Configuring the LHCb Readout Network Using a Database

Lana Abadie

Abstract—**The LHCb readout system is composed of hundreds of electronics boards, an event-building network based on Gigabit Ethernet switches and an online processing farm. The Experiment Control System (ECS) configures the system from the Online Configuration database. This database contains device parameters, the hierarchical structure and the connectivity information of the system. In addition the switches in the event-building network require routing tables that have to be generated according to the connectivity. We apply the Entity Relationship model to represent the connectivity of the system. SQL code builds the routing tables using the information contained in the Configuration database.**

Index Terms—**DAQ network, database, routing table.**

I. Introduction

THE LHCb experiment [1], [2] is one of the four experiments at the CERN Large Hadron Collider (LHC). The LHC, currently under construction, will accelerate two beams of protons in opposite directions close to the speed of light, reaching an energy of 7 teraelectronvolts (TeV), the highest energy ever observed in a laboratory. The LHCb experiment is situated at one of the places around the LHC ring where the beams collide. For more information, please see [3]. The LHCb detector is designed to exploit the large number of b-hadrons produced at the LHC in order to make precision studies of CP asymmetries and of rare decays in the B-meson systems. The Online Data Acquisition (DAQ) system collects data from the front-end electronics and transfers them to a CPU farm for the execution of the software trigger algorithms. There are two kinds of traffic intermixed corresponding to the data for Level-1 trigger (L1) and for the high level trigger (HLT). Details about the DAQ system can be found in [4].

The DAQ as shown in Fig. 1 is composed of the following:

- about 300 front-end electronics boards (FEs) containing diskless embedded PCs the size of a credit-card (Creditcard PCs);
- about 20 switches to reduce the number of links (multiplexing layer);
- about 100 SubFarms to process data—a SubFarm is composed of a SubFarm Controller (SFC)—and about 20 diskless PCs (SubFarm Nodes);
- a readout network to interconnect the data sources (FEs) and the SubFarms.

The monitoring, configuration and operation of all the experimental equipment will be handled by the Experiment Control System (ECS) [5]. All configurable information required for the control of the equipment is stored in the Configuration database [6].

In this paper, we focus on an effective way to configure the LHCb Online Data Acquisition (DAQ) network using this database.

II. Configuration of the DAQ: Methodology

A. Objectives and Requirements

The behavior of any networking device in a DAQ system has to be predictable to control and manage the routing of data. It is also necessary to keep track of all routing tables and configuration files to enable the system to recover after a crash and also to analyze the cause of the failure afterwards.

To satisfy these requirements, we need to configure the routing table for each switch of the DAQ network and reduce the use of the switch "automatic" or "self-management" routing mode (this feature allows the router to choose the routing of data).

The FE boards and the subfarm nodes are diskless. Diskless means they boot via the network—in our case using etherboot [7]—and they use a network file system (NFS) [8] tree installed on a remote server (not visible in Fig. 1) to mount their root filesystem. This remote server is located on a Controls PC. Using NFS means that you cannot use a true dynamic IP address but you need a fixed IP address. This IP address will be assigned using the dynamic host configuration protocol (DHCP) [9] which is based on a configuration file (DHCP configuration file). Etherboot will send a dhcp request to ask for an IP address. The FE boards and subfarm nodes will be identified by their MAC (Ethernet) address. We need to generate the DHCP configuration file to allow the FE boards and the Subfarm nodes to get their IP address and their filename for the boot image. This filename corresponds to the location of their rootfile system, in the NFS server. See the Appendix I for an example of a DHCP configuration file. We also need to generate other configuration files, for the Domain Name System DNS [10] for instance. Thus configuring our DAQ network consists of the following:

- generating routing tables for each switch of the DAQ, i.e., the switches in the readout network and in the subfarms;
- generating configuration files.

In the next sections we will focus on generating routing tables.

B. Use of a Database

To satisfy the previous requirements, we use the Online Configuration database [6] to configure the DAQ system since databases are nowadays a powerful and safe technology to save data for large systems. Information stored in the database should be:

- complete, to be able to start up the network after a crash;

Manuscript received June 27, 2005; revised March 6, 2006, 2005.

The author is with the CERN, CH-1211 Geneva 23, Switzerland and also with GET/INT, 91011 Evry cedex, France (e-mail: lana.abadie@cern.ch).

