Network Lab (9)

Student Intro and Guidelines

The lab purpose is to introduce you to the concept of event building in High Energy Physics experiments.

I. Networking Introduction

Every data acquisition system has, at its core, a computer network to gather and filter collision data from the detector. Usually a DAQ network has one (for small DAQ systems) or a few (for more complex DAQ systems) core switches/routers and lots of pizza box switches in order to connect every computer in the network.

To simulate the core of a DAQ network we will use an HP Procurve switch. Switches map the Ethernet addresses of the nodes residing on each network segment and then allow only the necessary traffic to pass through the switch. When a packet is received by the switch, the switch examines the destination and source hardware addresses and compares them to a table of network segments and addresses. If the segments are the same, the packet is dropped ("filtered"); if the segments are different, then the packet is "forwarded" to the proper segment.

Switches help us connecting computers in the same area which are not too further apart in a LAN (local area network). However standard switches does not care about teams or applications. To be able to divide computers in groups by teams or applications we have to use VLANS. So actually having switches that supports VLANs is having a switched network that is logically segmented on an organizational basis, by functions, project teams, or applications rather than on a physical or geographical basis. What this is saying is that a VLAN is not defined by any physical restrains or needs, it can span an entire country or can be in the same floor in an office. VLANs are formed for administrative purposes and not geographical purposes.

To be able to monitor the networking devices in a network (ex: switches and routers) the most used protocol is SNMP (Simple Network Management Protocol). SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications. In typical SNMP use, one or more administrative computers have the task of monitoring or managing a group of hosts or devices on a computer network. Each managed system (also called Slave) executes, at all times, a software component called an *agent* (see below) which reports information via SNMP to the managing systems (also called Masters).

The information gathered via SNMP protocol (ex: number of MB/s going in or out from one port, errors, discards, interface speed etc) can be stored in a standard database (like Oracle, MSSQL, MySQL etc) or with the help of RRD files (Round Robin Database). In order to create and store data in an RRD file you will have to use rrdtool application. RRDtool refers to Round Robin Database tool. Round robin is a technique that works with a fixed amount of data, and a pointer to the current element. Think of a circle with some dots plotted on the edge. These dots are the places where data can be stored. Draw an arrow from the center of the circle to one of the dots; this is the pointer. When the current data is read or written, the pointer moves to the next element. As we are on a circle there is neither a beginning nor an end, you can go on and on and on. After a while, all the available places will be used and the process automatically reuses old locations. This way, the dataset will not grow in size and therefore requires no maintenance. RRDtool works with with Round Robin Databases (RRDs).

