Operational Model of the ATLAS TDAQ Network

S. M. Batraneanu [a,b] A. Al-Shabibi [c], M. D. Ciobotaru [d,b], M. Ivanovici [e], L. Leahu [a,b] , B. Martin [a], S. N. Stancu [d,b].
[a] CERN, Geneva, Switzerland,
[b] University "Politehnica" of Bucharest, Bucharest, Romania,
[c] Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland,
[d] University of California, Irvine, Irvine, USA,
[e] University Transilvania of Brasov, Brasov, Romania.

***Abstract*—The ATLAS TDAQ network consists of four separate Ethernet based networks which together total over 4000 ports with 200 edge switches and 6 multi-blade chassis switches at the core. System checks are invoked at every level of the installation. The full installation is described in different static databases. Tools are provided to automatically cross-check these for consistency. The configuration management is centralized: configuration files stored in a database are distributed to all devices and the actual settings are periodically verified. Monitoring systems are deployed to validate the connectivity, identify malfunctions and confirm the resources availability upon request from TDAQ control. Relevant operational statistics (e.g. port status and throughput) are continuously logged and made available to TDAQ control. Watches and alarms are set for dynamic threshold violations and the complete instantaneous status can be viewed at different levels of abstraction in a 3D fly-through. A tool-set has been developed to demonstrate aggregate achievable cross-sectional bandwidth for TDAQ-specific traffic profiles, as well as to analyze traffic flows and hot spot behaviour.**

I. Introduction

The architecture of the ATLAS TDAQ networks has already been described in [1] and the first stage of the installation has been functioning since autumn 2006. A set of tools and support procedures are being put in place to ensure that the installation is consistently described, that all the elements are functional, and that failures are rapidly diagnosed, to minimize any potential down time. Statistics are gathered in real time to indicate the performance of the network, to assist in system analysis and to provide reliable status information to the experiment's run control. Status information is displayed in both 2D and 3D form. Specific tools have been developed to probe the behaviour of targeted portions of the network and determine realistic achievable throughput.

II. Cross Checking Installation Databases

The installation is described graphically using NetDesign [2], a Microsoft Visio® based drawing package. The physical elements such as switches and processors are described in two separate databases. The ownership, serial numbers and maintenance contacts are stored in the MTF database [3], while their geographical location is stored in Rack Wizard [4]. In addition, all network connected devices have their service settings described in the LANDB database [5], and cables are described in the cable database [6].

Each of these databases has its own data access interface and is maintained by many different people, each with their own particular focus. It is therefore likely that there will be omissions and errors at the time the data is first entered. Even if extensive cross checking is done there will inevitably be a divergence over time as material is moved, repaired or modified and only a sub-set (if any) of the relevant databases is updated. To avoid this, and ensure that the data across the various databases is consistent at all times, a tool has been developed to automatically cross check all these static databases.

Fig. 1 illustrates the block diagram of the database consistency check application. It is written in Java, mainly to take advantage of the supported interfaces, such as Java Database Connectivity (JDBC) [7] API, Simple Object Access

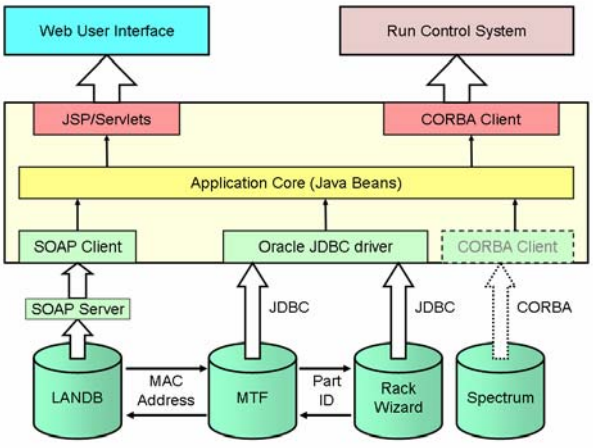

Fig. 1. Database consistency checking.

Protocol (SOAP) [8] API, and the Common Object Request Broker Architecture (CORBA®) [9] API, required by the proliferation of different database access requirements.

To allow for an easy extension of the application to cover other databases, or additional clients, the data processing core is clearly separated from the data access components and the client Web components that use JavaServer Pages (JSP) [10] technology.

