# Reliability of datagram transmission on Gigabit Ethernet at full link load

## LHCb Technical Note

| | |
|---|---|
| Issue: | 1 |
| Revision: | 0 |
| | |
| Reference: | LHCB 2004-030 DAQ |
| Created: | 17th Oct 2003 |
| Last modified: | 31st Mar 2004 |

**Prepared By:** A. Barczyk, A. Carbone, J-P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, G. Peco, V. Vagnoni

## Abstract

The reliability of datagram transmission over Copper Gigabit Ethernet using commodity hardware at sustained Gb/s rate is crucial for the functioning of the software trigger layers of the LHCb experiment. We aim to demonstrate that a nowadays available high-end commodity PC can be employed to achieve the required network performance, in particular to implement a sub-farm controller node. To evaluate all components ranging from the physical medium up to the operating system running on the sub-farm controllers, several issues are addressed, such as transmission error rate, packet drop in switching hardware, protocol handling on reception.

## Document Status Sheet

| **1. Document Title: Reliability of datagram transmission on Gigabit Ethernet at full link load** | | | |
|---|---|---|---|
| **2. Document Reference Number: LHCb 2004-030 DAQ** | | | |
| **3. Issue** | **4. Revision** | **5. Date** | **6. Reason for change** |
| 1 | 0 | 31st Mar 2004 | First released version |
| Draft | 0 | 17th Oct 2003 | Document created |

# Table of Contents

# List of Figures

## List of Tables

# 1. Introduction

## 1.1. The LHCb online system

The LHCb experiment needs an efficient and selective trigger system in order to collect the B-meson decay modes useful to reveal CP violation phenomena. Since the full description of the LHCb online system can be found in [1], only a brief summary shall be given here.

The LHCb trigger is organized in three hierarchical levels. The purpose of the first trigger level, called Level-0, is to reduce the LHC beam crossing rate, of about 40 MHz, to the rate of about 1MHz, at which all the sub-systems can be used for deriving a trigger decision. The Level-0 trigger is fully synchronous and is implemented in custom hardware.

At the 1 MHz output rate of the Level-0 trigger, data goes to the Level-1 trigger, which provides a nominal accept rate of 40 kHz. Upon positive Level-1 decision the complete event is processed by the High Level Trigger (HLT) with a final rate of 200 Hz. The Level-1 and HLT selection algorithms will be implemented on a commodity processor farm of about 2000 CPUs. The aggregate data throughput for both triggers will be around 6-7 GB/s. The technological challenge in the system implementation consists of handling the high data throughput using commercial, and to large extent, commodity equipment.

## 1.2. Ethernet as link layer

Copper Gigabit Ethernet is used throughout the system as the link technology. A large switching network provides the connectivity between the data sources and the processors of the farm. To reduce the size of the central network, multiplexing switches are foreseen before entering the main readout network switch. Data are then pushed through a large high-performance Ethernet switch to a sub-farm controller. For maximum scalability and ease of implementation, in particular of the front-end custom hardware, a connectionless datagram protocol has been chosen [2].

The farm of processors will be partitioned in sub-farms, each one being interfaced to the readout network switch by a “sub-farm controller” (SFC). The number of sub-farms is primarily determined by the aggregate throughput and will be of the order of 100.

Datagram, just as raw packet protocol, is by definition unreliable in the sense that neither success nor failure to receive the data is reported by the destination to the sender. Transport errors, like packet drop in a switch due to internal buffer congestion, or simple bit errors due to electrical noise on the wire, go undetected by the sender. Error rates are not higher than when using so-called reliable protocols like TCP, but they are handled in a different way. Transmission errors on short distance links can however be expected to happen very rarely, and packet loss in a switch depends highly on the buffer space available and on traffic shape. Thus, in favourable conditions (short links, large buffers), datagram transmission can be expected to be highly reliable, and can even help to avoid traffic congestion, since no acknowledgment and retransmission frames use up valuable bandwidth.

## 1.3. Transport protocol overheads

As one motivation for the choice of the transport protocol for the LHCb online system was given by the overheads introduced by each protocol layer, we briefly summarise them here.

At the lowest level[1], Layer 2 in the OSI (Open Systems Interconnect) Reference Model, the overheads include a 14 Bytes long Ethernet header, and the Frame Check Sequence (FCS) of 4 Bytes. Each frame transmission has to start with a preamble of 7 Bytes and a 1 Byte long Start Frame Delimiter. In addition, each frame has to be trailed by an Inter Frame Gap of at least 96 bit cycles, according to the IEEE 802.3 standard [4]. This amounts to at least 38 Bytes overhead in the raw Ethernet protocol. The maximum payload length, called Maximum Transfer Unit (MTU) is defined to be 1500 Bytes, but can be set smaller. It should be mentioned that some Gigabit Ethernet hardware supports so called Jumbo frames, i.e. frames longer than 1500 Bytes (typical MTU values are 9000 and 16000 Bytes).

The next higher protocol, Internet Protocol (IP), also called Layer 3 in OSI, introduces an at least 20 Bytes long header for each transmitted frame. It supports packet lengths of up to 64 kB, but due to the underlying Ethernet protocol restriction, the IP packets, if longer than MTU, have to be fragmented into frames fitting the MTU. Each frame has to have its own Ethernet and IP header.

The connectionless User Datagram Protocol (UDP) sits on top of IP, and introduces additional 8 Bytes overhead.

TCP (Transmission Control Protocol) is a connection-oriented protocol on top if IP, and in terms of overhead adds 20 Bytes in the header of a packet.

For the LHCb online system, in order to minimise protocol overheads and thus enhance the payload link utilisation, it has been chosen to use plain IP protocol. The choice was motivated by the IP being widely used in particular in high-end switches, a standardised fragmentation procedure, and the decision not to use a retransmitting protocol (TCP). UDP would not add any useful information for our purposes to the data. However, for convenience in programming the applications, UDP has been used in some of the tests presented in this report.

## 1.4. Sub-farm Controller

The sub-farm controllers are located at the down-stream end of the readout network. They receive all fragments of a sub-set of events and assemble them in complete event structures. They distribute the events to the compute nodes connected to them via another Gigabit Ethernet switch, collect the trigger decisions, notify the control system and route the retained events to permanent storage.

The challenge in the implementation of the SFC software will come from the fact that it has to receive incoming data at high link load in a reliable manner, while at the same time assemble the fragments into complete events and forward the latter to one of the farm nodes. The total data flow (in and outgoing) inside the SFC related to the event building process alone will be ~120 MB/s, but will peak for short times at ~250MB/s. In addition, events accepted by the HLT will pass through the SFC on their way to permanent storage. The system has thus to be capable of processing large amounts of data without affecting its capability to receive event fragments at any time.

[1] We skip the discussion of the "physical layer" here, which would correspond to Layer 1 in OSI, as it describes the physical medium interface, and cannot be viewed as a source of overheads in the strict sense.

Several issues arise in this context:

- CPU power needed to build the event in real time: The SFC cannot accumulate incoming traffic for 'later processing'; the system must be capable of merging the event fragments under sustained high data flow.

- System HW layout: The motherboard must guarantee enough throughput on the PCI bus connecting the CPUs and memory to the Network Interface Card.

- Network Interface HW: Gigabit network controllers vary in buffer space and interrupt coalescence features.

- Interrupt rate: Incoming fragment rate will be ~80 kHz. The network interface hardware, by default issues an interrupt to the CPU with each incoming or outgoing packet. The resulting interrupt rate is very high for a custom processor, resulting in poor performance of the system. Modern Ethernet controllers implement interrupt coalescence in order to increase performance.

- Operating System: Packet loss may result from too slow response of the system to the incoming traffic, which may occur in particular when the CPU is under heavy load. Guaranteed latency falls into the domain of real-time operating systems[1]. Although Linux does not belong to this category, the latest kernel versions include features such as low latency scheduling and kernel preemptibility which minimise the response time to external events.

[1] To be clear, it should be noted here that 'real-time' doesn't mean 'fast', but 'predictable'. Slow hardware cannot be compensated with real-time features.

# 2. Characteristics of the system and data fluxes to the SFC

In this section we describe the traffic pattern at the input to the SFC as can be expected from the current base line solution to the LHCb online system described in [1].

At the output of the switch, both Level-1 and HLT traffic are mixed; all data is transferred on one link. The two streams will have different characteristics when arriving at the SFC:

- L1 data will be received in rather short bursts of ~130 frames, each of size close to MTU (1500 Bytes). The time between two bursts is given by the number of sub-farms. A possible scenario is to distribute Level-1 events in round-robin mode among the sub-farms. With e.g. 100 sub-farms, 130 sources and a mean data payload of 1100 Bytes, the link would be loaded at 100% for about 1.2 ms, with a period of ~1.3 ms without transmission, on average.

- HLT data will be distributed according to a load balancing algorithm, and is much less predictable. Its common feature shared with the Level-1 traffic is that all modules will start transmitting the event fragments simultaneously to the same destination. The number of sources will be ~320. Due to the significantly larger event size, the time to receive a full event is longer, but the lower Level-1 accept rate makes the period between the HLT burst much larger.

The combined characteristic of the input stream is currently being evaluated in simulation, and will be published in a separate note [3]. First estimates indicate an average incoming throughput of ~60 MB/s, with bursts of several ms at 100% link load (125MB/s).

Each SFC will be equipped with at least two Gigabit Ethernet NICs: one for the incoming data stream, the second to connect it to the sub-farm nodes. The trigger decisions (L1 and HLT) coming back from farm nodes as well as complete accepted events destined for permanent storage are expected to contribute at percent level to the total network traffic through the SFC.

# 3. HW issues

In order to choose the proper platform for the implementation of the SFCs as well as farm nodes, different hardware issues have to be addressed. Today's commodity PCs typically are implemented using motherboards based on the PCI or PCI-X bus, with a CPU from Intel or AMD. A typical layout of a PC is shown in Figure 1. The processor bus (often called Front Side Bus, FSB) is coupled to the system bus (e.g. PCI) via the so-called Northbridge. A Southbridge connects the system bus to peripheral busses such as ISA and devices like the serial and parallel port drivers. Chips connected to the system bus constitute what is commonly referred to as the chipset. Due to restrictions on the physical length of the PCI bus, and thus on the number of connected devices, PCI-to-PCI bridges are used to interconnect PCI devices in a structured manner.

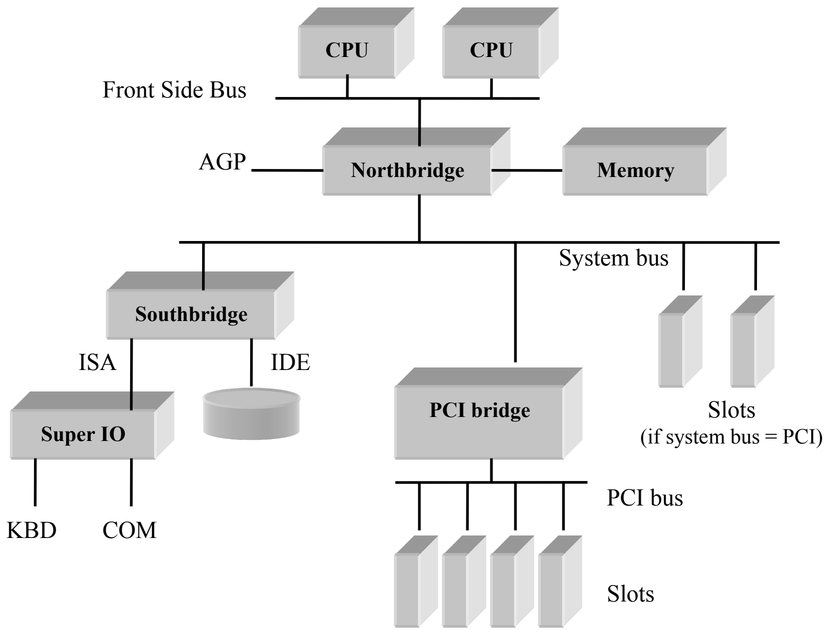

Figure 1: A typical layout of a PCI based computer system.

Many modern high end systems deviate from the layout as depicted in Figure 1, in the sense that the system bus is replaced by separate I/O interconnects providing bandwidth according to the needs of the connection. In such cases, the FSB is often referred to as the System Bus. Details of the systems studied for this note will be given in Section 4.1.

## 3.1. Hyper-Threading and HyperTransport

For our investigation we have selected two competing technologies: IA32 (Intel Xeon) with Hyper-Threading (HT) [7], and AMD64 (AMD Opteron) with HyperTransport$^{TM}$ [9] technology.

