GETB, a Gigabit Ethernet Application Platform: its Use in the ATLAS TDAQ Network

Matei Ciobotaru, Stefan Stancu, Micheal LeVine and Brian Martin

***Abstract*— The ATLAS Trigger and Data Acquisition system (TDAQ) will employ a large high-speed Ethernet-based network, comprising several hundred nodes. Ethernet switches handle all the data transfers, their performance being essential for the success of the experiment. We designed and implemented a system, the GETB (Gigabit Ethernet Testbed), which can be used to assess the performance of network devices. This article introduces the architecture and the implementation of the GETB platform, as well as its applications. These include the GETB Network Tester and the ATLAS Read-Out Buffer Emulator. The features of the system are described and sample results are presented.**

***Index Terms*— ATLAS TDAQ, computer networks, Gigabit Ethernet, development platform, FPGA, Handel-C, traffic generator**

I. Introduction

THE ATLAS Trigger and Data Acquisition system (TDAQ) is responsible for the real-time filtering and transfer of the data generated by the ATLAS detector. The input event rate of 40 MHz produced by the detector is progressively reduced by the TDAQ system down to 200 Hz. The filtering is done using massive computing resources which are interconnected in a large high speed network. Fig. 1 shows a schematic diagram of the central part of the TDAQ network.

The system is organized in three levels that process detector data. Level 1 (LVL1) uses dedicated hardware to perform the first selection of events. The interesting data selected by LVL1 are stored temporarily into Read Out Buffers (ROBs) over 1600 links. Data fragments related to an event are distributed to all the ROBs. The next filtering stage is performed by algorithms running on the Level 2 computing farm (LVL2). The LVL2 processing units (L2PU) use the Region of Interest (RoI) information available from LVL1 to collect data from only a relevant subset of the ROBs and to analyze the event. When the L2PU reaches a decision, it is announced to the LVL2 supervisor (L2SV). The L2SV forwards the LVL2 decision to the DataFlow Manager (DFM), which coordinates the assembly process of the validated events. The DFM assigns a Sub-Farm Interface node (SFI) to gather all the data available for an event selected by LVL2. The SFI asks the ROBs to deliver all 1600 event fragments. When a full event is stored in the SFI, the DFM sends a "clear" message to all the ROBs announcing that they can free their buffers (the DFM also sends "clear" messages for events rejected by the LVL2). The SFI is the entry point to the third level of filtering, the Event Filter, which performs more complex physics algorithms on complete events.

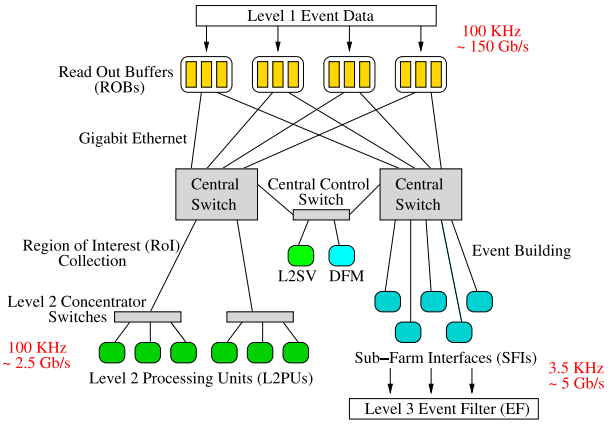

Fig. 1. The ATLAS TDAQ Network – Schematic block diagram.

The sustained data rates required for the system to operate properly are quite demanding (see Fig. 1); for example an SFI receives data from the ROBs at a rate of ≈60MB/s [1]. The technology chosen for the TDAQ network is Gigabit Ethernet (GE) [2]. The core network will be Layer 2 only; meaning that all data transfers will be handled by Ethernet switches (no Layer 3 IP routing). The need for minimal network latency and minimal packet loss imposes strict performance requirements on the switches. Therefore, each device that will be used in the network will undergo an evaluation process to make sure that it meets system requirements. In this paper we present the system created for this purpose.