Digital Object Identifier 10.1109/TNS.2006.873309

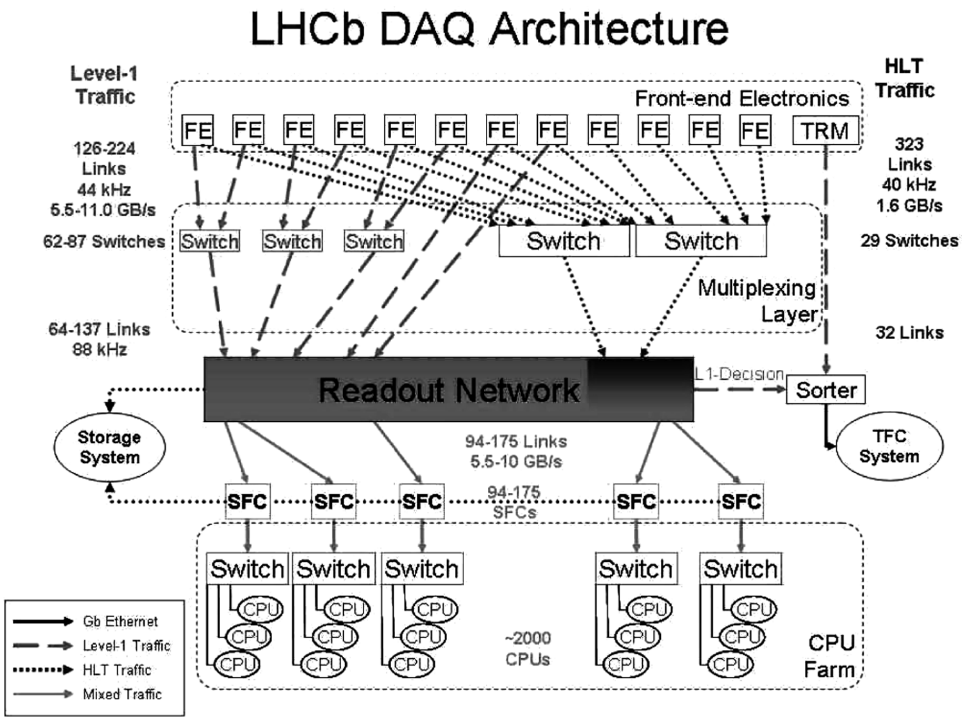


Fig. 1. The LHCb DAQ architecture.

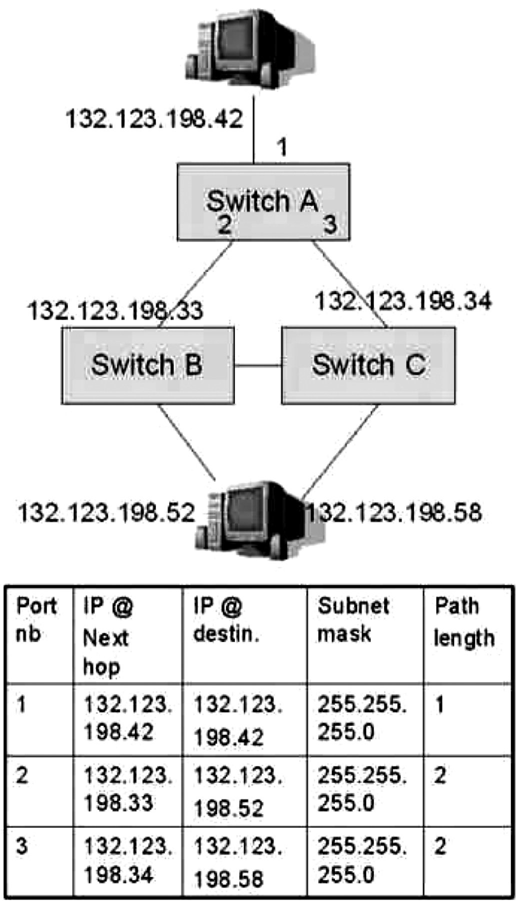
- scalable, to support any extension or changes to the network topology or in the hardware of the components;
- consistent, to avoid errors and incompatibilities in routing tables or in configuration files;

C. *Routing Table Definition*

A **routing table**, stored in a router or a switch, keeps track of routes to particular network host interfaces. There is one routing table per switch or router. We can distinguish two types of routing tables (we describe the different entries of routing tables as they are stored in swicthes and routers).

