Measuring TCP bandwidth on top of a Gigabit and Myrinet network

Juan J. Costa, Javier Bueno Hedo, Xavier Martorell and Toni Cortes
{jcosta,jbueno,xavim,toni}@ac.upc.edu

December 17, 2009

Abstract

In this article we measure the bandwidth achieved by the TCP protocol on top of a Gigabit and Myrinet network. We have created a synthetic benchmark, consisting on a server and a client that both produce and consume data, to measure bandwidth.

Four different versions of this benchmark are evaluated: (a) a simple one-threaded version; (b) a version that can consume data if the producer is unable to produce data; (c) a version with the Nagle algorithm disabled; and (d) a version where producers and consumers are threads on their owns.

The results shows that on both networks: (1) it is necessary to drain the network to avoid deadlocks; (2) the Nagle algorithm is useful and gets smaller variation in results; and (3) adding an extra thread for reading reduces the maximum bandwidth achievable and it gets better bandwidth for bigger messages and worse for smaller ones when a great number of messages are sent.

1 Synthetic Benchmark

1.1 Original version

The benchmark is a well-known client-server code. Server and clients are producer-consumer threads. The server has three parameters:

messages Number of messages to send.

nodes Number of nodes where messages will be sent.

size Size in bytes of each message sent.

The server algorithm is as follows: (1) it sends a fixed number of requests (*messages*) of fixed size (*size*) to all nodes (*nodes*); and afterwards (2) it waits for an acknowledgment per each request. The server code can be seen in Figure 1.

```
for (j=0; j<messages; j++)
     for (th=0; th<nodes; th++)
       sendto (th, WORK)   //Send WORK request

for (j=0; j<messages; j++)
     for (th=0; th<nodes; th++)
       recv (ACK)
```

Figure 1: Synthetic benchmark Server algorithm.

```
while (true)
{
   recv (&WORK)
   sendto (WORK.from, WORK)   //Retransmit WORK request
}
```

Figure 2: Synthetic benchmark Client algorithm.

The client (which code is shown in Figure 2) just connects to the server and waits forever for requests arrival. When any request arrives it is just retransmitted to the server.

1.2 Synthetic Benchmark with draining

The original version sends a lot of data without receiving any data. This behavior can fill the internal buffers and hang the application.

This version tries to solve this problem. It detects the case when the buffers are full and any sending operation would block, and it drains a request from each connected socket, before trying to send data again.

1.3 Synthetic Benchmark with an extra draining thread

In this version an extra thread is used to continually drain the sockets. A shared request queue is used between the server thread and the draining thread. The access to this queue is protected through the use of atomic counters and semaphores.

2 Bandwidth calculation

To calculate the bandwidth, a timer is set at the beginning and end of the server algorithm, and the bandwidth is calculated as the number of bytes transmitted in the middle. In order to get statistically significant values when measuring the time it takes to execute, the algorithm is repeated a fixed number of times (MAX).

The final bandwidth formula is shown in equation 1.

$$bandwidth = \frac{2 \times MAX \times messages \times nodes \times size}{time} \tag{1}$$

3 Testbed

All our benchmarks have been executed on top of a cluster of Power PC 970MP processors at 2.3 GHz and 4Gb of memory. Nodes are connected through two interfaces: a Full duplex Gigabit ethernet network and a Myrinet network.

4 Gigabit Performance Evaluation

4.1 Synthetic Benchmark

Figure 3 shows the bandwidth achieved by the synthetic benchmark when sending messages of 4096 bytes. It shows that the bandwidth increases steadily when the number of messages sent is between 1 and 32. From there on, the slope decreases till it stabilizes in a maximum value of 170MB/s when sending from 256 to 512 messages. But at that point the benchmark hangs and it does not continue any further.

The problem is that the server sends too much data without acknowledging it. The clients answer each request with a new one. This fills the internal receiving buffers at server with pending data from clients. The clients are informed (by the TCP algorithm) that they can not send any more data and they start to fill their own receiving buffers till they are full. Finally the server is informed that the clients can not receive any further data and blocks any send operation, arriving to a deadlock situation.