Consistency checks operate by cycling through records in each database and using common key properties to identify the corresponding records in the other databases. This is done for each database in turn. The common key between LANDB and MTF is the MAC address of the device, while between MTF and Rack wizard it is the PartID. In this way, records in one database that have no corresponding records in the others, or for which the corresponding records have inconsistencies such as different names, can all be identified. Once the Spectrum[1] server is fully functional it is intended to add a CORBA interface to it to check if all the objects discovered by Spectrum have corresponding entries in the static databases. Work to incorporate the NetDesign database check is also foreseen.

The consistency check can be run manually, for example after updating the databases to ensure that all relevant entries have been completely registered, or automatically at regular intervals as a background check.

III. Network Configuration

The scale of the networks deployed in the TDAQ system (order of 200 managed devices) calls by itself for the automation of the configuration task. Not only is the system large, but its gradual deployment will impose the need to simultaneously manage different flavours of equipment and/or software versions. Currently there are several commercial solutions for network configuration management (NCM), offering an extensive set of features. Since only a reduced sub-set of these features is essential for the particular case of configuring the TDAQ networks, the cost of deploying commercial software cannot be justified. The complexity of the configuration management task is significantly reduced by the "static" nature of the TDAQ networks configuration. Once installed, the configuration of a certain device does not need to change "on the fly". Regular upgrades and changes are performed during foreseen maintenance periods. Thus the ability to perform dynamic changes on the network configuration is a desirable feature, but not a "must" for the NCM system. The emerging open source utilities for configuration management [11] appear as a promising alternative for this purpose in the close future.

Currently a set of custom Python [12] scripts is used for configuring the network devices. Since this approach is unlikely to scale as the system grows, we plan for a coherent architecture for network configuration management (see Fig.2). A database is used to store the inventory of all the devices in the network (and eventually topology information), including repositories for device configuration files and firmware images.

The Front-End and Configuration Engine (FCE) provides the network administrator with a representation of the network (based on information from the database), and manages the three essential management operations together with the appropriate logging facilities.

A. Applying configuration commands

A Common Configuration interface (CCI) enables performing regular configuration tasks (e.g. enabling or disabling ports, configuring VLAN membership) using the same "language", regardless of the device type and manufacturer. This language should be scriptable, i.e. allow the user to apply a sequence of configuration actions to a desired set of devices. The CCI implementation translates the common "language" into configuration actions specific to each device through NETCONF [13] (if available), SNMP (Simple Network Management Protocol) and ultimately through the command line interface (CLI).

In order to provide full flexibility, the FCE allows the operator to bypass the CCI and directly perform changes on the device (through the CLI or SNMP).

B. Configuration file management

Based on the information from the network representation databases, the FCE should be able to make snapshots of the

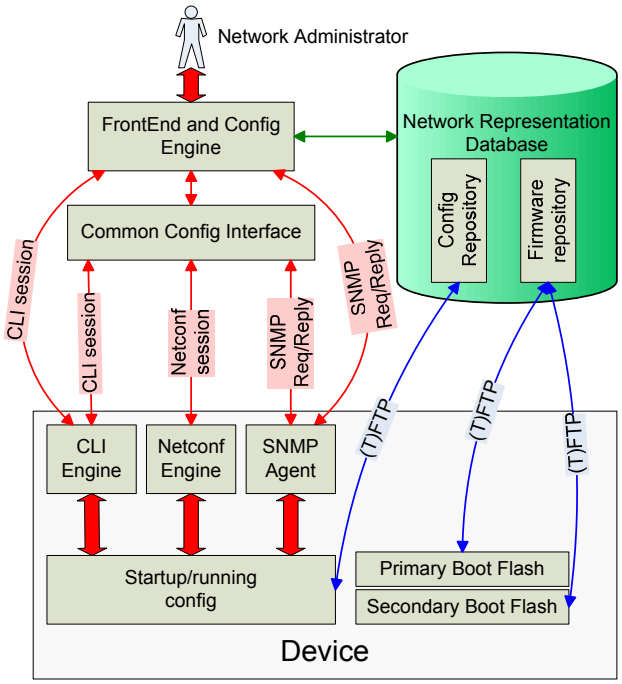

Fig. 2. Network configuration management.

[1] Spectrum is the software package deployed for network health monitoring (see Section IV-A).

configuration files for all the devices, either at pre-configured intervals or on demand. These snapshots can be stored in the Config Repository of the database, and the FCE should be able to "push" a known revision from the database onto the network devices. Configuration files can be uploaded to or downloaded from network devices using a file transfer protocol, like TFTP (Trivial File Transfer Protocol).

