

Testing the Force 10 E600 switch for the LHCb Readout Network

Internal Note

Issue: 1
Revision: 0

Reference: CERN/LHCb/Latex/InternalNoteClass
Created: October 20, 2005
Last modified: October 20, 2005

Prepared by: K. Hennessy, Dr. R. McNulty

Abstract

The E600 switch from Force 10 is a candidate switch for the LHCb Readout Network and was tested to determine if it is suitable for this purpose. The MAP2 supercomputer at University of Liverpool was used as a test-bed for this switch. The switch was viewed as a black box and its suitability was inferred by successful network throughput. Rates comparable to LHCb were achieved. The maximum rate achieved for 3TB of packets was 16Gbps.

1 Introduction

LHCb will require a readout network that can successfully transfer the high data rates coming from the various sub-detectors to a dedicated computer farm for processing and subsequent storage. The Force10 E-600 has been selected as a potential candidate for this job. This note outlines a series of tests that have been carried out at the MAP2 computing facility in the University of Liverpool to assess the E-600's suitability for this task.

In section 2 we briefly review the LHCb readout requirements. Section 3 describes the test-bed configuration at Liverpool. The software is described in section 4, the tests in section 5 and the results are presented in section 6. Section 7 summarises our conclusions on the suitability of the switch for LHCb.

2 LHCb Readout Requirements

The full LHCb readout structure is shown Figure 1. Data flows out of the detector to the front-end electronics. There it is digitised and assembled into network packets. The packets are then routed through the network switch to the computer farm for event reconstruction. The only significant cause of packet loss is due to congestion in the switch¹. The switch is considered suitable if this congestion never occurs.

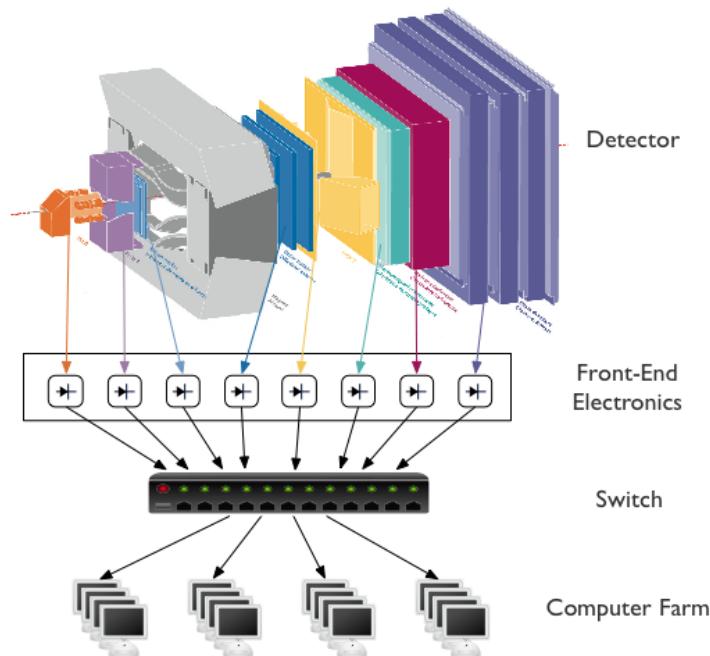


Figure 1 Schematic showing the flow of data through the LHCb Trigger

To facilitate delivery of the packets a communication protocol must be chosen. LHCb requires a protocol with little overhead, unidirectional transport, and has non-blocking I/O. For this reason UDP is the current best choice,

¹Bit errors may also occur due to the unshielded transmission lines but this occurrence has been shown empirically to be less than 10^{-14} [Trigger-TDR]

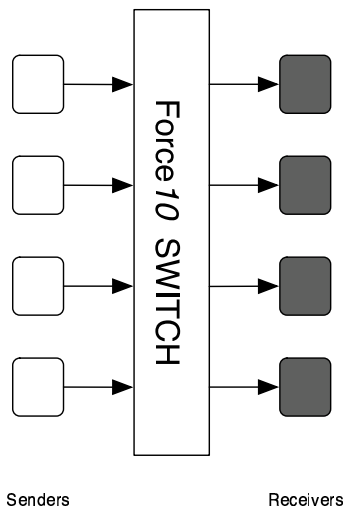


Figure 2 Unidirectional data flow through the switch

since it deals with all of the aforementioned requirements. In contrast, TCP, the primary other choice, is a reliable transport mechanism and has built-in congestion avoidance. TCP causes much overhead in spite of that, and a good deal of “back-talk” is resultant from the protocol. A backlog of data occurs when trying to use TCP for the kind of data throughput necessary for our setup. Previous tests confirm this [Shorten et al.]. This makes TCP particularly unsuitable for LHCb.

