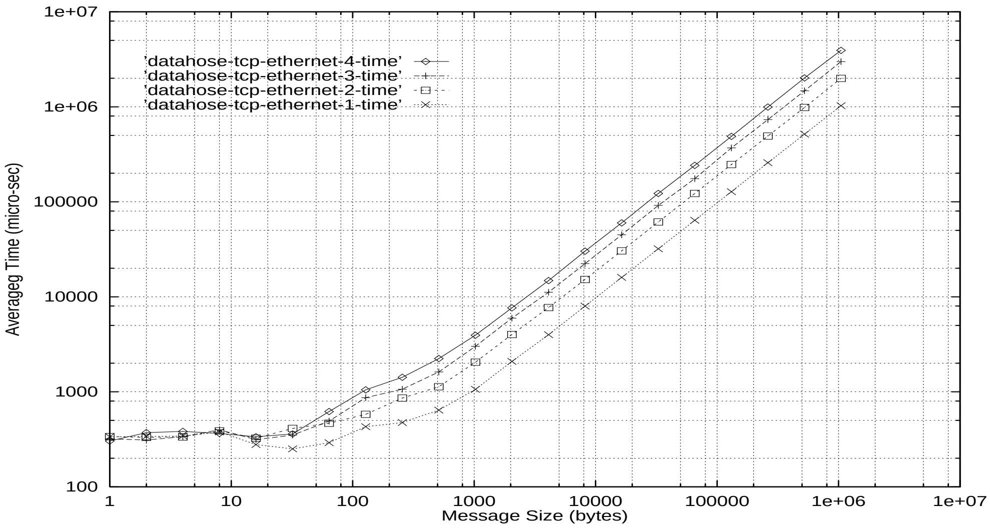

Figure 33: Datahose : TCP/IP : Ethernet : Time per Iteration

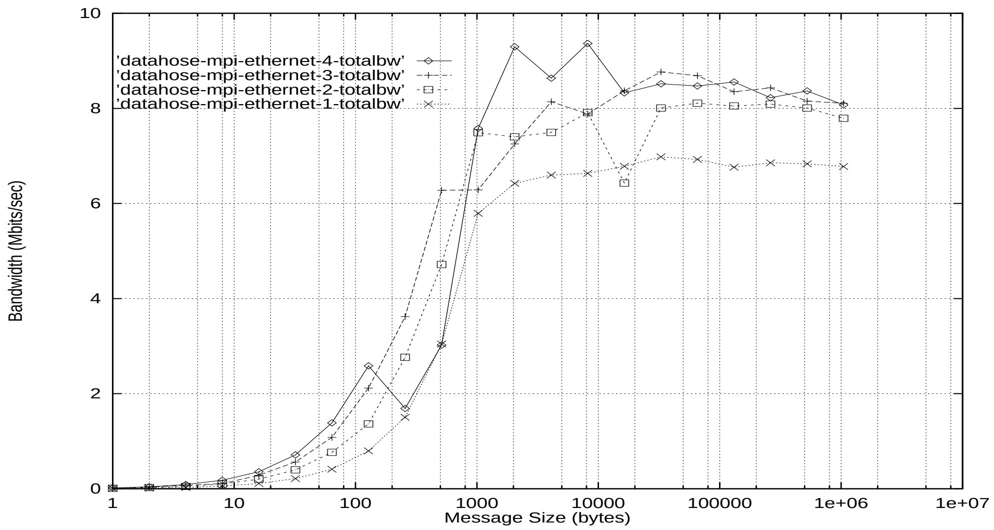

Figure 34: Datahose : MPI : Ethernet : Total Bandwidth

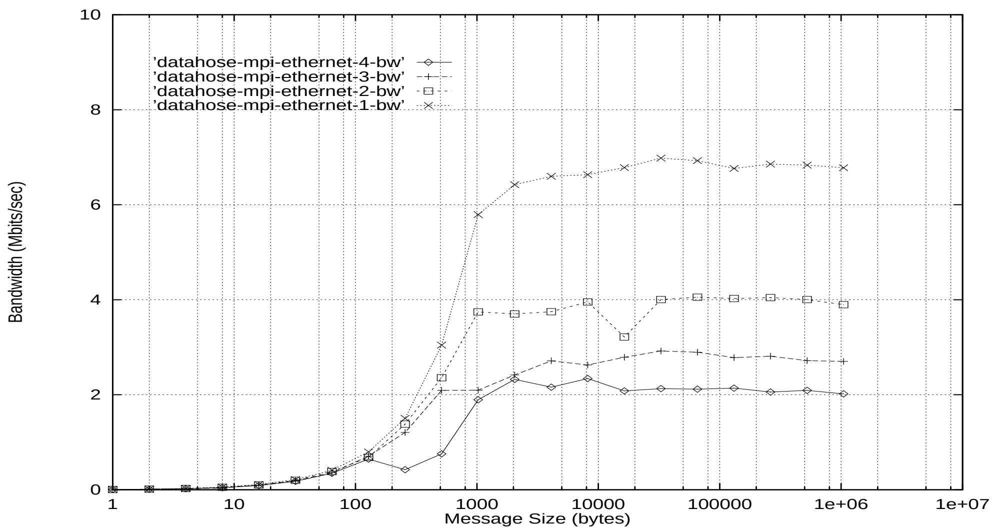

Figure 35: Datahose : MPI : Ethernet : Bandwidth per Pair of Processes (Master-Slave)