Intel has introduced the Hyper-Threading technology to address the question of unused CPU cycles, typically arising when the executing thread is forced to wait for data, e.g. as a result of a page fault. The physical CPU shares its resources among two logical processing units, which can increase the overall CPU use by up to 25% depending on the application [8]. The Linux 2.4 kernel is HT aware, and uses the logical CPUs in the same way as if running on a true Symmetric Multi-Processing (SMP) machine.

AMD, on the other hand, increases the performance of its CPUs by integrating a Northbridge logic into the processors, which interfaces the CPU core directly to the memory controller and the HyperTransport$^{TM}$ technology interface[1]. It has first appeared with the Opteron processor, but has been also implemented in the Athlon 64. Since the Northbridge acts as the interface between the processor and the system bus, and in particular the system memory, integrating this device on the same die as the CPU aims at improving data throughput to and from the CPU.

## 3.2. Chipsets, PCI(-X) bus, motherboard layout

The motherboard used in a SFC has to guarantee enough bandwidth between the NICs, memory and the CPUs. The standard bus technology to be found in commodity PCs is PCI [2]. It comes in 4 variants: 32bit/33MHz, 32bit/66MHz, 64bit/33MHz and 64bit/66MHz, with the corresponding bandwidth of 132 up to 528 MB/s. High-end PCs use an extension of PCI, called PCI-X [6], which increases the clock rate to 100 or 133MHz, resulting in maximum throughput exceeding 1GB/s (for 64bit/133MHz).

Clearly, a 32bit/33MHz bus poses a serious bottleneck for a gigabit Ethernet interface. Experience has shown that at least 64bit/66MHz bus has to be used per network interface to guarantee loss free transmission at gigabit speed.

According to the standard specifications, a PCI device designed for operating at clock rate R has to be capable of operating in the range [0..R]. This allows to connect devices with different nominal clock speeds on the same bus, with the slowest device dictating the bus clock rate. PCI-to-PCI bridges can be used to separate slow and fast devices, allowing them to operate at optimal speed.

Since all data movements go through the system bus, 1 GB/s is quickly reached, in particular in multi-processor systems. In high end PCs, therefore, other solutions than PCI are found, the system bus being basically defined by the chipset used, while PCI(-X) remains as the de facto standard peripheral interconnect.

## 3.3. Interrupt coalescence

Two different Gigabit Ethernet controllers were used in our tests: the Broadcom 570x and Intel 8254x. Both of them have interrupt coalescence features implemented by means of parameters passed to the corresponding software driver. The parameters have to be chosen according to the expected traffic shape – too high interrupt coalescence (i.e. too low rate) will result in buffer overflow, while a too low coalescence setting (i.e. too high rate) will unnecessarily strain the system and lead to data losses due to missing CPU power available for data processing.

[1] The HyperTransport$^{TM}$ technology is an advanced high-performance point-to-point link for integrated circuits, its specifications managed by the HyperTransport Technology Consortium [9].

### 3.3.1. Broadcom 570x

The parameters determining interrupt coalescence of the 570x are:

- Number of ticks (of 1µs) the network interface waits after frame reception (transmission), before it issues an interrupt to the CPU. If other frames are received (transmitted) in this time span, no additional interrupts are generated.

- Maximum number of frames, after which an interrupt is generated. This parameter, if set properly, assures that a burst of short frames does not lead to overrun in receive or transmit descriptors.

- Adaptive coalescence (on/off). If this parameter is set to on, the hardware issues interrupts in a dynamic way, depending on the current traffic shape. This parameter is OFF by default, if NAPI (described in Sec. 5.1.3) is used, i.e. in kernel 2.4.20 and above.

The first two parameters can be set independently for receipt and transmit interrupts.

### 3.3.2. Intel 8254x

The Intel Gigabit Ethernet Controllers implement three ways of moderating the interrupt rate [13]:

- An absolute timer, which starts upon reception (transmission) of the first packet. An interrupt is issued when the timer has expired. Its purpose is to assure interrupt moderation in heavy traffic conditions. The timer is set in steps of 1 µs.

- A packet timer, which starts upon reception (transmission) of every packet. An interrupt is generated only when this timer expires. Since it is reset at every packet received (transmitted), it is not guaranteed to generate an interrupt if the network traffic is high. Its purpose is rather to lower the latency in low traffic situations. The timer is set in steps of 1 µs.

- An interrupt throttle mechanism can be used to set an upper boundary on the interrupt rate generated by the controller. The corresponding parameter is the maximum interrupt rate.

While the first two timers work independently for reception and transmission of frames, the throttle mechanism can be used to ensure that despite quickly varying traffic conditions in both directions, the total interrupt rate does not increase above the given value.

# 4. The test bed setups

For the purposes of our measurements, we have set up two test beds[1], shown schematically in Figure 2. The Bologna group is using Intel Xeon based PCs for traffic generation as well as reception. The PCs are connected via 3 Gigabit Ethernet connections to the switch (HP Procurve 6108, 8 ports), which also provides the uplink to the campus network.

In the CERN setup, Intel Xeon and AMD Opteron based PCs are used. Network processors hosted on PCI cards serve as data sources. 11 such modules are used to generate data, which are fed through the switch (3com 4924, 24 ports) to the PCs. An additional network processor is used to receive data from the SFCs, and can act as a sub-farm emulator.

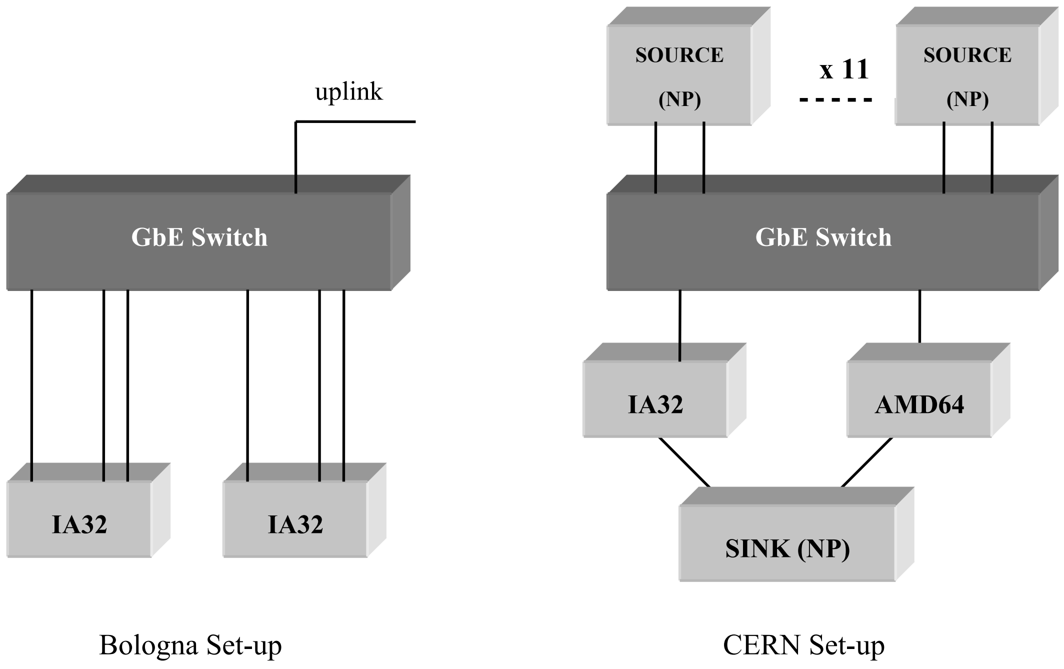

Figure 2: Schematics of the two test bed setups used in our measurements: Bologna setup on the left, CERN setup on the right. For clarity only Gigabit Ethernet connections are shown.

Both setups are equivalent for the purpose of testing the functionality of a SFC candidate. The CERN setup enables also to perform stress tests of the switch, thanks to the possibility of feeding data at high link utilisation on all ports.

[1] As one test bed is operated by the LHCb-Bologna group and the other by the CERN LHCb-DAQ group, for simplicity we will refer to them as the Bologna and CERN setups, respectively.

## 4.1. SFC Hardware

We describe in this chapter the hardware used in the test bed activities. A summary of the main parameters is given in Sec. 4.3.

### 4.1.1. Intel Xeon based SFCs

For the tests we have considered two motherboards using different chipsets. The Bologna SFC motherboard uses the Intel 7501 chipset. The block diagram with the main components is shown in Figure 3. The main features of the PC used in the tests are the 2.4 GHz CPU clock rate, 400 MHz front side bus (3.1 GB/s, unidirectional), 3.1 GB/s data throughput on the memory bus (unidirectional), 1 GB/s interface between the Northbridge and the PCI-X hub.

Although the motherboard has an integrated Gigabit Ethernet controller (Intel 82545EM), the tests were performed using the Intel 82546EB dual Gigabit Ethernet server class controller on a PCI-X card.

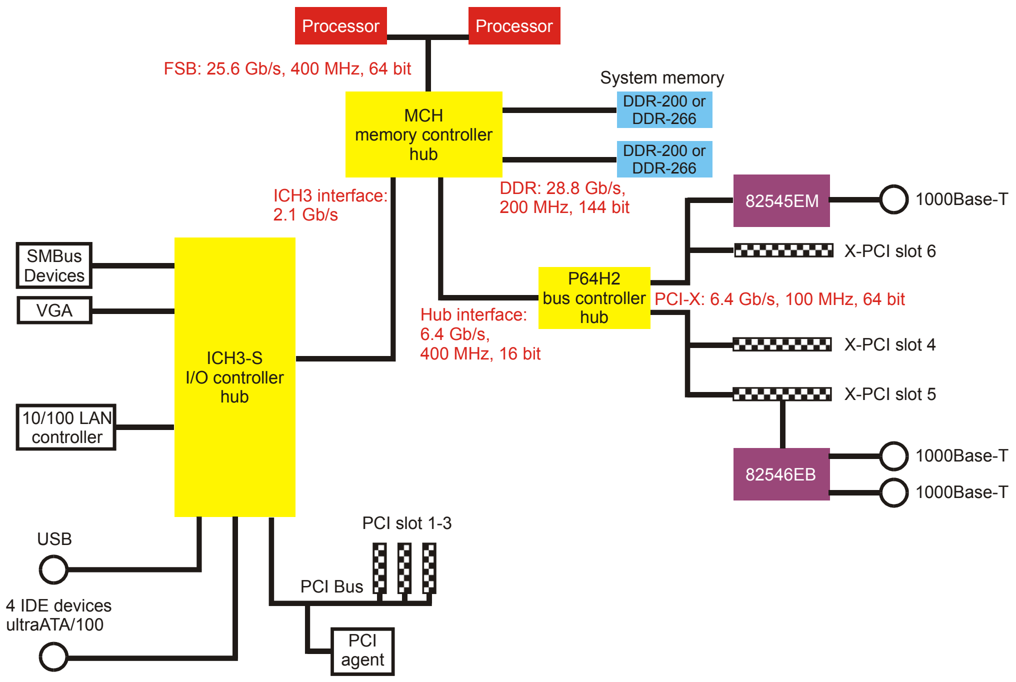

Figure 3: Schematic layout of the motherboard used in the SFC candidate in Bologna.

The SFC candidate used in the setup at CERN features a SuperMicro P4DL6 motherboard with the ServerWorks Grand Champion LE chipset. The block diagram is shown in Figure 4. The specs are very similar to the ones of the Intel chipset used in the Bologna SFCs. The major difference is the higher data bandwidth on the bus between the Northbridge and the PCI-X bridge, where the ServerWorks chipset offers 3.1 GB/s.

Also here the Intel 82546EB dual Gigabit Ethernet controller is used.

