Modeling large Ethernet networks for the ATLAS high level trigger system using parameterized models of switches and nodes.

Piotr Golonka Krzysztof Korcyl Frank Saka

Abstract— **Large local area Ethernet networks are strong candidates to connect data sources and processing nodes in high energy physics experiments. In the high level trigger system of the ATLAS LHC experiment several Gbytes/s of data, distributed over 1700 buffers, have to be delivered to around a thousand processing nodes. Due to the network size, its performance and scalability can only be assessed by modeling. To avoid lengthy simulation runs, and concentrate only on characteristics important for network transfers, the components of the system need to be parameterized.**

The network performance depends on traffic patterns generated by processing nodes and switching capabilities of the network, we therefore evaluated and modeled both processing nodes and switches. We have developed a parameterized model of a class of switches, where a limited set of parameters, collected from measurements on real devices, is used to model switching characteristics. Another set of simple measurements is used to collect values for parameters used to model processing nodes running the Linux operating system and the TCP/IP communications protocol suite.

In this paper we present the set of parameters used in the models together with measuring procedures used to calibrate our models. Calibrated models are used to model small test-bed setups with random traffic to validate our approach.

Keywords— **Ethernet, switches, TCP, Atlas**

I. Introduction

THE ATLAS detector is designed to investigate many different physics processes at the Large Hadron Collider (LHC) and is due to begin operation in 2005. Proton-proton bunch crossing will occur at 40 MHz, of which 100 kHz rate of interesting event will be filtered out by the first level trigger (LVL1). The high level trigger (HLT) composed of level 2 (LVL2) and Event Filter (EF) further limit the rate of accepted events to tens of Hz, the rate manageable for the tape/disk drives. To reduce bandwidth requirements for data transfers the LVL2 system will use information from the LVL1 to read full granularity data from a subset of detector buffers, typically 100 out of a total of 1700, containing interesting signatures (*Regions Of Interest*). These data have to be delivered to a member of the LVL2 processor farm (a set of 550-600 commodity PCs) for further processing. Events accepted by the LVL2 (1 kHz) require that data from all detector buffers are collected by an event builder to a PC belonging to a subfarm for final processing. The network connecting 1700 buffers and around thousand LVL2 processors and EF collecting nodes, has to provide a throughput of more than 6 GBytes/s throughput. A possible architecture for such a network using Ethernet technology has been presented in [1]. A number of concentrating switches aggregating data from detector buffers connected to Fast Ethernet will be linked via Gigabit Ethernet uplinks to the central Gigabit Ethernet switch. A similar concentration is foreseen for connecting the LVL2 processing PCs on Fast Ethernet and connected to the concentrating switches with the Gigabit Ethernet uplinks to the central switch. It is expected that more than a hundred concentrating switches will be necessary to provide the required throughput. The size of the network excludes building a prototype. Performance of such a large network can only be estimated by modeling. In our approach to building a model of the whole ATLAS HTL system we developed the parameterized model of switches [1], [2]. In the first part of this paper we review operation of the model with recently added extensions. The extensions were made to accomodate switches which split Ethernet frames into small cells when transfering them internally.

In the course of our studies on the ATLAS HLT system the issue of whether high level communication protocol such as TCP/IP [4] could be used for data transfers arose. This led us to design and the development of a parameterized model of TCP/IP stack. In the second part of this paper we present our approach to parameterization of the TCP behaviour relevant to the operation in the HLT system.

II. The parameterized model of Ethernet switches.

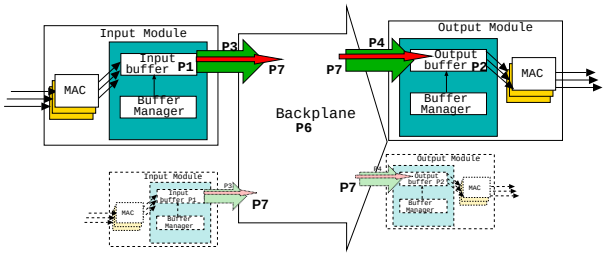
The parameterized model reflects the hierarchical architecture of the switch (ports are gropued into modules, modules are connected by a backplane to provide inter-module transfers) and implements store-and-forward mode of operation. In the Figure 1 the parameterized model of a switch for inter-module communication is presented together with a full list of parameters (including parameters for intra-module transfers). Full description of parameters can be found in [1]. The operation of the parameterized model is based on calculations using parameters representing buffering and transfer resources in the switch. The operation also assumes that any limitations of the transfer resources