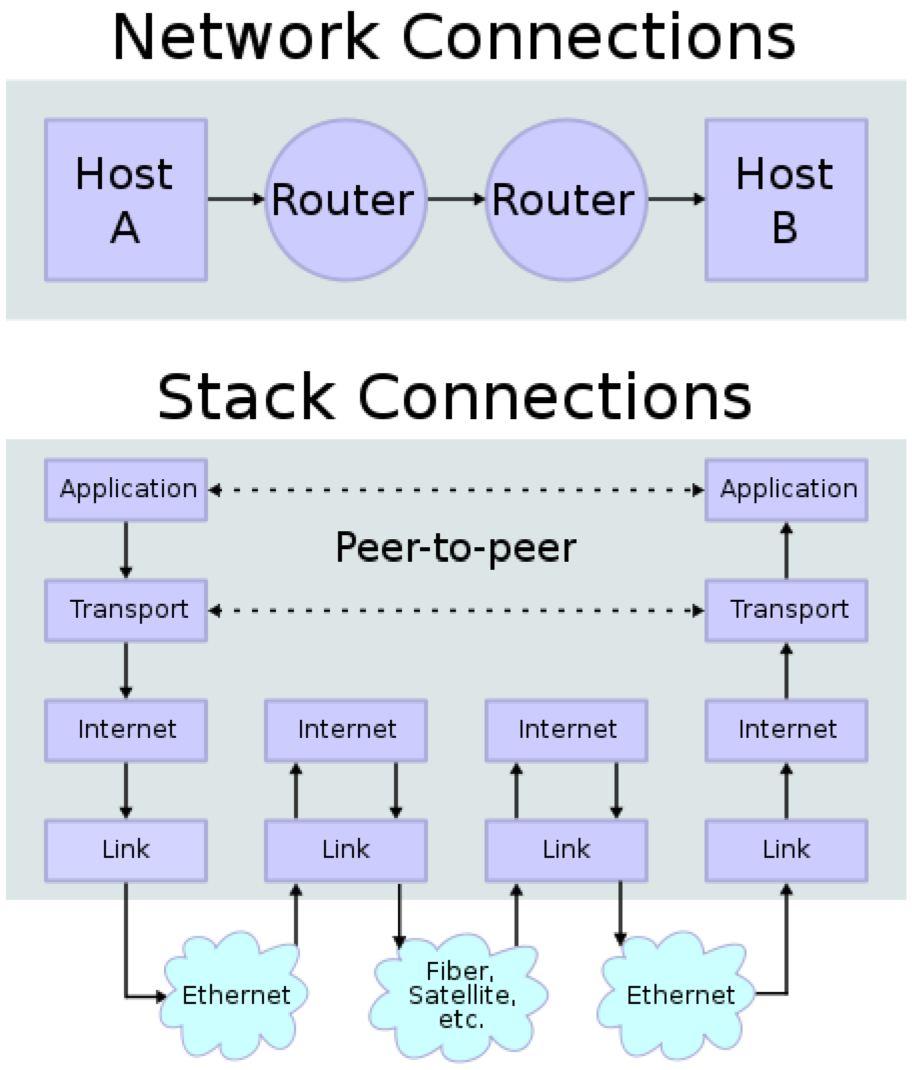

Illustration 1: Data transmission over a routed network

From a protocol point of view typically, the link protocol is E*thernet*, the Internet protocol is *IP*, the transport protocol are mainly *UDP* and *TCP*, and then an application protocol example is *http*. In order for data to be transmitted through the network, we encapsulate them into the protocol related to the layer we are using. This encapsulation consists in adding headers and footers to the data. Then, when a packet passes through the different layers, each layer read its related header and is able to process the following data.

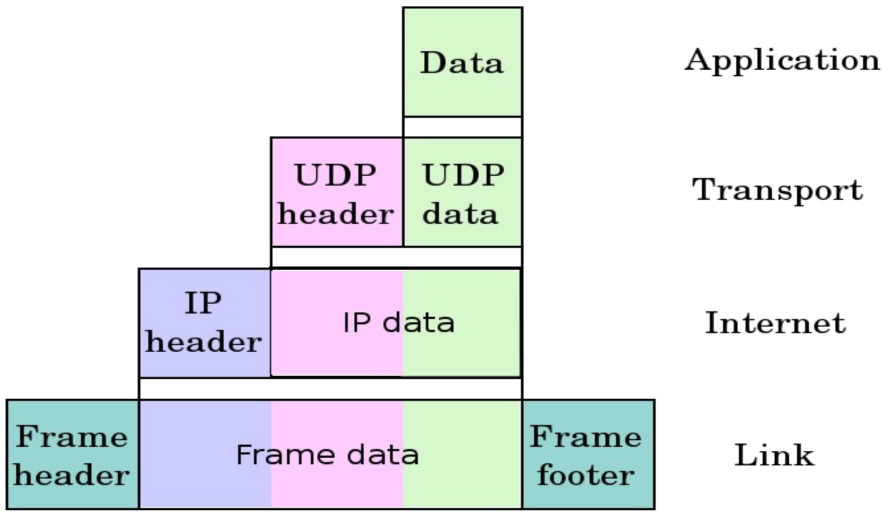

Illustration 2: Protocol stack

II. Event Building Introduction

Large experiments consists of a very complex detector, made of sub-detectors. Each sub-detector has a specific behavior and identifies different particles.

We want to get, for each collision, a picture of the complete detector. So each sub-detector is read-out by a device. The connection between this Readout device and the sub-detector is generally made of custom links which have to be resistant to radiations.

The readout devices perform a first data analysis before to format them following networking standards, because we're now in a radiation safe area and standard devices are very efficient.

To perform further processing, HEP experiments relies on a computing farm, i.e. each single event will be processed by a single core, which will decide if the event is interesting or not. If the event is interesting, it will be written to a temporary storage system, else it will be discarded. This architecture means that there is no parallelism in the processing.

In order to perform the event building, a computing farm core needs to get the full picture of the detector, i.e. the information from all readout devices.

This is achieved mainly using 2 different “protocols” which are either **push** or **pull**. Pushing means that the readout devices will send their information, for an event i, to a single selected core. This core can be elected according to a round-robin rule. This is quite limited because this core can be busy. An improvement is possible using some back-pressure mechanisms. A core would advertise if it's available or busy.