C. Firmware management

An approach similar to the one for the configuration file management is taken. Most of the devices support booting from two memory locations (primary and secondary). The firmware (FW) upgrade policy should make sure that one of the boot flashes contains the "last validated FW version", while the other one receives the upgraded version.

D. Logging System

A coherent logging system is important for troubleshooting and understanding the sequence of occurrence of certain events. Thus, devices must have their clocks synchronized using the Network Time Protocol (NTP), and make use of a common syslog [14] server.

IV. Network Monitoring

A. Spectrum

Spectrum [15] is the commercial network monitoring package employed, and provides various services, e.g. polling, alarm notification and data archiving. Its main tasks are to maintain a model of the full network in its core application, called the SpectroServer, and to constantly poll the network devices for information using SNMP. Spectrum is based on a client-server architecture, where the server is the SpectroServer and the clients are independent applications that connect to the server and are provided access to the data in the SpectroServer knowledge base. The OneClick Console is a client application used for visualizing the status of the network in a 2D layout. For example, Fig. 3 depicts a snapshot of a TDAQ data network, where all devices but one are working properly.

The concept that Spectrum relies on when building and updating its knowledge base is the *model*. Every network device, component, and application is represented as a model inside the SpectroServer. Keeping an up-to-date representation of the network is the task of the SpectroServer. This can be done synchronously, with SNMP requests being issued at five minute intervals, or asynchronously when it is notified by the network device itself of relevant events via the trap mechanism[2]. It is also possible for SpectroServer to change a writeable value in a device's Management Information Base (MIB) using the same protocol, SNMP. It is worth mentioning that the acquired version of Spectrum offers SNMP support for authentication, encryption and 64-bit counters. The last

[2] When detecting abnormal conditions, the SNMP agent on the device can issue asynchronous alerts (SNMP traps).

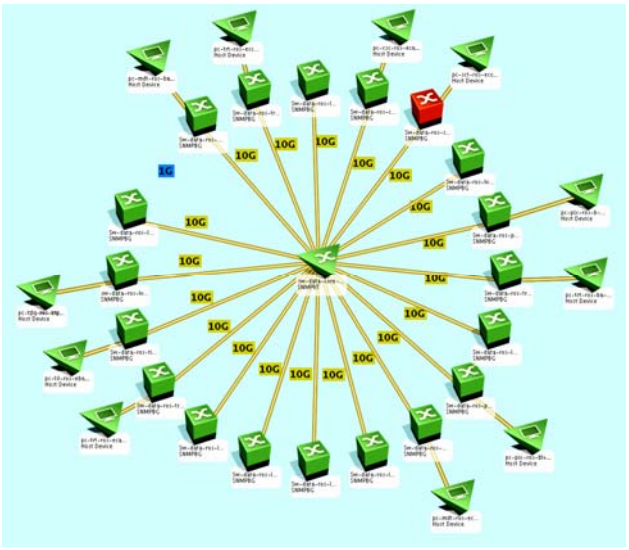

Fig. 3. 2D network visualization.

capability is important when monitoring interfaces with high speed links because it reduces the problems caused by a too rapid cycling and wrap around of the relevant counters.

One important role of Spectrum is to provide network specialists with the quickest possible way to detect, isolate and resolve problems. When a network problem is detected or reported by a device via an SNMP trap, the SpectroServer can generate an alarm for the model or models involved. In the event of multiple simultaneous error conditions, Spectrum will correlate them and derive the root cause of the failure, then issue an alarm for just that device.

Spectrum offers custom application access to the SpectroServer data via a CORBA based API. Using this mechanism, useful network information can be published to the TDAQ control applications or any other software that requests it.

Traffic statistics are needed for performance monitoring at the highest practicable rate. Even at low rates, maintaining these statistics in the knowledge database would rapidly overrun the storage capacity and so a different approach is needed. The SSLogger client program is used to request traffic statistics from each node and to update the received statistics in raw data files (plain text). These are updated every 30 seconds and then added to a Round Robin Database [16]. This limits the maximum storage requirements, but also the length of time the data can be stored. With the current settings this is of the order of one week for all the monitored ports at maximum time resolution.