P.G. is with CERN, on leave from the Faculty Of Nuclear Physics and Techniques, Univesity Of Mining and Metallurgy, Cracow, Poland; e-mail:Piotr.Golonka@CERN.CH

K.K. is with CERN on leave from the Institute Of Nuclear Physics, Cracow, Poland; e-mail:Krzysztof.Korcyl@CERN.CH

F.S. is with CERN, on leave from the Royal Holloway University of London, Egham, UK; e-mail:Frank.Saka@CERN.CH

CERN, European Organisation for Nuclear Research, 1211 Geneva 23, Switzerland.




```
P1 = Input buffer length (#frames)
P2 = Output buffer length (#frames)
P3 = Max to backplane throughput (MBytes/s)
P4 = Max from backplane throughput (MBytes/s)
P5 = Max throughput for intra-module(MBytes/s (not shown)
P6 = Max backplane throughput (MBytes/s)
P7 = Inter-module transfer bandwidth (MBytes/s)
P8 = Intra-module transfer bandwidth (MBytes/s)(not shown)
P9 = Inter-module fixed overhead (not shown)
P10= Intra-module fixed overhead (not shown)
```

Fig. 1. Switch parameterized model for inter-module communication

in the switch can be modeled as input queuing. When the frame arrives at the switch, a check is made to see whether there are enough resources to buffer the frame (the current count of frames buffered in the buffer should not exceed the parameter P1). If result of the check is negative, the frame is dropped. Once the frame is buffered in the input buffer, the current count of buffered frames in the source module is increased and the routing decision is made. Depending whether it is an intra-module or inter-modules transfer, the corresponding parameter P9 or P10 is used to model the fixed overhead time for taking the routing decision. Currently there are 4 types of transfers: inter-module unicast, inter-module multicast, intra-module unicast and intra-module multicast. The routing decision is followed by the resource calculations made using P3, P4 and P6 or P5, depending on the transfer type. For example: in the case of inter-module unicast transfer, the resources for a single frame transfer from the input buffer of the source module to the backplane (limited by the P3), transfer over the backplane (limited by the P6) and transfer from the backplane to the output buffer of the destination module (limited by the P4) will be necessary. The frame transfer is seen as a request to provide a certain amount of bandwidth needed to commence the transfer for inter-module transfers. The requested bandwidth is represented by the parameter P7, and for an intra-module transfers, the requested bandwidth is represented by the parameter P8. Frames currently being transferred occupy some part of the throughput represented by parameters P3, P4 and P6 for the inter-module transfers and P5 for the intra-module transfers. Together with evaluation of the transfer resources, another check is made to verify if there is enough buffering capacity in the output buffer of the destination module. If the available throughput is larger or equal to the requested bandwidth, and there is buffering available, then the frame can start to transfer. Newly inserted frames reduce the available throughput by a fraction corresponding to the parameter P7 or P8, depending whether they are inter or intra-module transfers. The current count of buffered frames in the output buffer is also incremented. Once the resources have been allocated, calculations are made to obtain the resource occupancy times. These times are calculated based on parameters P7 and P8 and the frame size. The larger the frame size, the longer the resource will be allocated. When the available throughput is less than that requested throughput, the frame has to wait until the necessary resources become available (usually when another frame leaves the switch). If there are more frames waiting for resources, it is up to the input buffer manager to decide which frame will be transferred next. The input buffer manager may implement different policies to make a decision: the frame waiting the longest time, the high priority frame etc.

Many modern switches split a frame received from a network into small cells (typically 64 bytes) and use these cells for inter-module transfers. It may happen that two or more transfers originating from different source modules and destined for a common destination module will overlap in sense that the link from the backplane to the destination module will be used to interleave cells from these transfers. As a result the link bandwidth will be shared by the transfers resulting in longer latencies - the more concurrent transfers there are, the smaller the bandwidth available for each one and longer latency. The same scenario can be applied to more than one transfers originating from the same source module and going to different destinations. In this case the shared link is from the module to the backplane. In the parameterized model the limitation on a link throughput is represented by P3 and P4 parameters. When modeling switches with shared links these parameters are not used. In the model we always accept a request for a transfer. Any time a new transfer is added, the new share of bandwidth for each current transfer is calculated and used to recalculate transfer completion time. Each time a transfer completes, the remaining transfers share equally the link bandwidth and new calculations for their completions are made.

