Studies for the ATLAS Second Level Trigger using Data-Strobe Link Technology

Thesis submitted in accordance with the requirements of
the University of London for the degree of
Doctor of Philosophy

by

Candidata Scientiarum
Nina Anette Hagen Madsen

University of London
Royal Holloway College, Physics Department
1999

it is a dry white season
dark leaves don't last, their brief lives dry out
and with a broken heart they dive down gently headed for the earth,
not even bleeding.
it is a dry white season brother,
only the trees know the pain as they still stand erect
dry like steel, their branches dry like wire,
indeed, it is a dry white season
but seasons come to pass.
Mongane Wally Serote (1974)

Abstract

A study of the ATLAS second level trigger using Data-Strobe Link technology is described together with software tools developed for this network technology.

ATLAS is one of the two general purpose detectors proposed for the Large Hadron Collider at CERN. The second level trigger will receive events at rates up to 100 kHz and must reduce this by around two orders of magnitude. The second level trigger architecture chosen for this study is called the local-global model. This architecture has four local processor farms, one per sub-detector, each connected by its own network to data buffers. The local processors are connected via another network to a global processor farm where results from the local processing farms are collected and evaluated.

The architecture has been emulated on two large hardware platforms, Macramé and GPMIMD. Both platforms use DS-Link technology. Improvements to the running of these platforms by designing, implementing and testing a diagnostic software utility called Netprobe are described. This utility facilitates hardware, board and system-setup debugging.

Macramé is a test-bed for studying network traffic patterns. It consists of C104 routers and specially-designed traffic-generating nodes and timing nodes. These components can be configured into a variety of topologies.

Each of the five networks were mapped to Macramé one at a time and the traffic patterns expected in each case were run through the network. The results obtained are presented and interpreted.

A process model of the second level trigger was developed and implemented on GPMIMD, a general purpose parallel computer, and is described together with benchmark results of the components. Results from running this model with both test and detector specific data are presented.

Acknowledgement

I would like to thank my joint supervisors J.A. Strong and R.W. Dobinson for their support and guidance during the past years. I would like to thank the ATLAS level-2 trigger community for support and fruitful discussions. I am particular grateful to R. Hauser, R. Heeley, P. Maley, J.C. Vermeulen, R.H. Jones and J.R. Hansen.

For financial support during my study, I would like to thank the Danish Research Academy, CERN, ATLAS and the ESPRIT projects MACRAME and ARCHES.

I would like to express my sincere thanks to my husband D.M. Steele, for his encouragement and support throughout my study. I am also most grateful to all the friends I made at CERN during my stay. Finally, I would like to thank all the people over the years who taught me to never give up.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter contains a short description and the motivation behind ATLAS. Afterward, the problem statement and objective of this work are presented. Finally, the structure of the thesis is described, including details on which parts are the author's contributions.

1.1 Motivation for the ATLAS Experiment

ATLAS [1, 2], A Toroidal LHC ApparatuS, is a proposed general purpose proton-proton experiment at the Large Hadron Collider (LHC) at CERN, Geneva, Switzerland. The experiment expects to start data-taking in 2005. The LHC offers a wide range of physics possibilities, from supersymmetry and Higgs searches to investigations of CP violation in B-decays and top quark studies.

Many of the above mentioned searches/analyses require a high luminosity. The LHC is designed with a nominal luminosity of $10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$, three orders of magnitude more than the Tevatron run at Fermilab, USA in 2000 [3], and four orders of magnitude larger than the last run at the SPS, CERN [4]. The LHC will provide a high-rate environment, the detector will, in order to deal with this, provide many signatures: e, γ, μ, jet, missing energy and b-quark tagging. The design of the ATLAS detector tries to make the most of the LHC physics potential while using cost-effective technologies.

The most intriguing issue of the LHC is the origin of spontaneous symmetry-breaking mechanism in the electroweak theory, as this will provide information to the question: "What is the origin of mass?" A possible measurable effect of the spontaneous symmetry-breaking is the existence of a Standard Model Higgs boson (H) or e.g. a family of Higgs

bosons ($H^{\pm}$, h, H and A) in the Minimal Supersymmetric extension of the Standard Model. The search for the Higgs boson(s) has first priority when optimising the detector design.

