LHCb 2002-049

Load balancing and Gigabit Ethernet Performance Studies for LHCb DAQ sub-farms

LHCb Technical Note

Issue: 1
Revision: 0

Reference: LHCb 2002-049
Created: 17 June 2002
Last modified: 24 September 2002

Prepared By:

Niko Neufeld
CERN/EP

Wulf Thannhaeuser
Summer Student
CERN/EP

Abstract

This document first analyses the performance of Gigabit Ethernet Network Interface Cards, in the way they will be used in the sub-farms of the LHCb Data Acquisition system. Studied are the impact of PCI bus-systems, the tuning of parameters like IRQ coalescence and frame length as well as the performance difference between TCP and UDP.

After this initial analysis of the Gigabit Ethernet performance, different versions of a load-balancing algorithm are implemented. This algorithm will be used by a sub-farm controller to distribute events evenly among its nodes.

The experiments and benchmarks performed show that the envisaged data throughput in a DAQ sub-farm is achievable with current hardware.

Document Status Sheet

1. Document Title: Gigabit Ethernet Performance Studies			
2. Document Reference Number: [Document Reference Number]			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
1	0	24 September 2002	Initial public release

Table of Contents

INTRODUCTION - LHCB DAQ AND EVENTBUILDING **4**

1 GENERAL GIGABIT ETHERNET PERFORMANCE OPTIMISATION **6**

1.1 OVERVIEW OF POTENTIAL PERFORMANCE BOTTLENECKS 6
1.2 TEST SET-UP 6
1.3 TCP / UDP COMPARISON 7
1.4 REDUCING THE TCP/IP OVERHEAD USING JUMBO-FRAMES 8
1.5 MINIMISE CONTEXT SWITCHES USING INTERRUPT COALESCENCE 9
1.6 REDUCE TCP/IP OVERHEAD WITH CHECKSUM OFFLOADING 10
1.7 THE PCI BUS BOTTLENECK 10
1.8 INCREASING NETWORK THROUGHPUT WITH FASTER PCS 10

2 IMPLEMENTING THE LOAD-BALANCING ALGORITHM FOR A SUB-FARM **11**

2.1 TEST SET-UP 11
2.2 PERFORMANCE MEASUREMENTS USING NETPERF IN A SUB-FARM 12
2.3 PERFORMANCE MEASUREMENTS USING A LOAD-BALANCING ALGORITHM 13
2.3.1 Load balancing algorithms 13
2.3.2 Results of the Load-balancing experiments 14
2.4 CONCLUSION 18

ACKNOWLEDGEMENTS **19**

REFERENCES **20**

Introduction - LHCb DAQ and Eventbuilding

The LHCb Data Acquisition system (DAQ) is responsible for the selection of "interesting" events among all events captured by the LHCb Detector. An event is considered "interesting" if a particle collision worth further investigation occurs. This evaluation is done by investigating the data collected by the detector before it is written to permanent storage. This process is called "triggering".

The trigger system has several levels. The level 0 and level 1 triggers are implemented in hardware and investigate only a small part of the data collected by the LHCb Detector. After these first two trigger levels, the whole detector is read out by the Data Acquisition System and the data needs to be passed through the read-out network to a set of DAQ sub-farms, where the additional trigger levels 2 and 3 are performed in software. These trigger levels include the entire data collected by the detector in their evaluation of an event. Figure 1 shows the LHCb DAQ system architecture. For a full description of the online system, please refer to [1].

