PROCEEDINGS OF SCIENCE

Using UDT for High Energy Physics Data Transport

Barney Garrett[1]
The University of Edinburgh
James Clerk Maxwell Building, Mayfield Road, Edinburgh. EH54 6TD. UK
E-mail: `barney.garrett@ed.ac.uk`

Brian Davies
Lancaster University
Department of Physics, Lancaster. LA1 4YB. UK
E-mail: `b.g.davies@lancaster.ac.uk`

eScience applications, in particular High Energy Physics, often involve large amounts of data and/or computing and often require secure resource sharing across organizational boundaries, and are thus not easily handled by today's networking infrastructures. By utilising the switched lightpath connections provided by the UKLight network it has been possible to research the use of alternate protocols for data transport. While the HEP projects make use of a number of middleware solutions for data storage and transport, they all rely on GridFTP for WAN transport. The GridFTP protocol runs over TCP as the layer 3 protocol by default, however with the latest released of the Globus toolkit it is possible to utilise alternate protocols at the layer 3 level. One of the alternatives is a reliable version of UDP called UDT. This report presents the results of the tests measuring the performance of single-threaded file transfers using GridFTP running over both TCP and the UDT protocol.

Lighting the Blue Touchpaper for UK e-Science - Closing Conference of ESLEA Project
The George Hotel, Edinburgh, UK
26-28 March, 2007

[1] Speaker

PoS(ESLEA)006

1. Introduction

TCP uses what it calls the congestion window to determine how many packets can be sent at one time. The maximum congestion window is related to the amount of buffer space that the kernel allocates for each socket. If the buffers are too small for the network connection the TCP congestion window will never fully open up resulting in never reaching the maximum potential of the network connection. In Long Fat Networks (LFN), such as UKLight the default kernel settings within Linux are inadequate. The Linux kernel can be tuned[1] for LFN's with the Bandwidth Delay Product (BDP) being used to give an appropriate buffer size. This is calculated using:

```
BDP (bytes) = Bandwidth (bytes) * RTT (seconds)
```

The socket receive buffer space is shared between the application and kernel. TCP maintains part of the buffer as the TCP window, this is the size of the receive window advertised to the other end. The rest of the space is used as the "application" buffer, used to isolate the network from scheduling and application latencies. By default this overhead is a quarter of the buffer space that the kernel is configured to use.

The txqueuelen is another buffer in the kernel stack that can affect performance; especially of TCP transfers. When the system is sending out too much data from the IP layer to the Ethernet device driver layer, this buffer, txqueuelen, may overflow. In TCP, this has the effect of a congestion event causing the congestion window to halve.

As can be seen from this description achieving the maximum throughput on a LFN requires an amount of work by the administrator of each end host involved, and if multiple links with different characteristics are involved the settings used may be suboptimal. This work can be avoided by creating multiple simultaneous connections however this can lead to other issues including file fragmentation if using multiple streams for a single bulk data move.

The aim of these experiments is to provide an alternative bulk transport mechanism that can be dropped into a running environment and provide high speed transport without requiring significant tuning to achieve maximum performance.

1.1 UDT

UDT[2] is an application level data transport protocol which uses UDP to transfer bulk data and it has its own reliability control and congestion control mechanism. It is not only for private or QoS-enabled links, but also for shared networks since it is TCP friendly.

1.2 GridFTP and Globus XIO

The Globus toolkit[3] provides the data management component GridFTP and a common runtime component XIO. GridFTP is a high-performance, secure, reliable data transfer protocol base on FTP that is optimized for high-bandwidth wide-area networks. Globus XIO is an extensible input/output library written in C for the Globus Toolkit. It provides a single API that supports multiple protocols, with these protocol implementations encapsulated as drivers. XIO drivers can be written as atomic units and stacked on top of one another. The latest Globus implementation of the GridFTP server implements XIO which allowed the replacement of TCP with UDT as the layer 3 protocol for the purposes of these tests.

2. System Component Testing

The hardware configuration for these tests is two Supermicro X6DHE-G2's with dual Xeon 3.2Ghz dual core CPUs, 2GB ECC DDR RAM, LSI MegaRAID SATA 300-8x RAID controller. The disks are six Western Digital Raptor 74GB SATA disks connected to the RAID controller and a seventh Western Digital 80GB SATA disk as the system Disk. The RAID controller is seated in PCI-X slot 1 so that it is on a separate PCI bus interface to the Gigabit LAN connection and thus not competing for bandwidth on the bus.