The LHCb readout network will consist of 323 front-ends synchronously sending data packets every $25\mu\text{s}$ through 96 links to a computer farm for event reconstruction. The total data transfer speed amounts to a maximum of 48Gbps.

3 Liverpool MAP2 Test Setup

The MAP2 cluster was used as a test-bed for the Force10 switch. The MAP2 supercomputer is an ideal test-bed as it uses the Force 10 E600 to control its network. According to the technical specifications of the E600, this switch is capable of 56Gbps bandwidth per rack unit. A unit typically consists of no more than 48 ports, giving a possible bandwidth of over 1Gbps per network port. Assuming the data rate for LHCb is 40kHz^2 , or a 1.0-1.5KB packet every $25\mu\text{s}$ from about 100 source ports, this gives a total expected bandwidth between 32 and 48Gbps. According to the specifications at least, the E600 switch should be capable of transferring the data rates that LHCb expects to see.

The test-bed has the following configuration:

- 40 Linux Nodes - 2.4.29, Pentium 4, 3GHz
- 1Gbps Ethernet cards using PCI-X slots
- Force 10 E600 network switch
- Bandwidth 1Gbps
- Packet Size 1.0 - 1.5 KB

Figure 2 is a simple schematic showing how data flows through the switch. Data flows in only one direction; from sender to receiver. Factors which influence this data flow are as follows:

CPU speed Can the computer maintain the rate of data at which needs to be sent?

²Note - this is expected to change

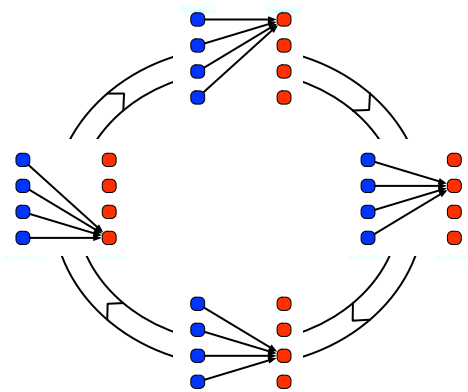


Figure 3 Round-robin data sending pattern

Buffers Hardware and software buffers: the computer has to have somewhere to temporarily store data while a decision is made on how to deal with it.

Synchronisation Are all the senders sending at the same time (as they should be)?

The test-bed was setup to simulate what will be happening at LHCb. Regarding switch configuration, LHCb will have roughly 100 front-end input ports sending data to 100 output ports. In our test setup, one fifth of the number of switch ports are being tested (approximately) compared with the number that LHCb will actually employ, and the tests following in section 5 are scaled accordingly.

On average, every $25\mu\text{s}$, each sending machine sends one data packet (1KB) to a receiving machine. This is done in a round-robin fashion (figure 3). Each sends to the “first” receiver; waits $25\mu\text{s}$; then send to the next receiver and so on. So each receiver gets a large chunk of data (100KB) every 2.5ms ($= 100 \times 25\mu\text{s}$). On the MAP2 supercomputer, this has been scaled down; there are 20 senders and 20 receivers in this case. The $25\mu\text{s}$ gap still exists, so the machines send and receive at the same rates overall, since the round-robin cycle completes in one fifth of the time.

4 Software Design

Two programs were created; a client and a server. The client sends data to the server (client and sender are synonymous; similarly for server and receiver). These were written in C and use the UDP protocol to transfer data. A data set of random numbers is created when the client starts. This prevents any caching or forward look-ahead of the data (if such construct have been put in place).