Pulling consists for a core in requesting the event fragment to each readout device. Therefore only an available core can have make the request. It can be enhanced by requesting only a part of the event, analyzing it and if it looks interesting, requesting the full event.

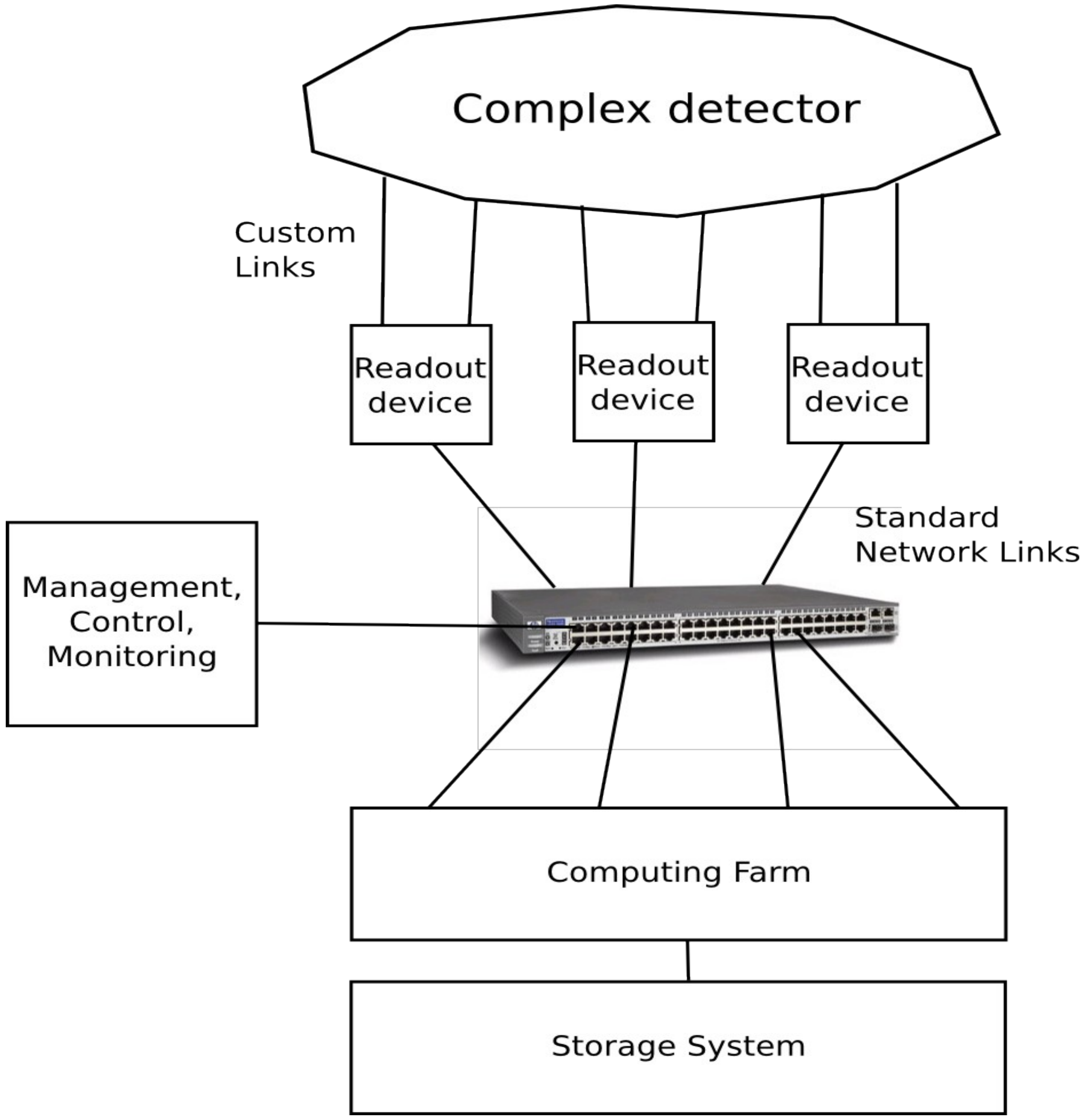

Illustration 3: Simplified architecture of a DAQ network

III. Laboratory Objectives

In a first part you will have to configure your system, mainly the switch, so data can be transmitted from a data injector to the processing computers.

Then you have to set up a simple system to gather traffic information from the switch so that you will be able to monitor the traffic flow in you network.