When the frame arrives at the output buffer of the destination module, it frees the allocated transfer and buffering resources in the input buffer of the source module. It is then up to the output buffer manager to decide which frame from the output buffer will be sent out next. Similar to the operation of the input buffer manager, the output buffer manager can implement different policies when making its decision. When the frame finally leaves the switch via the Media Access Controller (MAC), the current count of frames in the output buffer decrements. The allocation of resources for multicast or broadcast transfer might be different from the single frame transfer. The policy of handling the multicasts and broadcast transfers is strongly bound to the internal detail of switch, and we have not found any generalizations there. Currently, the model creates a copy of the multicast (broadcast) frame for each remote module housing at least one destination port and it is tretaed in the same way as unicast.

A. Measurements of the parameters for the switch model.

In the paper [1] we reported limitations on measurements imposed by using PCs. Using programmable network inter-

face cards ("intelligent" NICs) [2] we were able to generate traffic at GigabitEthernet line speed. Problems of saturating switches with large valencies were solved by a FPGA-based testers [2] capable of generating line speed traffic on 32 Fast Ethernet ports. The basic measurements used for calibration of parameters are ping-pong and streaming.

- The Ping-Pong measurements: client sends a variable size request to a server which replies with a response of the same size. The latency of the frame's round trip time is measured and used to calculate the transfer bandwidth.
- The streaming measurements: with flow control disabled, the client(s) generate multiple frames continuously with a fixed inter-frame time. Reducing the inter-frame time we look for coresponding banwidth at which the switch starts loosing frames. This defines the limits of switch throughput.

We show how results from basic measurements are used to extract values for parameters in the table:

Setup of the measurement	What is measured	What can be learned from the measurements
Ping Pong		
Intra-module	Latency L	$P10$ and $P8$ from $L = \frac{frame\ size}{P8} + P10$
Inter-module	Latency L	$P9$ and $P7$ from $L = \frac{frame\ size}{P7} + P9$
Streaming		
Intra-module	Achieved rate	Maximum throughput: $P5$
Inter-module	Achieved rate	Maximum throughput: $P3$,$P4$,$P6$

B. *Comparison of parameterized model with measurements.*

In the Figure 2 we compare latency distribution produced by the parameterized model using parameters collected on a 48 port Fast Ethernet switch to the latency measured in the test setup with a real switch[1]. The switch under test uses small cells for inter-module transfers, as described in the section II. In the test setup, we used 32 port Fast Ethernet tester generating random destination traffic with inter-packet time picked up randomly from a negative exponential distribution with an average corresponding either to 50% or 75% of maximal load for two data sizes 700 Bytes and 1500 Bytes respectively. The latency distribution is presented as a fraction of frames not reaching their destination within time shown on the latency axis. The biggest discrepancy for the 50% load can be seen for latencies in the range between 400 and 440 μs. Model predicts that one frame per 1000 will have latency greater than 440 μs, whereas measurements show that 1 frame per 1000 arrives to the destination later than 420 μs (error is still less than 5% for one permill of frames).

III. Parameterized model of nodes.

High level communication protocols may have substantial impact on the type of traffic and performance of the