2.1 Disk Subsystem

The disk subsystems were tested using IOZone[4] which is a file system benchmark tool. It generates and measures a variety of file operations including write, rewrite, read and reread.

For the results to have the maximum relevance to the production systems used by the GridPP sites we configured the disks for RAID 5 and used an ext3 file system. IOZone tests were performed using the command:

```
iozone –a –g 16G –i 0 –i 1
```

These showed that once the file size exceeded the cache size write speeds in the order of 125MB/s [Figure 1] and read speeds of approximately 155MB/s [Figure 2] are possible, which is just fast enough so as not to be the bottleneck on a 1Gb/s network connection.

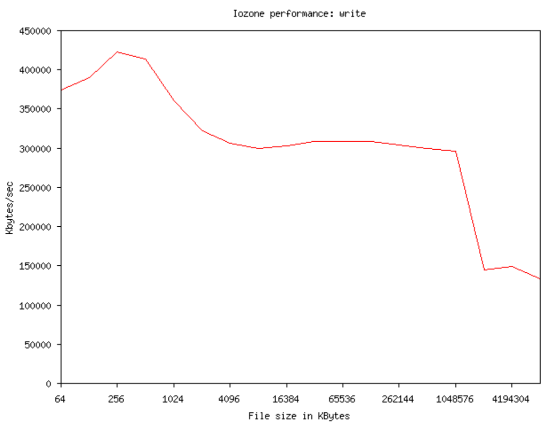


Figure 1

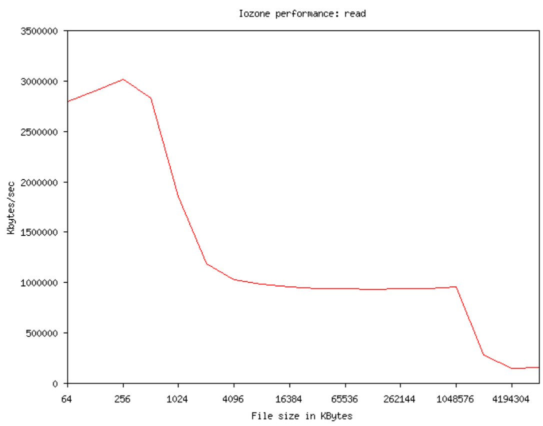


Figure 2

2.2 Networking

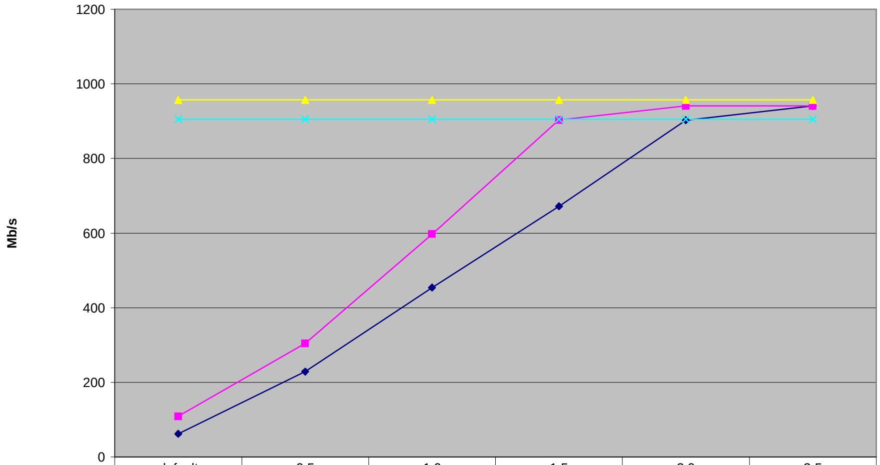
The raw network link was tested using Iperf [5] for the TCP and UDP protocols and XIOperf [6] for the UDT protocol. XIOperf is a tool similar to Iperf, and measures the performance characteristics of a transfer and reports them to the user. It is written on top of Globus XIO so it has all of the dynamically loadable transport driver functionality which allowed the testing of the UDT driver that will be used in GridFTP in later testing. These tests were carried out to profile the UKLight connection and set the baseline for later comparisons.