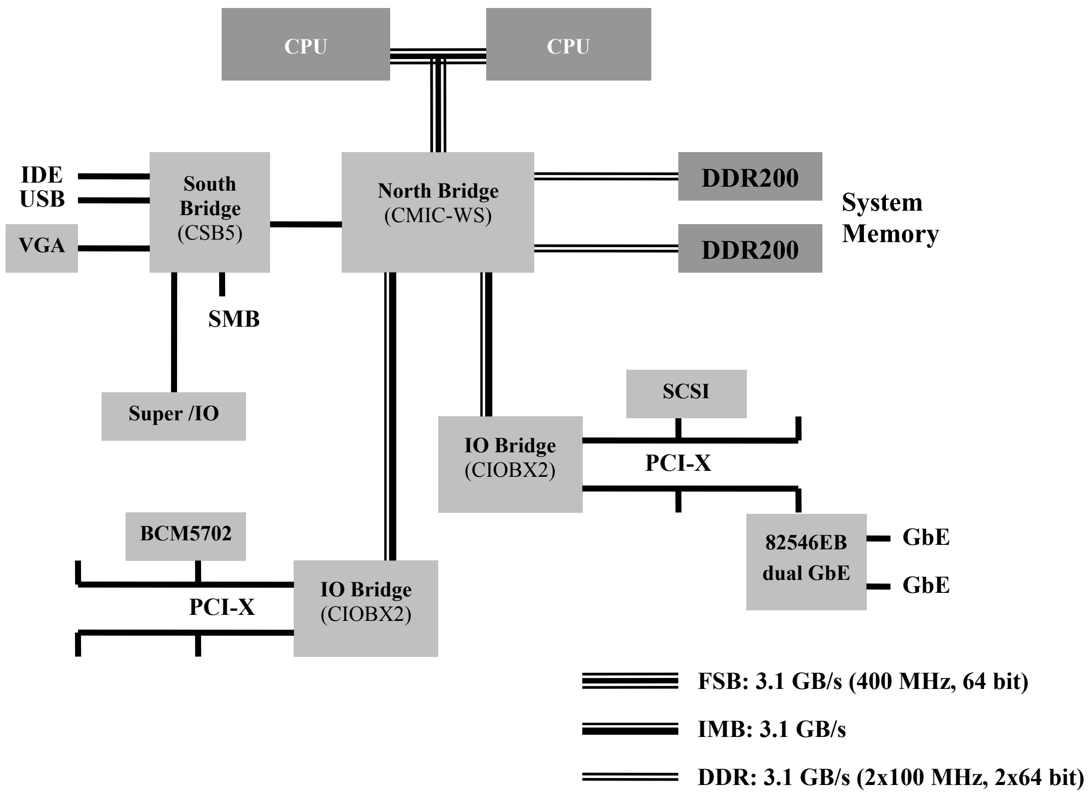

Figure 4: Schematic layout of the Intel Xeon based system, with ServerWorks GC-LE chipset used in the tests at CERN.

### 4.1.2. AMD Opteron based SFC

The other SFC candidate studied at CERN is based on an AMD Opteron processor and makes use of a completely different bus architecture. As mentioned in Sec. 3.1, AMD has opted for the HyperTransport as system wide bus technology. It is used to interconnect the CPUs as well as other system components. HyperTransport is a tunnelling technology, the AMD 8131 PCI tunnel connects to only one CPU directly, data to/from the other CPUs (if present) is routed through a cross-bar inside the first one. Each CPU can have its own memory, as indicated in Figure 5, while the tunnelling concept allows to access any memory block from any CPU or DMA capable device. This is referred to as Non Unified Memory Access (NUMA).

It's worth mentioning that the HyperTransport link offers 6.4 GB/s total bandwidth (bidirectional, i.e. 3.2 GB/s in each direction simultaneously).

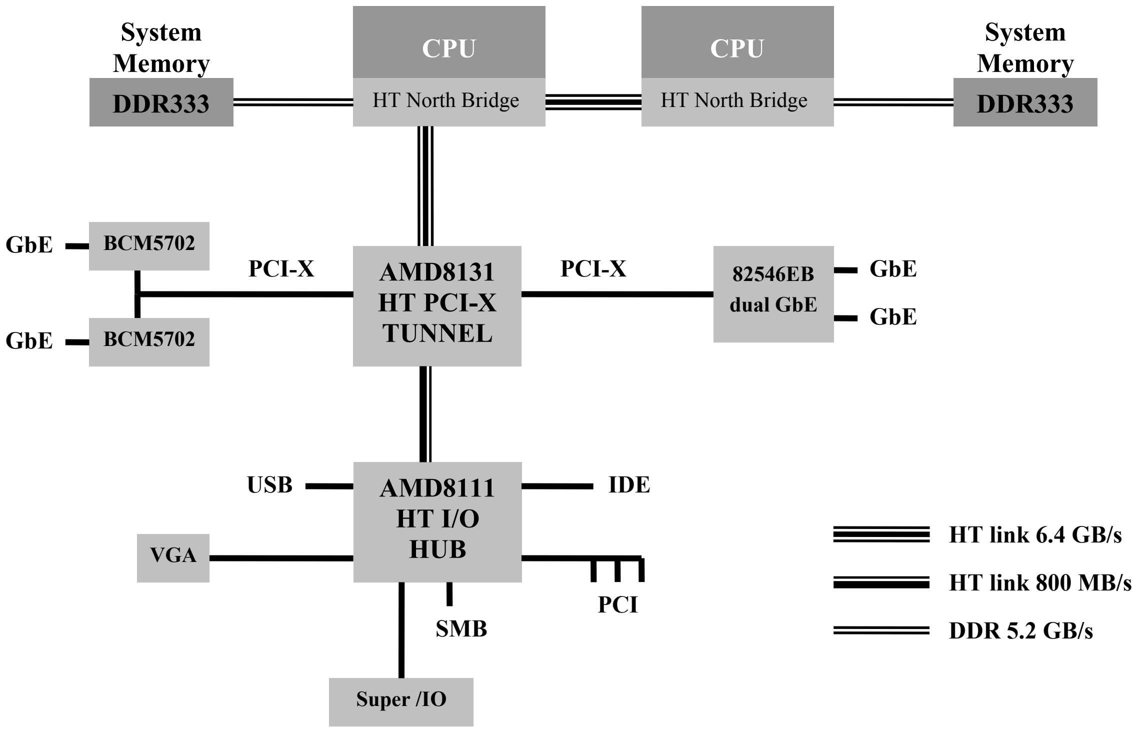

Figure 5: Schematic layout of the AMD Opteron based system used in the tests at CERN.

## 4.2. Data source modules (CERN setup)

An IBM NP4GS3 [15] Network Processor (NP) is used as data source, capable of generating data streams at up to line speed. The NP is embedded in a PCI card developed by the S3 company [16]. Three out of four Gigabit Ethernet ports are available for external connection, while the fourth is wrapped to the PCI connector via a Broadcom controller. Since the NP4GS3 is certified for OC48 (2.48 Gb/s), two ports can be used to generate data streams at up to 100% link utilisation each. In our setup the third external port remained unused.

The Network Processor is a very flexible device, and highly suitable for our purposes. Data patterns are programmable, any protocol can be implemented. For the tests described here, raw Ethernet packets, as well as IP packets were generated. In case of IP, the payload was formed as event structures, defined by the raw data transport format of LHCb [2]. Internal counters have been used to gain frame rate, throughput and error statistics in all tests.

In addition, the GP-IO pin available on the NP board was used for synchronisation between the data sources at frame level. Frames can be synchronised to within ~100ns, of ~12 Bytes at Gigabit speed. This feature is used to emulate the data flow from the L1 boards, i.e. is of importance for the evaluation of aggregation switches.

More details on the NP based data generators used in the tests can be found in **Error! Reference source not found.**.

## 4.3. Summary

A summary of main parameters of the components used in the test bed setups is given in Table 1.

| | **BOLOGNA, IA32** | **CERN, IA32** | **CERN, AMD64** |
|---|---|---|---|
| **CPU** | Intel Xeon (dual) | Intel Xeon (dual) | Opteron (dual) |
| **L2 Cache** | 512 kB | 512 kB | 1 MB |
| **CPU clock rate** | 2.4 GHz | 2.4 GHz | 1.4 GHz |
| **Motherboard** | SuperMicro X5DPL-iGM | SuperMicro P4DL6 | RioWorks HDAMA |
| **Chipset** | Intel E7501 | ServerWorks GC-LE | AMD 8131/8111 |
| **OS** | RedHat 9 + 2.6.0 kernel | RedHat 9 + 2.6.0 kernel | SuSE 8 + 2.6.0 kernel |
| **Switch** | HP Procurve 6108, 8 GbE ports | 3com 4924, 24 GbE ports | |
| **Traffic generators** | PC | Network Processor PCI card | |

Table 1: Parameters of the test bed components

# 5. The Linux Kernels

In the first part of this chapter we will give a detailed description of the evolution of the network data handling in the Linux kernel between versions 2.2 and 2.6. The last part summarises the new features of the latest kernel release.

## 5.1. Network core processing

To understand the improvements in network processing, we summarize how the implementation has changed in successive Linux kernel versions. Much effort has been put in preventing congestion collapse condition, in particular the interrupt livelock [14]: all CPU resources are spent to serve the interrupts and to enqueue packets, so no resources are available to dequeue and process them; all system resources are therefore used and no work is performed.

### 5.1.1. Network core processing in kernel 2.2

Whenever the Network Interface Card (NIC) receives a packet, it generates an interrupt (unless an interrupt throttle mechanism is active on the NIC). Upon reception of an interrupt, the kernel executes the interrupt handler which services the hardware. During execution of the interrupt handler other interrupts are disabled, so if other packets arrive in the meantime, they are dropped. For this reason the interrupt handler needs to complete its task as soon as possible. It usually schedules DMA data transfer from the NIC to kernel space, but it cannot perform lengthy checks or IP processing, since the system is potentially loosing packets while interrupts are disabled.

Interrupt handlers are therefore divided in two parts: the "top half" (TH) is executed immediately, inside the interrupt context (in_irq() = true), with the other interrupts disabled; the "bottom half" (BH) is deferred to a later time, outside the interrupt context (in_irq() = false), with interrupts enabled again.

The TH disables interrupts, allocates a new sk_buff structure in kernel memory, fetches data from the card buffer to the allocated sk_buff (using DMA), and (netif_rx()) enqueues the packet descriptor in a backlog queue, then calls the mark_bh() system call to defer the execution of the BH at the next available opportunity with interrupts enabled, and finally enables interrupts again and terminates.

BHs that are waiting will be executed only when one of the following events occurs: the kernel finishes handling an exception, the kernel finishes handling a system call, the kernel finishes handling a hardware interrupt or the kernel executes the schedule() function to select a new process. The BH's mechanism has been implemented since Linux kernel 1.x. Kernel 2.2 has 18 BH handlers (networking, keyboard, console, SCSI and serial all used bottom halves directly). Its main limit consists in the strict serialization of the task execution among CPUs in a SMP machine (two BHs do not run at the same time); only one packet at a time can enter the system.

Network BH dequeues packet descriptors from backlog queue and performs further processing (e.g. IP processing). If the backlog queue (whose default length of 300 descriptors can be changed through /proc/sys/net/core/netdev_max_backlog) fills up completely, it enters the throttle state in which no more packets can be enqueued (packets are silently dropped). Backlog queue exits the throttle state only when it becomes completely empty, to prevent service disruption due to kernel overload (congestion collapse condition) and Denial of Service (DoS) attacks.

Dropped packet counters for the backlog queue are located in /proc/net/dev_stat: the first field contains the dropped packet counter, the second field contains the number of times the backlog entered the throttle state.

### 5.1.2. Network core processing in kernel 2.4.0 to 2.4.19: softnet

Since Linux kernel 2.3.43 the BH mechanism has been replaced by the softirq and tasklet mechanism [19]. Softirqs and tasklets are kernel software interrupts, i.e. pieces of code that can be executed on-demand by a kernel thread (ksoftirqd_CPU#) without strict response-time guarantees.

Softirqs are SMP versions of BHs: they can be executed on as many CPUs at once as required (this means that they need to deal with race conditions in shared data using their own locks). A bit mask is used to keep track of which softirqs are enabled, so at most 32 softirqs are available. Tasklets are like softirqs, except that they are dynamically registrable (you can have as many tasklets as you want), and it is also guaranteed that any tasklet will only run on one CPU at a time, although different tasklets can run simultaneously (unlike different BHs).

The network interrupt handler disables interrupts, allocates a new sk_buff structure in kernel memory, fetches data from the card buffer to the allocated sk_buff (using DMA), calls netif_rx() and finally enables interrupts again and terminates (see Figure 6).

The netif_rx() function enqueues the packet descriptor into the backlog queue for the current CPU (__skb_queue_tail()) and marks the NET_RX softirq through the __cpu_raise_softirq(cpuid, NET_IF_SOFTIRQ) system call, to schedule the execution of the remaining processing, at the next available opportunity, with interrupts enabled.

Five possible congestion levels describe the status of backlog queues: NET_RX_SUCCESS (no congestion), NET_RX_CN_LOW (low congestion), NET_RX_CN_MOD (moderate congestion), NET_RX_CN_HIGH (high congestion), NET_RX_DROP (critical congestion, packet dropped). When the critical congestion level is reached, netif_rx() starts a throttling policy to push back the queue into a non-congested status to prevent service disruption due to kernel overload and DoS attacks.

Softirqs that are waiting will be executed only when one of the following events occurs: the kernel finishes handling an exception, an application-level process invokes a system call, the kernel finishes handling a hardware interrupt or the kernel executes the schedule() function to select a new process.