[1]only FE ports were used

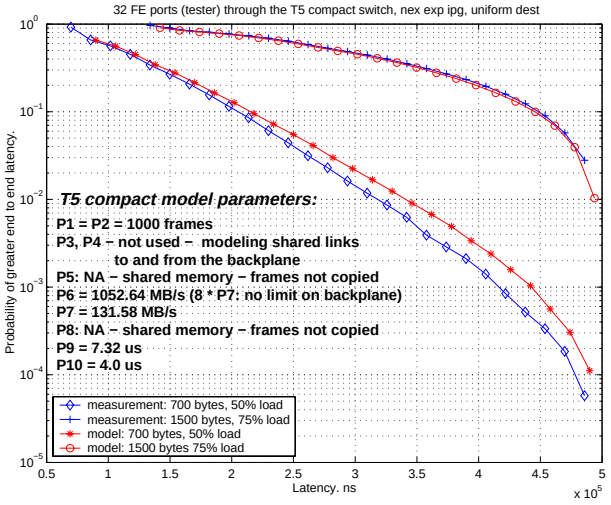


Fig. 2. Comparison between parameterized model and measurements

ATLAS HLT system. Protocol control messages may increase the overal number of packets in the system, protocols' algorithms may change the time distribution of packets (traffic patterns). Moreover, serving a stack of protocols requires additional computing power, thus it decreases CPU time available for computation of other tasks.

We use Linux implementation of TCP/IP for our studies and model calibration, as it is likely to be used by ATLAS. We tried to ommit system dependent details and perform our studies on higher level of abstraction, thus making it possible to model other implementations in the future.

A. *Network I/O : stack of protocols.*

We simplified the behaviour of the TCP stack in a single logical block called "TCPBroker". It models network processing steps ranging from the operating system application interface to the NIC. It is also able to model CPU loading due to communication. We used [3] as the main source of information about TCP/IP stack.

Data and control flow for Linux network input, together with parameters we use in our model are presented in Figure 3. One should notice, that the processing of an incoming packet in real system is split into two parts. When a new packet arrives at the network card, an interrupt service routine is executed. Then protocol-related, "bottom part" routines are executed with high priority, but outside the interrupt handler [5].

This two-step processing is modeled inside the TCPBroker block. As we perform our modeling in the time domain using discrete events, we calculate latencies and execute time-driven event sequences using queues of messages. The latency for the packet that comes from the network is accounted in steps, as the packet traverses the TCPBroker structure.

The first stage models interrupt handler execution. It increases the latency for each received packet packet by a constant value. It also sends a request for high-priority processing time to the CPU module (CPU usage modeling is described further in section III-C.) The frame is put in

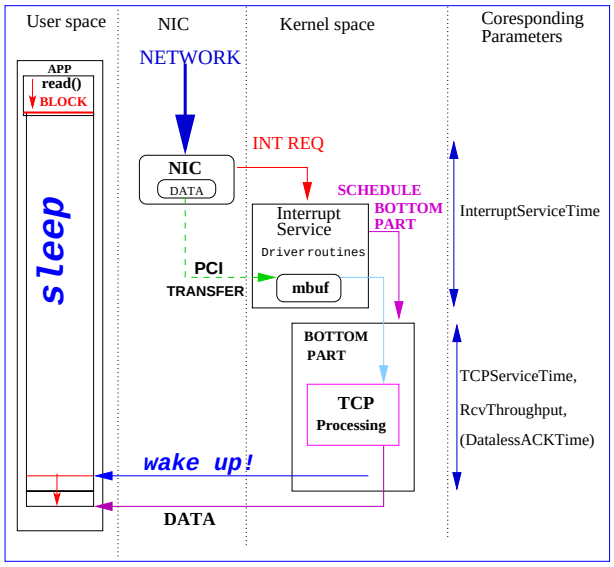


Fig. 3. Control and data flow for network input operation.

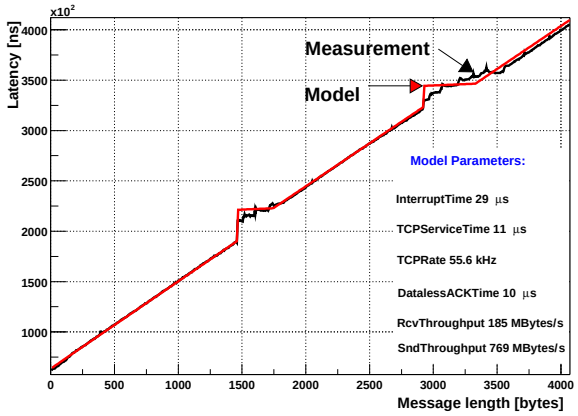


Fig. 4. Comparison of model and measurements for data latency in a simple request-response test.

a queue of packets that wait for "bottom part" processing.