4.2 Synthetic benchmark with draining

The bandwidth achieved when draining the sockets to avoid the deadlock in the synthetic benchmark is evaluated in this section. Two versions are used, one with the Nagle algorithm enabled, and another without it. The results are shown in Figures 4 and 5.

All figures shows bandwidth in function of the number of messages sent, and a curve is plotted for each message size used.

With Nagle algorithm

Figure 4a shows the bandwidth when sending messages between 2 nodes. There you can appreciate that *(1)* the maximum bandwidth achievable in-

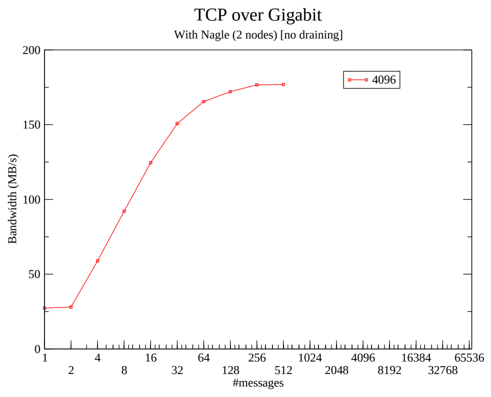


Figure 3: Gigabit bandwidth for the original Synthetic benchmark when sending messages of 4096 bytes between 2 nodes.

creases with the message size; and *(2)* there are two different behaviours depending on the message size:

1. If size is smaller than 2048 bytes then the bandwidth increases slowly till it stabilizes between 70–120MB/s. With the smaller sizes (256 and 512) there are two peaks, at 16 and 8 messagges respectively, with higher bandwidth than expected (I am not confident about that but these fit misteriously in a page of 4096 bytes. Is this the Nagle buffer?). Messages of 1024 bytes also show some peaks and valleys at 4, 16 and 128 messages that I am not able to explain.

2. If size is greater than 2048 bytes then the bandwidth increases quickly and finally stabilizes (with a maximum of 170MB/s).

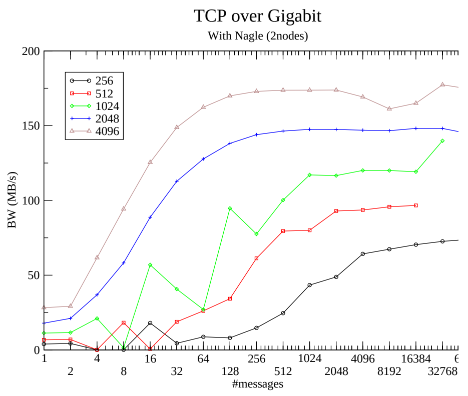
When more than two nodes are used, a new behaviour appears. For small message sizes (256–1024 bytes) the bandwidth increases slowly till it stabilizes around the 100MB/s. By contrast, bigger messages increases the bandwidth quickly till a maximum value is achieved, and then it drops and behaves like small messages. This behaviour can be seen in Figures 4b, 4c and 4d.

Without Nagle algorithm

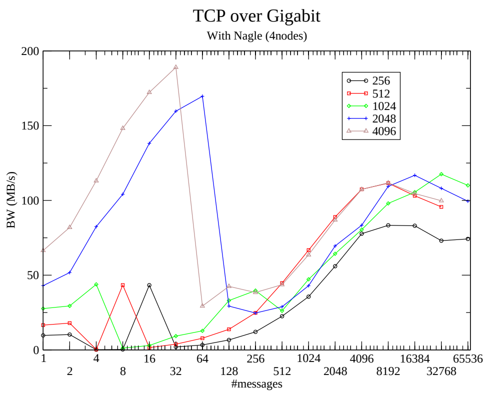
In this section the bandwidth for some nodes sending messages from 256 to 4096 bytes with the Nagle algorithm disabled are shown. Their behaviour is similar to the behaviour explained before with the Nagle Algorithm enabled.

