LHCb DAQ network upgrade tests

Flavio Pisani
Supervisor: Daniel Hugo Cámpora Pérez

September 27, 2013

Abstract

My project concerned the evaluation of new technologies for the DAQ network upgrade of LHCb. The first part consisted in developing and OpenFlow-based Clos network. This new technology is very interesting and powerful but, as shown by the results, it still needs further improvements.
The second part consisted in testing and benchmarking 40GbE network equipment: Mellanox MT27500, Chelsio T580 and Huawei CloudEngine 12804. An event-building simulation is currently been performed in order to check the feasibility of the DAQ network upgrade in LS2. The first results are promising.

Chapter 1

Development and testing of an OpenFlow-based Clos network

1.1 The Clos network topology

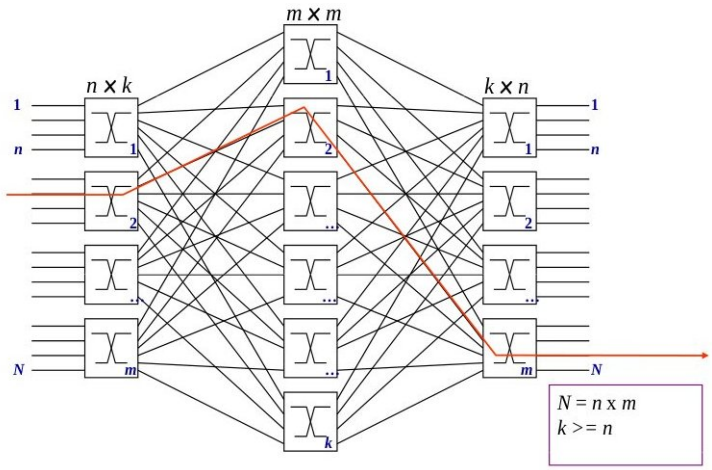

Figure 1.1: The Clos network topology

The Clos network is a three-layer network topology is composed by an ingress layer, a middle layer and an egress layer. The three layers are connected as showed in Figure 1.1. In order to achieve the maximum bandwidth, two packets can't share the same link, otherwise the bandwidth will be split between the two. Requiring maximum bandwidth implies more restrictive conditions on the number of devices in the middle layer. If we want to reach maximum bandwidth regardless of the traffic condition over the network without rearranging the previously established connections, the minimum number of middle layer devices is given by the *strict non-blocking condition*:

$$k \geq 2n - 1 \tag{1.1}$$

To use this kind of topology, we need to choose a free path from source to destination; to correctly do so we need to take into account the state of the whole network. This can be achieved with a centralized network controller.

1.2 Introduction to OpenFlow

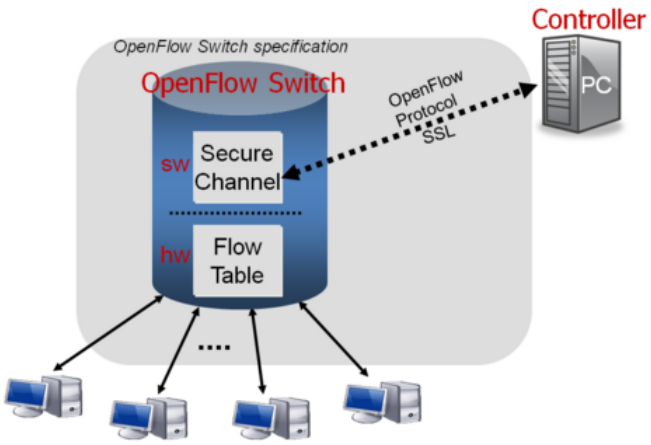

Figure 1.2: Typical OpenFlow configuration

OpenFlow is a communication protocol which gives access to the forwarding plane of a switch or a router. Using the OpenFlow protocol the switch is connected to the controller, as shown is Figure 1.2; all the data path is managed by the switch using the flow tables, while the control path is managed by the controller via the OpenFlow connection.
The flow tables are a more comprehensive version of the IP routing tables used by the common switches. Each flow table has matches, which can be layer-2, layer-3 or transport-layer fields of the header, and a chain of action which, together with selecting the output port, can also make changes to the header of the packet itself.
Every time a new packet is incoming, the switch tries to match it against the flow entries that are already defined; if the packet is matching, the defined actions are applied; if the packet is not matching, any entry is send to the controller, which is able to add, edit or remove flow entries.
This very powerful and flexible technology is however very young, and despite a very active community of developers is not ready for production yet. Further information about OpenFlow can be found at `http://www.opennetworking.org`.

1.3 Basic operation of the Clos network controller

The controller has been developed using the Trema framework. Trema is an open source, event-based framework for developing OpenFlow controllers in C and Ruby developed by NEC.
The controller is capable of forwarding packets from any to any host of a non-blocking three layer Clos network. The packets are destined according to their IP address. The IP-port configuration is loaded statically from three configuration files, one for each layer. The controller is split in three different parts, one for each layer of the network; on top of these three sections there is a dispatcher which passes the packet to the right part of the program. The association between network layer and packet is made checking the input port of the latter. The association between input and network layer must be provided in a configuration file. In order to keep the code modular the dispatcher also changes the data structure sent to the correct handler, so the three controllers could run as three independent programs.