The Higgs boson decay products include b-quarks, photons, Z, W, τ, jets and indirectly leptons and neutrinos. Over the majority of the Higgs mass range explorable by the LHC the cross-section of the Higgs processes are small, hence the need to operate at high luminosity and to detect and measure precisely the above mentioned particles.

Supersymmetry (SUSY) searches are also an interesting aspect of the LHC. To exclude the observation of the lightest stable SUSY particle, it is necessary for the detector to meet strict hermeticity and E_T^{miss} requirements.

Detection of new, heavy (up to 5-6 TeV) gauge bosons Z', W' necessitates charge identification and high-resolution lepton measurements up to a p_T of a few TeV. While the above mentioned new physics has been the main guideline for the detector design, the additional goal of making extensive beauty and top studies, has also imposed many constraints. This includes precise secondary vertex determination, reconstruction of final states with low-p_T particles, low-p_T lepton first-level trigger thresholds and second-level trigger track triggering. The basic design considerations can be summarised as follows:

- electro-magnetic calorimeter electron and photon identification and measurements,
- hermetic jet and missing E_T calorimetry,
- efficient tracking at high luminosity,
- stand alone, precision, μ momentum measurements up to the highest luminosity,
- very low-p_T trigger capability at lower luminosity,
- large acceptance in η coverage, and
- triggering and measurements of particles at low-p_T thresholds.

1.2 Introduction to the Sub-Detectors

The overall layout of the ATLAS detector consists of a inner detector, a calorimeter, a muon spectrometer, and two super-conducting magnets. Figure 1.1 contains a drawing

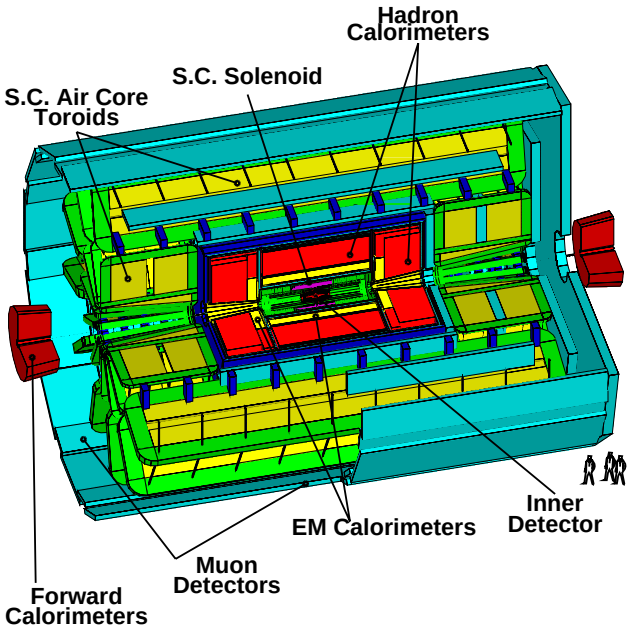


Figure 1.1: Cut-away view of the ATLAS detector.

of ATLAS in a cut-away view. To give a feel for the size of the detector, people have been included in the lower left corner.

The Inner Detector [5, 6] dimensions are a cylinder of 1.15 metre radius and 6.8 metre in length centred around the interaction point. The inner detector lies in a two Tesla axial magnetic field. The inner detector consists of three sub-detectors, pixel detectors at the smallest radii provide high-resolution 3-D points, the semiconductors tracking (SCT) detectors placed outside the pixel detectors provides fine granularity and high-precision momentum and vertex measurements. To achieve reliable pattern recognition for the track finding, a detector of straw tube trackers is placed outside the SCT. This Transition Radiation Tracker (TRT) provides continuous track measurements. The combination of these three detectors results in an inner detector with high-precision coordinates and very good pattern recognition capability. Figure 1.2 shows a simulated event in the barrel section of the inner detector.