Figure 5a shows three behaviours depending on the message size:

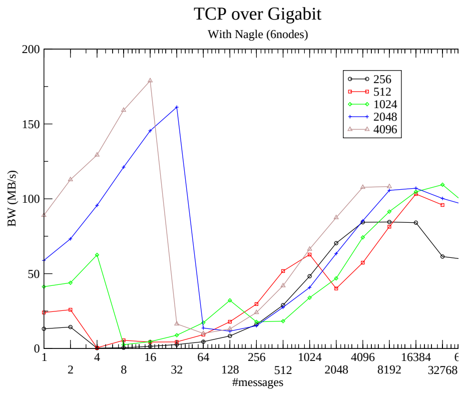
1. If size is smaller than 1024 bytes, the bandwidth makes a small hop from 4 to 32 nodes and from there on it keeps increasing slowly till it stabilizes at 40–50MB/s.



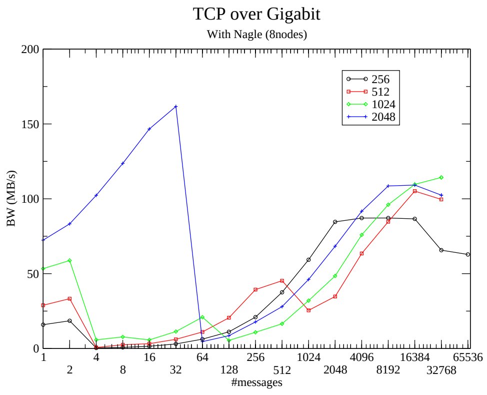
(a) 2 nodes.



(b) 4 nodes.

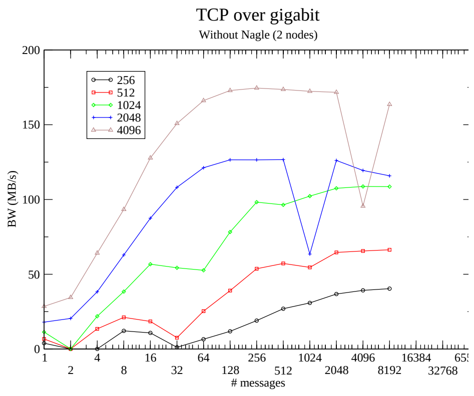


(c) 6 nodes.

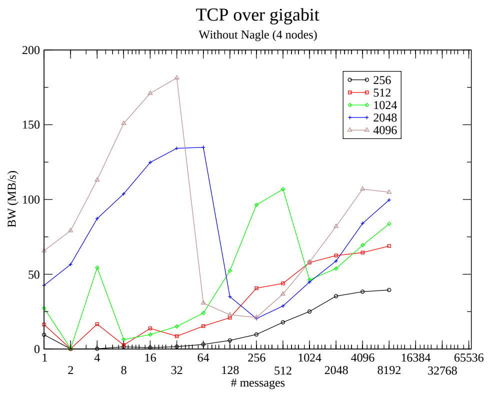


(d) 8 nodes.

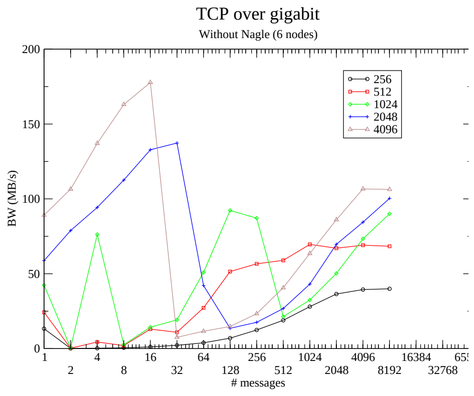
Figure 4: Gigabit bandwidth when the sender side drains the sockets and the Nagle algorithm is enabled.