Ingress/Egress layer controller

This part of the controller takes care of choosing the free route for the packet, deleting unused routes and updating the state of the network. Each switch in the ingress and egress layer is associated to an internal device number; this can be related to the IP address of a connected host by the controller, using hash tables. For keeping track of the state of the network, a three dimensional matrix $N_{dev} \times N_{dev} \times N_{mid}$ is used, where N_{dev} is the number of egress/ingress devices and N_{mid} is the number of middle switches, i.e. the number of different routes for given source and destination. Every time you take a particular route you have to increase its occupancy counter for each matrix element in the same line and row as your destination and source device. When the route is freed, the occupancy counter must be decremented accordingly. A route is free if the occupancy counter is zero.
When a new package is received, the controller does the following:

- deleting from the switch all the flows that use the same incoming port
- freeing the previous route, if present, in the network matrix
- getting the free route to the new destination from the network matrix
- open the new flow for all the traffic with the same source and destination
- forwarding the packet trough the chosen route

For the outgoing packet, static flows are defined as the switch connects.
The ARP traffic is forwarded using one of the possible routes in a round

robin way. No flows are opened for the incoming ARP requests in order to avoid overhead introduced by the controller and OpenFlow. The outgoing ARP packets uses static flows as for the IP ones.

Middle layer controller

The middle layer controller is simply opening the new flow and forwarding the packet according to its actual position a its destination. As for the Ingress/Egress, the ARP traffic is forwarded without opening a new flow.

1.4 Case study set up and results

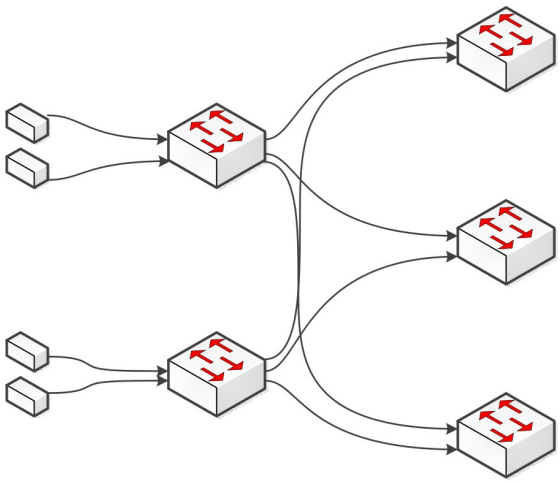

Figure 1.3: Tested network topology. .

The controller has been tested in a 4×4 non-blocking Clos-like network configuration as shown in Figure 1.3. Because all the links are full duplex and we are transmitting only in one direction the behaviour of the network is almost the same as a 4×4 non-blocking Clos network.
The OpenFlow-compatible switch used is the hp3500, a 48 ports 1 Gb/s TOR switch. This switch implements OpenFlow v 1.0 with some limitations. The most important thing to check is that all the flows are accelerated by the switch hardware and are not handled by the switch CPU. If your flow is software-implemented, the obtainable bandwidth will be very low because the CPU will bottleneck the switch.
For the hosts is used a 4 ports 1 Gb/s NIC installed on a PC running an AMD Atlhon X2 processor. The controller is connected on a virtual interface sharing the wire with one host.

1.4.1 Tests performed

One to one transmission bandwidth measurement

To perform this test one of the hosts has been replaced by a laptop computer with an 100 Mb/s NIC.
The bandwidth has been measured with *iperf*.

- **SD**: 94.2 Mb/s
- **FD**: 187.6 Mb/s

All to all traffic

This test has been performed using raw sockets for sending packets, in order to override the kernel routing and deliver packets to an interface connected to the same PC.
In this test we can see the limits of OpenFlow, at least in this particular configuration, and how the controller and the OpenFlow overhead can bottleneck the network. If the traffic is changing destination at a low frequency, the system is able to forward the packets properly; if the traffic is changing destination at a high frequency, many packets are queued in the OpenFlow queue, resulting in drops or higher latency. After some time the switch also fails in closing the connection to the controller; when the switch tries to reconnect the Trema framework does not handle the connection properly and the controller is not able to open new flows or forward packets to the switch. This bug is indeed reported on the Trema web page and a fix should be released in the latest version.

From the tests performed so far, we can conclude that OpenFlow is very powerful and flexible but, at least for this particular case study, is not fast enough for an implementation in the DAQ network.

Chapter 2

Testing 40 GbE networking equipment

2.1 Equipment under test

NIC tested:

- **Mellanox MT27500:** is an Infiniband/40Gbe adapter with RDMA support
- **Chelsio T580:** is a dual port 40GbE unified adapter, with full TOE support, RDMA support

Switches/Routers tested:

- **Huawei CloudEngine 12804:** is a Clos-architecture based core router, maximum 48 Tbit/s switching capacity, two 40G 24-port blades and a 48-port 10GE blade
- **Huawei CloudEngine 6800:** is a TOR switch with 48 10G electrical ports and 4x40G uplinks

2.2 Back-to-Back tests

All the results are obtained using iperf and the *Intel MPI Benchamrck Suite V3.2.4* for the RDMA tests.