- IP routing tables for routers (layer 3 in the OSI model [11]). The IP routing table has the following entries.
  - —Port number to forward data (Port nb in Fig. 2).
  - —IP address of the next hop (IP @ Next Hop in Fig. 2).
  - —IP address of the destination (IP @ destin. in Fig. 2).
  - —Subnet Mask [12]: property of a subnet i.e., a pool of networking devices (Subnet mask in Fig. 2).
  - —Path length or round trip down to the destination (Path length in Fig. 2).

  Fig. 2 shows the IP routing table of the switch A. The columns represent the entries quoted previously. A row of a routing table describes which port to take to reach a given destination. For instance, a packet destinated to 132.123.198.52 arrives in switch A. Switch A looks for 132.123.198.52 in the column **IP @ destin** of its routing table. Then, it forwards the packet to the port number (read from **Port nb** colum) associated with this destination, i.e., **port 2**, referring to Fig. 2.
- MAC routing tables for switches (layer 2 in the OSI model). The MAC routing table has the following entries:
  - —Port number to forward data;
  - —MAC address of the next hop;
  - —MAC address of the destination;
  - —VLAN (Virtual Local Area Network) [13] prefix which is a characteristic of a pool of networking devices;
  - —Path length or round trip down to the destination.



Port nb	IP @ Next hop	IP @ destin.	Subnet mask	Path length
1	132.123. 198.42	132.123. 198.42	255.255. 255.0	1
2	132.123. 198.33	132.123. 198.52	255.255. 255.0	2
3	132.123. 198.34	132.123. 198.58	255.255. 255.0	2

Fig. 2. Example of an IP routing table of switch A.

Different kinds of routing algorithms exist; in this paper, we use the most common one-the shortest path-which consists in minimizing the number of network devices crossed i.e., number of hops.

III. Representation of the DAQ Network

We will focus on determining the necessary information to be stored in the database and on finding an efficient table schema.

A. Network Definitions

A network can be seen as a graph composed of a certain number of **nodes** and **links**. Our network consists of the following.

- **Host nodes** (or "hosts") correspond to equipment which process data such as PCs. Host nodes are either sources (sending data) or destinations (receiving data).
- **Switch nodes** (or "switches") are intermediate nodes which forward data through the network.
- Two nodes may be inter-connected by a **link**. The end points of a link are called **ports** which are attached to nodes. The ports of a node are numbered.
  A link consists of the following:
  — a node and a port number for the start and the end of the connection;
  — a type corresponding to the data type that the link carries such as control or data traffic (Fig. 1);
  — an orientation: bidirectional or unidirectional (in our case all the links are bidirectional);
  — a status: either functional or dead.
- **A network** interface card (NIC) implements a port of a node. Each NIC is characterized by a MAC address and an IP address. Therefore, every port on a switch has a Mac address and an IP address.
- **A path** is a sequence of distinct connected nodes (or links). **A routing path** is a special type of path which starts from a switch node and terminates with a host node.
- **The maximum hops** correspond to the maximum number of links in a path. This parameter is a characteristic of the network. It is noted M.

B. Table Design Methodology

We use the the entity relationship model (ERM) because of its simplicity and its flexibility to model many systems. The ERM represents a collection of entities and their relationships. An entity represents an object or a concept. It is modelled using a table whose rows correspond to instances of this object. The table's columns represent the attributes of the object. In our case, **nodes, links, ports, network interfaces** are examples of entities. A relationship describes how entities are linked. For example, the parent-children relationship means that one or several attributes of the children tables will point to one or more attributes of the parent table (Foreign Keys).

In order to design the database schema, we collect use cases. A use case is an example of an application that will use the database schema. By collecting the main use cases, we can identify the requirements of a system. The main use cases for the DAQ network are as folows.