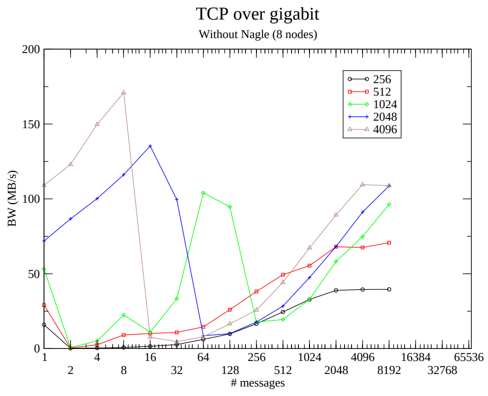
(a) 2 nodes.



(b) 4 nodes.



(c) 6 nodes.



(d) 8 nodes.

Figure 5: Gigabit bandwidth when the sender side drains the socket and the Nagle algorithm is disabled.

2. If size is 1024 bytes, the bandwidth increases quickly to 50 MB/s at 16 messages and stops growing till 64 messages where it quickly increases again till 256 messages where it stabilizes around 100 MB/s.

3. If size is greater than 1024 bytes, the bandwidth quickly increases till the 128 messages where the maximum bandwidth is achieved (170MB/s), and then it remains stable. There are a couple of glitches when sending 1024 messages of 2048 bytes, and 4096 messages of 4096 bytes.

When more than two nodes are used, the behaviour change substantially in function of message size.

1. If size is smaller than 1024 bytes, the bandwidth increases steadily till a maximum of 40–60 MB/s depending on the message size.

2. If size is 1024 bytes, the bandwidth follows the behaviour of smaller messages, but there is a point where it increases the growing rate

quickly till it arrives a maximum bandwidth of 100MB/s at 512 messages and drops to less than 50MB/s afterwards, increasing steadily from there. It also has a high bandwidth transfer with 4 messages (50MB/s).

3. If size is greater than 1024 bytes, the bandwidth quickly increases till the maximum bandwidth is achieved (at 64–16 messages depending on the number of the nodes), and then it drops dramatically below 50MB/s and it keeps growing as the small messages.

5 Extra draining thread

Figure 6a shows the bandwidth for two nodes sending messages of different sizes from 256 to 8192 bytes. As expected the maximum achievable bandwidth depends on the message size. The behaviour is similar to previous work, but in this case, the achieved bandwidth is smaller for the same message size. With a message size of 4096 bytes a maximum bandwidth of 150MB/s is achieved.

When more than 2 nodes are used, a similar behaviour as before is obtained. The only difference is that after the drop in bandwidth greater messages tend to have a higher bandwidth of 150MB/s in contrast to the 100MB/s without the extra thread.

6 Myrinet Performance Evaluation

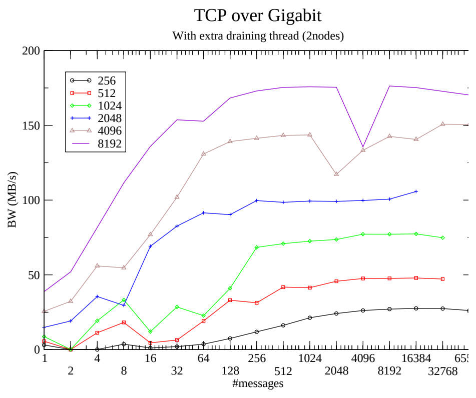
6.1 Synthetic benchmark

Figure 7a shows then bandwidth for the synthetic benchmark when executed on top of a Myrinet network. The sending of 1–2 messages have a high bandwidth (10–50MB/s depending on the message size).

