Generating the DHCP config file using confDB
LHCb Note

Internal Note

Issue: 1
Revision: 0

Reference: LHCb-2006-038
Created: May 8, 2006
Last modified: June 26, 2006

Prepared by: Lana Abadie

Abstract

The present note describes a method to generate the dhcp config file using the information contained in the configuration database. It also presents how it can be used according to the network topology.

Document Status Sheet

1. Document Title: Generating the DHCP config file using confDB LHCb Note			
2. Document Reference Number: LHCb-2006-038			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
Draft	1	May 8, 2006	First version.
Draft	2	May 19, 2006	Correct English mistakes.
Draft	3	June 8, 2006	New subsection added

Contents

List of Figures

List of Tables

1 Introduction

A DHCP server allows a client PC to get its IP address and its boot image. Thus all the sub-farm nodes, TELL1 boards and readout supervisors will need to be configured. In the LHCb online system, in principle, each controls PC will host a DHCP server. A DHCP server will configure all the clients controlled and monitored by the controls PC where it sits. This note describes a method and its implementation to dynamically generate the dhcp config file using the information stored in the configuration DB.

1.1 DHCP protocol

This protocol [1] allows a host which connects to the network to dynamically obtain its network configuration. The DHCP server will attribute an IP address, an IP name (or hostname), a boot image (file which will allow the host to get its configuration) to the newly connected host. When a host starts up, it has no network configuration. It will send a DHCPDISCOVER message (special broadcast with IP destination equal to 255.255.255.255) to know where the DHCP servers are located. The DHCP server will respond by a DHCPOFFER (also a broadcast message as the host may not have an IP address) which suggests an IP address to the host (DHCP client). The host sends a DHCPREQUEST to accept the IP address. The DHCP server sends a DHCPACK to acknowledge the attribution. The DHCP server can attribute dynamically an IP address or statically or both. It is fixed by the network administrator. If the address is attributed dynamically, it will be valid for a certain period. Moreover in the case of a dynamic attribution, it can take time or even fail (if all IP addresses are taken). In the case of a static attribution, the DHCP server has a dhcp config file defined by the network administrator which looks like as shown in Fig. 1.

In LHCb DAQ, the DHCP config file will be static to avoid any failures or wrong configuration at start up.

1.2 DHCP config file structure

When a host sends a DHCPDISCOVER message, the DHCP server will look for the entry corresponding to the MAC address of the host in the dhcp config file. It will provide with all the information namely (referring to Fig. 1):

- IP address which corresponds to the **fixed-address** information (which is static, always valid);
- IP name of the host which corresponds to **host**;
- Mac address of the host which corresponds to **hardware ethernet**;
- IP address of the gateway which corresponds to **option routers**;
- IP address of the tftp-server given by **server-name**;
- IP address of the NFS server (for boot image location) which is given by **next-server**;

```
#
# 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;
   }
  }
```

Figure 1 Example of a DHCP config file

- The boot image which is given by **filename**.

At the beginning of the dhcp config file, some generic options are fixed. Then there are options related to IP subnets. Then groups are defined. A group is a set of hosts which have the same filename and server-name. In LHCb DAQ, there will be one NFS server, one tftp-server and one gateway. The hosts will load their boot image from the tftp server. The nfs server is used as local disks for the sub-farm nodes (in the DAQ, sub-farm nodes are diskless). So this information will be part of the generic options.

2 Methodology

2.1 Getting the host nodes

In the dhcpd.conf, we have to specify the hosts which will be configured by the given DHCP server. To find all the hosts that a DHCP server needs to configure, the Perl script generates the destination table of the given DHCP server using the **CONNECTIVITY** table stored in the configuration DB. It means that we search for all the possible hosts that can be configured by the given DHCP server. The current topology (see Fig. 2) is that each controls PC is connected to all the nodes of a sub-farm via a switch.