The Calorimeter [7, 8, 9] consists of an inner electro-magnetic calorimeter (ECAL) and an outer hadron calorimeter (HCAL). The first will absorb electrons and photons,

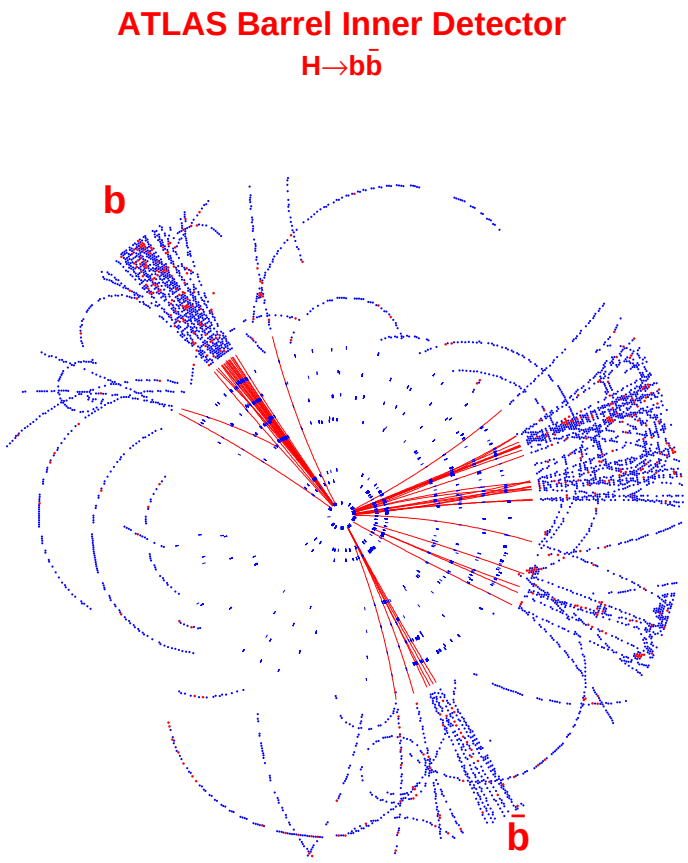


Figure 1.2: A simulated Higgs event decaying to a b-quark and a anti b-quark in the barrel of the inner detector.

while the latter will absorb hadrons, decay products of taus and most other particles except neutrinos and muons. With the calorimeter, the amount of energy deposited per unit length is measured. Particles not immediately stopped can be tracked by the calorimeter, and the shower shape can be used for particle identification. The calorimeter contributes to very good jet and missing E_T performance. The calorimeter is two metre deep in the barrel section, corresponding to an active calorimeter depth at $\eta = 0$ of $9.5\,\lambda_{\text{abs}}$. The HCAL contains the solenoid flux return iron yoke integrated into its support structure.

The Muon Spectrometer [10] is the outer layer of ATLAS, defining its overall dimensions (radius 11 metres, length 42 metres). The toroidal magnet system generates a large field volume and strong bending power with a light and open structure. A very good muon momentum resolution is achieved with three layers of high-precision tracking

chambers. Fast trigger chambers complement the tracking chambers. The layout of the chambers in the barrel has been divided into large and small, with the small chambers placed over the coils. The chambers overlap so they can be aligned with respect to each other using tracks from traversing muons.

The Magnet System consists of a super-conducting solenoid around the inner detector and eight super-conducting barrel and sixteen end-cap air core toroids placed inside the muon spectrometer. The solenoid magnet produces a two Tesla field inside the inner detector. The return field goes through the calorimeter. The air core toroid system creates a toroidal field around the beam inside the muon spectrometer. These magnetic fields cause charged particles to be bent, thereby enabling charge identification and particle momentum measurements.

1.3 The LHC Beam Characteristics

The energy at injection of protons into the LHC [11] is 450 GeV. The protons are, thereafter, accelerated to 7 TeV, which is the energy at collision. The luminosity, which is determined by the density of particles and the collision rate, will be $10^{33}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$ initially. The primary goal is to run at $10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$, but the LHC is designed to go beyond even this. Each beam consists of a set of bunches. The bunch separation is 25 ns in time. At chosen interaction points, one of which is inside the ATLAS detector, the bunches collide creating events in the detector at the collision rate of 40 MHz. At high luminosity, the interaction rate is approximately twenty-five per bunch crossing.