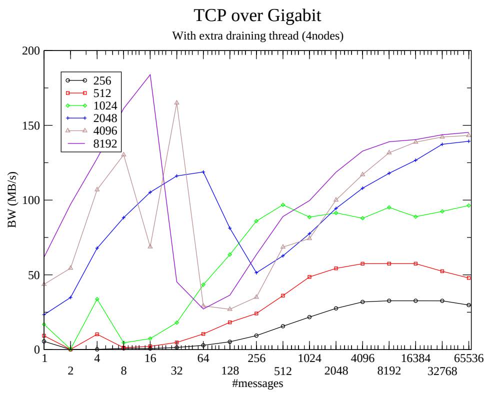
For small messages of 256 bytes, it has a linear increase in bandwidth till the 32000 messages where it achieves the maximum bandwidth of 100MB/s and falls a little bit.

Messages of 512 bytes have a strange behaviour. It has a 75MB/s peak at 32 messages, a valley of a few MB/s afterwards, and back again to 100MB/s at 128 messages. After this the bandwidth keeps increasing to a maximum bandwidth of 175MB/s and falling again, to finally stabilize at 100MB/s.

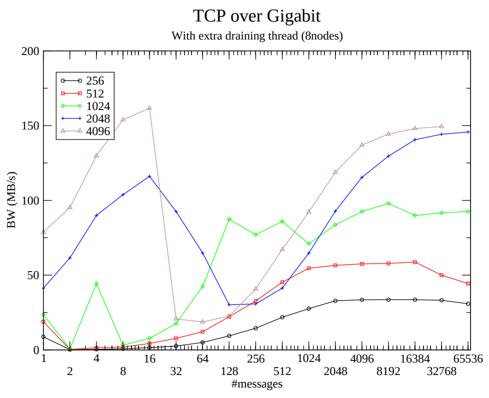
In contrast, the messages of 1024 bytes have a behaviour similar to the smaller messages. It starts with a low bandwidth and has an exponential growth till 225MB/s (at 8192 messages). After this, it falls and recovers slightly to 150MB/s. Bigger messages also have a quick increase in bandwidth till the maximum bandwidth is achieved (around the 350MB/s), and then they fall (100–150MB/s) and recover a little bit.



(a) 2 nodes.

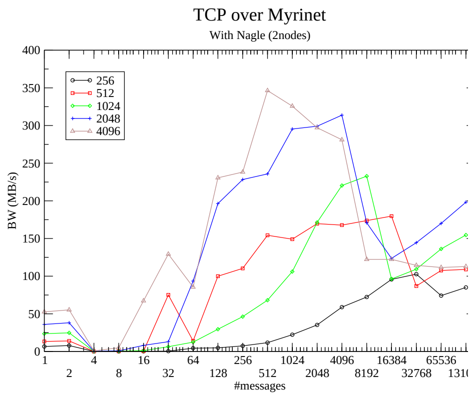


(b) 4 nodes.

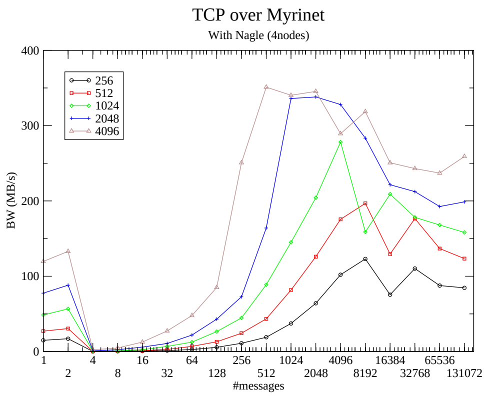


(c) 8 nodes.

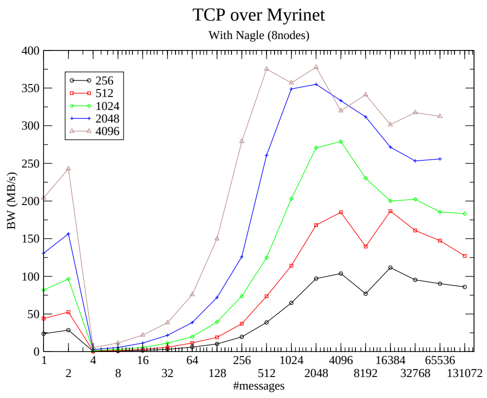
Figure 6: Gigabit bandwidth when an extra thread for draining the sockets is used.