This table is generated using a PL/SQL script. To get more details about the algorithm, please refer to this document [2]. Besides the host name, the destination table contains a column which indicates from which network interface the host will be configured. A network interface can be viewed as a port. A node can have one or several network interfaces. In the case of the LHCb DAQ, each node has two network interfaces, a controls network interface and a data network interface. The DHCP server will configure only the controls interface. The configuration of the data network interface is not fixed yet. A network interface is uniquely identified by a MAC address.

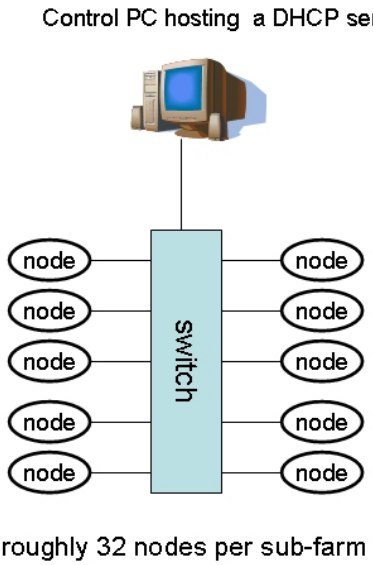

Figure 2 Current topology of the control network

Then, on the fly, the Perl script queries against the configuration DB for the IP address, IP name and Mac address related to each pair of (host name, network interface).

2.2 Getting the boot image

The boot image information is usually linked to the device type (CPU architecture, kernel version for instance). In other words, in most cases, all the farm nodes will have the same boot image as they are of the same type. The same remark can be done for the TELL1 boards (per subsystem) and readout supervisors. However it may occur that a host has its specific boot image. To get the right boot image, we perform the 2 following steps:

- Step 1 : we look if the given host name has a specific boot image in the **DEVICE_BOOTING_TABLE** stored in the configuration DB. If we find an entry corresponding to this node in this table, we select it. If no, we go to step 2.
- Step 2 : if there is no boot image associated specifically for this host node, we attribute it the boot image associated with its type. We get it from **DEVICETYPE_BOOTING_TABLE**.

So at the end, for each host, we have the required entries, **hardware ethernet**, **fixed-address** and **filename** to generate the dhcp config file.

2.3 Getting the subnet ID

In the configuration DB, we store the subnet mask instead of the subnet ID. However, one can compute the subnet ID given the subnet mask and the IP address. The idea is very common. Let's take the following example :

IP address : 137.192.25.15

Subnet Mask : 255.255.255.0

- 1. Convert the IP address and the Subnet Mask to binary formats:

  IP address : 100001001.11000000.00011001.00001111

  Subnet Mask: 11111111. 11111111.11111111.000000000

  We consider these two numbers as 2 vectors (32×1).

- 2. Perform a pointwise multiplication of the 2 vectors.
  The subnet ID is then equal to 100001001.11000000.00011001.00000000
- 3. Convert it into decimal format : 137.192.25.0
  Then we group the hosts by subnet IDs as follows:
  subnet 137.192.25.0 netmask 255.255.255.0 {
  group {
  host pctest32 {
  hardware ethernet 00:00:A2:11:25:B4; fixed-address 137.192.25.15;}
  host { ... } }
  subnet ...

At this stage, we have all the necessary information to build up the dhcp config file. The next section will focus on the implementation.