Mellanox MT27500

Without RDMA

- **SD:** 37.5 Gb/s
- **FD:** 74.9 Gb/s

RDMA test

- **SD:** 33.5 Gb/s
- **FD:** 66.6 Gb/s

The NIC is working well without RDMA and is a bit slower using RDMA.

Chelsio T580

Without RDMA

- **SD:** 37.8 Gb/s
- **FD:** 75.7 Gb/s

average CPU usage

- **SD sending:** 5.98
- **FD:** 14.72
- **SD receiving:** 5.52

With TOE

- **SD:** 35.6 Gb/s
- **FD:** 32.4 Gb/s

average CPU usage

- **SD sending:** 1.53
- **FD:** 3.36
- **SD receiving:** 4.70

RDMA test

- **SD:** 33.5 Gb/s
- **FD:** 66.6 Gb/s

Average CPU usage

- **SD sending:** 1.53
- **FD:** 3.36
- **SD receiving:** 4.70

The NIC is working well without TOE and as the MT27500 using RDMA. TOE is having problems in the FD test, this issue should be fixed with the next firmware release.

2.3 Huawei switch tests

For comparison the bandwidth has been tested connecting the two NICs with the switch in the middle. The switch has been configured in the *snake* configuration, all the 48 ports of the switch are connected in a chain. The following results show the bandwidth achieved in this configuration.

Mellanox MT27500

Without RDMA

- **SD:** 37.5 Gb/s
- **FD:** 68.1 Gb/s

RDMA test

- **SD:** 28,7 Gb/s
- **FD:** 52.0 Gb/s

The system is working as expected.

Chelsio T580

Without RDMA

- **SD:** 37.8 Gb/s
- **FD:** 74.9 Gb/s

With TOE

- **SD:** 27.3 Gb/s
- **FD:** 19.5 Gb/s

The system is behaving as expected.

The complete set of tests performed on the switch can be found on the wiki page:`http://lbonupgrade.cern.ch/wiki/index.php/Huawei` .

2.4 Event building simulation test

This test is a simulation of the event-building network traffic. At every tick the event fragments are received by a different node, while at the same time all the other nodes send an UDP packet to the receiver, containing the ID of the sending node and the event number. The event number is the same as the tick number. If every node is sending al line speed, the average data received by each node will saturate the bandwidth, but, because of the

bursty nature of the traffic, the instantaneous data rate can be up to $N-1$ times the bandwidth of the line (where N is the number of nodes). In order to avoid, drops the switch must supply an adequate buffer for queueing the outgoing packets.

2.4.1 The sender

The sender sends the UDP packets to all the nodes in the set up in a round robin way. To achieve synchronization, the sender starts sending packets after a given epoch time. To improve synchronization, the destination is changed after a given amount of time calculated from the hardware clock. The payload of the packets is composed by two 32 bit unsigned integers and filled up with trailing zeroes: the first is the machine ID (from 0 to $n-1$), the second one is the progressive event ID. The packets are sent using system calls and regular sockets.
The time interval between to packets can be set at run time.

2.4.2 The receiver

The receiver polls on port 10000 and takes out the machine ID and the event ID from the payload. The fragments received are stored in a ring buffer grouped by event number and machine number. When the buffer is full, the fragments from the oldest event are checked: the missing fragments and received fragments counters are subsequently updated. If a fragment from an event older than the oldest event in the buffer is received, this packet is marked as late. Each late packet is also marked as lost.
Each ten seconds the program prints the various packet counters for each machine and for the whole system, as well as the achieved bandwidth.

2.4.3 Preliminary tests

The first test is to see the maximum bandwidth that can be reached by the system using only one node, to avoid synchronization issues. The bandwidth reached using 9000 B packets is 31.1 Gb/s.

The synchronization problem

In order to be processed in the right way by the receiver, the fragments of an event have to be sent before the next tick arrives. Keeping different machines synchronized for a long period with very low tolerances is a very difficult task. The system used is stable with a tick period of 6 μs; the tick period can be lowered to 4 μs but this makes the system less stable. Sending packets at this rate gives a bandwidth of 11 Gb/s. In the tests performed so far the system seems to be stable. However, more tests are needed such

as: Performing long run test for checking the stability of the system; and increasing the bandwidth using multiple processes in parallel.

Conclusions

The OpenwFlow controller works fine, but is not fast enough for a possible use in the LHCb DAQ network. Further tests can be done to see if the speed improves, such as changing the developing framework, the switch or the controller architecture.
The 40GbE network equipment functions correctly, and all the benchmarks show that it nearly meets specifications. TOE is non performing very well on the Chelsio NIC, but this should be fixed with the new firmware. The Huawei ClouEngine performs well; the event-building simulation is still running but the first results are promising.