- We need to know in advance the data path connecting a FE board (source) to a SFC (destination). Referring to Fig. 2, if the PC identified by IP address 132.123.198.42 sends data to the destination 132.123.198.52, we should know that the data will first enter the switch A on its port number 1 and get out on its port number 2, then enter switch B via its interface 132.123.198.33 and get out from the port number 5 (for instance, although not specified in the figure) and finally reach its destination. This is different from the default operation of a network in which a node is likely to dynamically determine the routes for a given data packet.
- The switches in the DAQ network can be either routers or switches. In the case of LHCb, this has not yet been defined. So we need to set up both IP and MAC routing tables for each switch. It means that we have to provide the entries as defined in Subsection II.C.
- In our DAQ network (see Fig. 1), a FE board (source) cannot directly send data to a SubFarm PC but only to a SFC which then will process the data and forward it to the SubFarm PCs. Same remark for the switches in the SubFarms. The SFC distributes the event to the Subfarm PCs which run trigger algorithms. Then the trigger decisions are sent back to the SFC. So in the routing tables of the readout network, switches will have the entries-MAC address destination and IP address destination-corresponding to FE boards and SFCs. The routing table of the switches in the SubFarms will have the entries—MAC address destination and IP address destination-corresponding to SubFarm PCs and to SFCs. Given these requirements, a routing path is a sequence of distinct **intermediate nodes** (or links) which starts from a switch—the one we want to generate the routing table for-and ends with a host node. A routing path has **one and only one host node** which is the terminal node.

C. Relational Database Design for the DAQ System

Fig. 3 shows an optimized database schema which represents our DAQ network system and allows an easy implementation of the use cases mentioned earlier.

The primary key of a table, used to refer to other tables, is in bold. The arrows represent the relationships (foreign keys) between the different entities (or tables).

In the database, we distinguish the following two types of tables.

- Tables filled by the user (typically the network administrator of the DAQ). The names of these tables are in upper case in Fig. 3.
  The name and characteristics of the nodes (host and intermediate) are stored in the **DEVICE** table. The links between node interfaces and their properties are stored in the **CONNECTIVITY** table. The **IP_MAC** table contains the IP and MAC adresses corresponding to the node interfaces and the **PORT** table contains port properties. This type of information cannot be generated automatically.
- Tables derived from the information stored in the database (table names in lower case in Fig. 3) such as routing tables, configuration files. Section IV describes the algorithms used to derive these tables.
  The **routing_table** table contains both the MAC and IP routing tables of a switch: there is one **routing_table** table per switch.

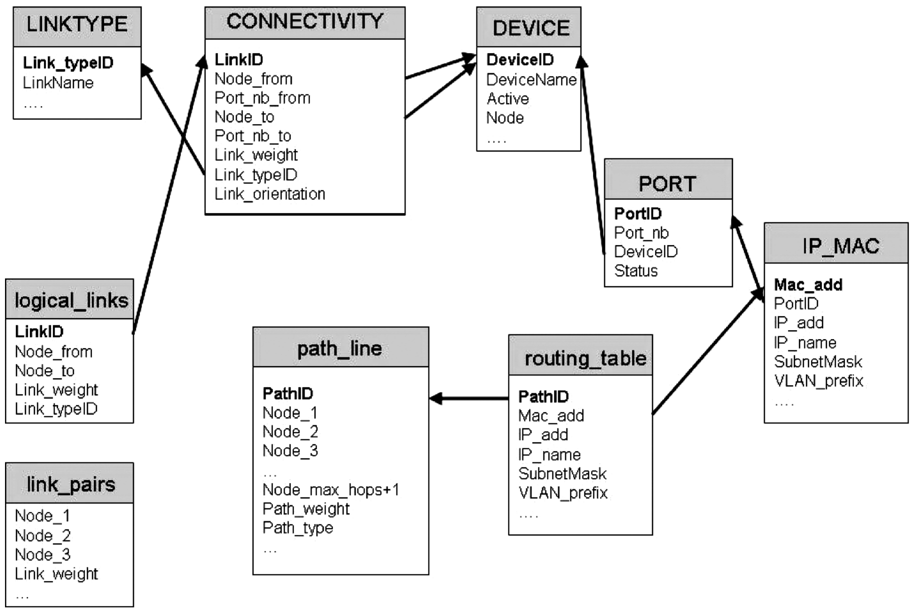


Fig. 3. Table design for the DAQ system based on the ERM.

The **path_line** table contains all the complete routing paths starting at a given switch: there is one **path_line** table per switch. Although the **path_line** table does not comply with the relational rules (because the number of columns depends on M), we decided to proceed with this design for efficiency reasons.

IV. Generating a Routing Table

This section explains how we generate the routing table for a particular switch node using the table schema of Fig. 3.

The main idea to generate a routing table for a switch A is to determine all the routing paths-designated by their nodes-starting from switch A and with a length less than M (maximum number of hops) which will be stored in the **path_line** table.