The do_softirq() function checks a bit mask: if the bit corresponding to a given softirq is set it calls the corresponding handler routine. If the bit mask contains NET_RX_SOFTIRQ, the handler net_rx_action() is called. The net_rx_action() function dequeues the first sk_buff from the current CPU's backlog queue and calls the relevant processing functions (e.g. ip_rcv() for IP packets). If the backlog queue contains more than one packet, net_rx_action() loops over the packets, until either a maximum number of packets has been processed (/proc/sys/net/core/netdev_max_backlog) or the maximum time interval has been spent (1 jiffy, usually 10 ms).

If net_rx_action() exits the loop leaving a non-empty queue, the time squeeze counter is increased and the NET_RX_SOFTIRQ is enabled again to allow for processing to be resumed at a later time.

The counters of packets dropped from backlog queues are located in /proc/net/softnet_stat. In the table, each line corresponds to a CPU-specific backlog queue: the second column contains dropped packets counters, the third column contains time squeeze counters (the number of times net_rx_action() breaks the loop leaving a non-empty queue and reschedules itself for later execution), the fourth column contains the number of times the backlog entered the throttle state.

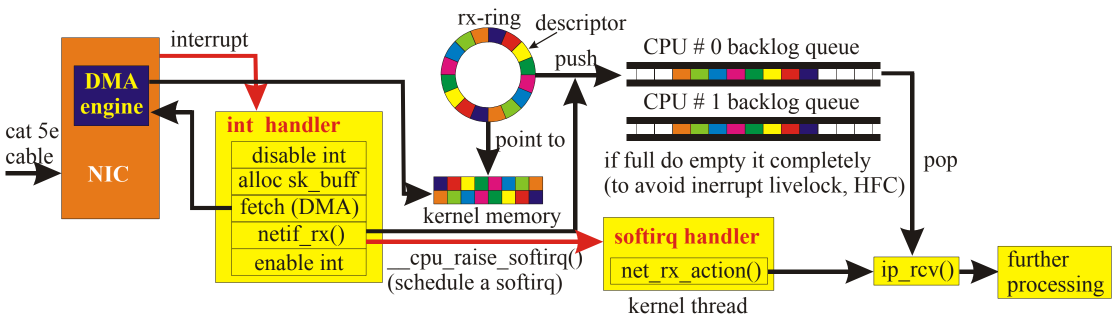

Figure 6: Schematic description of the softnet mechanism (kernel ≤ 2.4.19).

## 5.1.3. Network core processing in kernel 2.4.20: NAPI

NAPI stands for New Application Program Interface. It is an interrupt moderation technology based on a mixture of interrupt and polling mechanisms [20], [21]. It has been introduced in kernel 2.5.3 and then back-ported to kernel 2.4.20.

Until kernel 2.4.19 (softnet), packet extraction from the backlog queue was driven by a softirq scheduled by a hardware interrupt generated by the NIC. Since kernel 2.4.20 (NAPI) the backlog queue has disappeared and packet descriptors are left in the rx_ring. The interrupt handler simply registers the interface into a kernel poll_list (see Figure 7). Then, in the softirq handler, devices registered in the poll_list are polled to get packets.

Interrupt mechanisms improve latency under low load but make the system vulnerable to congestion collapse if the load exceeds the Maximum Loss Free Forwarding Rate (MLFFR). Conversely, polling mechanisms introduce more latency under light load and abuses the CPU by polling devices that have no packet to offer, but prevents congestion collapse under heavy load.

When the first packet in a batch reaches the NIC, an interrupt is generated. First of all, the network interrupt handler disables interrupts that can be generated by new packets arriving (rxint) or by a packet arriving and finding no DMA buffers available (rxnobuff). Any new packet arriving when the DMA ring is filled will be dropped without disturbing the system. The network interrupt handler then allocates a new sk_buff structure in kernel memory, fetches data from the card buffer to the allocated sk_buff (using DMA), calls netif_rx_schedule() and terminates (see Figure 7). The netif_rx_schedule() function, in turn, puts a reference to the device in the poll_list attached to the interrupted CPU and schedules a softirq. The net_rx_action() function, called by the softirq handler, polls all devices registered in the poll_list to get packets out from the rx_ring. All interfaces are given an opportunity to send up to a configurable number of packets (quota). When the device has no more packets to offer, it is taken off the poll_list and allowed to interrupt again. If the quota is exceeded and a device has still packets to offer, the device is put at the end of the poll_list.

Under low load, before the MLFFR is reached, the system converges toward an interrupt driven system: the packets/interrupts ratio is lower and latency is reduced. Under heavy load, the system takes its time to poll registered devices. Interrupts are allowed as fast as the system can process them: the packets/interrupts ratio is larger and latency is increased.

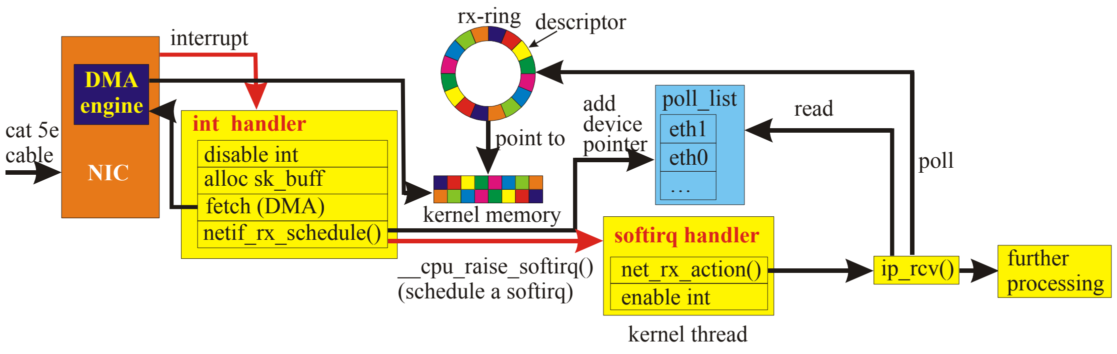

Figure 7: Schematic description of the NAPI mechanism (kernel ≥ 2.4.20).

### 5.1.4. NAPI back-compatibility

As NAPI has changed the driver-to-kernel interface, its introduction should require all network drivers to be rewritten. In order to accommodate old device drivers (NAPI-less) allowing for a gradual migration, the old interface (backlog queue) is still available (back-compatibility). Backlog queues, when used in back-compatibility mode, are polled just like other devices (see Figure 8).

It should be emphasised, however, that even a NAPI-aware kernel (≥ 2.4.20), when used with old (NAPI-less) device drivers, works as previous kernels, using backlog queues.

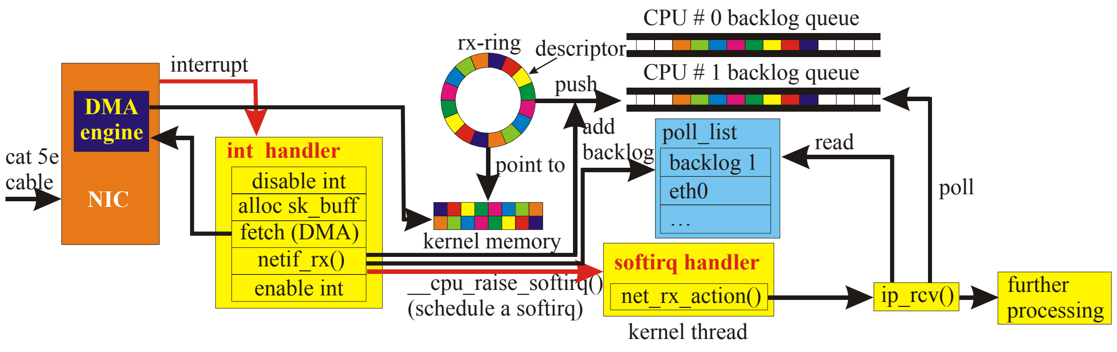

Figure 8: Schematic description of NAPI back-compatibility mode (kernel ≥ 2.4.20, NAPI-less driver).

## 5.2. New kernel 2.6 features

Several new features were introduced in kernel version 2.5 (the development version leading to the stable 2.6 release) in order to lower the kernel latency, some of which have been back-ported to version 2.4.20. The main changes relevant to the event building process involve NAPI as the default network API, kernel preemptibility, O(1) scheduler, increased internal clock granularity and improved CPU affinity. A summary of all new features of the new kernel can be found in the “post-Halloween” document [23].

### Low latency, kernel preemption

Up to version 2.4 of the Linux kernel, a process entering Kernel Mode could not be preempted. A process handling an interrupt or exception might have spent milliseconds running in Kernel Mode, thus preventing other time-critical processes from running. The new kernel is fully preemptible, resulting in lower latencies.

### The O(1) scheduler and CPU affinity

The task of the scheduler is to decide which process is the next one to execute on a given CPU. O(1) means in this context that the time for decision taking is independent of the number of processes in the run queue. The new scheduler does also distinguish between logical CPUs (Hyper-Threading) and true SMP, and distributes the load among physically different processors. The CPU affinity has been improved – a process will only be migrated from one CPU to another in order to resolve imbalances in run queue length [26].

### Change in kernel internal clock frequency

Linux uses clocks for two main purposes: (1) keeping the current time up to date; (2) maintaining mechanisms (timer) to notify the kernel or a user program that a certain time interval has expired. Timers are used for time-sharing CPU among runable processes: each process is given a *quantum* of time: if the process is not completed within the quantum, the schedule() function selects the new process to run. The Linux kernel makes use of four different clocks on the IA-32 (80x86) platform:

1. RTC: The Real Time Clock (RTC) sends periodically (the frequency of its own oscillator ranges from 2 to 8192 Hz) a global interrupt (IRQ8); it is the only clock still active when the computer is switched off (the power being supplied by a small battery).

2. TSC: The 64-bit Time Stamp Counter (TSC) is implemented on the CPU itself (starting from Pentium) and is increased at every CPU clock tick (the reference clock signal is sent to the CPU CLK pin from an external oscillator).

3. PIT: The 16-bit Programmable Interval Timer (PIT) sends periodically a global interrupt (IRQ0) at a frequency stored in the HZ macro based on its own internal oscillator.

4. APIC: The 32-bit local Advanced Programmable Interrupt Controller (local APIC), present in recent Intel CPUs, sends periodically (at a frequency stored in the HZ macro) a local interrupt (LINT0) limited to the CPU itself (in multiprocessor architecture), based on the APIC bus clock signal. TSC, which is more accurate than PIT is used to correct PIT in system calls related to timing measurements.

On a single processor PC, all time-keeping activities are handled by the PIT. In a SMP system, general activities, like software timers, are handled by the PIT, while CPU-specific activities, like determination of running time of the currently running process, are handled by the local APIC.

Since CPU time-shared activities are handled by the PIT (in single processor systems) or local APIC (in SMP systems), a change in HZ macro modifies scheduler performance: HZ is basically the granularity of the system. In general, shorter ticks produce higher resolution timers and therefore better performance of I/O multiplexing (polling) and improved system latency in process preemption, but introduce more timer overhead (more frequent timer interrupts) and more context switches between processes.

Since kernel version 2.5.25, the internal clock frequency on Intel IA-32 platform has been increased from HZ = 100 to HZ = 1000 (i.e. granularity has been increased from 10 ms to 1 ms). On other platforms, the internal clock (HZ) varies between 100 (e.g. M68K) and 1024 (IA-64).

# 6. Measurements

In this chapter we report on the measurements performed. Sec. 6.1 shows the measurement of the transmission error rate on copper link, Sec. 6.2 presents the protocol reliability test. In Sec. 6.3 we present the performance evaluation of the SFC candidates in the context of data transport, and Sec. 6.4 shows the performance of two Gigabit Ethernet switches in terms of data loss under heavy traffic conditions.

## 6.1. Transmission error rate

The reliability of copper link as physical medium has been tested in a simple setup involving only a network processor based frame generator. A 100 m long category 5e cable was used to interconnect two ports of the frame generator, and placed close to electrically noisy devices.

Two kinds of error conditions have to be distinguished: transmission errors and equipment malfunctioning. The first one can occur during normal operation, e.g. due to noise pickup on a long copper wire. The rate at which these transmission errors happen should be limited in order for the data acquisition system to function as desired. The second one is typically a result of a breakdown of one of the components in the data path, and cannot be estimated in the design phase of the experiment. A careful evaluation before the purchase of the equipment is needed in order to minimise the Mean Time Between Failure (MTBF). We will refer to the first error source simply as errors, and as faults to the second one. In this note we are interested only in the error rate.

Assuming a correctly formed frame at time of sending, the only two errors of interest are receive and checksum errors. The checksum (or CRC) error indicates that at least one bit flip occurred during frame's transmission, and was detected by the Media Access Control (MAC) device. A receive error on the other hand is signalled by the physical layer device (PHY) if it detects an error condition. The exact meaning of the receive error is specific to the used PHY chip. The error condition is not bound to the data carried by a frame, but can also happen between frames, i.e. in idle state of the link. Experience has shown that usually the two error conditions are correlated, noise induced on the wire can result in a PHY detected receive error, and would corrupt a frame being currently transmitted, thus resulting in a checksum error. Detailed description of the MAC and PHY layers can be found in [4].