B. sFlow

The SNMP statistics provided by Spectrum offer an aggregate view on the traffic levels in various places of the network. Average link occupancies can be measured, but SNMP does not provide any information on where this traffic is moving from or to, or what type of packets make up the

flow. When troubleshooting network congestion issues or just for simple health monitoring, it is often useful to know the contents of the data streams in more detail. For the TDAQ network it is intended to use sFlow [17] for this purpose. sFlow is a technology based on statistical sampling. Packet descriptor samples are saved by network switches and submitted to a central collector. The samples are then analyzed by the collector and traffic characteristics are inferred. A simple collector and analysis package has been developed to explore this technology. Its most important feature is the ability to identify network conversations, i.e. packet exchanges between pairs of source and destination IP or MAC addresses. Conversations can further be classified based on UDP/TCP port numbers if necessary. For each port in the network, a histogram of the per-conversation traffic is produced.

Having the traffic profile for each switch port, pie-charts as shown in Fig. 4, and traffic matrices can be generated. The "top-users" of a link as well as any abnormal traffic patterns can be immediately detected.

The package is implemented in Python and this facilitates any future functional extensions or integration with other applications. For example, one possible direction for future work is to use the TDAQ specific protocol headers to classify the traffic.

C. *High-speed SNMP monitoring*

For specific monitoring requiring high sampling rate (i.e. a few problematic ports of the network), the YATG (Yet Another Traffic Grapher) package [18] has been developed. YATG has been designed to poll SNMP counters very fast and thus produce bandwidth utilization plots with a fine time resolution.

It has been implemented as a multi-process, multi-threaded application. Multi-processing is used to simultaneously poll the ports that have been selected for analysis. Within each process there are two threads. One will loop on issuing an SNMP request to the port in question and then posting the subsequent request upon reception of the response. It is therefore running at the capacity of the port, without saturating the process CPU just waiting for the responses. The second thread is a timer process that sets the bound of the total sampling time. When the polling thread is terminated by the timing thread, the package generates the traffic plots for the selected ports.

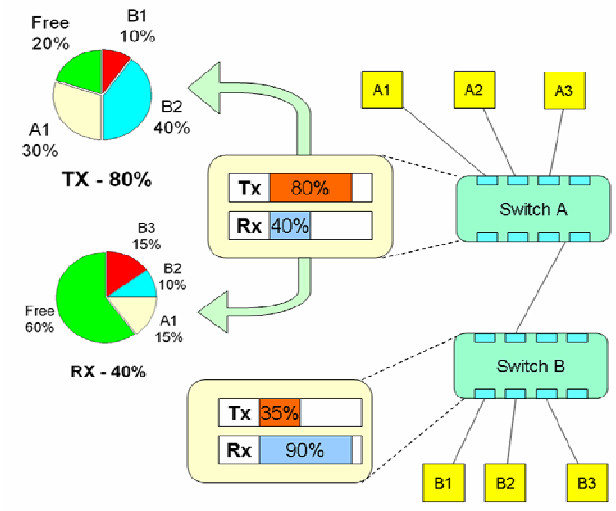

Fig. 4. sFlow monitoring.

This method was shown to be the most effective for obtaining the best results from any given switch and rates of up to 100Hz per port have been obtained. However it also exposed a problem with SNMP messaging in general. Although the counters in the switches are being maintained at hardware speeds, the SNMP server that reads them runs on the management CPU of the device. It is usually allocated a fairly low priority in order to free the CPU for time-critical operational tasks. Attempts were made to reduce the SNMP rate by making a single grouped request instead of a request per port. In this case however, the server will usually truncate the response to what can be fitted into a single Ethernet frame. Although some switches can respond at up to 100Hz, for others the figure is as low as 1Hz and the worst was measured at 0.2Hz. This response time represents the limiting factor for SNMP based monitoring. To achieve higher speeds, methods closer to the hardware need to be employed (see Section VII-B).

V. Status Visualization

Gathering the relevant statistics and status of a system is only part of a much larger problem: deciding which information is needed by different users and in what form it is to be presented. At one extreme, detailed knowledge of which ports or devices are reporting chronic or sporadic failures is needed for maintenance purposes. However, global views of traffic overloads and lost packets are needed by system analysts to determine if load sharing is working properly or if the data-taking processes themselves are overloading the system. Operators merely need to know that all or most of the system is working within design limits.

With over 4K ports being monitored over four separate networks and many processing devices connected simultaneously to two or three of those networks, there is a real concern that visualizing any of the data in detail on health, throughput or errors is only possible for limited parts of the system which means that the global view of what is happening at a system wide level would be lost.