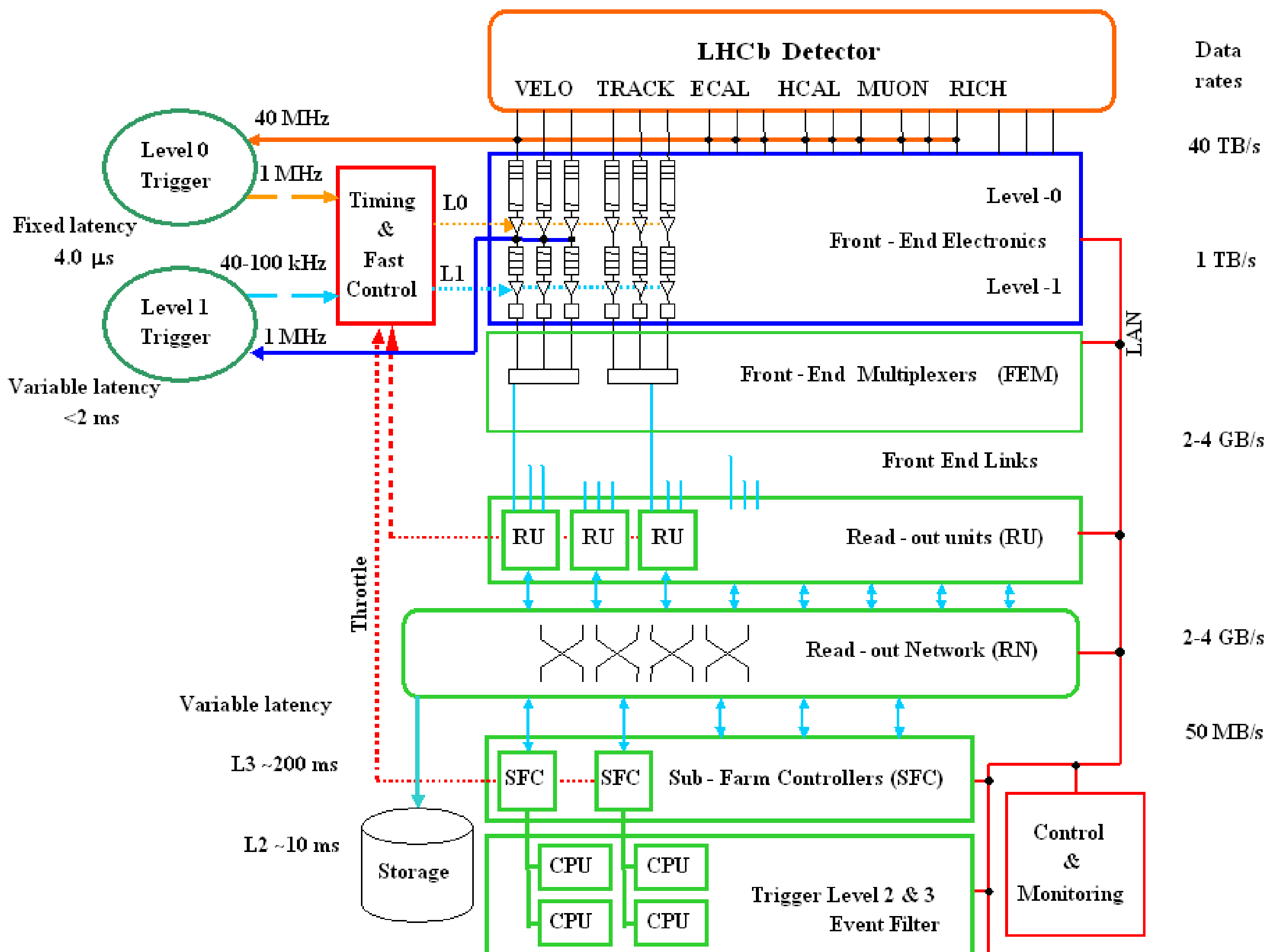

Figure 1 The LHCb DAQ system architecture

After the event-fragments have been collected by the Read-out Units (RU), they are passed through the Read-out Network (RN) in such a way that all data belonging to one particular event reaches one specific Sub-Farm Controller (SFC). There will be ~ 25-30 sub-farms, each containing a Sub-Farm Controller (SFC)

and ~ 10-20 PCs or "nodes". As shown in Figure 1, any data traffic from or to the nodes has to go through and is controlled by the SFC of the sub-farm.

A SFC therefore receives multiple segments of data belonging to the same event. It assembles them and reorders them forming an event data-structure, which now contains all the data collected about the actual event. This process is called "Event-building".

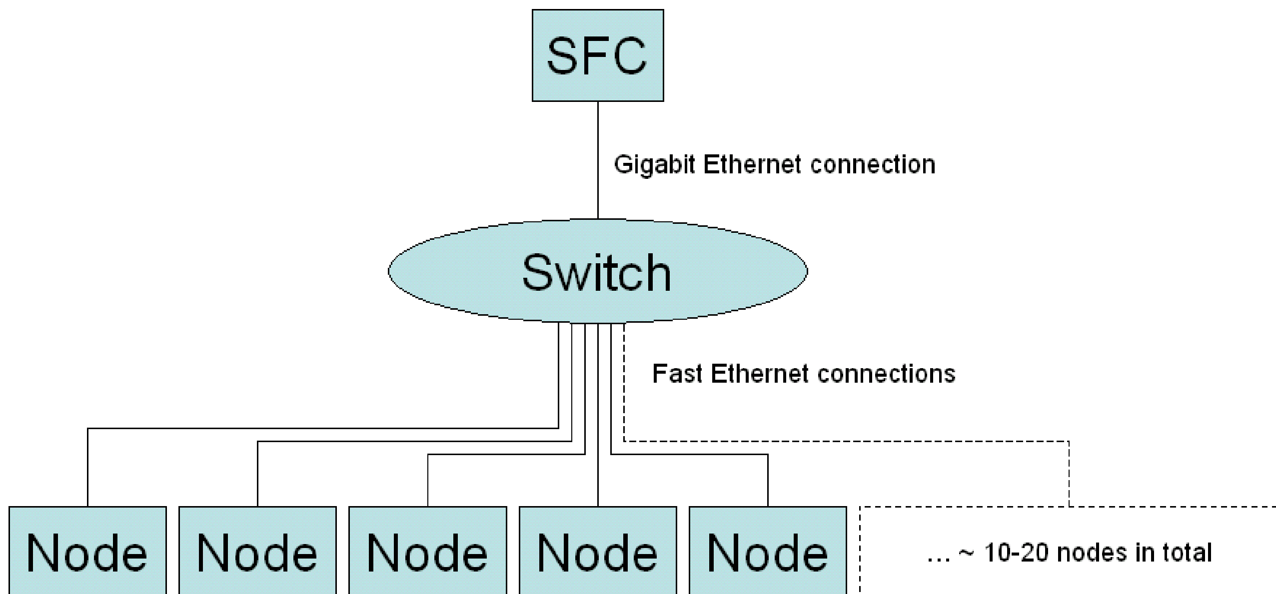

Figure 2 Sub-farm architecture

After a SFC has built a new event it will pass it on to exactly one of its nodes. The node receiving a new event will perform the level 2 and 3 trigger algorithms and, if the event is accepted, it will write back the processed event data to permanent storage. As usual, this data traffic goes through the SFC as well.

Figure 2 shows the architecture of the sub-farms. The SFC is connected to a switch, which links it to its nodes. The link between the SFC and the switch is a Gigabit Ethernet connection (unidirectional nominal throughput: 1000 Mbit/s). The links between the switch and each of the sub-farms are fast Ethernet connection (nominal throughput per link: 100 Mbit/s).

Each SFC should support the forwarding of events at ~ 80 Mbyte/s (640 Mbit/s). Notice that this implies an incoming data traffic of 80 Mbyte/s and an outgoing traffic of 80 Mbyte/s at the same time.

In Part 1 it will be discussed how to optimise the performance of Ethernet links, focussing especially on Gigabit Ethernet connections. The aim is to investigate the experimentally achievable throughput of Gigabit Ethernet Links in order to have a reference value for Part 2.

In Part 2 the load balancing algorithm, used by the SFC to distribute the events among its nodes, will be described and then implemented. The algorithm covers the full data handling protocol foreseen for the final DAQ system. (The trigger algorithm could be simulated by adding an appropriate delay, but since this note aims to determine the obtainable network throughput, there will be no such delay in the experiments described later on.) The experiments done using this load-balancing algorithm will allow a more accurate prediction of the achievability of the throughput necessary to handle the data traffic in the sub-farms. It is worth noting here that all tests are done with current hardware, and that the actual PCs used as nodes will most likely be more performant than the ones used for these tests. The Ethernet connections linking them together will however be very similar in the actual DAQ system.