3 Implementation

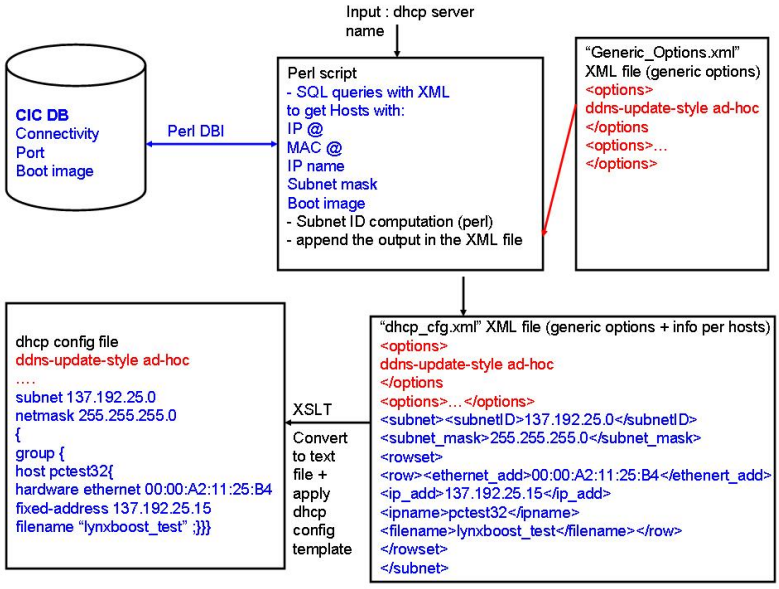

Figure 3 Implementation design

Fig. 3 describes the principles of the implementation.

3.1 Prerequisites

To be able to use this application, you need:

- Linux operating system
- Perl
- Perl DBI module to query against the configuration database
- Perl XSLT module to allow the use of XSLT functionalities
- Connectivity and boot image information tables filled accordingly in the configuration DB [4]
- Tns_names.ora to locate the configuration db

3.2 Handling generic options

First, the user enters the generic options by editing the ”dhcp_options.xml” XML file provided with the application. There should be one option per line. These options will be properly formatted (put between <options> ... </options>and written in another XML file, called ”dhcp_file.xml”. Referring to Fig. 3, the generic options are written in red. Then, the user executes the Perl script, with the dhcp server name as input argument. It is case-sensistive. The Perl script performs three important steps. It queries the necessary information, writes them in ”dhcp_file.xml” and generates the dhcp config file using XSLT.

3.3 Use of XML

The script queries the necessary information against the configuration DB using XML features of Oracle. It means that the results are already encapsulated with XML tags as it is shown in blue in Fig. 3. After querying the information, a Perl routine is called to compute the subnetID. Then hosts with network configuration information are regrouped by subnet ID. Everything is printed and appended in ”dhcp_file.xml”, i.e. after the generic options as shown in Fig. 3.

3.4 Use of XSLT

XSLT is used to convert the content of ”dhcp_file.xml” into a dhcp config file. To do so, there is a xsl file which reads the XML file. It converts the XML tags into words which are understandable by a dhcp server.

For instance, the XML tag <ethernet_add> is converted into hardware ethernet, <ipname> to host, etc. The output of this operation is the dhcpd.conf. All the files created and used are located in the same directory. The dhcpd.conf will not be copied in /etc/... It is for security reasons.

4 Example of usage

4.1 Excluding hosts

If for any reasons, you want to exclude hosts to be configured by a given dhcp server, you can do so by disabling them. There is C-function (with bindings to Python and PVSS) called **UpdateMultipleDeviceNodeUsedByType** or **UpdateMultipleDeviceNodeUsed** .

The first one is to be used if the hosts are characterized by a common prefix. Conversely it means if a host starts with this prefix, then it will be excluded. If it’s not the case, use the second function.

When you have disabled all the hosts you don’t want to be configured by the given DHCP server, execute the Perl script.

Then once this step is performed, include the hosts back again. You use exactly the same C-function as previously. These two functions are documented whether you use it in C, Python or PVSS in this web page [3]. Let’s consider the following example.

Assume that your connectivity schema looks like Fig. 4. There are two DHCP servers, DHCP server 1 and DHCP server 2. You want the DHCP server 1 to configure the nodes from **node_1_1** to **node_1_32** and DHCP server 2 the nodes from **node_2_1** to **node_2_32**. Practically, i.e. from a network point of view, both DHCP servers 1 and 2 can configure all the nodes. So the destination table of DHCP server 1 which, in that case, is the same as DHCP server 2, will contain all the host nodes, i.e. from **node_1_1** to **node_1_32** and from **node_2_1** to **node_2_32**. The generated destination table for both DHCP servers 1 and 2 contains too many reachable hosts. Consequently the generated dhcp config file will be wrong. To solve the problem, you have to disable hosts. Let’s take the example of the DHCP server 1 (similar thing for DHCP server 2). The destination table of DHCP server 1 should contain only hosts from **node_1_1** to **node_1_32**. Hosts from **node_2_1** to **node_2_32** must be excluded, using **UpdateMultipleDeviceNodeUsed**. Then execute the Perl script to generate the dhcp config file for DHCP server 1. Then include the nodes excluded back so that you can generate the dhcp config file for DHCP server 2. The C code can look like:

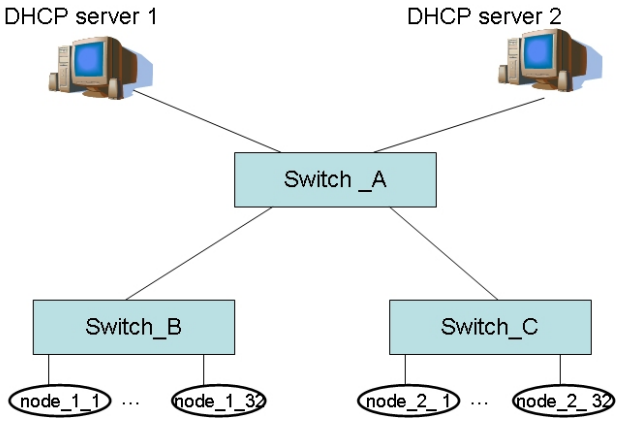

Figure 4 Example of use case where one needs to exclude host nodes

```
#include <iostream>
#include<stdio.h>
#include<unistd.h>
#include <ctime>
#include <time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "cicDB.h"

int execute_dhcp(char* dhcp_name)
{
  int res=0;
  execl("/usr/local/bin/perl","perl","dhcpCfg_generate.pl",dhcp_name,NULL);
  std::cout<<"couldn't execute the perl script and err="<<std::endl;
  res=-1;
  return res;
}

int main()
{
  pid_t pid_pere,pid_fils,pid_first,pid_1;
  int res=0;
  int res2=0;
  int len_array=1000;
  char* nto_list;
  char ErrMess[1000];
  int status=0;
  int status1=0;
  int i=0;
 char nodeName[50];
char nodeName_bis[50];

char dhcp_server1[50]="DAQ_CTRL_PC_01" ;
char dhcp_server2[50]="DAQ_CTRL_PC_02" ;

  res=DBConnexion(DB_NAME,login,pwd,ErrMess);
//disabling the host nodes from node_2_1 to node_2_32
  for(i=1;i<33;i++)
 {
sprintf(nodeName,"node_2_%d",i);
if(i = =1)
```

```
// the second parameter (nodeused) is set to 0 to disable the node.
res_query=UpdateMultipleDeviceNodeUsed(nodeName,0,1,0,ErrMess);
else
{
if(i==32)
resquery=UpdateMultipleDeviceNodeUsed(nodeName,0,0,1,ErrMess);
else
resquery=UpdateMultipleDeviceNodeUsed(nodeName,0,0,0,ErrMess);
}
}
pid_first=fork();
if(pid_first==-1)
    {
      perror("error fork");
      res=DBDeconnexion(ErrMess);
      exit(1);
    }
  else
    {

       if(pid_first==0)
 {
   std::cout<<"before generating the first dhpc config file ..."<<std::endl;
   res=execute_dhcp(dhcp_server_1);
   std::cout<<"Error in generating the first dhpc config file..."<<std::endl;
   res=DBDeconnexion(ErrMess);
   perror("exec");
   exit(1);
 }
       else
 {
   res=wait(&status);
   if(WIFEXITED(status)!=0)
   {
      for(i=1;i<33;i++)
{
sprintf(nodeName,"node_2_%d",i);
sprintf(nodeName_bis,"node_1_%d",i);
if(i = =1)
{
res_query=UpdateMultipleDeviceNodeUsed(nodeName,1,1,0,ErrMess);
res_query=UpdateMultipleDeviceNodeUsed(nodeName_bis,0,0,0,ErrMess);
}
else
{
if(i==32)
{
resquery=UpdateMultipleDeviceNodeUsed(nodeName,1,0,0,ErrMess);
resquery=UpdateMultipleDeviceNodeUsed(nodeName_bis,0,0,1,ErrMess);
}
else
{
resquery=UpdateMultipleDeviceNodeUsed(nodeName,1,0,0,ErrMess);
resquery=UpdateMultipleDeviceNodeUsed(nodeName_bis,0,0,0,ErrMess);
}
}
}
}
```

```
 pid_1=fork();
 if(pid_1==-1)
   {
     perror("error fork");
     res=DBDeconnexion(ErrMess);
     exit(1);
   }
 else
   {
     if(pid_1==0)
       {
       std::cout<<"before generating the second dhcp cfg file ..."<<std::endl;
       res=execute_dhcp(dhcp_server_2);
       std::cout<<"Error in generating the second dhpc config file..."<<std::endl;

       res=DBDeconnexion(ErrMess);
       perror("exec");
       exit(1);
       }
     else
       {
res=wait(&status1);
if(WIFEXITED(status1)!=0)
 {
 for(i=1;i<33;i++)
  {
sprintf(nodeName,"node_1_%d",i);
if(i = =1)
// the second parameter (nodeused) is set to 0 to disable the node.
res_query=UpdateMultipleDeviceNodeUsed(nodeName,1,1,0,ErrMess);
else
{
if(i==32)
resquery=UpdateMultipleDeviceNodeUsed(nodeName,1,0,1,ErrMess);
else
resquery=UpdateMultipleDeviceNodeUsed(nodeName,1,0,0,ErrMess);
}
}


     res=DBDeconnexion(ErrMess);
}
else
{
std::cout<<"child process 2 exited abnormally..."<<std::endl;
res=DBDeconnexion(ErrMess);
}
       }

   }


 }
   else
   {
std::cout<<"child process 1 didn't exit properly..."<<std::endl;
res=DBDeconnexion(ErrMess);
```

```
perror("exec");
exit(1);
   }
 }
}
}
```