(a) 2 nodes.



(b) 4 nodes.



(c) 8 nodes.

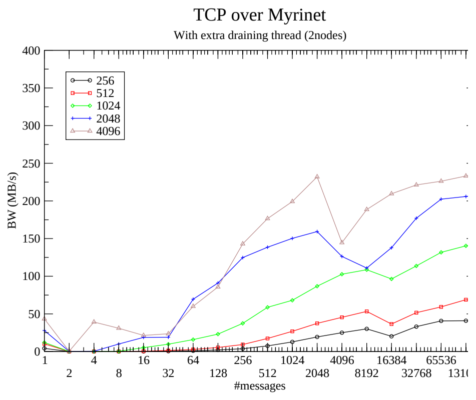
Figure 7: TCP over Myrinet bandwidth for the synthetic benchmark when the server side drains the sockets.

When the number of nodes is increased (Figures 7b and 7c) the curves get clearer. Each curve drawing a message size steps on the smaller size with just a few overlaps. All of them have exponential increase of bandwidth with the number of messages sent, till they arrive to a maximum bandwidth (350MB/s), fall down a little bit and stabilizes.

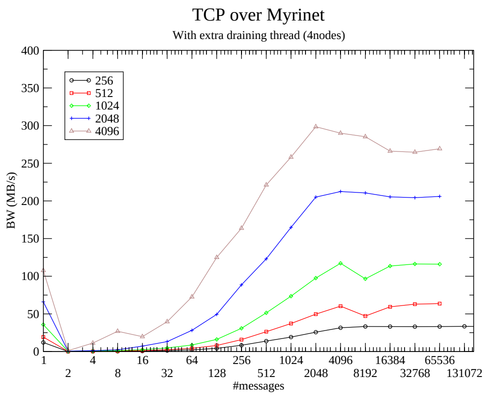
6.2 Synthetic benchmark with an extra thread draining

The behavior when adding an extra thread is similar to what we obtained in the gigabit network. When using 2 nodes, the bandwith is similar for all message sizes. In the case of messages of 4096 bytes, it increases steadily till a maximum of 250MB/s and falls a little bit, stabilizing again at 250MB/s.

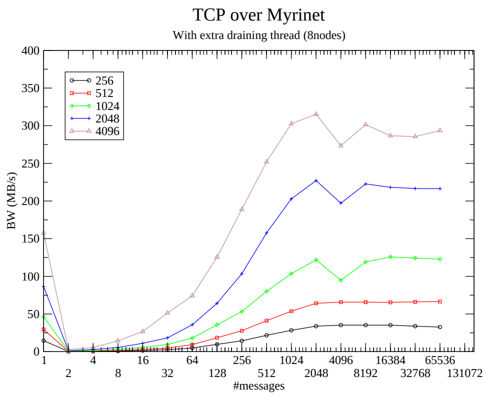
When more nodes are used, the maximum bandwith increases a little bit more than before. The behavior is that the bandwidth increases till the maximum bandwidth and then stabilizes there. In the case of 4096 bytes



(a) 2 nodes.



(b) 4 nodes.



(c) 8 nodes.

Figure 8: TCP over Myrinet bandwidth when an extra thread for draining is used.

the maximum bandwitdth achieved is 300MB/s.

7 Conclusion

We have tested the synthetic benchmark on top of a Gigabit and Myrinet network with different versions and obtained the following conclusions:

- We need to drain the communication socket in order to avoid deadlocks.
- When using the Nagle algorithm the bandwidth for different number of nodes has less variance and it tends to an average bandwidth of 100MB/s.
- When the Nagle algorithm is not used, the bandwidth has more variance and big messages tend to have an average of 100MB/s while the

smaller ones remain at 50MB/s.

- If an extra thread is added, then the total bandwidth is reduced, but for a great number of messages the big messages tend to have an average bandwidth of 150MB/s while the smaller ones remain at 50MB/s.