1 General Gigabit Ethernet Performance Optimisation

1.1 Overview of potential performance bottlenecks

Gigabit Ethernet (GbE) offers a nominal throughput of 1000 Mbit/s in both directions at the same time. Considering full duplex communication the GbE standard should therefore support a throughput of up to 2000 Mbit/s. However, an average "off the shelf" PC is not necessarily able to support such data rates at present. Multiple bottlenecks limit the experimentally achievable throughput:

– Data transfer protocols such as TCP/IP have a per-packet overhead. Any data that needs to be sent has to be "packed" into packets or frames adhering to the protocol specifications. This process of segmentation and reassembly is costly.
– Checksums have to be computed for every packet that was sent or received, in order to detect transmission errors.
– Context switching can be a very costly process for a PC. The CPU of a network traffic receiving PC for example is interrupted every time a network packet is received, to inform it of the arrival of this new packet. Whenever this happens, the CPU has to buffer numerous registers and the stack, and switch from "User" to "Kernel" mode in order to retrieve and process the new data that has just arrived. Even GHz Processors in current PCs are not usually able to handle an interrupt frequency above O(100 kHz).
– Network Interface Cards (NICs) in PCs are usually connected to the PCI bus. This implies that whatever throughput is supported by the NIC might be limited by the throughput of the PCI bus since all data that should reach the PC's memory must be transmitted using the PCI bus. A typical "off the shelf" PC has indeed a fairly slow PCI bus, unable to cope with the full throughput of what the NIC would be able to achieve.

Section 1.2 will present the test set-up used to obtain the benchmark results presented in the following sections. These benchmarks will help in the process of investigating the actually achievable throughput of Gigabit Ethernet.

The following sections will then present optimisation options and benchmark results that show how the significance of the bottlenecks described above can be reduced.

1.2 Test Set-up

The benchmark results presented in Part 1 of this note where obtained using two PCs linked together by a simple point-to-point connection.

PC 1 (pclhcb71) is an Intel Pentium III, 800 MHz, with 512 MBytes of RAM.
PC 2 (pclhcb91) is an Intel Pentium 4, 1.8 GHz, with 256 MBytes of RAM.

Both PCs are running:
Linux version 2.4.18-3 (Red Hat Linux 7.3 2.96-110)

Both PCs have the following Gigabit Ethernet NIC from 3com installed:
" 3Com Corporation 3c985 1000BaseSX (SX/TX) (rev 01) "
They are linked using an optical cable.

The NIC driver used on both PCs is:
AceNIC v0.85, which is freely available at [2].

The benchmark results were obtained using the network-benchmarking program Netperf, which is freely available at [3]. The version used was Netperf 2.2 alpha.

Finally, it is worth noticing that the benchmark results obtained were usually better if you used the slower one of the two PCs as the sending client, while the faster PC was used as the receiving host. This shows that the receiving process is computationally more expensive than the sending process. The results presented in this paper will be the ones obtained using this configuration.

1.3 TCP / UDP comparison

TCP is a reliable connection-oriented protocol supporting full-duplex communication [4].
"Reliable" means here:

- All messages are delivered.
- Messages are delivered in order.
- No message is delivered twice.

UDP is an alternative to TCP [4]. It is an unreliable protocol providing a (connectionless) datagram model of communication. It is "unreliable" because:

- Messages may be lost
- Messages do not necessarily arrive in order
- Messages may be delivered twice.

The above description makes it obvious that TCP is a more sophisticated protocol. Unfortunately, due to TCP's additional capabilities, it is also a computationally more expensive protocol. The per-packet overhead in TCP is higher than it is for UDP. Therefore, if throughput was the only concern, UDP might be preferable to TCP.

However, after events have already been accepted by the level 0 and level 1 triggers, they are very valuable and connection reliability is therefore a concern, since this data should not get lost anymore. It is also worth mentioning that professional applications using UDP rather then TCP, such as for example NFS, usually implement additional protocols on top of UDP to work around the shortcomings of UDP's unreliability. These additional protocols will then most likely take away the benefit of UDP's reduced computational complexity. (Indeed, NFS Version 3, which is not yet widely available, already reverted to the use of TCP rather than UDP for its network communication because of this reason).

It is nevertheless interesting to investigate the performance differences between UDP and TCP. Benchmark results doing exactly this will be presented together with another optimisation in the next section.

1.4 Reducing the TCP/IP overhead using Jumbo-frames

Section 1.3 discussed that TCP/IP has a considerable per-packet overhead. Even though this per-packet overhead itself might be unavoidable, it is possible to reduce the total overhead by reducing the amount of packets. With a given amount of data to transmit, this implies increasing the size of each packet.