Second step of modeling simulates "bottom part" processing, i.e. handling all aspects of network protocol stack. At this stage, the latency of processed packet is increased again and CPU time is requested, according to a set of parameters described in the next section. In real system, "bottom part processing" execution may still be altered by a network interrupt. We model this behaviour as well: when a packet arrives at NIC while second step processing is simulated, additional latency must be accounted for, in the packet processed in second step, in order to model suspension of processing due to the interrupt.

Second step of processing has high priority, thus no data may be passed downstream to lower priority routines (user space/application), while any frame is pending for processing. Due to this fact, we need to add another queue in our model. When the second step processing is completed, it is put to this queue, waiting until all pending frames execute their "bottom part". In this case, the latency for a waiting frame is increased accordingly in our model. If there are no more frames pending for second step processing, all frames from the queue are passed to a model of the application (see section III-C).

B. *TCP model parameters.*

In our studies we need a simple behavioural model of TCP, as ATLAS trigger network will be a local area network. We decided to implement the following parts of the TCP in our model: connection establishing phase, address and TCP ports demultiplexing, splitting/reconstruction of data stream to/from packets.

A peculiarity of TCP protocol for request-response type of traffic is sending acknowledments for received data. These acknowledgments may use either special, dataless packets (we use a term "dataless ACK packets"), or may be "piggybacked" to a data packet. The whole mechanism was modeled. Neither the retransmission mechanism, nor roundtrip-time estimation algorithms were implemented, as they are not of great importance to ATLAS. We found that introducing the following parameters is sufficient to model traffic patters in request-response data transfers:

- *TCPServiceTime, RcvThroughput:* control receive process. When a packet arrives from a network, coresponding latency is calculated as: $TCPServiceTime + \frac{RcvThroughput}{packetlength}$
- *TCPRate, SndThroughput:* control send process. When certain amount of data is to be sent, it is split into packets, then latency for each packets is calculated as: $TCPRate^{-1} + \frac{SndThroughput}{packetlength}$
- *DatalessACKTime:* time taken when an acknowledgment packet with no data ("dataless ACK") needs to be sent [2].
- *InterruptServiceTime:* time taken in network interrupt handler routine. It is accounted for any frame that is received from network.

In the figure 4 we present comparison of results given by our model with measurements for simple request-response test. Values of parameters used in simulation are presented as well.

C. *CPU and application modeling.*

Apart from network traffic patterns, we also model CPU utilization on computation nodes. Our measurements indicate that huge amount of CPU time is consumed for network communications.

In addition to TCPBroker, our model of computation node contains simplified model of a CPU and multi-tasking operating system. One or more[3] application models assigned to a node may request processing time from CPU. TCPBroker block may also send "interrupt" requests to the CPU, as mentioned in previous sections, thus delaying the application's requests.

When the amount of CPU time requested by the application is accepted and elapsed, the application model is informed about this. When no application requests CPU, and no interrupts are raised, the CPU is in idle state. By

[2] In fact this is the only parameter that is peculiar to TCP protocol.
[3] functionality for multiple applications sharing the same processor will be implemented soon

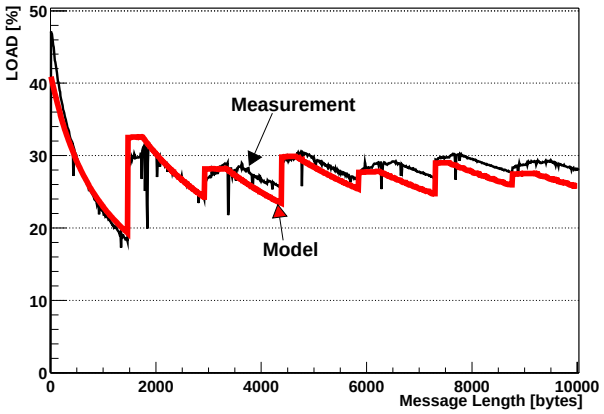


Fig. 5. Comparison of model and measurements for CPU load in a simple request-response test.

comparing non-idle time to total time for a CPU we can estimate the average CPU load.