In the end you will have to implement a simple event building software on the processing computers, which receives data from the network, decode it and then write them to a permanent storage system.

IV. Network Configuration Guideline

a) Power up the switch. Connect cables between computers and switch. Check the connectivity light. (It doesn't matter what port numbers you chose to use for connecting the PCs);

b) Login to the network monitoring PC (NETMON) using *user: student, password: student*;*

c) Connect the serial cable (DB9-DB9) from the serial port on the switch to the serial port on the NETMON computer;

d) Start "screen" application to connect to the switch:

   i. student> **screen /dev/ttyS0**

   ii. sw-daqcluster-c1> **enable (user: admin, password: admin*)**

   iii. sw-daqcluster-c1# **show running-config**

e) Check that the switch has the ip address set to 10.128.2.2 (netmask 255.255.0.0). If the ip is not set please set it:

   i. sw-daqcluster-c1# **configure terminal**

   ii. sw-daqcluster-c1(config)# **vlan 1**

   iii. sw-daqcluster-c1(vlan-1)# **ip address 10.128.2.2 255.255.0.0**

f) Make sure that the NETMON PC can request information via SNMP from the switch. You can check that by having a look in the configuration file after the following lines:

   i. ip authorized managers IP NETMASK

   ii. management-vlan 1

g) Have a look at the VLANs already configured on the switch (show running config). Are the connected ports all in the same vlan? Is it important to have all the ports in the same vlan? If necessary please make adjustments to where the cables are connected;

h) Try to ping the switch;

i) Try to request some simple information from the switch via SNMP:

   i. student> **ping 10.128.2.2**

j) Try to request some simple information from the switch via SNMP:

   i. student> **snmpget –v 2c –c public 10.128.2.2 sysDescr.0**

k) Use wireshark to have a detailed look at the network traffic (hint: use menu command on the switch);

l) Activate port mirroring on the switch so that the traffic will be duplicated for monitoring purposes:

   i. sw-daqcluster-c1# **conf**
   ii. sw-daqcluster-c1#(conf)# **mirror-port 1**

m) Use wireshark to have again a detailed look at the network traffic.
   i. Student> **sudo /usr/local/bin/wireshark**

V. Network Monitoring Guideline

a) From the console change directory to swmon
   i. student> **cd ~/swmon**
b) Have a look at the following files:
   i. **switch_stats_create.sh, switch_stats.sh, switch_graph.sh**.
c) Run switch_stats_create.sh and check for newly created switch_stats.rrd file:
   i. student> **./switch_stats_create.sh**
d) Have a look at the file switch_stats.rrd
   i. student> rrdtool dump switch_stats.rrd | less
e) Run switch_stats.sh to start polling the switch:
   i. student> **./switch_stats.sh &**
f) Open another console and run switch_graph.sh in to start generating traffic plots.
   i. student> **./switch_graph.sh &**
g) Open index.html file in a browser (ex: firefox)
h) Modify **switch_stats_create.sh, switch_stats.sh, switch_graph.sh** to start monitoring only on the active ports (the one connected to the event building farm and the event injector)

Decoding for the SNMP OIDs related to traffic information

OID .1.3.6.1.2.1.2.2.1.10 =

.iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2).ifTable(2).ifEntry(1).ifInOctets(10)

OID .1.3.6.1.2.1.2.2.1.16 =

.iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2).ifTable(2).ifEntry(1).ifOutOctets(16)

VI. Event Building Guideline

Our data acquisition system uses the network stack presented in Illustration 5. The transport layer implements MEP (Multi Event Packet). It is a simple protocol which stores several event fragments, from *1* to *m*.

Different data sources, from *1* to *n*, are transmitting such packets to the event-builder. It means that an event builder computer will receive *n* packets of *m* events and will have to write them to process them.

In order to store these data and to be ready to write them, or to discard them in case of problems, you'll need to store them in a data structure. An example of a solution is presented in Illustration 4.

A template source code is available in */home/student/NETWORKLAB/meprx_tpl.py*. It gives a basic python structure and the first steps of data processing. Using this as an example, you have to implement the complete network decoding and data storage.

The language used for this software is Python. You will find in annexes several information to help you for the implementation.