In a run of $2\times10^{11}$ frames of 1518 Bytes each, at 100% link load, no transmission errors were detected. All frames were correctly received. This number is equivalent to $2.4\times10^{15}$ bits transmitted. Extrapolating these numbers to the full size of the readout network as described in [1], we can expect less than one transmission error in ~12 hours of operation of the experiment.

In another test run, 4 receive errors were detected within $3.6\times10^{11}$ frames transmitted (of 1436 bytes each). No frame loss was registered. During this run mechanical work (network installations) was carried out around the test setup. The cable used in the test was displaced. Since all the 4 errors occurred within a time span of just the two days where the installations were done, while the full measurement needed more than 2 weeks of run time, it is reasonable to assume that the errors were indeed induced by mechanical rather than electrical noise. This is equivalent to 1 transmission error in ~4.5 hours of operation of the full size system. Even if we believe that this number is too pessimistic, it nevertheless represents an acceptable error rate.

## 6.2. Protocol reliability, packet reception

The operating system used for these tests was Linux Red Hat 9A with two versions of the kernel: 2.4.20-18.9smp and 2.6.0-test11. All daemons (X11 server, crond, atd, sendmail, etc.) were stopped, except the server of the testing software (p2pserver). Flow control was activated on both NICs and on the switch. Datagrams used for these tests had an IP payload of 4096 Bytes (three Ethernet frames).

### 6.2.1. First results (Linux RedHat 9A, kernel 2.4.20, default setup).

First benchmark results were obtained with standard Red Hat 9A setup. Only socket send buffer size and socket receive buffer size were increased from default 131072 B up to 524288 B.

The Intel e1000 network interface driver version was 5.0.43-k1 (as supplied by the Red Hat 9A distribution), the number of descriptors allocated by both the driver tx_ring and rx_ring was 256, the pfifo_fast qdisc queue length was 100.

Results of benchmarks performed this way showed a rather high throughput of 999.9 Mb/s but with the non-negligible datagram loss rate of $5.1\times10^{-5}$ (one datagram lost every 19500 datagrams sent). This is too large for the LHCb L1 trigger needs.

Results of repeated benchmarks showed big fluctuations (see Figure 9), and the distribution of the results was multi-modal. Once magnified, the main peak showed a 6-peak sub-structure (see Figure 10) that shows that certain data loss rates are much more frequent than others.

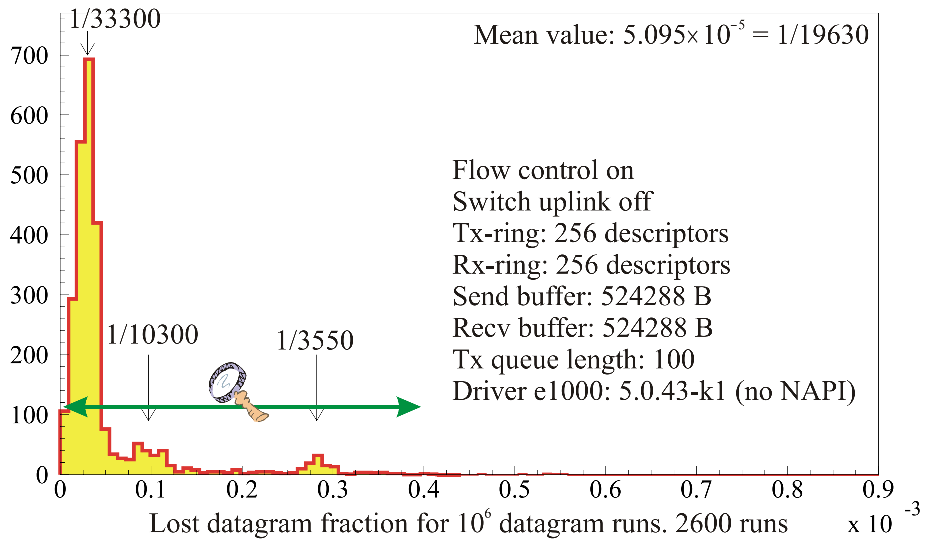

Figure 9: Histogram of repeated measurements of datagram loss rate. The region magnified in Figure 10 is also indicated.

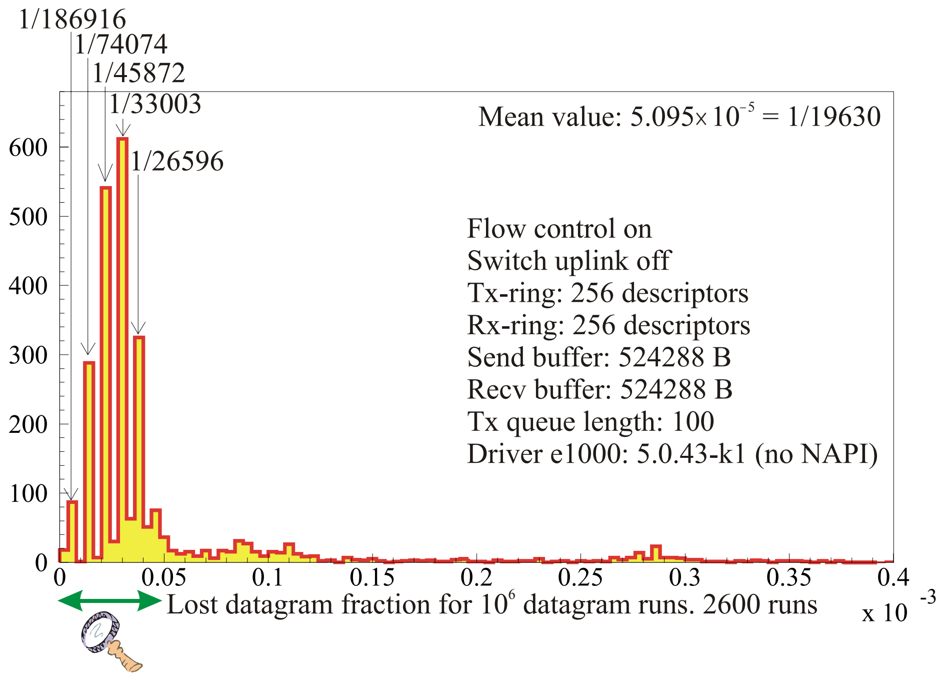

Figure 10: Histogram of repeated measurements of datagram loss rate (magnified). The region magnified in Figure 11 is also indicated.

Magnifying again the histogram, each of the 6 sub-structures showed a finer structure (see Figure 11). These peak structures on datagram loss rates distribution are likely due to throttling policies (see below): to avoid service disruption due to kernel overload and DoS attacks, as queues enter a congested state, they start a throttling policy in which the empty queue state is restored by dropping a group of packets.

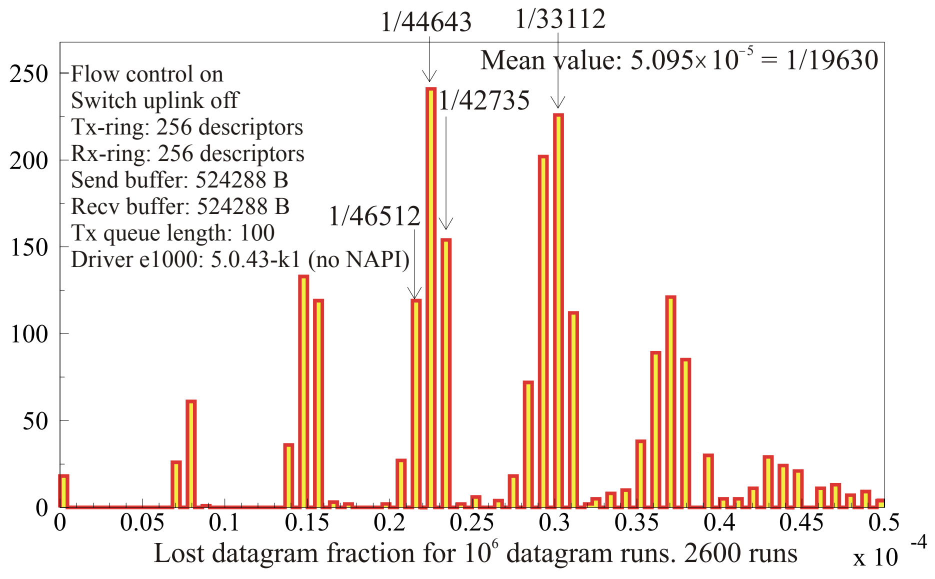

Figure 11: Histogram of repeated measurements of datagram loss rate (magnified).

Further tests were performed with special tunings of the queue parameters. Performance improvement was noticed, but not enough for the LHCb L1 trigger needs. The big improvement was however achieved by changing the network driver from softnet to NAPI. Once NAPI was enabled with kernel 2.4.20, benchmarks were repeated and most of the datagram losses disappeared. We do not show here the results as they were superseded by the ones reported in the next section using kernel 2.6.

### 6.2.2. Results with kernel 2.6.0 and NAPI

Setting the number of descriptors allocated by the driver tx_ring and rx_ring to 4096 (maximum value allowed), the IP send buffer size to 524288 B and the IP receive buffer size to 1048576 B, maximum throughput was 999.9 Mb/s, while datagram loss was dramatically decreased to a rate of $7.1\times10^{-10}$ (101 datagrams lost for $1.4\times10^{11}$ datagrams sent).

Figure 12 shows the maximum data rate achieved as a function of the datagram size. The black line represents UDP payload rate. The red line represents the total rate, all overheads as described in Sec. 1.3, up and including the UDP header. It represents therefore the effective Ethernet wire load, which reaches 100% at 500 B datagram size. The decrease of the throughput under 500 B datagram size indicates a bottleneck in PC hardware and/or kernel. Discontinuities in UDP payload rate are due to the increase in overhead when an additional Ethernet frame is required by the fragmentation process. The minimum Ethernet frame size of 64 Bytes requires padding for frames carrying less data, thus lowering the payload rate.

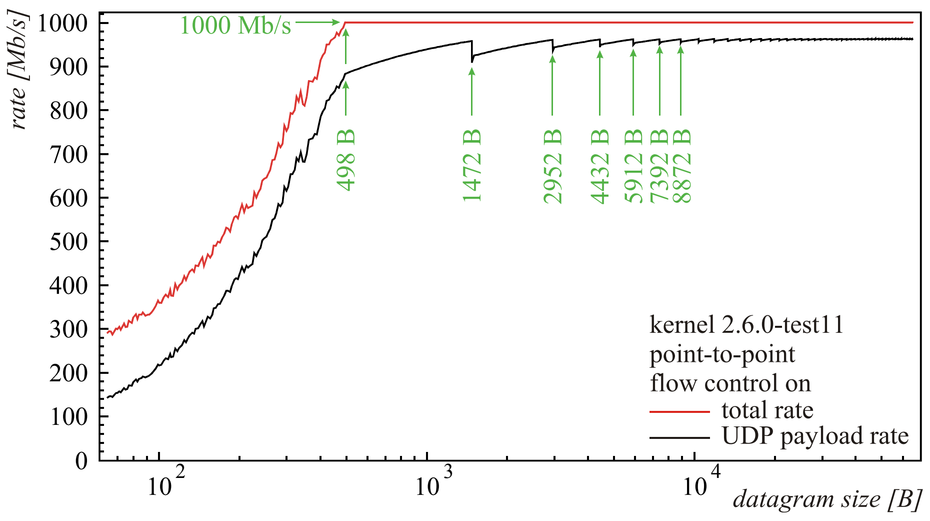

Figure 12: Maximum data throughput versus datagram size. The discontinuities are due to the fragmentation process.

Figure 13 shows the maximum Ethernet frame rate as a function of the UDP payload size. The highest rate of about 280 kf/s corresponds to the shortest Ethernet frames (64 Bytes). When all frames of a datagram have the maximum size, the maximum rate is about 80 kf/s.