First we define the concept of link and path weights. Then, we introduce the concept of **logical_links** and **link_pairs** which are used to build the routing paths. Finally, we present an algorithm to find optimal routing paths.

A. Link and Path Weights

In the **DEVICE** table (Fig. 3), the **Node** attribute, set by the user, is equal to 1 if it is a host node and 0 if it is an intermediate node.

In the **CONNECTIVITY** table from Fig. 3, the **Link_weight** of a link L (noted $W(L)$) is set to the folowing:

- 0 if the link is between 2 intermediate nodes;
- 1 if the link is between a host node and an intermediate node;
- 2 if the link is between 2 host nodes (although not used here).

Fig. 4 illustrates the concept of link_weight.

The path weight $(W(P))$ is defined as the sum of the link weights along the path (see Fig. 5).

By using the definition of the routing path, we can derive the following theorem which will be used to find the subset of routing paths from paths.

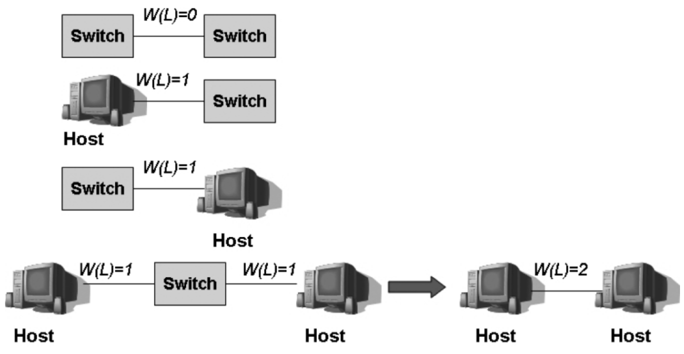


Fig. 4. Link weight concept.

A path P of length J is a routing path of length J

$$\Longleftrightarrow \begin{pmatrix} W(P) = \sum_{i=1}^{i=J} W(L)_i = 1 \quad \text{and} \\ W(P) = \sum_{i=1}^{i=J-1} W(L)_i = 0 \end{pmatrix}$$

where $W(L)_i$ corresponds to the weight of the i-th link in the path P. The proof is given in the Appendix II.

B. Algorithm Initialization

The input parameters of the routing algorithm are the name of the intermediate node and M (**max_hops** in Fig. 3). This concept of maximum length is necessary as finding all the paths in a graph is a NP-complete (Non-Polynomial) problem [14].

The algorithm to generate the routing table is based on the following steps.

- Create the **logical_links** table which is the logical (or macroscopic) view of the system. It is derived from the **CONNECTIVITY** table (cf Fig. 3).
  In the **logical_links** table, multiple physical links between two nodes are represented by one single link. This step is performed to reduce the redundant paths which leads to inconsistent routing tables. Referring to Fig. 6, the four

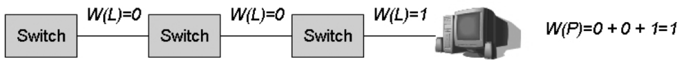
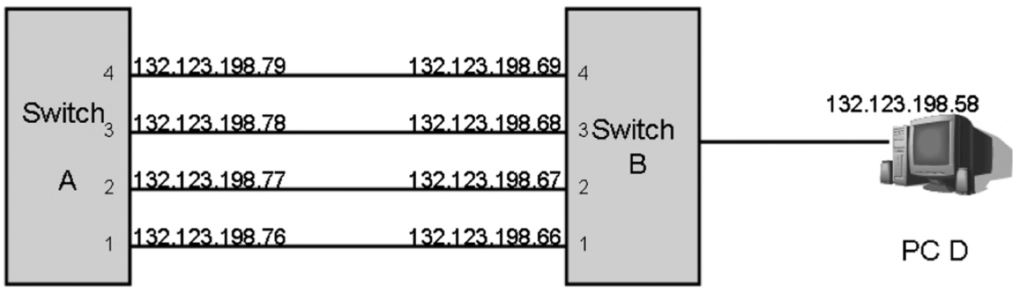
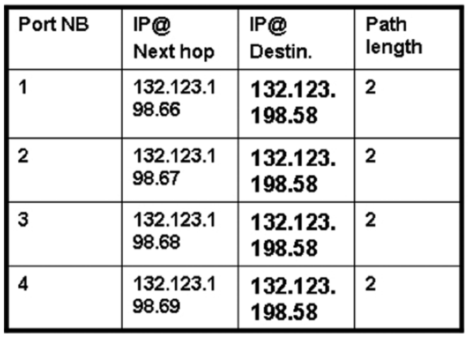


Fig. 5. Example of a routing path.