II. Motivation

A set of performance metrics and test procedures has been defined in order to check the compliance of candidate devices for the ATLAS TDAQ network [3]. These procedures try to characterize the performance aspects that will be relevant for the final ATLAS data-taking phase. Commercial network testing equipment such as the Ixia Optixia or Spirent Smartbits is mainly oriented towards protocol compliance and raw performance, lacking the flexibility needed in defining ATLAS-like traffic patterns.

After researching the market we understood that a programmable platform would better suit our needs, because in

Manuscript received June 16, 2005.

M. Ciobotaru is with CERN, 1211 Geneva 23, Geneva, Switzerland and with "Politehnica" University of Bucharest, Bucharest, Romania (email: Matei.Ciobotaru@cern.ch)

S. Stancu is with University of California, Irvine, Irvine, CA 92697-4575 (email: Stefan.Stancu@cern.ch)

M. LeVine is with Brookhaven National Laboratory, Upton, NY 11973-5000 (email: levine@bnl.gov). The work of Micheal LeVine was supported by the U.S. DOE under Contract No. DE-AC02-98CH10886.

B. Martin is with CERN, 1211 Geneva 23, Geneva, Switzerland (email:Brian.Martin@cern.ch)

addition to a testing system we also envisioned other networking applications (emulation, monitoring, etc). Based on our previous experience with hardware-accelerated networking applications (the FPGA-based FastEthernet Tester and the CPU-based Gigabit Tester [4], [5]) we designed a new FPGA-based platform called the Gigabit Ethernet Testbed (GETB).

This new platform is fully programmable and delivers Gigabit wire-speed performance; it also allows us to build a high port density testbed. In the following we describe the architecture of the GETB platform and we present the projects currently using it. First we describe the Network Tester, a tool that is being used at present to evaluate network equipment for the ATLAS TDAQ network. The ATLAS ROB Emulator will be presented afterwards. A third project, a Network Emulator, is described in detail in [6].

III. Architecture

The GETB uses custom-built PCI cards and control software to provide a platform for GE applications. The hardware and software designs are presented in this section.

A. Hardware Platform

The core of our system is the GETB card; it is a custom PCI card that contains an FPGA, two Gigabit Ethernet ports and local memory. The hardware design was done at CERN. Fig. 2 shows the card with the main components highlighted.

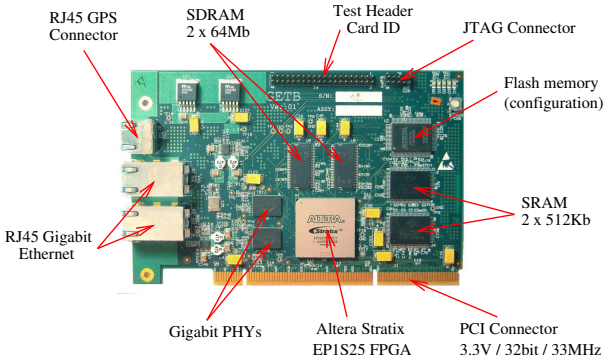

Fig. 2. The GETB Card.

The central FPGA controls all the resources available on the card. User applications can access 128 Mb of SDRAM and 1 Mb of SRAM. The SDRAM is used for tasks which require large amounts of storage space (buffering, traffic description), while the SRAM is more appropriate for time-critical operations (histograms). Depending on the application, the two GE ports can be fully independent or data can flow from one port to the other. The third RJ45 port is used for clock synchronization among multiple cards (via a clock distribution system); the PCI bus can also be used to synchronize cards in the same chassis.

The main component is the Altera Stratix FPGA, a device containing 25k logic elements. To simplify the board layout the Ethernet MAC and PCI functionality are embedded in the FPGA (using IP cores from MoreThanIP and PLDA respectively). The firmware contains blocks common to all projects (MAC, PCI, control) plus application dependent parts (see Section IV-B). Most of the high-level functionality is implemented in the Handel-C hardware description language [7]. All projects share a common low-level library that provides primitives for accessing the memories, routines to simplify the creation of Ethernet packets and a simplified PCI interface. All of the applications make heavy use of the dual-port SRAM blocks provided by the Stratix FPGA to implement queues between concurrent and asynchronous processes.