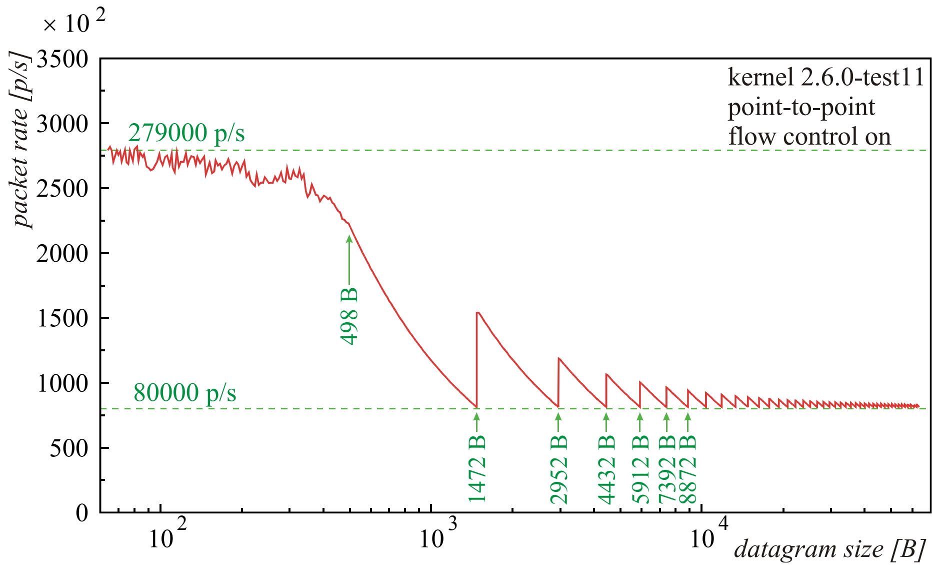

Figure 13: Maximum Ethernet frame rate versus datagram size. The discontinuities are due to the datagram fragmentation process.

Figure 14 and Figure 15 show the datagram loss rate and transmission error rate, respectively, as a function of the datagram size. Transmission errors can be divided in two groups: receive errors as reported by the NIC and protocol handling related errors (failed IP reassembly and UDP protocol errors). The first category includes CRC errors and frame losses due to overflow in NIC internal buffers. Errors reported by the higher level (layer 3 and above) protocol handlers typically mean a failure to reassemble a datagram, usually as a result of missing fragment, thus are usually correlated with the NIC reported errors. They have been observed in earlier kernel versions for large datagram sizes, but the improvements in 2.6.0-test11 kernel seem to have eliminated this problem, as can be seen in Figure 15.

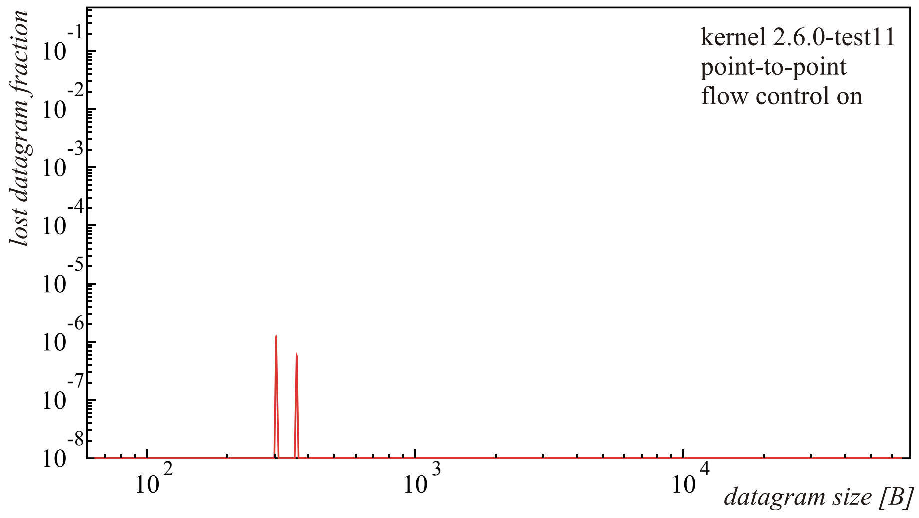

Figure 14: Datagram loss rate versus datagram size.

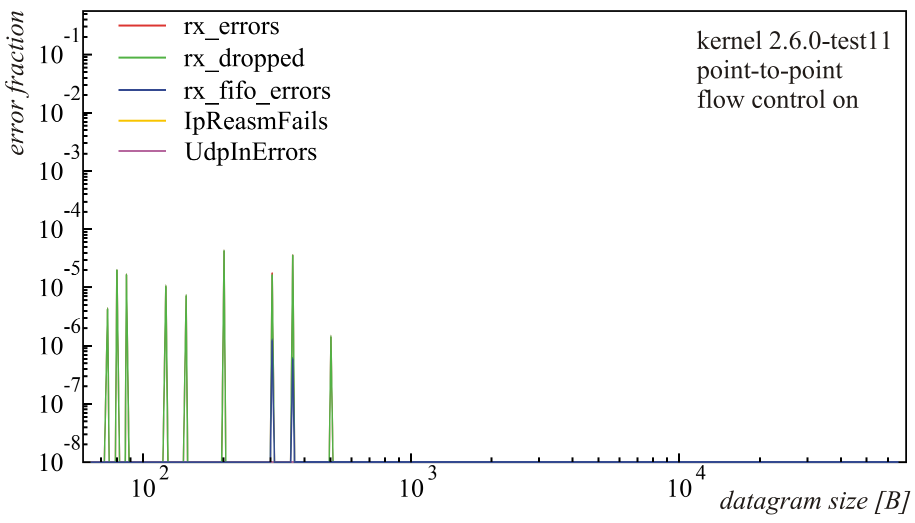

Figure 15: Transmission error rate versus datagram size.

## 6.3. Performance and packet forwarding tests at CERN

Since the Sub-Farm Controller will have to simultaneously receive and send data, we have chosen to test it under this realistic condition. Raw Ethernet packets are received on one network interface and forwarded to a different port. Apart from the event building process, which is well approximated by a constant load on the CPU, this corresponds to the real data flow through the SFC.

### 6.3.1. SFC behaviour under load

Clearly the receiving host, be it a farm node or a Sub-Farm Controller, will have to perform also some handling of the received data, which introduces additional load on the system and adds latency. Under heavy load, packet loss may increase. This has been observed on hosts running the 2.4.20 kernel and NAPI compliant device driver. The new low-latency features of the 2.6 kernel have improved the situation.

In addition to the frame forwarding application, additional CPU load has been generated to reach a total of ~98%. On both the Xeon and Opteron based machines, no packet loss has been observed in $2x10^{10}$ frames (of 1436 bytes each) under full link load. Default coalescence settings have been used, which resulted in ~6 kHz interrupt rate.

### 6.3.2. Influence of the interrupt rate

As described in Sec. 3.3, interrupt rate can be modulated by means of parameters given to the network driver. The adjustment of coalescence settings provides means to tune the packet handling latency without overburdening the CPU with interrupts. Therefore, in principle, we are interested in as high an interrupt rate as possible, but without loosing received frames even at highest link loads. We have thus measured the impact of the interrupt rate on packet drop under heavy network traffic conditions. For this test, we have programmed the data sources to fill the wire at 100%, while the frame size was varied in a range between 800 and 1500 payload Bytes, i.e. in a range where we do not expect packet loss due to transmission rate itself (c.f. Figure 12 and Figure 13). Packet drop has been measured at low (~20%) and high (~98%) CPU load.

Figure 16 shows the packet loss as a function of the interrupt rate on both the Xeon and Opteron SFC candidates under ~20% and ~92% CPU load. The Intel 82546 Gigabit controller was used. At high CPU load, the loss-less limit (0 packets lost in $4x10^{10}$) has been found to be around 10 kHz on the Intel Xeon, and ~120 kHz on the AMD Opteron. The interrupt rate has to be kept low in particular on the Xeon based system, where already 20 kHz lead to significant packet loss ($3x10^{-5}$).

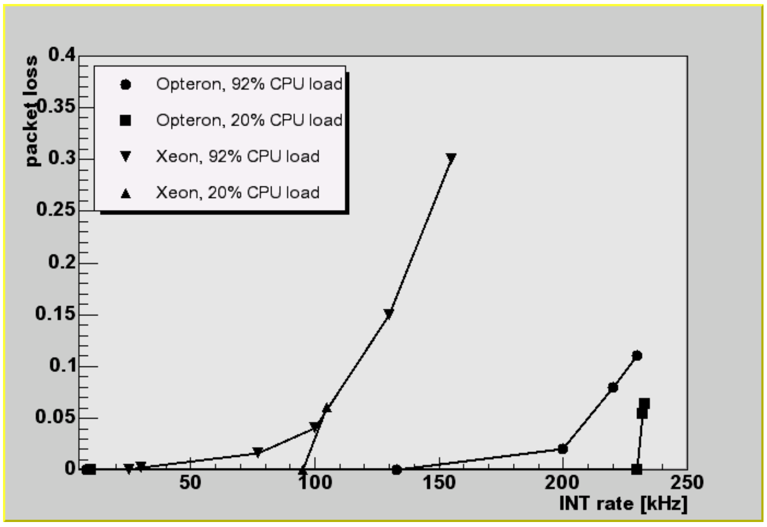

Figure 16: Packet loss as a function of Interrupt rate on the Xeon, and Opteron based systems.

Tests were also carried out using the integrated Broadcom 570x gigabit controller. Two different software drivers are available for this controller: the native Broadcom bcm5700 (v7.0.0) and the tg3 (v2.2) from RedHat. Both drivers are NAPI compliant, the tg3 driver lacks in interrupt moderation features, however. Using both drivers, a packet loss rate of $10^{-8}$ was observed. NIC statistics indicate overflow in receive buffers. In addition, it has been found that the RioWorks motherboard (Opteron) has both Broadcom controllers placed on a 32 bit/66 MHz PCI bus, which clearly creates a bottleneck for two Gigabit streams.

## 6.3.3. IP socket buffer occupancies

As another measure of performance, we have investigated the socket buffer occupancies in the case of IP forwarding. Raw IP packets of 1548 Bytes payload have been generated, i.e. consisting of two Ethernet fragments, in order to force IP packet reassembly. The results are shown in Figure 17.

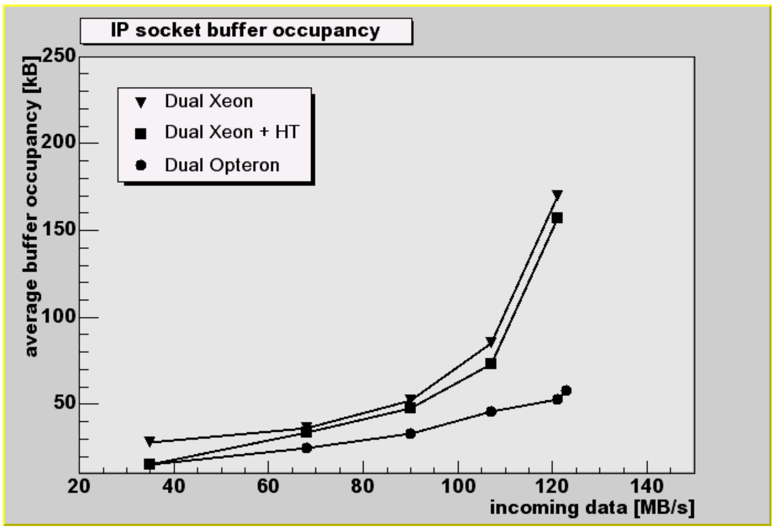

Figure 17: IP buffer occupancies as function of incoming data rate.

In particular at full link load, the Opteron is performing clearly better, with the buffer occupancy around 50 kB. As can be seen from the two Xeon plots, enabling Hyper-Threading brings a performance increase of merely ~10%. This is not surprising, given that the application is not very CPU intensive, but the load lies rather on the I/O capacity.

## 6.4. Packet loss in switch

A potential source of data loss is the possibility of packets being dropped in the switch. This can happen in particular in congestion conditions, i.e. if the output ports are temporarily overloaded for a time sufficient to overflow the output port buffers. We report on the measurement of packet drop rate of 4 Gigabit Ethernet switches: the 3com 4924 routing switch, the HP ProCurve 6108 and 2824 managed switches, and the 3com 2824 unmanaged switch.

### 6.4.1. 3com 4924

In the CERN setup, we have also investigated packet loss rate in the switching hardware. All 24 ports of the switch were connected to data generators. Two tests were performed: a full-mesh test and an aggregation test.

In the full-mesh test, the data generators were programmed to send 1436 byte packets to each other through the switch at maximum rate (~86 kHz). The data throughput per link was 123 MB/s, i.e. 100% link load when the Inter-Packet Gap is taken into account. The traffic pattern was defined with 23 source ports sending to one destination port, while the latter sends a frame to its predecessor[1]. The destination is chosen in a round-robin mode among all the 24 ports. This traffic pattern is valid only for a relatively short time.

[1] The switch "swallows" incoming frames with the destination port being the same as the source port, thus it is not possible to send from all 24 ports to one output.

Since there's no synchronisation between the sources possible at this rate, the resulting traffic pattern becomes random after some minutes already.

No packet drop was observed in $3x10^{11}$ packets. Since also no transmission errors were detected, we can also claim that the bit error rate on short (3 m) cat. 5e cables is below $10^{-15}$, compatible with the result shown in 6.1.