Port NB	IP@ Next hop	IP@ Destin.	Path length
1	132.123.1 98.66	132.123. 198.58	2
2	132.123.1 98.67	132.123. 198.58	2
3	132.123.1 98.68	132.123. 198.58	2
4	132.123.1 98.69	132.123. 198.58	2

Problem : More than one routing path to PC D.

The routing table of switch A as defined here is ambiguous

Fig. 6. Routing table of switch A: an ambiguous routing table.

links between Switch A and Switch B are replaced with one single link. Our switches have the "Link Aggregation" feature [15]. It means that we can group several physical links and consider them as one "logical" link. The switch will then forward the packet through this logical link (it's the switch which decides to which physical port the data should be forwarded). The reason of this "aggregation" operation is to ensure that there is only one routing path per destination and to avoid misconfiguration which often leads to broadcasts and finally screws the network up. For instance, assume that a packet destinated for 132.123.198.58 arrives at Switch A. Switch A looks for 132.123.198.58 in its routing table and sees four possible ports to forward the packet (1, 2, 3 or 4). If these 4 physical links are not aggregated, switch A will not know from which port the packet should get out.

- Create the **link_pairs** table in which all valid pairs of successive links are stored. To create the **link_pairs** table, we perform a self-join of the **logical_links** table. It is like a Cartesian product of the table with itself with the following constraints.
  - Link1 is defined by (**Node_1, Node_2**) and Link2 is defined by (**Node**_2, **Node**_3) (referring to Fig. 3) where **Node**_2 corresponds both to Node_to of link1 and to Node_from of link2.
  - The link_weight of link1 must be equal to 0 because we want to find routing paths (i.e., it starts from a switch and, as we are looking for pairs of links, we exclude the switch-host links).

  The **logical_links** and **link_pairs** tables are not switch dependent. They are re-generated only when a change occurs in the **CONNECTIVITY** table.
- For each switch, a **path_line** table is created and initialized with the elements from the **link_pairs** table which have this switch as a starting node. We then have found paths which have a length equal to 1 or 2. These paths are inserted in the **path_line** table. If the length path is equal to 1, then the path is inserted as a row into the **path_line** table using the columns **Node_1, Node_2**. If the legnth path is equal to 2, then the path is inserted as a row into the **path_line** table using the columns **Node_1, Node_2, Node_3**.

C. Algorithm Body

This subsection explains how we find the routing paths.

We iterate over i which represents the length. At each iteration i, a join between the **link_pairs** and the **path_line** tables is executed. It means that a path P, with $W(P)=0$ (i.e., having not reached a host) is completed with an element from **link_pairs** whose first link is equal to the last link of P. If no such pair exists, the path P is removed. There may be more than one pair which verify the conditions. Thus if there are N possible pairs, these N possible pairs will be appended to P and there will be N new paths (i.e., N new rows in the **path_line**). At the end of iteration i, we have found all the paths of length i and inserted them in the **path_line** table and we have filled the $i+1$ Node columns of **path_line** table.

For each iteration i, the detailed description of the steps is as follows.

- **Step 1**. In the **path_line** table, select the paths P where $W(P)=0$. (The last column filled is **Node_*i***).
- **Step 2**. Find all the possible pairs of links where (**Node**_$i-1$, **Node**_i) is equal to (**Node_1, Node_2**) of **link_pairs** table and check that there is no cycle (i.e., a node appearing twice in the path).

- **Step 3**. Insert these new valid paths in the **path_line** table. So the **Node_1** to **Node_**$i+1$ columns are filled in.
- **Step 4**. Delete the old paths where $W(P) = 0$ and **Node_**$i+1 = 0$.
- **Step 5**. Increment i by 1.
- **Step 6**. Stop the loop if i is greater than M or if all the paths are routing paths, i.e., all paths verify $W(P) > 0$.
- **Step 7**. Finally, we resolve multiple paths to a given destination by selecting the shortest routing path. These shortest path are stored in a temporary table called **shortest_routing_path**. This table is not mentioned in the Fig. 3.