In the figure 5 we present comparison of preliminary results produced by the model with measurements for request-response test. Although the results produced by a model does not precisely refect measurements, we can observe aproximate value and data length dependency of CPU load caused by network I/O. The reason for this discrepancy is that our model of an application was very simple: it used only one parameter to represent the time used by application. We stress that these results were obtained using parameters related to data latency only (no operating system related parameters, e.g. process switching time) have been modeled.

D. *Parameters measurement procedure.*

The setup for the measurements consisted of two PCs connected directly by Fast or Gigabit Ethernet. Both of them were of the same type to assure symmetric setup: using 400MHz Intel Pentium II with 128 MBytes of main memory. We used Linux kernel versions 2.4.0-test10 and 2.4.2 and performed our tests using IP version 4 and standard Ethernet frames without VLAN tags. A variation of ping-pong test, which was described in previous chapter, was performed: client and server were exchanging a message of variable size using already established TCP connection. Round trip time was measured and divided by two to estimate latency for sending packet one way. In order to measure CPU usage, another thread: "CPU burner" was run on client machine. Its only task was to increment a counter. By comparison of counters indications for "idle" (i.e. not communicating) and "busy" (running ping-pong test) on client, we found the fraction of CPU time used for communication. To trace activity of a real system performing TCP/IP communication we used kernel profiling facility called *Linux Trace Toolkit* [5]. We found this tool very useful to look inside sequences of events that occur on a machine during transmit and receive.

TCPServiceTime and *TCPRate* have been measured using the latency plot (Fig. 4) for the zero length message (fixed overhead). Proportion between *TCPServiceTime* and *TCPRate* was found using the LTT tool. *RcvThroughput* and *SndThroughput* contibute to the slope of latency plot for messages shorter than one Ethernet segment. The *RcvThroughput* can be measured from the slope of latency plot for messages spanning across two Ethernet segments (*SndThroughput* contributes there as a fixed overhead). *InterruptServiceTime* can be measured from the height of step at the Ethernet segment boundary from the same plot. Different heights of the steps for 1460 and 2920 can be used to measure the *DatalessACKTime* (reception of a second segment makes TCP produce and send the dataless ACK. [6]

Another tool which was used was `tcpdump`, which is a standard utility program found in many UNIX distributions. We used it to understand traffic patterns generated at the client and at the server nodes.

IV. Conclusion.

In this paper we presented our work to develop simulation tool to be used in modeling the ATLAS HLT system.

A parameterized model of switches has been verified on a number of commodity off the shelf products. The model has been recently extended to include devices using small cells to transfer Ethernet frames internally. A good agreement has been reached between measurements and results from the modeling of small setups with multiple switches.

Modeling of networking nodes is still in progress. However, simulation of the TCP protocol stack seems to adequqtely represent measurements for simple applications including CPU load. This leads us to conclusion, that our simplified, parameterized model of protocol stack performs quite well. No scalability issues for nodes modeling were assessed in the paper, this important will be addressed in the future. For further references and slides from the conference poster, please see [6].

Acknowledgments

We would like to acknowledge a contribution to our work from R.W. Dobinson, C. Meirosu, M. LeVine and R.F. Beuran.

References

[1] Dobinson, R.W ; Korcyl, K. ; Saka, F. ; *Modeling large Ethernet networks using parametrized switches.* OPNETWORK 2000 conference, Washington DC, 28 Aug - 1 Sep 2000, `http://nicewww.cern.ch/korcyl/opnetwork2000/op2k_paper.pdf`

[2] Dobinson, R W ; Haas, S ; Korcyl, K ; Le Vine, M ; Lokier, J ; Martin, B ; Meirosu, C ; Saka, F ; Vella, K ; *Testing and Modeling Ethernet Switches and Networks for Use in ATLAS High-level Triggers* CERN-OPEN-2000-310 ; Pres. at: Workshop on Network-Based Data Acquisition and Event-Building, Lyon, France, 20 Oct 2000

[3] Stevens, W.R., *TCP/IP Illustrated , vol.1,2* Adison-Wesley Publishing Company, 1994. ISBN 0-201-63346-9

[4] Postel, J.B., ed.1981c. *Transmission Control Protocol* RFC793

[5] Details and code for the Linux Trace Toolkit can be found at `http://www.opersys.com/LTT/`

[6] `http://cern.ch/Piotr.Golonka/conferences/RT2001.`