The second test was performed with synchronised sources, and aims at testing the behaviour of the switch in conditions as are expected in the aggregation layer of the LHCb readout network. The incoming frames are synchronised to ~100ns, which at Gigabit rate corresponds to ~12 Byte, i.e. overlap of frames is guaranteed. Traffic pattern was set for aggregation rates of 23:1 and 18:6, i.e. with 23 ports sending to one output port, and 18 ports sending to 6 output ports, respectively.

The frame rate was set to 40.5 kHz, and the incoming packet sizes varied between the minimum Ethernet frame size of 64 Bytes and the maximum frame size possible for 100% output link occupancy. Results are shown in Table 2, no packet drop was observed.

| 23:1 aggr., 40.5 kHz input, 930 kHz output frame rate | | | | 18:6 aggr., 40.5 kHz input, 121 kHz output frame rate | | | |
|---|---|---|---|---|---|---|---|
| | Output | | | | Output | | |
| Frame size [B] | Through-put [MB/s] | link load [%] | Lost frames | Frame size [B] | Through-put [MB/s] | link load [%] | Lost frames |
| 64 | 56 | 62 | 0/1678719446 | 64 | 7 | 8 | 0/1313283798 |
| 80 | 70 | 74 | 0/1676610875 | 379 | 43 | 39 | 0/1314816922 |
| 96 | 85 | 86 | 0/1678424471 | 694 | 80 | 69 | 0/1315031256 |
| 112 | 99 | 98 | 0/1678940572 | 1006 | 115 | 99 | 0/1313678986 |
| 114 | 101 | 100 | 0/1677027681 | 1009 | 116 | 100 | 0/1313147736 |

Table 2: Results of aggregation tests for 23:1 and 18:6 aggregation.

### 6.4.2. HP ProCurve

The switches used by the Bologna group, HP ProCurve 6108 and 2824, on the other hand showed a significant rate of packet loss at already two links loaded with unidirectional traffic, as can be seen in Figure 18 and Figure 19.

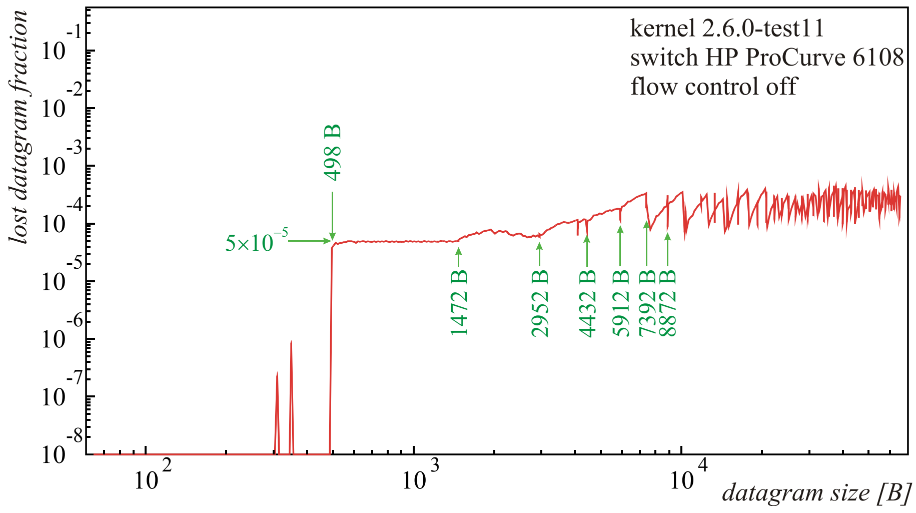

Figure 18: Datagram loss rate versus datagram size using the HP Procurve 6108 switch with flow control disabled.

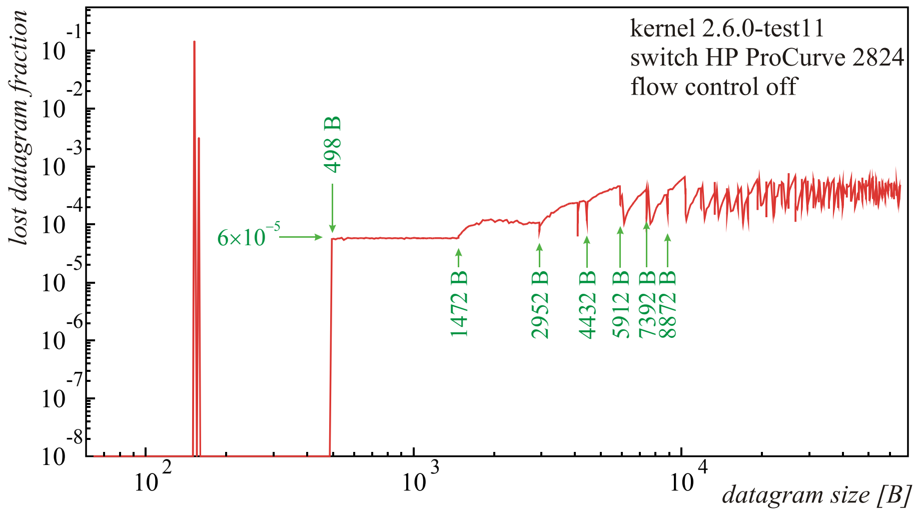

Figure 19: Datagram loss rate versus datagram size using the HP Procurve 2824 switch with flow control disabled.

Switching on the flow control on the switch, but keeping it disabled on the sending NIC, unveiled the occurrence of pause frames sent by the switch. This indicates that the switch cannot handle more than 98% link load and tries to reduce the data rate.

The reported NIC and protocol errors are shown in Figure 20 and Figure 21. In the packet size range [498 B, 1472 B] no errors are counted by kernel counters, being datagram composed by only one Ethernet frame.

In this range datagram loss is therefore due completely to packet drop in the switch. For datagram size greater then 1472 B reassembling errors and UDP errors arise from missed fragments in datagram, also due to dropped packets in the switch.

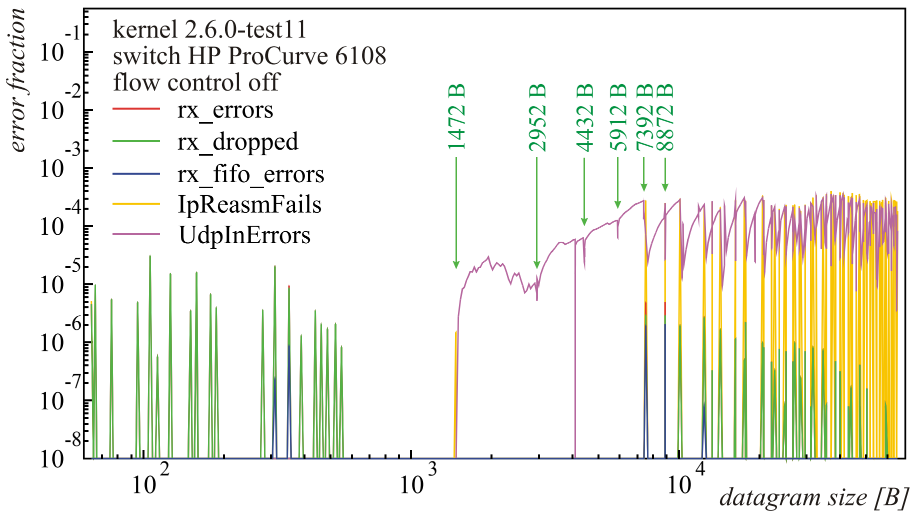

Figure 20: Transmission error rates versus datagram size for connection via the HP ProCurve 6108 and flow control disabled.

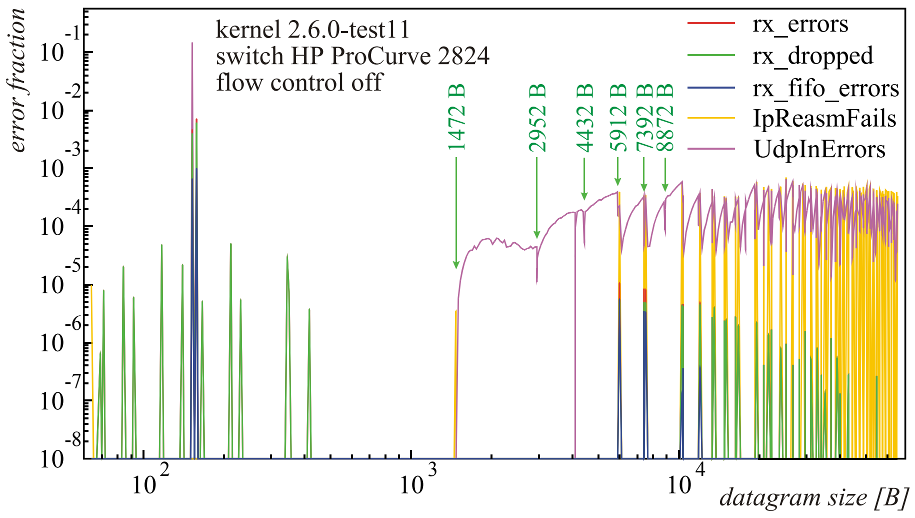

Figure 21: Transmission error rates versus datagram size for connection via the HP ProCurve 2824 and flow control disabled.

Once flow control is switched on, loss as well as error rate decrease significantly, with loss-less transmission for datagram size above 490 B. The observed loss rate in this case is shown in Figure 22, while Figure 23 shows the corresponding error rates, for transmission via the HP ProCurve 6108 switch. Measurements using the HP ProCurve 2824 switch led to similar results.

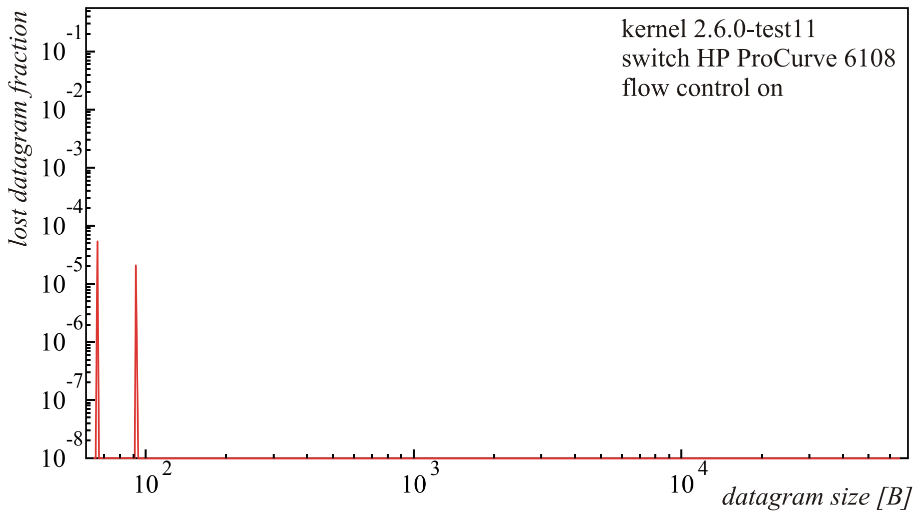

Figure 22: Datagram loss rate versus datagram size for connection through the HP ProCurve 6108 with flow control enabled.

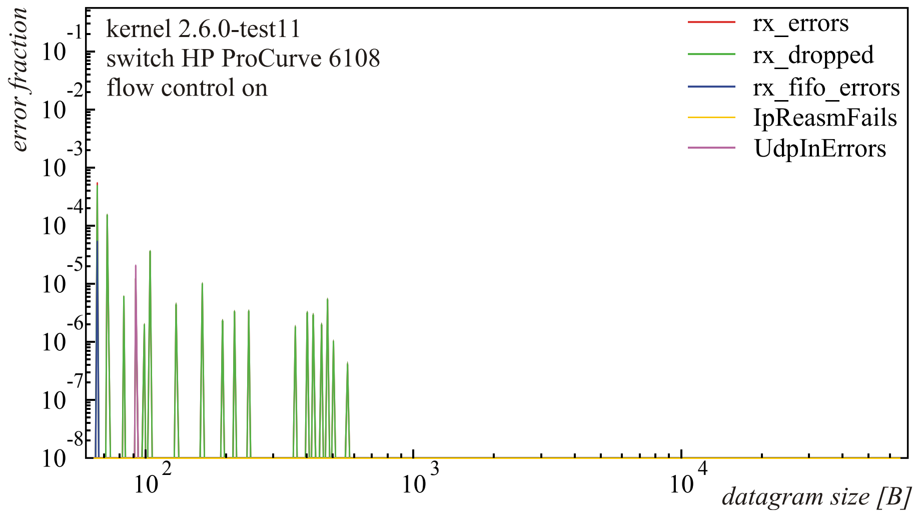

Figure 23: Transmission error rates versus datagram size for connection via the HP ProCurve 6108 with flow control enabled.