This algorithm works in all network architectures including full mesh layouts. A more detailed version in "pseudo-code" is given in Appendix III.

D. Routing Table

The **path_line** table contains all the routing paths of a switch. The **shortest_routing_path** table contains the shortest routing path to a given destination. Then we do a mapping of the information stored in the **shortest_routing_path** table to the two types of routing tables (IP and MAC).

V. Implementation and Test

The Configuration database is a central database based on Oracle technology (Oracle 9i) [16] because it is well supported at CERN. Also Oracle is operating system independant. To generate the routing tables, we have developed a PL/SQL package. PL/SQL (Procedural Language) is a proprietary but portable language developed by Oracle [17]. PL/SQL is a server side language interpreted by (Oracle) DBMS servers. The main advantage to use PL/SQL is we can build functions and procedures which perform SQL queries in the server-side. The execution time is faster as we reduce the number of round trips over the network. We have tested this package with different layouts of the DAQ network system which has been represented with 2350 nodes and 4216 links. In our tests, M was set to 10 and the real maximum path length found was 7.

VI. Conclusion

This paper proposes a way to configure a network with a focus on how generating routing tables using a database. The next step is to implement other packages to generate configuration files by still using the same representation. One ongoing project is to automatically generate a DHCP configuration file.

Appendix I
Example of a DHCP Configuration File

```
#
# Sample dhcpd.conf file
# Check with your network
# admin for local settings
#
# main Class B network
subnet 127.238.0.0
# subnet mask (Class B)
netmask 255.255.0.0 { not authoritative; }
group {
# gateway address
option routers 127.238.1.1;
# the boot image
filename "vmlinuz-2.4.26-ccpc08.nbi";
# the tftp-server IP address
server-name "127.238.142.63";address
# the nfs-server IP address
next-server "127.238.142.63";
option root-path
"127.238.142.63:/usr/local/defaultroot";
# here now the entries for the Hosts
host pclbcc02 {
hardware ethernet "01:02:03:04:05:05";
fixed-address 127.238.142.77;
          }
      }
```

The usual and necessary entries we find in a dhcp configuration file are in bold. Comments are in italic.

Appendix II
Proof of the Theorem

Proof of: *A path P of length J is a routing path of length J*

$$\Longleftrightarrow \begin{pmatrix} W(P) = \sum_{i=1}^{i=J} W(L)_i = 1 \quad \text{and} \\ W(P) = \sum_{i=1}^{i=J-1} W(L)_i = 0 \end{pmatrix}.$$

A path P of length J is a routing path of length J $\Longleftrightarrow$ P is a sequence of J + 1 nodes which are all intermediate nodes (J nodes) except the terminal one which is a host node.

So P is a routing path of length J $\Longleftrightarrow$ P is a sequence of J-1 links between intermediate nodes and one link intermediate-host nodes.

As the link weight between two intermediate nodes is equal to 0 and the link weight between an intermediate node and a host node is equal to 1, we have the folowing.

P is a routing path of length J $\Longleftrightarrow$

$$\forall i \in [1, J-1], W(L)_i = 0 \quad \text{and} \quad W(L)_J = 1$$

As all the link weights are greater or equal than 0, we have

$$\forall i \in [1, J-1], W(L)_i = 0 \Longleftrightarrow \sum_{i=1}^{i=J-1} W(L)_i = 0$$

and we also have

$$\forall i \in [1, J-1], W(L)_i = 0 \quad \text{and} \quad W(L)_J = 1$$
$$\Longleftrightarrow \sum_{i=1}^{i=J} W(L)_i = 1.$$

Hence, *A path P of length J is a routing path of length J*

$$\Longleftrightarrow \begin{pmatrix} W(P) = \sum_{i=1}^{i=J} W(L)_i = 1 \quad \text{and} \\ W(P) = \sum_{i=1}^{i=J-1} W(L)_i = 0 \end{pmatrix}.$$