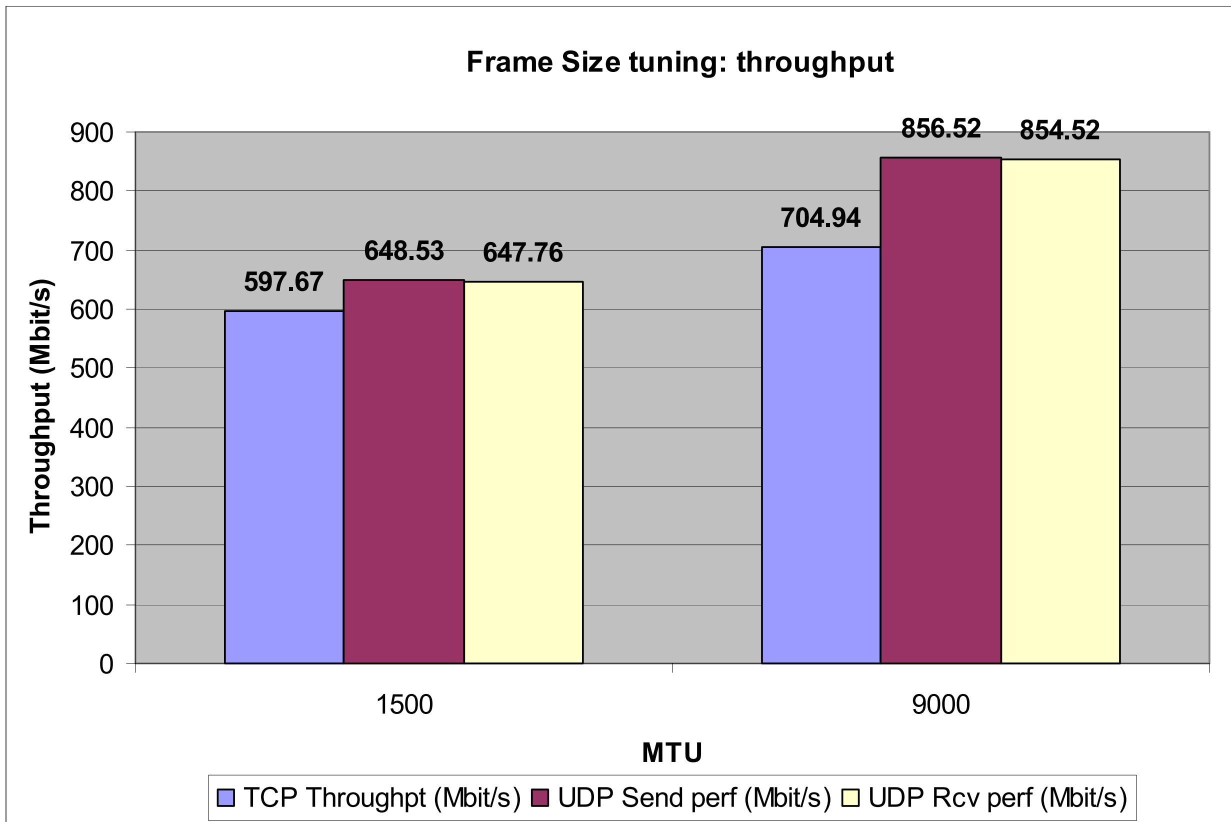

Figure 3 UDP and TCP throughput with and without Jumbo-frames

The Ethernet Standard [5] specifies a Maximum Transmission Unit (MTU) of 1500 bytes. This means that no frame sent through an Ethernet network is allowed to exceed this frame size. There is however a proposed and de-facto respected vendor-specific extension to this standard, which increases this MTU to 9000 bytes. These frames with a bigger MTU are called “Jumbo Frames” and are supported by more and more GbE NICs. (Notice that Jumbo Frames are only applicable to GbE, not to Fast Ethernet and that obviously both the sending and the receiving end have to support them in order for it to work.)

Using Jumbo Frames, the number of packets necessary to transmit a given amount of data could therefore be reduced by a factor of six. This implies six times less TCP/IP overhead, and as can be seen in Figure 3, Jumbo Frames allow indeed a throughput increase of more than 30% for UDP and ~ 17% for TCP.

Referring back to section 1.3, where TCP was compared to UDP, it is also interesting to note that the throughput achieved using UDP is indeed considerably better than the one achieved with TCP. However, comparing the UDP send performance with the UDP receive performance shows that data has been lost, something that is not tolerable in the LHCb DAQ system.

1.5 Minimise context switches using interrupt coalescence

As mentioned in section 1.1, the CPU is interrupted by the NIC every time a new packet arrives through the network. Whenever such an interrupt occurs, the PC has to perform a "context switch" in order to retrieve and process the data that just arrived through the network. Such a context switch involves buffering the register and the stack as well as switching from "User" to "Kernel" mode. This process can be time-expensive.