VHDL glue code is used in the firmware to inter-connect the various blocks (Handel-C netlist, MAC and PCI cores, SDRAM controllers, etc); for the entire project the size of this glue code is significant (≈2200 lines). Any design modifications would require manual changes of the code, an error prone process. To simplify its maintenance we created a tool that takes as input the names of the blocks and the relationships between them and generates the required source code (VHDL_Gen, [8]); this tool has saved considerable effort.

The GETB card fits into a standard 3.3V PCI slot. The PCI interface is used to configure the application firmware, to collect statistics and to perform firmware upgrades. The large scale testbed is built using many cards mounted in industrial PCs which are interconnected in a local network. The typical configuration for the large scale GETB testbed is shown in Fig. 3. Our current setup has 128 GE ports, but there is no technical limitation on the number of ports in the system.

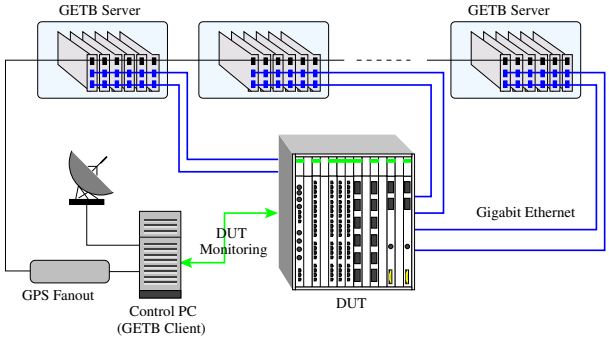

Fig. 3. Typical Testbed Setup (block diagram).

B. Control Software

The GETB provides a common control infrastructure for all the derived projects. In order to drive the system using automated procedures the control system was built using the Python scripting language [9]. This allows the user to create scripts for running tests in a very simple and flexible manner.

The machines hosting GETB cards utilize Linux and a Python server application which is responsible for the configuration, constant monitoring of the cards and the handling of remote client connections; in Fig. 3 the client is the Control PC. For the low-level interaction with the cards via PCI we use a custom Linux kernel module and its associated library (IO_RCC, [10]).

The management client runs on any machine with a Python interpreter. On the client side each of the two ports of a GETB card is seen as an individual entity and can be independently

configured. The entire system is controlled from the client PC either using the command line or by running scripts. A graphical interface is available for displaying the statistics from all the ports in the system.

IV. The GETB Network Tester

The main application of the GETB platform is the Network Tester. The aim is to allow us to evaluate devices (switches) from various manufacturers and to identify those which best suit the ATLAS TDAQ network needs.

A. Features

The Network Tester uses the GETB card to implement a traffic generator and measurement system. It can send and receive traffic at Gigabit line speed while computing real-time averages for the most important parameters of a flow (packet loss, latency and throughput). Being an FPGA-based architecture, all processing is done on the hardware without any CPU-load on the host system. Two transmission modes are supported: one in which each port is fully independent of the others (Independent Generators, IG mode) and one in which a port transmits only when requested by another port (Client Server, CS mode).

1) Transmission – Independent Generators: In the IG mode each port in the system sends traffic according to a list of packet descriptors loaded into the SDRAM of the card. Each descriptor is used to build one packet. The firmware uses the fields in each descriptor to build the outgoing packets. The user can configure the Ethernet and IP headers, the inter-packet time (IPG) and packet size for each descriptor independently, allowing a wide range of traffic patterns to be generated. For example, a negative-exponential random number generator used for the IPG produces a Poisson traffic stream. In addition to raw Ethernet and IP packets, special frames like Flow Control and erroneous frames can be transmitted.

2) Transmission – Client-Server: The second mode of operation – the client-server (CS) or request-reply mode – emulates the traffic produced by the data-intensive applications in the ATLAS TDAQ network. Some ports of the tester are configured as servers and the others as clients. The servers send data only in reply to requests coming from the clients. The load on the network is regulated by the client, which uses a token based mechanism for issuing requests. At initialization the client receives HT (maximum number of tokens), and issues requests, using one token per request, until it runs out of tokens. The client recovers a token for each message (reply) received from the server. When enough replies have been received (number of current tokens=LT (low token parameter)), requests are issued again within the limit of available tokens. The HT and LT parameters control the load created on the network. Moreover the burstiness of the traffic is determined by the value of the LT parameter.