The commercial tools that address this typically display a hierarchical 2D representation of the network with color coded status information at every level. However, as the network size increases they are forced to create ever deeper levels of hierarchy just to keep the display on one screen. This makes navigation extremely cumbersome. Devices such as those in

the TDAQ installation, with multiple network connections make it even more difficult to create a readable display. What is needed is a tool that will allow for varying levels of detail to be visible as a function of the chosen point of view. Seeing the whole TDAQ network displayed in this manner would just not be possible at any meaningful level of detail.

The chosen approach was to benefit from recent advances in 3D flythrough visualization software that allows rapid and smooth scaling from the very large to the very small with enhanced operator control of the 3D scene. X3D [19], an emerging ISO standard for real-time 3D visualization, was adopted since it is flexible and powerful enough to fulfill the demanding requirements of this large scale visualization system.

A hierarchical 3D prototype of the network has been developed with two layers of abstraction. The top layer which offers the overall picture of the network at a glance is made of two types of containers: processor farms and core switches. Fig.5 shows a screen shot of the prototype top layer. The bottom layer offers more detailed information about particular devices and is made up of processors and switches grouped by functionality as shown in Fig. 6.

Using a static 3D model of the network installation authored in 3D Studio Max [20] as input and the X3D prototypes for primitives (such as panels, processors, switches), the X3D description of the top-level containers with their interconnections was generated. The hierarchy of objects that need to be visualized and some relevant characteristics of each object or object group such as, for example, the object type (processor, switch) are stored in a small-scale MySQL [21] database. The application uses the MySQL data and the X3D

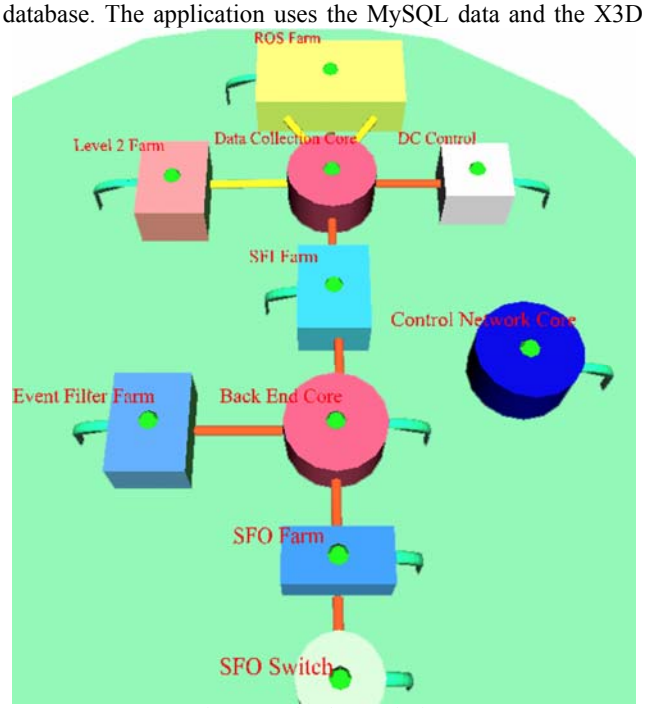

Fig. 5. Top-level network view.

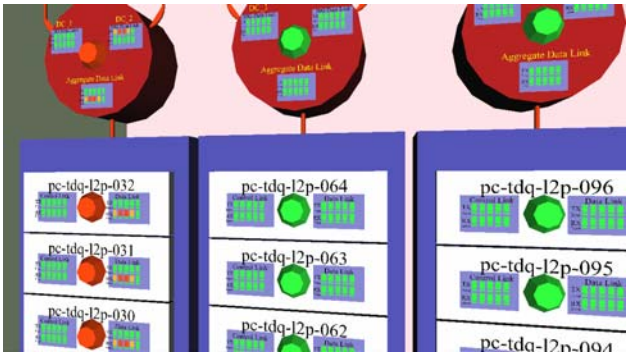

Fig. 6. Close-up view of processor farms.

prototypes to populate the top-level containers at initialization time. It also performs the automated placement of the components inside the container thus relying only on the container coordinates and on a set of placement rules and dimension parameters. Adding the real-time variables to the scene is done using the standard Scene Access Interface (SAI) [19]. This allows the application to modify object attribute values as a function of the status information obtained by mining the relevant data from the different databases.

Both top and bottom level components carry virtual 'status lights' which show the aggregate health status of the component and traffic and error information panels which reveal input and output throughput colored according to predetermined thresholds. Errors that exceed critical thresholds propagate upwards so as to be visible from the top-level containers. Thus a view of the whole system will reveal where potential concerns exist and the operator can 'fly in' for a closer look at the cause and scale of the problem.