For the TCP tests tuning was carried out to determine what was required to achieve maximum throughput for a single transfer using only a single stream. Using the calculation for BDP:

```
BDP (bytes) = Bandwidth (bytes) * RTT (seconds)
BDP = 134217728 * 0.01
BDP = 1342177.28
```

Multiple test runs were made using multiples of the BDP to determine the effect on the throughput. TCP transfers were tested both with the buffer sizes as calculated above and then also taking into account the overhead that is reserved for the application.

Figure 3 shows the results of these tests. It can be seen that without any tuning of the kernel UDP, which features no congestion control and is not a reliable protocol, is capable of 957Mb/s which is about 97% of the available bandwidth, UDT is slightly slower at 905Mb/s which is about 92% of its available bandwidth, and finally TCP only manages to achieve 62Mb/s or about 6%. It is only after the buffer sizes have been increased to over 1.5 times the BDP corrected for overheads that TCP reaches its maximum performance of 941Mb/s, almost matching the performance of UDP.



	default	0.5	1.0	1.5	2.0	2.5
TCP	62	229	454	672	902	941
TCP+SF	108	304	597	903	941	941
UDP	957	957	957	957	957	957
UDT	905	905	905	905	905	905

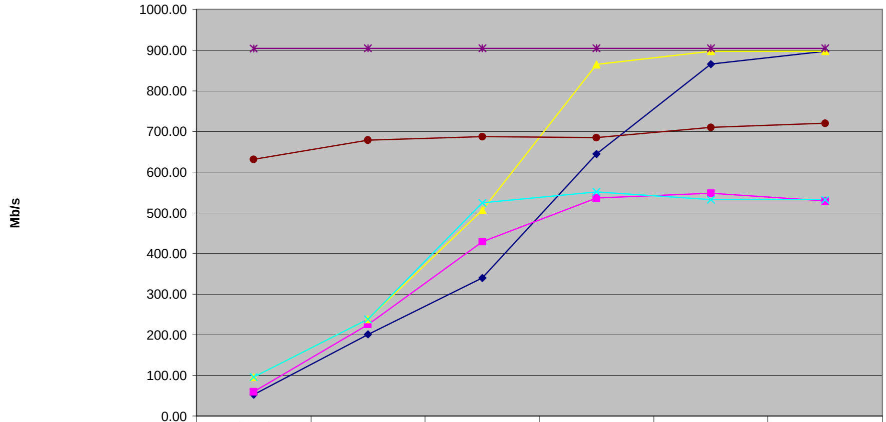
TCP Buffer Size (x BDP)

Figure 3

3. Putting it all together

Once the baseline performance of the individual components had been determined tests began on actual file transfers using GridFTP with both UDT and TCP and the layer 3 transport protocol. The first test runs were made using /dev/zero and /dev/null to determine what effect using GridFTP would have on memory to memory transfers, similar to those done using Iperf and Ieper, and they showed that there was a slight drop in throughput when using GridFTP.

Finally transfers were done from file system to file system, again using increasing kernel buffer sizes to maximize the throughput for TCP. Figure 4 show the complete results.



	default	0.5	1.0	1.5	2.0	2.5
TCP zero->null	52.81	200.91	339.43	644.51	865.59	897.34
TCP File Transfer	59.80	225.49	428.59	536.21	548.13	529.83
TCP+SF zero->null	95.76	238.30	506.59	864.85	897.35	897.35
TCP+SF File Transfer	95.76	238.30	524.44	551.60	532.46	532.46
UDT zero->null	904.39	904.43	904.43	904.40	904.45	904.47
UDT File Transfer	631.52	678.78	687.18	684.63	709.86	720.11

TCP Buffer Size (x BDP)

Figure 4

By looking at these figures it can be seen that when the network and the file system are being loaded at the same time there is a bottleneck causing a significant slowdown in the throughput of the transfer. Further testing showed that while receiving data from the network and writing to disk was possible at line rate reading data from the disk while sending it to the network is only possible at about 50% - 70% of the available bandwidth. Why this is the case is unknown at present.

References

[1] TCP performance tuning - how to tune Linux http://www.acc.umu.se/~maswan/linux-netperf.txt

[2] UDT http://udt.sourceforge.net/

[3] Globus Toolkit http://www.globus.org/

[4] IOZone http://www.iozone.org/

[5] Iperf http://dast.nlanr.net/Projects/Iperf/

[6] XIOperf http://globus.org/alliance/publications/papers/xioperf.pdf