To synchronise the clients, each client program waits for a broadcast signal before it starts sending data. A broadcast packet gets sent to every machine on the network subnet. Only one packet is sent and is “heard” by all the relevant nodes simultaneously. The clock timings in each computer are slightly different. These timings are based on the oscillations of a crystal inside the computer [Henderson et al.]. The crystals can differ somewhat (by how much is detailed in section 5.2.2). In order to avoid complications that may arise from this, the broadcast message is sent every 2.5ms to resynchronise the nodes.

UDP has no error checking built in, so it was necessary to add some checks to ensure the data is arriving and arriving intact:

Checksum to ensure each packet hasn’t been corrupted.

Sequence numbers to ensure no packets were dropped.

Statistics use large enough data sets to weed out anomalies.

5 Tests

A series of tests were performed in order to answer the following key questions:

- What factors affect network throughput?
- Can the senders correctly mimic LHCb frontends?
- Can the receivers match the rate of transmission?

5.1 Identifying factors which affect network throughput

5.1.1 Delay Effect

It would be expected that introducing a delay between packet transmissions will have an effect on data throughput. To assess this effect the inter-packet delay was varied and the corresponding throughputs were recorded.

By choosing a default buffer size of 100KB on each of the Linux nodes, a test was performed to analyse the effect of imposing such a delay. In this test there were 20 senders sending to 1 receiver. Table 1 shows that the added delay prevents loss of data and a longer delay allows the receiver more time to handle the data. With no delay³, the receiver receives 10.20% of the data and with a delay of 50 μ s, this successful throughput increases to 79.88%. All of the data arrives when the delay is set to 100 μ s. It is clear from this that imposing a delay has a critical effect on data throughput.

| Delay / μ s | Number of packets | % success |
|-----------------|-------------------|-----------|
| 0(8) | 20,000 | 10.20 |
| 50 | 20,000 | 79.88 |
| 100 | 20,000 | 100.00 |

Table 1 Success rate with increasing packet transmission delay

5.1.2 Buffer Effect

Packets will arrive at the receivers in clumps, followed by a rest period. For data sets greater than the available buffer size, the receiving buffer usually overflows resulting in data loss. The buffer has to be large enough for the computer to store data temporarily before it deals with it. By creating large enough buffers more data should arrive safely. The buffers were set to 8MB per node. The buffers inside the switch are on average 5MB per port. The consequence being that any loss that occurs due to buffering would happen in the switch rather than the computer nodes.

5.2 Senders as LHCb frontends

For our tests to realistically mimic LHCb type behaviour each sender must synchronously transmit a packet every 25 μ s with as little error as possible. Two factors can potentially prevent us from doing this:

5.2.1 System interrupts

Figure 4 shows the time lapse between successive transmissions by one of the senders. Figures 4(a) and (b) are the same except the Y-axis in (b) is a log scale. The requested sender delay was set to 25 μ s and 4(a) shows that the average is exactly this. Viewing with a log scale, clearly most packets are sent every 25 μ s as desired. However, some of the packets are sent much later. The reason for this was tracked to Linux scheduling interrupts. The code was written to compensate for such situations and following a long-delayed packet, subsequent packets will be sent at a faster rate to make up for the “lost time”. The data points below 25 μ s show this compensation (Fig. 4(b)). So on average, the senders *are* sending every 25 μ s.

³There is a small inherent delay due to the time it takes a node to send a packet, hence the 8 in braces.