1.4 The Trigger/DAQ

The main task of the trigger [12, 13, 14] is to filter the events, such that only the most interesting events are read out to permanent storage for later off-line analysis. The filter in the trigger must reduce the event rate by 5-6 orders of magnitude, from 40 MHz to 10-100 Hz. A fuller description of the trigger and DAQ system is given in chapter 2.

1.5 Problem Statement and Objective

The work presented in this thesis, is an emulation study of the processor and network usage in the second level of the ATLAS trigger. The emulation has been carried out using DS-Link technology.

One purpose of the emulation studies is to determine at which rate message patterns, corresponding to ATLAS second level trigger traffic, can be sent through currently available technology. The second purpose is to study the effect of implementing the protocol on transputers. Software optimisation issues are also investigated.

Network aspects are studied on the Macramé test-bed and the processor aspects on GPMIMD. The Macramé test-bed is a switching network of up to 1024 nodes, which can evaluate the network performance of traffic patterns. GPMIMD is a sixty-four node parallel computer. The nodes are T9000 transputers.

1.6 Thesis Outline

This chapter has given a overview of the Large Hadron Collider and the ATLAS experiment. The problem and objective for the thesis have also been presented.

The following chapter contains a description of the proposed ATLAS trigger/DAQ system. This chapter also reviews the requirements of the second level (level-2) trigger, outlines which aspects of the level-2 trigger are studied in this thesis and describes the different architecture solutions under investigation.

The third chapter describes the technology used in the emulation. First, the IEEE 1355 DS-Link standard is introduced and general network topologies are explained. Following this, we focus on the DS-Link systems, the GPMIMD machine and the Macramé test-bed used for the emulation studies. Finally, the control and host options for DS-Link systems are explained.

Chapter four, details the author's work on the test and diagnostic software tool, Netprobe. The introduction to this chapter contains the motivation behind developing Netprobe. The sections in this chapter describe the command line interface, spying, verifying and configuring a DS-Link network, and the support which was added for HS devices. Finally, the outcome of the work with Netprobe is discussed.

Chapter five describes the input which has gone into the emulation. The main topics

here are the configuration of the sub-detectors, benchmarks of the trigger algorithms and finally the simulated data used for driving the emulation.

The next two chapters, present the author's emulation study first on the Macramé test-bed and, thereafter, on the GPMIMD machine. The conclusions are summarised and presented in the last chapter.

Chapter 2

ATLAS Trigger/DAQ System

This chapter provides a basic description of the trigger and data acquisition system needed by an LHC experiment, like ATLAS, as well as information on some of the design options for the level-2 trigger. Finally, a more detailed description is given on the part of the trigger which was emulated.

The trigger and the data acquisition system have different tasks and priorities. The task of the trigger is to select the most interesting events. The trigger must choose only as many events as the data acquisition system can transfer to permanent storage; usually tapes. One task of the DAQ system is to organise the data flow though the system and to provide data to the trigger system. Another task is to move the data from events that were chosen by the trigger to permanent storage. Finally, a monitoring task is also performed.

2.1 Overview of the Trigger/DAQ System

This section will provide an overview of the trigger and data acquisition system architecture. A general description of the main components is also given.

The ATLAS trigger/DAQ system is organised into three layers refered to as the first and the second level trigger, followed by the event filter. The data acquisition is responsible for the data-flow through the system. Figure 2.1 shows a diagram of the trigger/DAQ architecture.