In addition to the SAI which offers the real-time visualization capabilities, X3D offers other important features related to scalability and performance. Levels of detail can change as a function of the distance of the viewpoint. This limits what needs to be drawn for any given scene. Proximity sensors allow the control of navigation speed depending on the position of the viewpoint and are used to decrease speed when navigating inside containers.

Time sensors control the status refresh rate and allow concurrent refresh of the status information. X3D Proto structures allow the creation of user-defined object types, each with its own set of attributes which can be accessed externally. Using this feature, the primitives and their dynamic attributes – such as color and description text – were defined and this almost eliminated redundant X3D descriptions, reducing the X3D files size from tens of MB to hundreds of KB.

The application is currently using the Xj3D toolkit [22] comprised of a Java-based 3D viewer and an open source implementation of the Scene Access Interface. The Xj3D toolkit, which relies on the OpenGL [23] 3D rendering engine, is used as a testing ground for the developers of the X3D standard and is still in its initial stages. Although sustained

efforts are made for its improvement, the toolkit focuses mainly on proving that the newly developed X3D features can be implemented rather than on performance and scalability. For this reason it is foreseen to migrate to a commercial toolkit in the near future.

VI. Reporting System Information

The system information gathered by Spectrum (see Section IV-A) is required not just for visualization but also for other consumers such as run control and error reporting. Fig. 7 shows the data flow from the monitoring system to the end users. The basic status and health of the system is made available to the Network Initial Panel in the operator GUI (Graphical User Interface) of the Online Software[3]. All the network related alarm messages are displayed on this panel for the operator's attention, and also made available through the Error Reporting System (ERS). When a run partition[4] is spawned, there is a Network Partition panel spawned with it. The default is for this to also run a check to see if the resources required by the partition, i.e. processors and their connectivity, are available. The Services Manager maintains an up-to-date copy of the currently available resources detected by the SpectroServer to be used for a fast cross check with the requirements held in the Configuration Database (ConfigDB[5]).

Once the partition is running, all the alarms that are relevant to that partition are directed to the network partition panel. SSLogger runs the network traffic statistics collection. The resulting plots are published to a web interface which can be read from anywhere including the Network Panels. The OneClick Spectrum console offers 2D connectivity and status display and can be accessed by the network administrator for advanced troubleshooting.

The Services Manager also handles the statistics produced by the Report Gateway and compiles tables of the individual port statistics as well as the aggregate values and the threshold comparisons used by the Status Visualisation program (see Section V) to produce the 3D status. The link to the Detector Control System (DCS) via the information services (IS) is used to correlate switch or network failure with the status of power in the relevant racks so that the appropriate alarm message may be generated.

VII. Targeted Diagnostics

The monitoring so far described is essentially reactive, status evolves and monitoring hopefully keeps up with the changes. However the sampling methods used are orders of magnitude slower than the typical packet transit time from detector to processing node. More specific tools are needed to calibrate and monitor particular connections or network areas that are shown to be causing problems by the general monitoring system. Data transmission problems can be due to material failure or, more often, to inadequate available bandwidth for the instantaneous load. This can be caused by asymmetric data distribution patterns or inappropriate traffic patterns caused by poor flow control between applications. It is not possible to increase the sampling rate of SNMP messages because of the switch CPU restrictions already mentioned in Section IV-C. Different approaches are required to gather system data that are more limited in scope but capable of greater resolution.

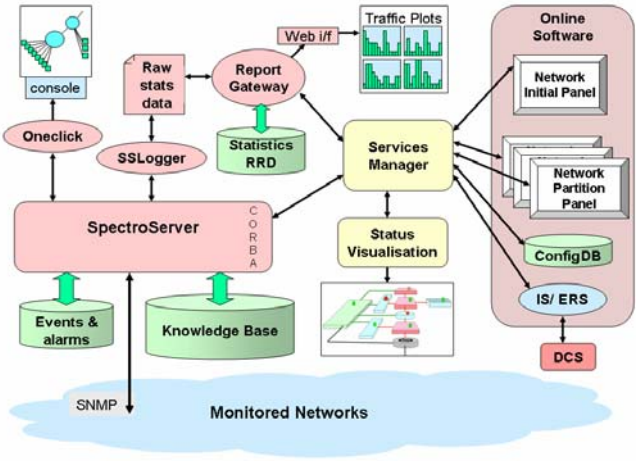

Fig. 7. Monitoring and reporting.