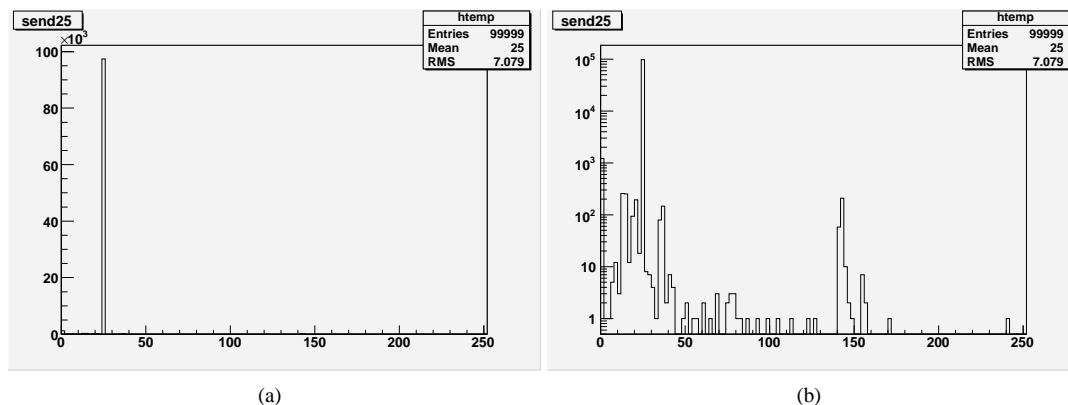


Figure 4 Recorded times taken to send packets, (a) linear and (b) logarithmic

5.2.2 Sender clock signal

An oscillating crystal is used to generate the clock signal in the senders, and since no two crystals are the same, a small difference is expected. This difference will cause the sources to de-cohere over time. We examine this decoherence effect and show that a broadcast signal must be used to resynchronise the senders.

If we assume the sources start off in sync and that all of the incoming packets from each source to receiver 1 are logged (the senders sending in a round-robin sequence every $25\mu\text{s}$). We would expect the time separation between two successive packets from the same sender to receiver 1 to be $\Delta t = 20 \times 25\mu\text{s} = 500\mu\text{s}$. That is, receiver 1 sees a packet every $500\mu\text{s}$ from a particular sender. Since the clocks are slightly different on each sender Δt will vary slightly from sender to sender. Δt was measured for each sender and graphed in figure 5.

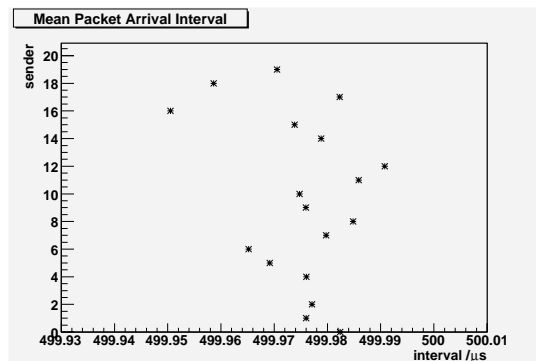


Figure 5 $500\mu\text{s}$ as timed by different sources (Δt)

The maximum deviation between the sources is about $0.05\mu\text{s}$, after $500\mu\text{s}$ has elapsed. If we introduce a broadcast signal every 2.5ms , the maximum deviation, D , amounts to:

$$D = \frac{0.05}{500} \times 2500 = 0.25\mu\text{s} = 1\% \quad (1)$$

Thus, employing such a broadcast signal will prevent our senders from de-cohering to any significant extent.

5.3 Receiver Rate Handling

It was necessary to determine the receiver behaviour at faster send rates to show that the network was not running too close to any threshold of performance. The system should not cause packet loss in the event of temporary faster data rates than expected. The following test explains this. The sender rate was varied and the incoming receive rate was monitored on the receivers. The graph in figure 6 shows a plot of the receive rate versus the sender delay. The line represents a 1:1 linear relationship as would be expected, i.e. the receive rate should match

the send rate. The data behaves as expected except at zero send rate. This is due to the processing time on the send (i.e. there is no real zero send rate). The graph shows that even for high data rates the network can cope with the load.

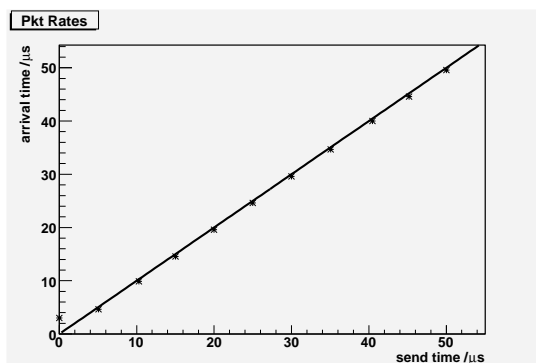


Figure 6 Packet arrival times versus sender times

6 Results

In the previous section we have shown that our test-bed very closely mimics the setup expected for LHCb. Following on from this, the next logical step was to stress the switch to the limits required by the experiment. We stress the switch by passing increasing amounts of data through it and recording the throughput.