The bursts are also determined by the length of the reply messages (which can span multiple frames). The requests are usually small frames (64 bytes) while the replies consist of one or more maximum sized frames (1518 bytes). Typically congestion is created towards the clients (many servers send large amounts of data to a small number of clients). To avoid stalling the system because of packet loss we implemented error recovery mechanism on both ends relying on packet loss detection and timeouts (see also Section IV-A.3).

The client-server mode tries to emulate the behavior of the SFI, L2PU and ROS applications in the ATLAS Network (Fig. 1). Choosing the appropriate values for the concentration ratio (number of servers vs. clients), values for the token limits and number of replies we can emulate more accurately different traffic scenarios in the ATLAS network.

Both modes of transmission (IG and CS) maintain individual statistics for the number of packets and bytes that are transmitted to each of the other tester ports in the system. All transmitted packets contain information that is used to detect packet loss and to measure latency (next section).

3) Receive path: The packet receive path is responsible for updating statistics and histograms. Each port keeps track of a set of global counters (total number of frames, bytes, different frame types, etc.) and a set of arrays of counters which are updated per source-destination flow: packet loss, average latency, average IPG.

The tester detects packet loss in real-time by embedding a sequence number into each packet sent (any gap in the received sequence numbers means usually that packets were lost[1]). The one-way latency is measured by marking each packet with a timestamp (the clocks are synchronized between cards, see Section III-A). While a test is running, the information about packet loss and latency is available for each source-destination pair.

For an in-depth analysis the user can define histograms (for latency, IPG, packet size and queue utilization). A set of configurable rules based on source ID or VLAN priority can be used to filter the frames to be logged in a histogram (conditional matching). For each histogram the user can set the minimum and maximum values, the resolution and the number of bins.

B. Implementation

Fig. 4 shows a block diagram of the Network Tester (for one GE port). Each block in the figure is a parallel Handel-C process running on dedicated hardware in the FPGA. By optimizing the code to make use of the parallelism as much as possible we have met our main requirements: to support line speed for both transmit and receive paths (TX and RX) for any kind of traffic pattern and to have the two built-in ports running simultaneously with the full set of features.

The TX processes use the SDRAM memory and the associated controller to fetch the traffic descriptors, decode them, build the packets and send them to the Ethernet MAC core (see Section IV-A.1). The time when a packet is sent is determined by the IPG time in the descriptor or, in the client-server mode, when there are enough tokens available (Section IV-A.2).

The RX updates the internal counters and in the client-server mode uses the feedback path to push requests in the transmission queue. All the processes are controlled and

[1]This can also be caused by packet reordering, but this should not appear in a Layer 2 switching device.

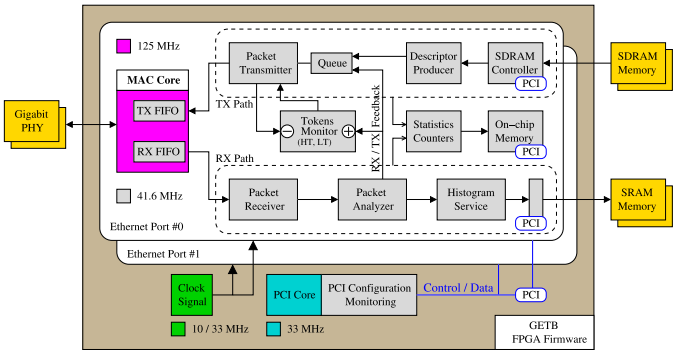

Fig. 4. The FPGA Firmware Block Diagram.

configured via PCI. The external and internal memories can also be read and written using the host PCI interface.

C. Operation

The tester operation is entirely managed by the control system described in Section III-B. Python scripts are available to run all the tests described in the ATLAS requirements document for testing candidate devices [3]. The tests are performed automatically by iterating over a parameter space (i.e. modifying the network load, traffic pattern, switch settings, etc). The results and the plots are automatically saved. At each test iteration consistency verifications are performed. In order to control the device under test (DUT) from the testing scripts, we developed a Python interface that is used to configure and monitor the tested device (sw_script, [11]). Using this feature the statistics reported by the tester are cross-checked with those reported by the DUT. This switch-dependent procedure is currently implemented for products of the major switch manufacturers.