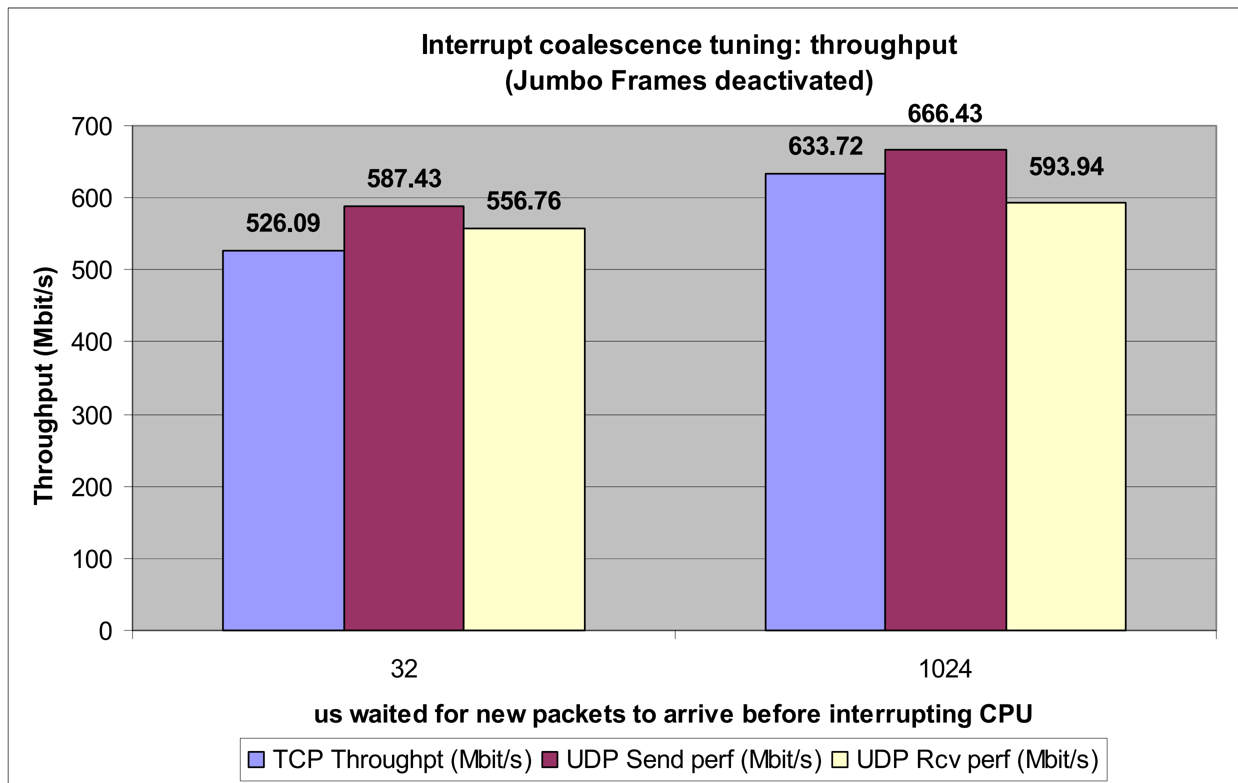

Figure 4 UDP and TCP throughput with more or less Interrupt coalescence

Because of all the tasks involved in doing a context switch, an average "off-the-shelf" PC cannot currently handle more than O(100 kHz) of interrupts. With Fast Ethernet connections, this is not usually a problem, since packets do not arrive through the network at a rate that the CPU could not handle. With GbE however, the packet receiving frequency increases drastically, and a normal PC might not be able to handle the frequency of interrupts sent by the NIC, which in turn would limit the experimentally achievable network throughput.

In order to deal with this performance bottleneck, it is possible to buffer incoming packets in the NIC for a while and only interrupt the CPU after a specified amount of time has elapsed or a specified amount of packets has been received. This technique is called "interrupt coalescence".

Obviously, it is necessary to specify both a time limit and a space limit after which the buffered packets will be transmitted, otherwise buffered information would either not be transmitted until new packets arrive or information might be lost because the NIC runs out of space to buffer new incoming packets. In the results presented below, only the time limit was modified, the space limit being set to the default value for this 3com NIC.

Figure 4 shows that by increasing the time limit for interrupt coalescence from 32 to 1024 microseconds, the TCP throughput could be increased by ca 20%. The UDP receive performance increase is not as impressive, but still noticeable.

Note also that for the above tests, jumbo frames have been deactivated to make the performance increase that interrupt coalescence allows more obvious. Both Jumbo Frames and interrupt coalescence work in the same direction, but mind that dependant on equipment, one may not always be able to use Jumbo Frames. Interrupt coalescence however is a technique that can help to increase the network throughput, even when this optimisation is only in place at one of the connection ends.

1.6 Reduce TCP/IP overhead with checksum offloading

The TCP/IP protocol adds a checksum to the tail of each transmitted packet. This checksum uniquely identifies the data transmitted. The receiving side recomputes this checksum with the data it has received. If the result differs from the checksum that was submitted in the tail of the packet, then a transmission error was detected.

If there is a considerable amount of traffic on the network, this checksum computing can take up a lot of CPU time. The idea of "checksum offloading" is that this process can be done in hardware on the NIC, rather then in software by the CPU. This allows the CPU to spend more time on other processes.

Checksum offloading was activated for all benchmarks described in this note.

1.7 The PCI bus bottleneck

As mentioned in the introduction, the PCI bus might be a bottleneck to network throughput, since the NIC has to communicate with the PC's memory via this bus. PCI busses in typical "off-the-shelf" PCs are usually 32 bits wide and run at a frequency of 33 MHz. This gives them a nominal throughput of 1056 Mbit/s. However, this theoretical maximum is usually not achievable, since the PCI standard imposes a burstsize on the data transfer, and PCI itself has an overhead as well. A more accurate approximation of the throughput achievable on such a PCI bus would probably be around 900 Mbit/s.

Since the default PCI bus size can be considered more or less optimal, it might be difficult to improve this throughput. There are however PCs with better PCI bus specifications:

- 64 bits / 32 MHz
- 64 bits / 66 MHz
- 64 bits / 133 MHz

1.8 Increasing network throughput with faster PCs

It is also worth mentioning that raw CPU power obviously improves network throughput as well (up to the limits of the Ethernet link). If a CPU is very fast, it might be able to handle more network traffic, independent of the optimisations in place on the network link.

2 Implementing the load-balancing algorithm for a sub-farm

Part 1 of this note looked at general Ethernet performance optimisation. The benchmarking was performed in a simple point-to-point test set-up between two PCs. Part 2 will tackle the problem of simulating the data traffic in an LHCb DAQ sub-farm more accurately in order to benchmark its performance. This will allow us to deduce whether the network throughput needed to handle the amounts of data produced by the detector is achievable with the current DAQ architecture and on current hardware.

Section 2.1 will present the test set-up used in the benchmarking processes. The next section will present the results of a couple of simple Netperf network performance tests. Then the load-balancing algorithm, used by the program simulating the network traffic in a sub-farm, will be presented. Finally, the network throughput results that the experiments have produced will be discussed.

2.1 Test Set-up

To simulate the architecture that will be present in an LHCb DAQ sub-farm, a part of the LHC Computing Grid Testbed structure was used. Figure 5 shows this Testbed.