Assuming a packet is sent every $25\mu s$ the bandwidth utilisation, B , is calculated as follows:

$$\begin{aligned}
 B &= \frac{N \times P}{t} \\
 &= \frac{20 \times 1KB}{25\mu s} \\
 &= \frac{20 \times 1024 \times 8bits}{25 \times 10^{-6}s} \\
 &= 6.10Gbps
 \end{aligned}
 \tag{2}$$

where N is the number of sending nodes, P is the packet size, and t is the time between packet transmissions. The tests were performed at this rate. Table 2 shows the results of the tests. The first test ran for 250ms and this time is directly proportional to the total amount of data sent. The running time/data set size was increased by a factor of 10 for each successive test. All of the tests completed 100% successfully with all data arriving at the receivers. The final test ran for 41.67 minutes and transferred 2TB of data without any loss.

| Delay | Time | Data Set Size | % success |
|-----------|----------|---------------|-----------|
| $25\mu s$ | 250ms | 200MB | 100 |
| $25\mu s$ | 2.5s | 2GB | 100 |
| $25\mu s$ | 25s | 20GB | 100 |
| $25\mu s$ | 4.17min | 200GB | 100 |
| $25\mu s$ | 41.67min | 2TB | 100 |
| $15\mu s$ | 25min | 3TB | 100 |

Table 2 Stress Test Results - successful rates and data sets achieved by Force10 E600

The maximum data rate that is quoted in the LHCb Trigger TDR is 56.7Gbps (or 7.1GB/s). Since MAP2 is roughly one fifth the scale of the LHCb setup, this amounts to ~11Gbps. A further test was performed to try to achieve this data rate on MAP2. The packet size was increased to 1.5KB and the delay was decreased to $15\mu s$. Thus, the data size was 3TB and the running time was 25 minutes. This resulted in a bandwidth of 16Gbps with 100% throughput, well above the expected 11Gbps for LHCb. It is important to note that although the MAP2 setup is one fifth the size of what is expected, the per-port data rates are comparable to that of LHCb. Each chunk of data (cluster of 20 packets) arriving at each port is $\frac{1}{5}$ the size of an LHCb cluster (100 packets), but that port will receive clusters 5 times as frequently (since the senders loop back to the first receiver 5 times as often).

During the previous tests, a foreseeable problem was tracked down to the network configuration. The switch has a default “arp” timeout of 30 minutes. It refreshes its arp table at this frequency. With a large network subnet, this may cause disastrous effects. A network “storm” occurs if data needs to be sent to a node with a five minute old arp entry. Substantial data loss occurs. Permanent solutions to this problem are being investigated, such as static arp tables, a much longer timeout, etc.

7 Conclusions

From the tests performed, a clear understanding of the network behaviour has been shown. The data obtained shows how it is important to configure the Linux nodes to properly test the switch. The senders were set up to closely mimic the behaviour of the LHCb front-end electronics. The receiver were setup to service the arriving data and log any loss of throughput.

Two terabytes of data was successfully transferred without error in 42 minutes with packets sent every $25\mu\text{s}$. An additional stress test ran three terabytes of data through the switch in 25 minutes with packets sent every $15\mu\text{s}$, also without error. Assuming that the switch configuration issue will be resolved, the stress tests show that the Force10 E600 switch would be suitable for the LHCb Readout Network.

8 References

- [Trigger-TDR] *LHCb Trigger System Technical Design Report* - CERN, 2003
- [Henderson et al.] W. Henderson, D. Kendall, A. Robson - *Accounting for Clock Frequency Variation in the Analysis of Distributed Factory Control Systems* - <http://ieeexplore.ieee.org>
- [Stevens et al.] W. R. Stevens, B. Fenner, A. M. Rudoff, - *Unix Network Programming, The Sockets Networking API, 3rd Ed.* - Addison-Wesley 2004.
- [E-Series Datasheet] *Force 10 E-Series Datasheet* - <http://www.force10networks.com>
- [Shorten et al.] R. N. Shorten, D. J. Leith, J. Foy, R. Kilduff, *Analysis and design of congestion control in synchronised communication networks*. Proc. 12th Yale Workshop on Adaptive & Learning Systems