D. Sample Results

We present a few results obtained using the GETB Network Tester.

1) Size of the MAC Address Table: The first example is the determination of the effective size of the Forwarding Database (FDB), the number of MAC addresses a switch can learn. The FDB contains the mapping between end-nodes and switch ports. When it is full, it leads to flooding (a packet is forwarded to all ports) and to an inefficient bandwidth use. The tester is configured to send a stream of packets with random values for source MAC addresses. This forces the switch to (try to) learn all the addresses; the FDB can be filled up in this way. The rate at which the packets are sent corresponds to 10% utilization of the GE link. Fig. 5 shows the number of addresses that are effectively learned as the number of offered addresses is increased (for the three address patterns indicated).

The results are surprising, given that the device advertises a table size of 16000 entries and a line-speed learning rate. We observe a deviation from the ideal curve when more than 5000 addresses are used. The central switches in ATLAS need to learn at least 4000 addresses; thus we should be aware of any limitations in the device.

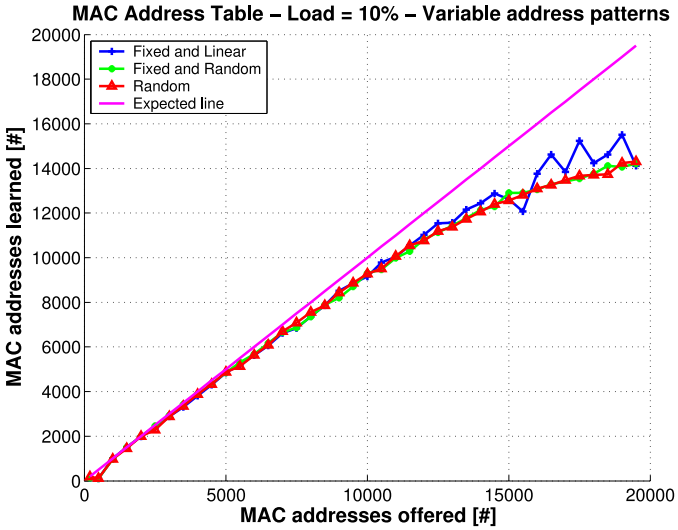

Fig. 5. Measuring the size of the MAC table.

2) Quality of Service (QoS) Tests: In the second example we show how to check the QoS features of the DUT. Congestion on a switch port is created using multiple traffic flows with different priorities. When the load of each transmitter increases above a certain threshold (depending on the number of senders, all sending at the same rate) the scheduling algorithm of the switch must allocate more bandwidth resources to the high priority flows. In this example 8 priorities have been used and the loss rate for each flow (Fig. 6) was measured. We observe that the switch properly implements a weighted round robin (WRR) scheduling scheme (which guarantees a portion of the bandwidth to each flow). The measurement is possible because the congested port keeps track of individual statistics for each incoming traffic stream. The ATLAS network may use QoS to optimize the flow of important messages (from the L2SV to the DFM for example).

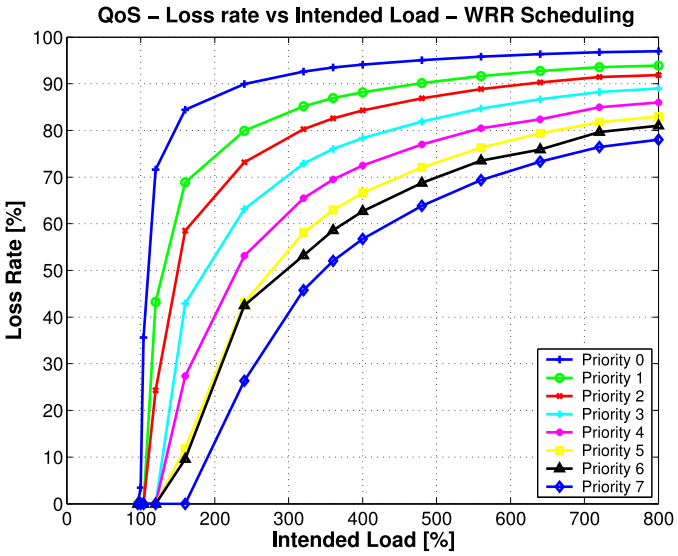

Fig. 6. Quality of Service measurement.