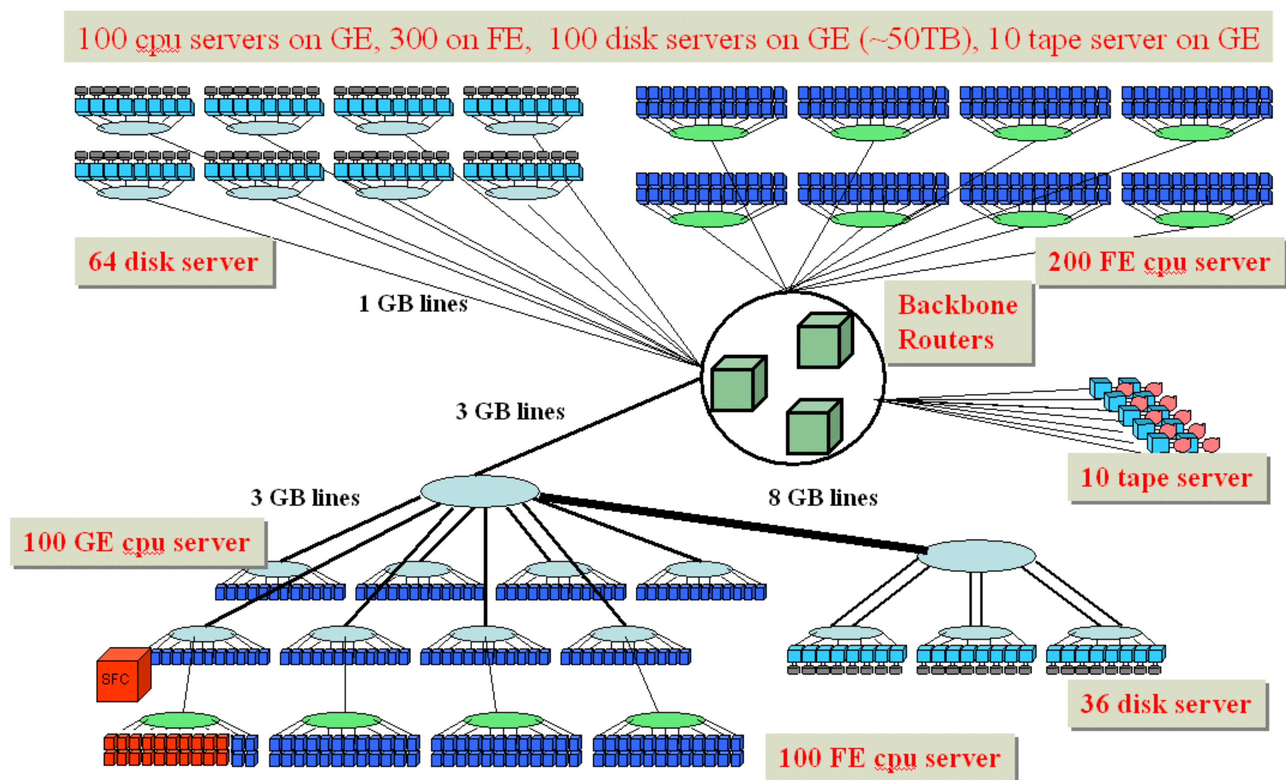

Figure 5 LHC Computing Grid Testbed Structure

As can be seen on the lower left side of the diagram, one of the PCs in the testbed was used as SFC. The SFC is linked to a Gigabit Ethernet switch, which is linked to a Fast Ethernet Switch with a Gigabit uplink. Twenty-four more PCs are attached to this Fast Ethernet switch. Twenty of those will constitute the “nodes”.

The PC used as SFC (lxshare085d) is a Dual Intel Pentium III, 1.0 GHz, with 1GByte of RAM.
The PCs used as nodes (lxshare0229-248) are Dual Intel Pentium IIIs, 1.0 GHz, with 512 MBytes of RAM.

The PC used as SFC is running:
Linux version 2.4.18-5ds1 (Red Hat Linux 7.1 2.96-98)
The PCs used as nodes are running:
Linux version 2.2.19-6.2.1.1smp (based on Red Hat Linux 6.1)

The PC used as SFC has the following GbE NIC installed:
"BROADCOM Corporation NetXtreme BCM5700 Gigabit Ethernet (rev 12)"
The NIC driver installed is:
Gigabit Ethernet Driver bcm5700 with Broadcom NIC Extension (NICE) ver. 2.2.20

The PCs used as nodes have the following (Fast Ethernet) NIC installed:
"Ethernet controller: Intel Corporation 82557 (rev 08)"
The NIC driver installed is:
eepro100.c: $Revision: 1.20.2.10

The switch that the SFC sits on is a Gigabit Ethernet switch.
The switch connecting all the nodes is a Fast Ethernet switch type 4400 from 3com.

For the Netperf benchmark results presented in section 2.2, the version of Netperf used was still Netperf 2.2 alpha, which is freely available at [3].

2.2 Performance measurements using Netperf in a sub-farm

In order to have a reference value of how much simultaneous network traffic the sub-farm can handle, 20 Netperf processes were simultaneously run on the SFC for a duration of 3 minutes. Each of these processes tries to send as much data as possible to a different node. The following benchmark results were achieved:

Individual bandwidths (per process/node) in Mbits/s:									
75.02	39.78	18.98	40.06	75.9	36.65	7.54	40.48	69.09	36.82
52.07	36.56	45.46	35.6	53.96	37.37	15.3	37.02	52.13	35.44
Total bandwidth in Mbits/s:									
									841.23

The results above show that:

- The total bandwidth is most likely limited by the Gigabit Link between the SFC and the Fast Ethernet Switch connecting all the nodes, since the throughput value is similar to the benchmarks performed in Part 1 of this document.
- The bandwidth is unfortunately not evenly distributed among the nodes. This might be due to unfair scheduling, i.e. some Netperf processes on the SFC might have been allocated more time by the scheduler than others, and would therefore have been able to transmit more data. Tests with longer running times however smoothened this distribution, both during Netperf tests and load-balancing experiments.

2.3 Performance measurements using a Load-balancing algorithm

In the LHCb DAQ sub-farms, the SFCs will receive events that have to be passed on to their nodes for processing. A node will then analyse the event using the level 2 and 3 trigger algorithms. This processing takes time, and the SFC therefore has to distribute the events more or less evenly between its nodes in such a way that the nodes have time to process their event(s) without falling behind too much. This process of distributing the computational load between several PCs is called "load-balancing".

In section 2.3.1 two different load-balancing algorithms will be presented: "round-robin" and a load-balancing algorithm using control tokens. Section 2.3.2 will present the network throughput results that have been obtained with the chosen load-balancing algorithm during the experiments. Finally, the implications of these results for the current LHCb DAQ architecture will be discussed.