A. Measuring achievable bandwidth

A tool has been developed to emulate the data acquisition traffic flow, which is characterized by request-response transactions. Physics events are stored in detector buffers and are retrieved on demand by the processing nodes [24]. The traffic pattern has a "funnel" shape: as the event data is scattered among a large number of detector buffers, each processing node (a client) receives data simultaneously from many sources (the servers). In order to regulate the data rates, a traffic shaping mechanism is employed [25].

The tool can be used to validate the network, not just in the sense that all nodes have the necessary inter-connectivity, but also that there is enough capacity to deliver the bandwidth required for a given TDAQ configuration. It emulates the behavior of DAQ applications for what concerns the use of the network, but does not depend on the entire DAQ control infrastructure to operate.

The tool can be used to find the upper bounds in terms of transaction rates supported by a given combination of network and computers. It can also be used to measure the maximum queue depths in switches. Thus, it is possible to cross-check that buffer management configurations are correctly reflected in the hardware.

[3] The Online Software is the term used to denote the framework that controls the ATLAS TDAQ system.

[4] The term partition denotes the subset of TDAQ components used for a particular run.

[5] The ConfigDB stores the information required for configuring and running a partition.

B. *Measuring queue occupancy development*

Resource management in any switch is a proprietary issue. Some manufacturers will have a freely allocatable pool of shared memory, some will allocate memory on a per-port basis and yet others will permit a degree of user programmability in the way that memory is allocated. It has also been observed that manufacturers may change their memory allocation models from one firmware release to the next. Depending on the conditions in force at any moment, any oversubscribed egress port may, sooner or later, run out of available memory and the overflow packets will be discarded with negative consequences for the parent application.

As previously mentioned, it is possible to measure the physical memory allocation by precisely directing known traffic distributions through a target switch. This is not enough however to judge to what extent the memory is sufficient for long-term error free operation. The actual flow to any given port will have some distribution as a function of trigger conditions and physics data flow management. This could possibly lead to bursts of oversubscription with enough duration and rate to overflow the available buffers. Profiling the queue occupancy will show how well the infrastructure copes with the reality of the load and, if needed, will indicate how load balancing can be optimized. However, slow polling will never reveal the details of queue occupancy. Typical buffer sizes are of the order of 10^3 to 10^4 frames for full size frames. At 10 Gbit/s, such a buffer can overflow in less than a millisecond so the sampling rate needs to be of the order of tens of microseconds. We implemented a system able to achieve such high rates.

The firmware of a pair of FPGA[6] based network interface cards, the GETB [26], has been modified to transmit and receive time-stamped probe packets. In this way, latency between two points of the network can be measured with very fine granularity, based on the transmission and reception time-stamps from the probe packets. For known frame sizes the latency is proportional to the queue occupancy. Note that the measurement method is inherently intrusive. A standard measurement scenario is shown in Fig. 8.

A typical hot spot may be the egress port of a "core" switch where data samples from multiple sources are concentrated together and forwarded to an "edge" switch. By injecting probe packets into the core switch and receiving them from a spare port on the edge switch, we can dynamically monitor the instantaneous queue occupancy at the egress port. This sampling can be done at very high rates (1.488 MHz for 64 byte probe packets).

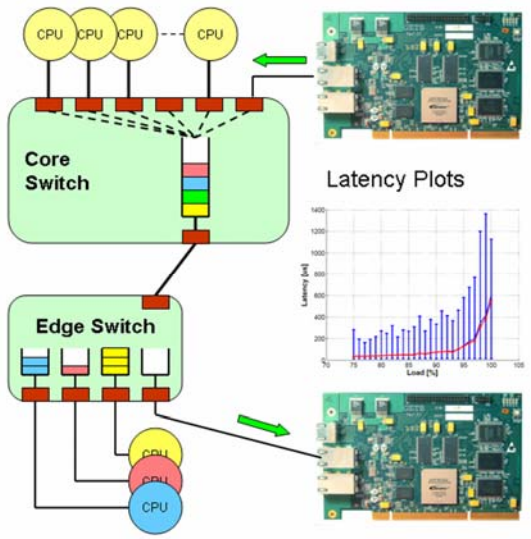

Fig. 8. Measuring queue latency.

VIII. Status and Future Plans

Operating, monitoring and maintaining this large network is accomplished using a set of tools that cover different needs. The database consistency checking application has been released and tested on the MTF and Rack Wizard databases. Network configuration is currently done by scripting and open source management tools are being studied. Spectrum, YATG and sFlow monitoring have all been tested as first releases. The bandwidth measuring application has been deployed in prototype form as has the GETB prototype for measuring queue development. The 3D status visualization package is in development.