APPENDIX III
THE CORE OF THE ALGORITHM

```
# Initialization: paths of length $< 3$ have
# been found using the link_pairstable.
# i represents the length.
$i = 3$;
WHILE ($\text{stop} = 0$ and $i \leq \text{max\_hops}$)}
      # steps 1 and 2 join between link_pairs
      # and path_line tables
      SELECT f.Node_1,..., f.Node_$i$, t.Node_3
      FROM link_pairs t, path_line f
      # 4 conditions to respect:
      # (Node1,Node2)=(Nodei-1,Nodei)
      WHERE
      $\text{t.Node\_2} = \text{f.Node\_}i$ AND $\text{t.Node\_1} = \text{f.Node\_}i\text{-1}$ AND
      # check $W(P) = 0$ and same link type
      $\text{f.Path\_weight} = 0$ AND $\text{f.Path\_type} = \text{t.Link\_type}$ AND
      # check there is no cycle,
      # i.e., twice the same node.
      $[\prod_{k=1}^{k=i}(\text{t.Node\_3} - \text{f.Node\_}k)] \neq 0$.
      # step 3:insert the result of select
      # (new paths of length i)
      INSERT the new valid paths found
      (of length $i$) in the path_line table.

      # step 4: delete old paths i.e., where
      # $W(P) = 0$ and path length $= i - 1$
      DELETE path_line the rows WHERE $\text{Path\_weight} = 0$
      AND Node_$i + 1 = 0$.

      COUNT the number of rows where $\text{Path\_weight} = 0$
      in the path\_line table.
      # step 5 and 6: stop the loop if
      # all the paths are routing paths
      # otherwise increment i by 1

      IF all the $\text{Path\_weight} > 0$ then
            $\text{stop} = 1$;
      ELSE
            $i = i + 1$;
      END IF;
END WHILE;
```

ACKNOWLEDGMENT

The author would like to thank A. Barczyk, J.-P. Dufey, B. Gaidioz, and N. Neufeld for their fruitful discussion on the LHCb DAQ network. The author would also like to thank supervisor E. van Herwijnen, for his careful reading of the paper. His numerous comments, suggestions, and corrections have substantially improved the quality of this text.

REFERENCES

[1] *LHCb Technical Proposal* (in The LHCb Collaboration), , Feb. 1998 [Online]. Available: http://lhcb-tp.web.cern.ch/lhcb-tp/index.htm
[2] *LHCb Reoptimized Detector Design and Performance* (in The LHCb Collaboration), , Sept. 2003 [Online]. Available: http://lhcb.web.cern.ch/lhcb/TDR/reoptdr.pdf
[3] *The LHCb LOI (Letter of Intent)*, [Online]. Available: http://lhc-b.web.cern.ch/lhc-b/loi/LOI.html
[4] *LHCb Online Computing, Trigger System Technical Design Report* (in The LHCb Collaboration), , Oct. 2003 [Online]. Available: http://lhcb.web.cern.ch/lhcb/TDR/TDR.htm
[5] C. Gaspar, "An integrated experiment control system, architecture, and benefits: The LHCb approach," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 3, pp. 513–520, Jun. 2004.
[6] *LHCb Online Configuration Database*, [Online]. Available: http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/
[7] *Etherboot Protocol Details*, [Online]. Available: http://www.etherboot.org/
[8] *NFS: Network File System Protocol specification - RFC 1094*, [Online]. Available: http://www.faqs.org/rfcs/rfc1094.html
[9] *Dynamic Host Configuration Protocol DHCP: RFC 2131*, [Online]. Available: http://www.faqs.org/rfcs/rfc2131.html
[10] *Domain Name System DNS: RFC 1035*, [Online]. Available: http://www.faqs.org/rfcs/rfc1035.html
[11] *The Seven OSI Layers Model Created by ISO*, [Online]. Available: http://www2.rad.com/networks/1994/osi/layers.htm
[12] *Internet Standard Subnetting Procedure: RFC 950*, [Online]. Available: http://www.faqs.org/rfcs/rfc950.html
[13] *Virtual LANs (Local Area Network) IEEE 802.1Q*, [Online]. Available: http://www.ieee802.org/1/pages/802.1Q.html
[14] *Finding Paths in a Graph, a NP Problem*, [Online]. Available: http://mat.gsia.cmu.edu/mstc/heurnote/node23.html
[15] *Link Aggregation Specifications (Ethernet Protocol 802.3ad)*, [Online]. Available: http://www.ieee802.org/3/ad/index.html
[16] *Oracle Technology Network Website*, [Online]. Available: http://www.oracle.com/technology/index.html
[17] *Oracle PL/SQL language Documentation*, [Online]. Available: http://www.oracle.com/technology/tech/pl_sql/index.html