2.3.1 Load balancing algorithms

2.3.1.1 Round-robin

The most intuitive way of distributing events evenly among nodes is very simple. The so-called "round-robin" algorithm takes an incoming event and forwards it to the node which has not received an event for the longest time. More intuitively speaking, the SFC forwards an event to node 1, then one to node 2, one to node 3, and so on until the last node has received an event. Then the process is restarted with node 1. This algorithm will indeed distribute the events evenly among all nodes: If for example a total of 2000 events were to be forwarded by the SFC, each of the 20 nodes would received exactly 100 events.

However, the round-robin approach has certain problems. The processing of an event does not take a constant amount of time. The level 2 and 3 trigger algorithms process the event by performing a series of tests on the event. If any of these tests fail, the event is rejected and the processing of this event is finished. It therefore takes longer to accept an event then to reject it. This implies that the order in which nodes become ready to process the next event is not necessarily identical to the order in which they have been passed events. A simple round-robin approach might therefore be inefficient.

2.3.1.2 Load balancing using control tokens

The problem of passing an event to a node which is not ready to receive it can be solved by using a queue of so-called control tokens maintained by the SFC. Here is an outline of the algorithm:

- The SFC keeps a FIFO queue of control tokens received by its nodes.
- Whenever the SFC wants to forward an event to one of its nodes to process it, it takes the first token of the queue and sends the event to the sender of this token.
- A node receiving an event processes it using the level 2 and 3 trigger algorithms and, once this is done, sends a control token to the SFC, indicating that it is ready to process the next event. The SFC receives this token and places it at the end of the queue.
- When nodes connect to the SFC for the first time they send a control token without having received an event. This ensures that the token queue is not empty once the SFC receives events.

- The control token also includes information about the results of the level 2 and 3 triggers on the last event processed. This information would be kept in permanent storage for future reference.

- For each event sent to a node, there will eventually be status information in the next token. That means that there is always an answer.

- In the event of a positive trigger decision (i.e. the event is accepted) the token will contain the data destined for permanent storage.

In theory, the use of control tokens ensures that only nodes ready to process a new event actually receive one. In practice however, not only tokens but also events need to be buffered in the SFC and the nodes: The SFC needs to be able to store events in case its token queue happens to be empty at a certain point in time. Also, the system might work more efficiently if the nodes start by submitting two tokens instead of just one, when they first connect. This way, they would always have a buffered event and wouldn't need to wait until they receive a new event in order to process data.

2.3.2 Results of the Load-balancing experiments

2.3.2.1 Implementation details

For the purpose of load-balancing within an LHCb DAQ sub-farm the algorithm using control-tokens (described in section 2.3.1.2) was chosen, since this is likely to be the algorithm of choice during the actual implementation phase of the LHCb DAQ system.

All data transfer through the network relied upon BSD Stream Sockets (i.e. using TCP).

The load-balancing algorithm using control tokens was actually implemented in three different ways to compare and experiment with the achievable network throughput:

- Version 1 implements the algorithm exactly as described in section 2.3.1.2. The process running on the SFC is subdivided into two threads. One of the threads is responsible for the reception of control tokens, while the other thread is responsible for receiving events from an event generator and forwarding them to the nodes. These two threads share a token queue, which is protected by a mutex. The event generator is a separate process, which, for the purpose of the experiments described here, also ran on the SFC. It was however implemented in a way that would also allow it to run on a different PC. Events received by the event-forwarding thread are first placed in an event queue and whenever tokens are available, events are forwarded to the appropriate nodes. Nodes immediately discard a received event and send back a control token without any delay.

- Version 2 does not have an event generator or an event queue anymore. It is implemented using a thread per connected node. Each thread running on the SFC is constantly waiting for a new control token. Whenever a thread receives a token, it immediately sends out an event to its node. The event being sent is always the same event, different from the others only because of its unique event ID. Nodes immediately discard a received event and send back a control token without any delay.

- Version 3 finally is very similar to version 2, except that there is no token traffic left. Each thread in the SFC is constantly sending events as fast as it can, the corresponding node on the other side does not send any tokens any more, but is just receiving events as fast as it can, immediately discarding anything it receives. (In this very basic form, the algorithm is similar to the 20-node Netperf test described earlier.)

During all experiments, events were assumed to have a constant size of 32 KBytes.

Normal tokens have a size of only 28 bytes. However, in order to simulate the fact that nodes might accept an event and send back processed event data with the token in this case, a random number generator is driving the token generator and on average every 200th token has a size of 64 KBytes.

The experimentally obtained throughput values presented in the following paragraphs will always be values reflecting the actual payload throughput, i.e. ignoring TCP/IP headers and tails.

The following values were the best values obtained during a long series of tests. This approach to presenting the results was chosen because the network and PCs involved were not fully optimised for our purposes and the results obtained showed variations of ~ 5%.

2.3.2.2 Version 1 results

Test forwarding 600 K events, ~ 3 minutes:

Events forwarded per second: 2975.7

Event throughput (excl. tokens) in MByte/s: 95.3
Event throughput (excl. tokens) in Mbit/s: 762.1

Test forwarding 2 M events, ~ 10 Minutes:

This test showed that there is a problem in the implementation of Version 1 of the load-balancing algorithm, since the forwarding of events seems to start at a very reasonable speed, but then soon decreases. Furthermore, the number of events that arrive when no token is available seems to increase. Version 1's behaviour is currently under investigation.