3) Buffering capacity and the ATLAS traffic pattern: The ATLAS TDAQ network contains a few places where a number of traffic streams are concentrated to a single destination (a funnel shaped traffic pattern, e.g., the Event Building traffic with data coming from all the ROBs to a single SFI). The performance in such conditions depends on the amount of buffering inside the switch.

We developed methods to measure the effective buffering capacity for a port on the switch. Fig. 7 shows how many packets can be stored in the device, for different packet sizes and for 3 different devices. The shapes of the curves can give us insight to the internal memory allocation policy. For example Device A seems to split and store packets in small cells while Device C uses a fixed-size slot for any packet size. Device B uses a combination of the two: a fixed slot for packets up to 1100 bytes and a fine granularity cell-based allocation for larger packets.

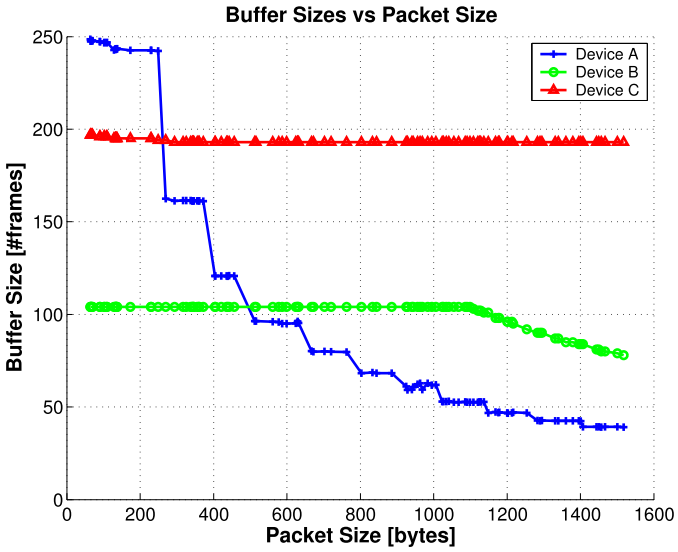

Fig. 7. Buffer capacity for different packet sizes and for different switches.

The raw buffer size is not enough to characterize a device. The way the switch deals with congestion internally is also important. Fig. 8 shows the loss rate in an ATLAS traffic scenario for 5 different buffering levels; these measurements are taken for a device that has user-configurable buffers. The concentration ratio[2] was 6 to 1 (six times more servers than clients). The x-axis represents the intended load to be obtained on the congested ports, and the loss rate is shown on the y-axis. We observe that for small numbers of available buffers, packet loss starts to appear at 85% load and this onset increases to 95% when more buffers are allocated. Although the loss rate is not large (2-3%), it carries a substantial performance penalty in the request-response dialog in ATLAS, due to the time lost in timeouts. Recalling that an event build requires 1600 response packets, a loss rate of even 0.1 per cent will result in timeouts for every event.

[2]The concentration ratio represents the number of servers over the number of clients, keeping the terminology from Section IV-A.2.

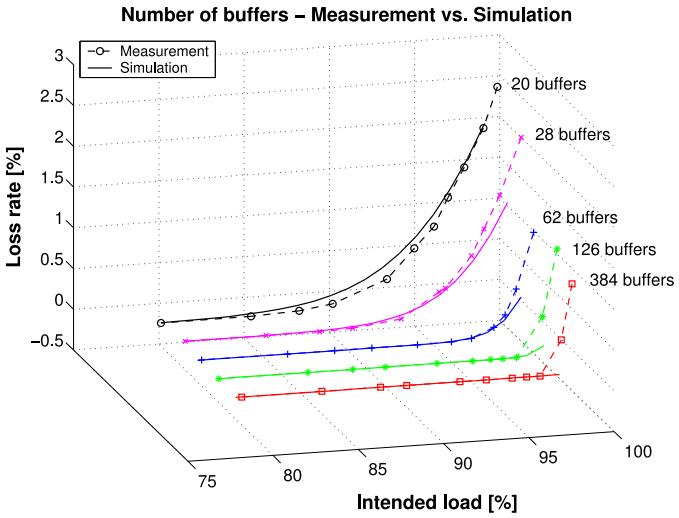

Fig. 8. Impact on buffer sizes on ATLAS traffic performance – Measurement vs Simulation.