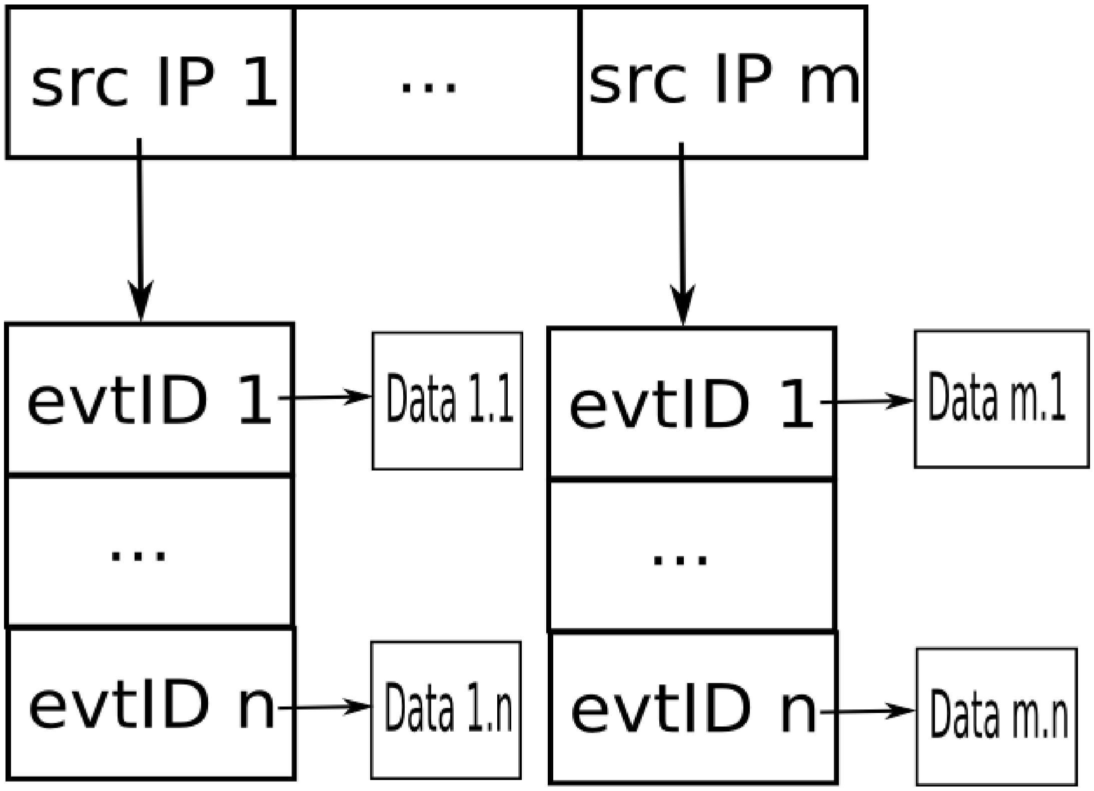

Illustration 4: Data structure to store event data

Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0
DA[7:0]	DA[15:8]	DA[23:16]	DA[31:24]
DA[39:32]	DA[47:40]	SA[7:0]	SA[15:8]
SA[23:16]	SA[31:24]	SA[39:32]	SA[47:40]
L/T[15:8]	L/T[7:0]	Version/ IHL	Type of Service
Total Length[15:8]	Total Length[7:0]	Identification[15:8]	Identification[7:0]
Flags/Fragment Offset[11:8]	Fragment Offset[7:0]	Time to Live	Protocol = 0xF2
Header Checksum[15:8]	Header Checksum[7:0]	IP-SA[31:24]	IP-SA[23:16]
IP-SA[15:8]	IP-SA[7:0]	IP-DA[31:24]	IP-DA[23:16]
IP-DA[15:8]	IP-DA[7:0]	L0ID / L1ID[7:0]	L0ID / L1ID[15:8]
L0ID / L1ID[23:16]	L0ID / L1ID[31:24]	Number of Events[7:0]	Number of Events[15:8]
Total Length[7:0]	Total Length[15:8]	Partition ID [7:0]	Partition ID [15:8]
Partition ID [23:16]	Partition ID [31:24]	EventID 1[7:0]	EventID 1[15:8]
Len 1[7:0]	Len 1[15:8]	Data1[7:0]	Data1[15:8]
Data1[23:16]	Data1[31:24]	…	…
…	…	Event ID2[7:0]	Event ID2[15:8]
Len 2[7:0]	Len 2[15:8]	Data2[7:0]	Data2[15:8]
Data2[23:16]	Data2[31:24]	…	…
…	…	…	…
…	…	…	…

Event Building protocolIllustration 5: Event Building protocol description. Ethernet in grey, IP in yellow, MEP in green, fragment header in teal and raw data in white.