Events sent in total	Time taken in seconds to send the last 100 K events	Total number of events that arrived when no token was available	Throughput in Mbit/s	Throughput in Mbyte/s
100000	33	7	775.76	96.97
200000	34	52	752.94	94.12
300000	54	56	474.07	59.26
400000	100	7186	256.00	32.00
500000	103	17557	248.54	31.07
600000	104	27920	246.15	30.77
700000	104	38296	246.15	30.77
800000	104	48663	246.15	30.77
900000	103	59037	248.54	31.07
1000000	89	63183	287.64	35.96
1100000	87	66885	294.25	36.78
1200000	118	75571	216.95	27.12
1300000	150	90589	170.67	21.33
1400000	145	105120	176.55	22.07
1500000	132	118229	193.94	24.24
1600000	186	136892	137.63	17.20
1700000	215	158380	119.07	14.88
1800000	215	179868	119.07	14.88
1900000	215	201379	119.07	14.88
2000000	215	222875	119.07	14.88

2.3.2.3 Version 2 results

Test forwarding 600 K events, ~ 3 minutes:

Events forwarded per second: 3405.5

Event throughput (excl. tokens) in MByte/s: 109.0
Event throughput (excl. tokens) in Mbit/s: 872.1

Test forwarding 2 M events, ~ 10 Minutes:

Events forwarded per second: 3480.5

Event throughput (excl. tokens) in MByte/s: 111.4
Event throughput (excl. tokens) in Mbit/s: 891.3

Test forwarding 200 M events, ~ 17 Hours:

Events forwarded per second: 3243.2

Event throughput (excl. tokens) in MByte/s: 103.8
Event throughput (excl. tokens) in Mbit/s: 830.6

This last test with a running time of 17 hours indicates that Version 2 does not suffer from the same performance decrease that Version 1 does.

Furthermore, the following statistic shows that the distribution of the bandwidth among the nodes of the sub-farm does indeed smoothen during a longer test:

Number of connected nodes: 20

Events sent to lxshare0229.cern.ch: 10425677, Throughput in Mbit/s to this node: 43.3
Events sent to lxshare0230.cern.ch: 10709545, Throughput in Mbit/s to this node: 44.5
Events sent to lxshare0231.cern.ch: 10023924, Throughput in Mbit/s to this node: 41.6
Events sent to lxshare0232.cern.ch: 9475635, Throughput in Mbit/s to this node: 39.4
Events sent to lxshare0233.cern.ch: 9465028, Throughput in Mbit/s to this node: 39.3
Events sent to lxshare0234.cern.ch: 9911658, Throughput in Mbit/s to this node: 41.2
Events sent to lxshare0235.cern.ch: 9896578, Throughput in Mbit/s to this node: 41.1
Events sent to lxshare0236.cern.ch: 10354954, Throughput in Mbit/s to this node: 43.0
Events sent to lxshare0237.cern.ch: 9696645, Throughput in Mbit/s to this node: 40.3
Events sent to lxshare0238.cern.ch: 10182680, Throughput in Mbit/s to this node: 42.3
Events sent to lxshare0239.cern.ch: 9937960, Throughput in Mbit/s to this node: 41.3
Events sent to lxshare0240.cern.ch: 10259920, Throughput in Mbit/s to this node: 42.6
Events sent to lxshare0241.cern.ch: 9651051, Throughput in Mbit/s to this node: 40.1
Events sent to lxshare0242.cern.ch: 9790147, Throughput in Mbit/s to this node: 40.7
Events sent to lxshare0243.cern.ch: 10462562, Throughput in Mbit/s to this node: 43.5
Events sent to lxshare0244.cern.ch: 10692188, Throughput in Mbit/s to this node: 44.4
Events sent to lxshare0245.cern.ch: 10021289, Throughput in Mbit/s to this node: 41.6
Events sent to lxshare0246.cern.ch: 9692120, Throughput in Mbit/s to this node: 40.3
Events sent to lxshare0247.cern.ch: 10209239, Throughput in Mbit/s to this node: 42.4
Events sent to lxshare0248.cern.ch: 9168862, Throughput in Mbit/s to this node: 38.1

2.3.2.4 Version 3 results

Test forwarding 600 K events, ~ 3 minutes:

Events forwarded per second:	3247.9
Event throughput (excl. tokens) in MByte/s:	104.0
Event throughput (excl. tokens) in Mbit/s:	831.8

Test forwarding 2 M events, ~ 10 Minutes:

Events forwarded per second:	3191.5
Event throughput (excl. tokens) in MByte/s:	102.2
Event throughput (excl. tokens) in Mbit/s:	817.3

2.4 Conclusion

The achieved throughout rates of up to 890 Mbit/s show that the envisaged data throughput in a DAQ sub-farm is already achievable with current hardware, and that technology is definitely on our side.

This does affirm that the network is currently able to achieve the necessary performance. Since none of the experiments actually performed any data processing, there were virtually no delays between the reception of an event by a node and the sending of the corresponding control token.

The PCs and switches used during the described experiments, were part of the Grid Test-bed and not optimised for the sole purpose of these experiments. Results were therefore subject to slight variations of ca 5%. (This explains for example, why the best results were achieved with version 2 of the code, even though version 2 was on average slightly less performant than version 3). It would be interesting to verify the experimental results presented above on a dedicated and optimised network, adhering to the following restrictions:

- None of the PCs involved in the experiments should have any other tasks to process.
- They should not need to handle any irrelevant network traffic.
- Separate NICs and networks should be installed for data transfer and process control (as is foreseen in the LHCb DAQ Online System).

These tests will be performed in due time.

Acknowledgements

We would like to thank Bernd Panzer-Steindel and CERN's ADC group for their support in using part of the GRID test-bed for the load-balancing experiments. We would also like to thank Clara Gaspar for all the useful discussions.

References

[1] LHCb Collaboration, “LHCb Online System, Data Acquisition and Experiment Control: Technical Design Report”, LHCb-TDR-7, CERN-LHCC-2001-040

[2] Jes Sorensen, “Alteon AceNIC / 3Com 3C985 / NetGear GA620 Gigabit Ethernet Adapter” http://home.cern.ch/~jes/gige/acenic.html

[3] Rick Jones, “The Public Netperf Homepage“ at http://www.netperf.org/

[4] Lawrence Besaw, “Berkeley UNIX† System Calls and Interprocess Communication”, January 1987, Revised September 1987 and January 1991 by Marvin Solomon

[5] IEEE Std 802.3, “Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications”, 2000 Edition