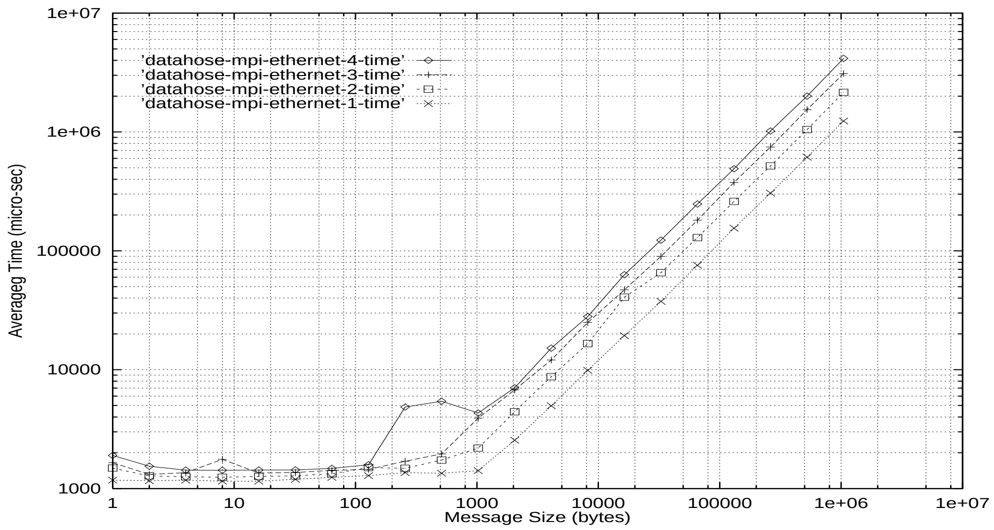

Figure 36: Datahose : MPI : Ethernet : Time per Iteration

5 Analysis of Results

Characterization and analysis of end-to-end network performance is an involved task. There are many factors that contribute to the final results: machine architecture, network protocol software, network card in the computer and its device driver, the external hub (concentrator for rings like FDDI) and so forth.

Many features that we observed in our results were predictable and intuitively appealing. They were:

- The time it took to process messages below a certain threshold (100 bytes for all the six combinations) was constant, probably because TCP_NODELAY prevents message coalescing. So all small messages are sent in a packet of their own and the small size makes the network latency a big factor (as opposed to bandwidth).
- FDDI and Fast Ethernet are an interesting comparison. Because of the different transmission mechanisms they performed differently on the same benchmarks. We analyse their TCP/IP performance here.
  - *Low network load*
    Fast Ethernet performed better (roundtrip: 51 Mbits/sec [Figure 7], datahose: 67 Mbits/sec [Figure 25]) than FDDI (roundtrip and datahose: 43 Mbits/sec [Figure 1] [Figure 19]) when the network was lightly loaded (1 pair of processes). One possible reason is that Ethernet (Fast or otherwise), based on CSMA/CD [4], does not wait for permission for each message. Thus a transmitting process, after sensing for absence of a carrier, wrote to the network hoping that a collision would not occur. For light-load conditions, collisions do not occur or were rare. Thus the high throughput. FDDI is based on token ring; each machine in FDDI has to wait until it can grab the token that is floating around on the ring. Only when a station has the token does it start transmission. This control mechanism slows down the transmission process even in a no-load situation.
  - *High network load*
    FDDI performed much better (roundtrip and datahose: 98 Mbits/sec [Figure 1] [Figure 19]) in a high load situation (3 pairs) than Fast Ethernet (roundtrip: 76 Mbits/sec [Figure 7], datahose: 81 Mbits/sec [Figure 25]). The reason again is the network hardware mechanism. As network load increased, so did the number of packet collisions in Ethernet. Thus even though the individual processes could put out packets faster onto the network, retries from collided packets degraded performance drastically. So the bandwidth per pair of processes dropped sharply for Fast Ethernet as load went up. FDDI is a collision-free network and hence this phenomenon did not affect its performance. We see almost 100% aggregate bandwidth utilization for FDDI.
- Although MPI was slower then TCP/IP, it was more consistent. TCP/IP had very abrupt spikes and dips in performance at various places. The corresponding MPI tests were generally much smoother.

- MPI was always (except for a few data points in Fast Ethernet, both roundtrip and datahose) slower than TCP/IP. The difference gives us the software overhead involved with using MPI. MPI reduced performance in the two faster networks (FDDI and Fast Ethernet) by about 10 Mbits/sec at the maximum load levels. However, MPI and TCP/IP were pretty close on Ethernet, giving the impression that the software penalty was masked by the much slower network.

- Of the three networks we looked at, FDDI seemed to be the most predictable in terms of performance. Abrupt discontinuities, which were present in the two Ethernets (Fast and Normal), did not plague FDDI.

While the results were generally consistent with expectations, the presence of repeatable performance spikes and dips in TCP/IP made the analysis interesting. We verified most of the TCP/IP discontinuities using the "ttcp" package. They seem to be consistent even across various machine types (for Ethernet). At present we do not have an insightful explanation for these anomalies. If you have some clue to the solution, we would appreciate hearing from you.

Finally, we would like to emphasize that our results are for specific implementations of the hardware and software standards. Other implementations may have different performance.

References

[1] Christos Papadopoulos and Gurudatta M. Parulkar, "Experimental Evaluation of SUNOS IPC and TCP/IP Protocol Implementation", IEEE/ACM Transactions on Networking, Vol. 1, No. 2, April 1993, Pages 199-216.

[2] Ralph Butler and Ewing Lusk, "User's Guide to the p4 Parallel Programming System", Version 1.3, Argonne National Laboratory, ANL-92/17, August 1993.

[3] D. W. Walker, "The design of a standard message passing interface for distributed memory concurrent computers", ParComp Vol. 20, No. 4, 1994. Pages 657-673.

[4] Andrew S. Tanenbaum, "Computer Networks", 2nd Edition, Prentice Hall, 1988.