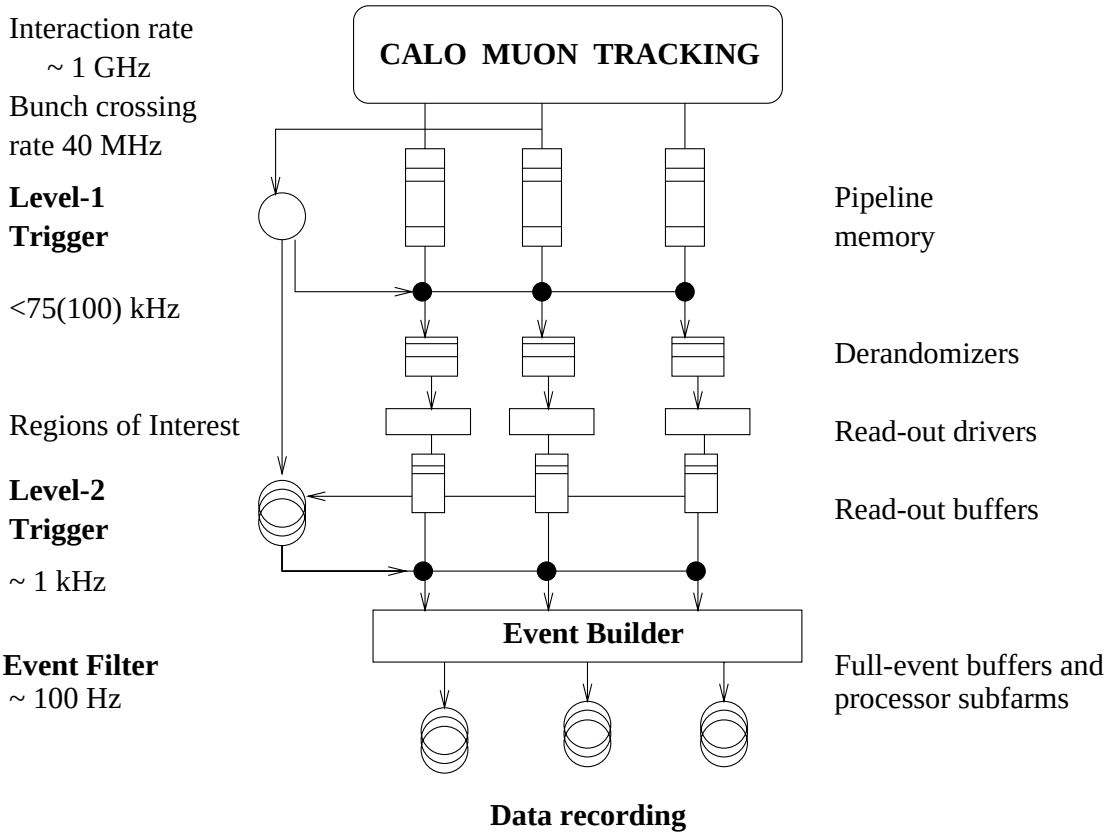


Figure 2.1: A sketch of the flow between the trigger and DAQ.

The Trigger The first level trigger uses special-purpose processors to act on a subset of the data at the full bunch-crossing rate of 40 MHz and must reduce the event rate to 75 kHz (upgradable to 100 kHz). The second level trigger must reduce the event rate down to about 1 kHz; this is done by processing data from regions pointed to by the first level trigger. The third level of the trigger, the event filter, uses all the event data, as well as available calibration data to make the final selection of events to be stored.

The First Level Trigger The first level trigger (LVL1) uses specially designed hardware and looks only at reduced granularity data from some of the sub-detectors. The latency of the first level trigger which is the elapsed time from time of interaction to time of distribution of decision, is about $2\mu s$. During this time, all the detector data is stored in on-detector memory. Upon accept from the first level trigger for an event, the event data are transferred to the read-out buffers, where they can be accessed by the higher level triggers. For each event accepted, the first level trigger provides information to the next level as to why this event was accepted. This information contains the numbers of signatures found, their position and type. This information is known as the Region of Interest (RoI) information.

The Second Level Trigger The second level of the trigger uses the RoI information from the first level trigger to guide its effort. The level two trigger uses full-granularity, full-precision data for its processing, but only from regions and sub-detectors which, according to the RoI information, are expected to contain useful information. If the RoI information mentions a jet-like feature in a certain cone of the detector, only data from the calorimeter detectors in that region are used by the second level trigger. See chapter 5 for more details. This guiding is essential to keeping down the processing and data-transfer requirements of this part of the trigger.

The Event Filter The event filter acts as the third level of the trigger. Its task is to take the final decision about which events are to be put on permanent storage. After level-2 has accepted an event all its data are collected, from the read-out buffers, in one of the event filter's processors. Here, a full event reconstruction is possible; the decision time is expected to be around one second. The event filter must achieve a data-storage rate of about 100 MBytes/s

The Data Acquisition (DAQ) System The DAQ system is responsible for the data flow from the front-end memory until the event data are stored on permanent storage.