Further work is planned with a view to ensuring that the tools described in this paper reach production maturity, and can be seamlessly integrated in the operation of the TDAQ system. Furthermore, the network monitoring data obtained will assist in the detection and diagnosis of wider system performance issues.

References

[1] M. Ciobotaru, L. Leahu, B. Martin, C. Meirosu, S. Stancu, " Networks for ATLAS trigger and data acquisition", in *Proc. Computing in High Energy Physics (*CHEP 06), Mumbai, India, Feb. 2006.

[2] B. Martin, E. Panikashvili, "Design tools for large networks", in *Proc. IEEE Real Time 2007 Conference*, Batavia, Illinois, 2007, p. (to appear).

[3] C. Delamare, A. Jimeno, S. Mallón Amérigo, E. Manola-Poggioli, P. Martel, B. Rousseau, D.Widegren, "Manufacturing and test folder:MTF", in *Proc. EPAC 2002*, Paris, France, 2002.

[4] F. Glege, "The Rack Wizard, a graphical database interface for electronics configuration", in *Proc. 9th Workshop on Electronics for LHC Experiments proceedings*, Amsterdam, Holland, Oct. 2003.

[5] The CERN IT/CS Network Services. [Online]. Available: https: //network.cern.ch/

[6] M. Hatch, "Cabling of the ATLAS experiment", in *Proc 9th Workshop on Electronics for LHC Experiments*, Amsterdam, Holland, Oct. 2003.

[7] Java SE - Java Database Connectivity (JDBC). [Online]. Available: http://java.sun.com/javase/technologies/database/

[8] Simple Object Access Protocol. [Online]. Available: http://www.w3.org/ TR/soap/

[6] Field Programmable Gate Array.

[9] Common Object Request Broker Architecture. [Online]. Available: http://www.corba.org/
[10] JavaServer Pages. [Online]. Available: http://java.sun.com/products/jsp/index.jsp
[11] ZipTie, an Open Source framework for Network Inventory Management. [Online]. Available: http://www.ziptie.org
[12] Python Programming Language – Official Website. [Online]. Available: http://www.python.org/
[13] R. Enns, Ed., "NETCONF Configuration Protocol", RFC 4741, Dec. 2006
[14] C. Lonvick, "The BSD syslog Protocol", RFC 3164, Aug. 2001.
[15] SPECTRUM® Network Root Cause Analysis & Performance Management Software. [Online]. Available: http://www.aprisma.com
[16] T. Oetiker, RRDtool. [Online]. Available: http://oss.oetiker.ch/ rrdtool/
[17] P.Phaal, S. Panchen, N.McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", RFC 3176, Sept. 2001.
[18] A. Al-Shabibi, C. Meirosu, S. Stancu, A. Topurov, "YATG: Yet Another Traffic Grapher", [Online]. Available: https://edms.cern.ch/file/839606/1/Yatg_Arch_Desc.pdf
[19] *Extensible 3D (X3D)*, ISO/IEC Std. 19775, 2004. [Online]. Available: http://www.web3d.org/x3d/specifications/
[20] Autodesk 3ds Max. [Online]. Available: www.autodesk.com/3dsmax
[21] MySQL database. [Online]. Available: http://www.mysql.com/
[22] The Xj3D Project [Online]. Available:http://www.xj3d.org/
[23] M. Segal, K. Akeley, "The OpenGL Graphics System: A Specification" [Online]. Available: http://www.opengl.org/registry/doc/glspec21.20061201.pdf.
[24] H.P. Beck et al,, "High-Level Description of the Flow of Control and Data Messages for the ATLAS TDAQ Integrated Prototypes", [Online]. Available: https://edms.cern.ch/file/391514/1.0/DC-012.pdf
[25] Christian Haeberli, "The ATLAS TDAQ DataCollection Software", PhD thesis, University of Bern, "Traffic Shaping" Section 9.3.5, page 53. [Online].Available: http://doc.cern.ch/archive/electronic/cern/preprints/thesis/thesis-2005-028.pdf
[26] M. Ciobotaru, M. LeVine, B. Martin, S. N. Stancu, "GETB, a Gigabit Ethernet Application Platform: its Use in the ATLAS TDAQ Network" in *Proc IEEE Real Time 2005 Conference*, Stockholm, Sweden, Jun. 2005