A further investigation was performed on switch HP ProCurve 6108 to measure the maximum loss free switching rate. A single unidirectional flow of datagrams (of 1472 B size) was sent through the switch at different rates (using a delay in the send loop) and dropped frames fraction was measured. Results, plotted in Figure 24, show that the switch can manage up to 980 Mb/s raw rate (including overhead and Inter-Packet Gap) without frame drop.

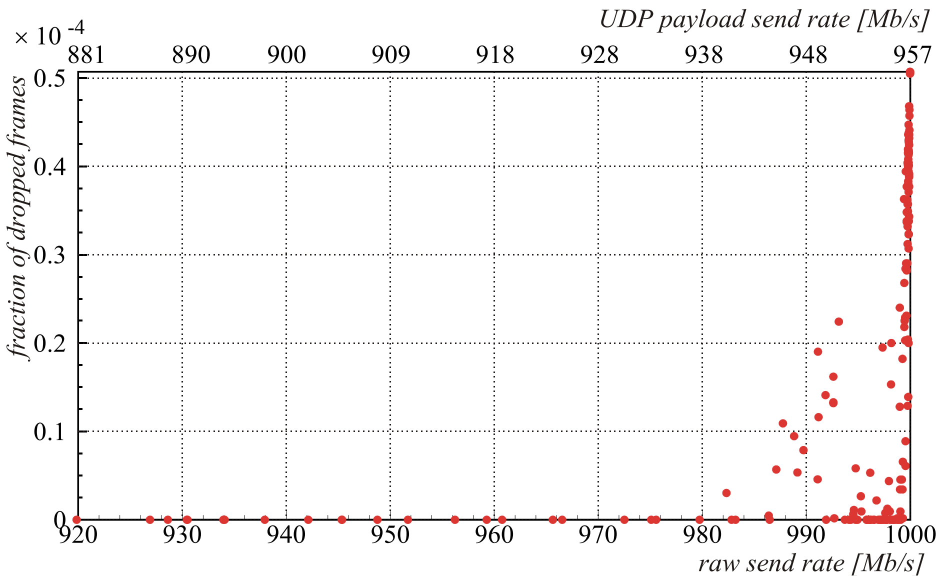

Figure 24: Fraction of dropped frames versus send rate for the HP ProCurve 6108 with flow control disabled (UDP datagram size: 1472 B). Each point corresponds to a 25 minutes run.

## 6.4.3. 3com 2824

The same tests as on the HP ProCurve switches were carried out on an unmanaged 24 port Gigabit Ethernet switch from 3com. All tests resulted in loss-less transmission across the switch at 100% link load.

# 7. Conclusions

Transmission error rate on Copper Gigabit Ethernet has been measured for connections over a 100m long cat. 5e cable. The expected error rate in the full LHCb DAQ setup has been found to be well within acceptable limits.

Evaluating four Gigabit Ethernet switches from two different manufacturers, we have seen clear difference in performance offered by the hardware. While the HP ProCurve 6108 and 2824 switches have shown high packet loss rates of $6\times10^{-5}$ at 100% link load, already with unidirectional traffic between two hosts, the 3com 4924 switch remained loss-less in $3\times10^{11}$ transmitted packets at 100% link load on all ports (simultaneous and in both directions).

Packet loss on server class PCs has been investigated using raw Ethernet (Layer 2), IP (Layer 3) and UDP protocols. With carefully tuned parameters, such as large receive buffers and optimal interrupt coalescence, no packet loss was observed in $2\times10^{11}$ packets, using raw Ethernet protocol and frames sizes close to MTU. UDP packet loss was observed to be $7\times10^{-10}$ with 4096 byte datagram size. Link load for both measurements was 100%.

We have therefore demonstrated that the LHCb readout network as proposed in [1] can be implemented with hardware commercially available even nowadays.

# 8. References

[1] "A common implementation of Level 1 trigger and HLT Data Acquisition", A. Barczyk, J-P. Dufey, B. Jost, N. Neufeld, LHCb 2003-079

[2] "Raw-data transport format", B. Jost, N. Neufeld, LHCb 2003-063 DAQ

[3] "DAQ Simulation Results", J-P. Dufey, LHCb note in preparation

[4] "802.3 IEEE Standard for Information Technology, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications", IEEE, Piscataway, NJ, 2002

[5] PCI Local Bus Specification, rev. 2.2, PCI Special Interest Group, 1998

[6] PCI-X Addendum to the PCI Local Bus Specification, rev. 1.0, PCI Special Interest Group, 1999

[7] "Hyper-Threading Technology on the Intel Xeon Processor Family for Servers", Intel Corp. White Paper, 2002

[8] "Intel Pentium 4 Processor with HT Technology", Intel Corp. White Paper, 2003

[9] http://www.hypertransport.org

[10] http://www.supermicro.com/PRODUCT/MotherBoards/E7501/X5DPL-iGM.htm

[11] http://www.supermicro.com/PRODUCT/MotherBoards/GC_LE/P4DL6.htm

[12] http://www.rioworks.com/HDAMA.htm

[13] "Interrupt Moderation using Intel Gigabit Ethernet Controllers", Intel Application Note, AP-450

[14] Intel® E7501 Chipset Memory Controller Hub (MCH) Datasheet, ftp://download.intel.com/design/chipsets/datashts/25192702.pdf

[15] IBM NP4GS3 Datasheet, http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerNP_Network_Processors

[16] S3 group, http://www.s2group.com/network_processing/copernicus

[17] "Use of Network Processors in the LHCb DAQ Test-Bed", A. Barczyk, LHCb 2004-024 DAQ

[18] "Eliminating receive livelock in an interrupt-driven kernel", J.Mogul and K. K. Ramakrishnan, Winter USENIX conference, January 1996

[19] "Understanding the Linux Kernel", second edition, D.P. Bovet, M. Cesati, O'Reilly, 2002

[20] "Beyond Softnet", J. H. Salim, R. Olsson, A. Kuznetsov, Proceedings of 5th Annual Linux Showcase & Conference, http://www.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf

[21] NAPI HOWTO, A. Kuznetsov, J. H. Salim, R. Olsson, ftp://robur.slu.se/pub/Linux/net-development/NAPI/NAPI_HOWTO.txt

[22] "Linux Kernel 2.6 – New Features III: Networking", slides 30-34, J. Cooperstein, Axian Inc. (http://www.axian.com/docs/linuxkerneltalk3.pdf)

[23] "The post-Halloween document", Dave Jones, http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt

[24] "Lowering Latency in Linux: Introducing a Preemptible Kernel", R. Love, Linux Journal, Issue 97, May 2002, http://www.linuxjournal.com/article.php?sid=5600

[25] "Introducing the 2.6 Kernel", R. Love, Linux Journal, Issue, 103, May 2003, http://www.linuxjournal.com/article.php?sid=6530

[26] "CPU Affinity", R. Love, Linux Journal, Issue 111, July 2003, http://www.linuxjournal.com/article.php?sid=6799

[27] Linux Kernel 2.6 Status - November 2nd, 2003, http://kernelnewbies.org/status/latest.html (see 2.5.25)

[28] Summary of changes from v2.5.24 to v2.5.25, http://www.kernel.org/pub/linux/kernel/v2.5/ChangeLog-2.5.25

[29] "Linux 2.5.25 changes », L. Torvalds, http://www.ussg.iu.edu/hypermail/linux/kernel/0207.0/0741.html

[30] "Linux Kernel 2.6 – New Features I", slides 31, J. Cooperstein, Axian Inc. (http://www.axian.com/docs/linuxkerneltalk1.pdf)

[31] "Driver porting: Timekeeping changes", LWN.net, http://lwn.net/Articles/22808/

[32] "Design of a Real-Time Communication Service for Local Area Networks, Clock Granularity", R. Koster, http://wwwagss.informatik.uni-kl.de/Projekte/Squirrel/da/node5.html

[33] "Linux Advanced Routing & Traffic Control HOWTO", B. Hubert, http://lartc.org/howto/index.html

[34] "Iproute2+ tc notes", M. Lamb, http://snafu.freedom.org/linux2.2/iproute-notes.html

[35] "Linux Advanced Routing & Traffic Control Manpages", B. Hubert, http://lartc.org/manpages/

# A1. List of Operating System's network queues

Since packets are dropped in queues, let's list all the Operating System queues used in networking.

## A1.1 Sender side

1. **Send socket buffer (in user space)**
   Buffer size can be changed with the command:
   ```
   setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (char*)&bS, sizeof(unsigned))
   ```
   Default size: 65535 B
   Default size can be changed by command:
   ```
   echo $newsize > /proc/sys/net/core/wmem_default
   ```
   Maximum allowed size can be changed with the command:
   ```
   echo $newsize > /proc/sys/net/core/wmem_max
   ```

2. **qdisc (queueing discipline)**
   Queueing discipline is used for traffic control (shaping, scheduling, policing, dropping, see http://lartc.org/) and can be accessed by commands "ip" and "tc" (traffic control) from iproute2 package.
   The queueing discipline set for an interfaces, can be retrieved with the command:
   ```
   ip link show
   ```
   Queueing discipline can be pfifo_fast, pfifo, tbf, cbq, red, sfq, prio, csz.
   Default queueing discipline is **pfifo_fast**. Its queue length can be read with command:
   `ifconfig eth0` (look at txqueuelength parameter)
   The default length of pfifo_fast qdisc is 100.
   Queue length of pfifo_fast qdisc can be changed by the command:
   ```
   ifconfig eth0 txqueuelen $size
   ```
   For pfifo_fast qdisc statistics are not available. To get statistics, the **pfifo_fast** qdisc must be replaced with a **pfifo** qdisc, with the command tc:
   ```
   tc qdisc add dev eth0 root pfifo limit $size
   ```
   where `$size` is the wanted pfifo queue length.
   Statistics from pfifo qdisc can be get with command:
   ```
   tc -s -d qdisc show dev eth0
   ```
   To restore the pfifo_fast queue:
   ```
   tc qdisc del dev eth0 root
   ```

3. **tx_ring**
   Statistics can be obtained with the command:
   ```
   ethtool -S eth0
   ip -s link show eth0
   ifconfig eth0
   ```
   The size can be changed during the installation of the network loadable module, e.g., for Intel e1000 module:
   ```
   modprobe e1000 TxDescriptors=$size
   ```
   or by adding permanently to /etc/modules.conf the 2 lines:
   ```
   alias eth0 e1000
   options e1000 TxDescriptors=$size
   ```

## A1.2. Receiver side

1. **rx_ring**
   Statistics can be obtained with the commands:
   ```
   ethtool -S eth0
   ip -s link show eth0
   ifconfig eth0
   ```

   The size can be changed during the installation of the network loadable module, e.g., for Intel e1000 module:
   ```
   modprobe e1000 RxDescriptors=$size
   ```

   or adding to /etc/modules.conf the 2 lines:
   ```
   alias eth0 e1000
   options e1000 RxDescriptors=$size
   ```

2. **backlog queue**
   Backlog queue(s) is present in both kernel ≤ 2.4.19 and kernel ≥ 2.4.20, but in kernel ≥ 2.4.20 it is used only for **back-compatibility** with NAPI-less network driver. It is **not used in kernel ≥ 2.4.20 with NAPI-aware network drivers**.

   In kernel 2.2, statistics from backlog queue can be obtained with command:
   ```
   cat /proc/net/dev_stat
   ```

   where the counters (one row, without labels) are: (1) drop count, (2) number of times the backlog entered the throttle state, (3) number of hits in fast routes, (4) number of success in fast routes, (5) number of defers in fast routes.

   In kernel 2.4 statistics from backlog queue(s) can be obtained with command:
   ```
   cat /proc/net/softnet_stat
   ```

   where the counters (one row for each CPU, without labels) are: (1) packet count, (2) drop count, (3) time squeeze counter (the number of times net_rx_action() breaks the loop leaving a non-empty queue and reschedules itself for later execution), (4) number of times the backlog entered the throttle state, (5) number of hits in fast routes, (6) number of success in fast routes, (7) number of defers in fast routes, (8) number of defers out in fast routes, (9) CPU collision.

   Default length of 300 can be changed by the command:
   ```
   echo $size > /proc/sys/net/core/netdev_max_backlog
   ```

3. **Receive socket buffer (in user space)**
   Buffer size can be changed with the command:
   ```
   setsockopt(sock, SOL_SOCKET, SO_RCVBUF, (char*)&bS, sizeof(unsigned))
   ```

   Default size: 65535 B
   Default size can be changed by command:
   ```
   echo newsize > /proc/sys/net/core/rmem_default
   ```

   Maximum allowed size can be changed with the command:
   ```
   echo newsize > /proc/sys/net/core/rmem_max
   ```