2.2 The Level-2 Trigger

This section will give an overview of the second level trigger which has been emulated. Choices had to be taken as to which architecture and protocol were to be emulated, since these are still not decided.

Requirements This paragraph will present some of the most important requirements which the second level trigger must fulfil, seen from the outside systems.

The level-2 trigger acts on events at the rate retained by the first level trigger. This rate is estimated to be 75 kHz, including a safety factor. The second level trigger must be scalable up to 100 kHz input rate. The role of the second level trigger is to reduce the trigger rate to a level which can be sustained by the event building system; of the order of a few kHz. In the design of the second level trigger, full granularity, full precision data from regions in the event, selected by the first level trigger, will be available to the second level trigger. From a cost and maintenance point of view the level-2 trigger must, where possible, consist of commercial, off-the-shelf-components.

The event processing can be split into a number of steps: feature extraction, object building, and trigger type selection. In feature extraction, data from one region of interest in one sub-detector are collected and processed to give a compact description of the data, e.g. convert hits into a track. Object building takes the features from a region of interest, from the relevant sub-detectors, combines them and produces an object signature and possibly a particle id. The combination of objects in an event is compared to a menu of physics selections; also known as the trigger menu. The event selection decision is based on the outcome of this comparison.

The trigger selection criteria will be considerably looser than those used for the final data analysis. To allow for efficient use of new knowledge the trigger menus must be flexible, i.e. changeable on a per run basis. The level-2 trigger will allow pre-scaling and forced acceptance for some events. The latter is for calibration purposes, while the former can be used to reduce the number of well-known high-cross-section events.

The level-2 Components The second level trigger consists of four main building blocks; the read-out buffer (ROB) complex, the supervisor system, the networks, and processing farm.

The read-out buffer complex receives data from the front-end electronics upon accept of the event from the first level trigger. The read-out buffers are used as source of data for both the event building and the second level trigger. The second level trigger can request data from the ROBs and clear data in the ROBs from events which it rejects.

The supervisor complex receives the RoI information from the first level trigger and combines it into a RoI record. The second task of the supervisor is to assign processing resources to the event and forward the RoI record to the part of the level-2 trigger which needs it for obtaining the RoI data. The third task is to receive level-2 decisions and broadcast them to the ROB complex and the event builder. In addition, the supervisor monitors the processor usage.

There are two logically distinct networks in the second level trigger; the network which carries the level-1 accept, RoI information, event data requests to the ROB complex, and the level-2 trigger decisions; and the network which carries the event data from the ROB complex to the processors.

The processing farm consists of processors connected to the network carrying the event data. The processing farm can be partitioned both logically and physically in various ways.

Possible Architectures For comparison purposes three architectures were proposed. These three architectures were selected to be orthogonal enough that other possibilities were mainly "combinations" of the first three. The three architectures were known as the data-driven, the local-global, and the single farm.

The data-driven architecture is partitioned in a local and a global part. Processing up to feature extraction is done in fast pipelined "data-driven" processors. Global processing is done in a farm of general purpose processors. Dedicated hardware is used for extracting data from the ROBs and pushing it to the data-driven processors. A general purpose switching network is used for data transfers between the data-driven processors and the global farm of processors.

The local-global architecture consist of local processing farms, one per sub-detector;