This code generates the dhcp config files for DHCP server 1 and DHCP server 2.

4.2 Excluding links

According to the network topology, it may be useful to disable a link. Let's take the previous example. To generate the dhcp config file of the DHCP server 1, host nodes from **node_2_1** to **node_2_32** have been disabled. Another possibility would have been to disable the link between Switch_A and Switch_C, using **UpdateMultipleLinkUsedLinks**. Then execute the Perl script. Enable back the link between Switch_A and Switch_C so that the dhcp config file of the DHCP server 2 can be properly generated. The choice between the two possibilities depends on the topology of the network and also on the distribution of the DHCP servers.

4.3 Adding new hosts

At the beginning of the LHCb experiment, there will be less sub-farms than foreseen. So little by little, the connectivity of the DAQ will evolve by adding new sub-farms and new sub-farm nodes. In that case, you just have to insert these new sub-farm nodes in the **CONNECTIVITY** table as it is described in this web page [4]. You may also need to insert boot image information if it's a new type of device or if a newly added host requires a special configuration. Then you execute the Perl script to get the dhcp config file related to the new connectivity.

4.4 Generating a dhcp config file irrespective of the dhcp server

Depending on the network architecture, the DHCP config file can be the same for all the DHCP servers. It is the case when the switches/routers are properly configured. It avoids generating different dhcp config files. To do so, execute the Perl script with "none" as input argument, instead of providing a dhcp server name.

5 Conclusion

This note presents a method to generate the dhcp config file using the information contained in the configuration DB. It has been tested with a real DHCP server and it has accepted the dhcpd.conf produced.

6 References

[1] "Dynamic Host Configuration Protocol" RFC 1541 http://www.ietf.org/rfc/rfc1541.txt

[2] L.Abadie, "Configuring the LHCb Readout System using a Database". August, 2005. http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/default.htm

[3] API documentation http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/download_libraries.htm

[4] How to save the connectivity in confDB http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/connectivity_part_get_started.htm