The figure contains also the results of simulations[3]. Our analytical model assumes that the packets are lost only because of queues filling up in the switch (lack of buffer space). We observe that for small amount of buffering we have very good agreement between measurements and predicted behavior. For larger number of buffers the curves diverge because in reality packet loss may appear because of other factors (for example a rate limitation on a component on the internal packet path of the switch). These analytical models can give us some hints on how to dimension the final system to run in the lossless region, provided that we know the depth of buffers and the traffic patterns.

V. The ATLAS ROB Emulator

The interface between the ATLAS detectors and Trigger-DAQ (TDAQ) is the ROB, an intelligent buffer management card that receives event fragments from detectors on three optical links, storing the fragments in local buffers. The ROB responds to requests for these fragments from L2 trigger processors (L2PU) and from the event builders (SFIs), and requests to free buffers associated with specified events. There will be over 500 ROBs in ATLAS. The ROB is connected to TDAQ in two ways: it is equipped with a GE port as well as a PCI interface to its PC host. Thus the ROBs may be configured as independent devices connected to the TDAQ network fabric, or they can be managed actively by their PC host, configured as a Readout System (ROS).

The hardware configuration of the TDAQ network as well as the software running on the L2PUs and SFIs, need to be validated under conditions similar to or more demanding than those expected under running conditions in ATLAS. Performance is an issue, as is robustness of the software in various failure scenarios, particularly buffer over runs in switches and the OS network stacks. A credible validation requires a test installation of a reasonable size. TDAQ has

[3]The loss probability of an M/D/1/K queue has been computed for the corresponding egress buffer depth, using the algorithm described in [12]

agreed upon a test installation (pre-series) comprising ten per cent of the total number of units of each type: 50 units in the case of the ROB.

The manufacturing process of the ROBs is beginning; the first units are in demand by the detectors for early commissioning. It is expected that a sufficient quantity of ROBs will not be available in order to provide 50 for the pre-series installation. For this reason, a ROB emulator (ROBE) has been implemented using the GETB card. A GETB card contains two independent ROBE implementations. In its simplest use, the ROBE emulates the ROB directly connected to the TDAQ network.

The ROBE responds to requests using the UDP protocol. The format for the event fragments delivered conforms to the ATLAS event format [13]. The length of the data payload in the fragments is configurable from the host and can also be specified by the requester, in order to generate realistic event traffic. The content of the data payload is without meaning. In order to emulate the ROB, it is capable of generating 20K responses per sec.

The ROBE capabilities have been extended to the emulation of a complete ROS. It handles requests that would be sent to a ROS, asking for the contents of multiple ROBs; responses are correctly formatted as ROS event fragments. Requests to delete event fragments are ignored, since no fragments are stored in this emulation. Responses, which can span multiple frames, are limited in length only by the limit of a UDP datagram (64K Bytes). The ROBE also responds to ARP requests for one or more IP addresses. The ROBE supports the VLAN protocol.

The clock speeds achieved in this application are modest (41 MHz), but still allow the ROBE to respond to back-to-back bursts of GE requests.

Requests are buffered in a 256-Byte FIFO provided by the MAC. Incoming frames are copied immediately to a private frame buffer. The contents of this buffer are parsed and, when a reply is required, converted to a descriptor, a 100-Byte structure that contains all the information necessary to construct a reply. These structures are written to a queue implemented in an on-chip 64 kByte SRAM. The queue has space for over 600 pending responses. To protect against overrun of the receive FIFO, when the queue level exceeds a high water threshold, a IEEE 802.3x PAUSE frame is emitted. The PAUSE frame is overridden when the queue level drops to a low water level.

The implementation comprises 3 concurrent processes: packet receiver, response generator, and packet transmitter. The response generator takes its input from the response descriptor queue, and formulates a response, which is fragmented into packets as necessary. Completed packets are placed in a queue, which is emptied by the packet transmitter.

The functionality of the ROBE is configurable using the PCI host interface. Ethernet and IP addresses, IP Port numbers, the number of ROBs in a ROS, are some of the parameters that can be set. The GETB infrastructure also provides statistics on number of requests received, responses generated, and errored frames, via PCI.

The ROB emulator has been tested in a dedicated test situation and has been proven to be robust and to meet performance requirements. It is ready to be installed in the pre-series test bed.

VI. Conclusion

We present the GETB, a platform that provides a flexible environment for the design and development of Gigabit Ethernet applications. A network tester capable of generating traffic similar to the one produced by the ATLAS Data Acquisition software has been designed and implemented based on the GETB platform. The tool proved to be extremely useful and efficient in evaluating switches from various manufacturers. In addition, a ROB emulator application allows large-scale testing of the data acquisition network before detector commissioning. Other networking applications making use of the GETB infrastructure are also possible: a network emulator is now being developed; a network analyzer is another potential application of the platform.

Acknowledgment

The authors would like to thank our colleague Jaroslav Pech whose skill in designing, debugging and preparing the GETB for production made this project possible. We would also like to thank the Networking Group at CERN and the ATLAS TDAQ Collaboration for their support.

References

[1] S. Stancu, M. Ciobotaru, and K. Korcyl, "ATLAS TDAQ DataFlow Network Architecture Analysis and Upgrade Proposal," in *Proc. IEEE Real Time 2005 Conference*, Stockholm, Sweden, June 2005, p. (in press).

[2] S. Stancu, B. Dobinson, M. Ciobotaru, K. Korcyl, and E. Knezo, "The use of Ethernet in the Dataflow of the ATLAS Trigger and DAQ," *ECONF, Proc. of CHEP03 Conference*, vol. C0303241, p. MOGT010, 2003. [Online]. Available: http://arxiv.org/pdf/cs.ni/0305064

[3] S. Stancu, M. Ciobotaru, and D. Francis, "Relevant features for DataFlow switches," CERN, Tech. Rep., 2005. [Online]. Available: http://sstancu.home.cern.ch/sstancu/docs/sw_feat_noreq_v0-5.pdf

[4] F. R. M. Barnes, R. Beuran, R. W. Dobinson, M. J. LeVine, B. Martin, J. Lokier, and C. Meirosu, "Testing Ethernet Networks for the ATLAS Data Collection System," *IEEE Trans. Nucl. Sci.*, vol. 49, pp. 516–520, Apr. 2002.

[5] R. Dobinson, S. Haas, K. Korcyl, M. J. LeVine, J. Lokier, B. Martin, C. Meirosu, F. Saka, and K. Vella, "Testing and Modeling Ethernet Switches and Networks for Use in ATLAS High-level Triggers," *IEEE Trans. Nucl. Sci.*, vol. 48, no. 3, pp. 607–612, 2001.

[6] M. Ivanovici, R. Beuran, and N. Davies, "Assessing Application Performance in Degraded Network Environments - an FPGA-based approach," in *Proc. of the MAPLD International Conference*, Washington, D.C., Sept. 2005, p. (in press).

[7] Celoxica – The Handel-C Language. [Online]. Available: http://www.celoxica.com/technology/c_design/handel-c.asp

[8] M. Ciobotaru. VHDL_Gen – Creating hardware using Python. [Online]. Available: http://ciobota.home.cern.ch/ciobota/project/vhdl_gen/

[9] The Python Language. [Online]. Available: http://www.python.org/

[10] M. Joos, "IO_RCC – A package for user level access to I/O resources on PCs and compatible computers," CERN, Tech. Rep. ATL-D-ES-0008, Oct. 2003. [Online]. Available: https://edms.cern.ch/document/349680/2

[11] M. Ciobotaru. sw_script – Unified interface for switch configuration and monitoring. [Online]. Available: http://ciobota.home.cern.ch/ciobota/project/sw_script/

[12] O. Brun and J. M. Garcia, "Analytical solution of finite capacity M/D/1 queues," *Journal of Applied Probability*, vol. 37, no. 4, pp. 1092 – 1098, Dec. 2000.

[13] C. Bee, D. Francis, L. Mapelli, R. McLaren, G. Mornacchi, J. Petersen, and F. Wickens, "The raw event format in the ATLAS Trigger and DAQ," CERN, Tech. Rep. ATL-D-ES-0019 v.3, Apr. 2005.