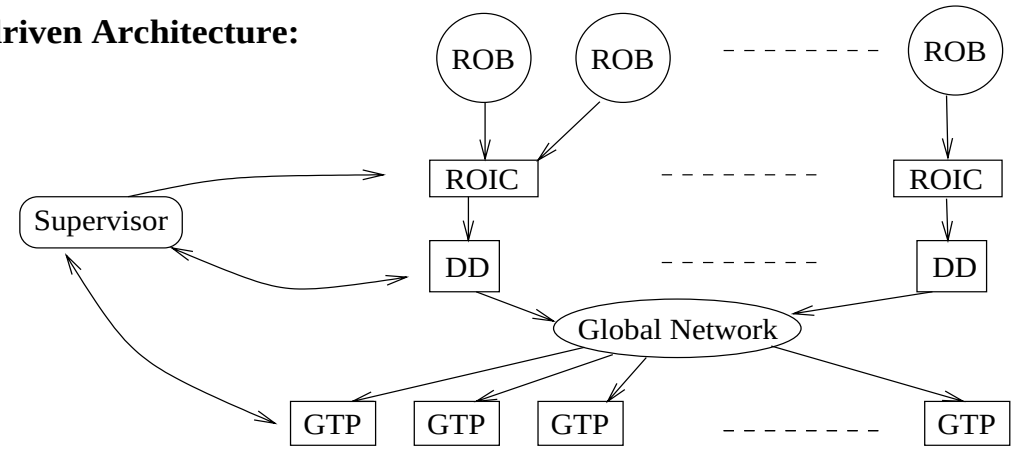
and a global processing farm. The processors in the farms are all general purpose processors. General purpose switching networks are used to connect the local farms to their respective ROBs, and to the global processing farm.

The single farm architecture consist of one large farm of processors connected via a general purpose switching network to the supervisor and the ROBs. A processor is assigned an event from the supervisor, and will then collect data from the ROBs for processing. The data collection and processing will happen interleaved, such that the processor asks for some data, then does some processing, then decides if the event can be rejected, and if not, which data it should now request.

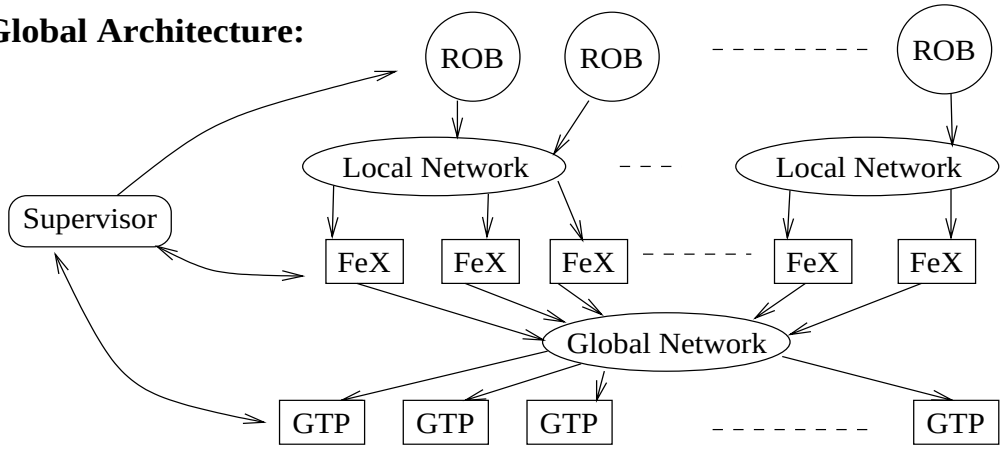
Figure 2.2 shows the three main architecture options for the second level trigger. For more information about these architectures see [13].

Time Scale Along with the technology choice in June 2001, the Technical Design Report for the higher level triggers in ATLAS must be submitted with a detailed specification of the final system. The construction of the higher level triggers are then expected to be finished by the end of 2003. In 2004, integration of the trigger, sub-detectors, DAQ and Detector Control System (DCS) must be completed to ensure operation in 2005 when ATLAS is supposed to start data-taking.

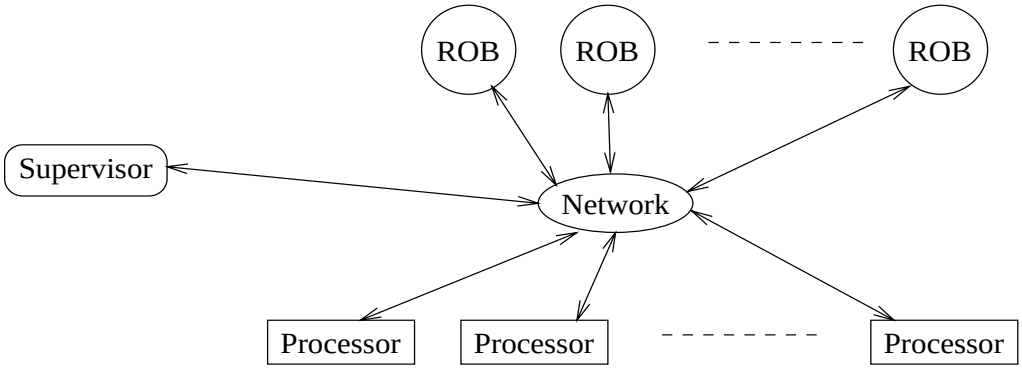
Data-driven Architecture:



Local-Global Architecture:



Single Farm Architecture:



ROIC: RoI collector
DD: data-driven feature extractor processor
FeX: Feature extractor processor
GTP: Global trigger processor

Figure 2.2: The three main architecture options for the second level trigger.

2.3 Architecture Choice

As the architectural and protocol choices for the second level trigger were not (and are still not) taken, we had for the emulation studies to decide which architecture(s)/protocol(s) to study. The choice fell on a architecture known as the local-global architecture with a "push" protocol. I shall try to quickly sketch the reasons for this choice.

The motivation for choosing the local-global architecture for the emulation lay in:

- the Macramé test-bed cannot deal with cause/effect i.e. the single farm architecture, where upon receiving data the processor must request more data,
- the data-driven architecture is designed for different technology,
- DS-Link technology was used in one of the implementations of the vertical slice for the local-global architecture test-bed [15], and in
- the Macramé test-bed size was a good match for the subnetworks in the local-global design.

The Push Protocol for the Local-Global Architecture The general idea behind this protocol is that the supervisor allocates feature extractor (FeX) processors and a global trigger processor (GTP) per event, and informs the ROBs where to send their data. The local and global processors wait to receive information from the ROBs and local processors respectively. They then collect together the fragments received, process these and pass on the results. The supervisor uses a Round Robin processor allocation scheme with feedback from the processors.

In this protocol, see figure 2.3, the data is pushed forward as soon as possible. Any receiving unit must, along with the data, receive information about how many fragments to wait for.

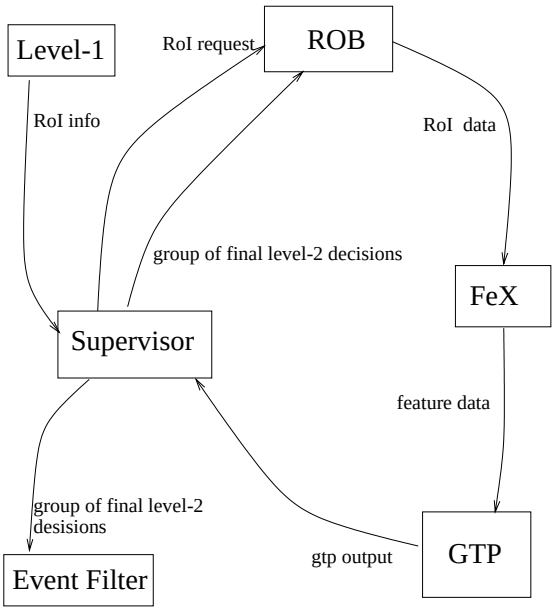


Figure 2.3: A sketch of the push protocol.

2.4 Concluding Remarks

The last section has sketched the different design options being investigated over the last three years, explained why the local/global architecture was chosen for the emulation.

Having introduced the system to be emulated, the next two chapters will concentrate on the DS-Link technology used for the emulation. After these chapters, the input assumptions etc. will be described before the results of the emulation are presented.

Chapter 3

Data-Strobe Link Technology

This chapter describes the hardware used for the emulation studies. Both the Macramé test-bed and the GPMIMD machine are implemented using DS-Link technology.

First, the IEEE 1355 standard is introduced, as this gives a basic level of understanding of the DS-Link technology. Next, the "Clos" network topology is described. Then, the Macramé test-bed and the GPMIMD machine are described in detail. Finally, the host and control systems are described.

3.1 The IEEE 1355 Standard

This section describes the basic concepts of the technology used for the emulation studies.

The IEEE 1355 [16] standard has been developed to exploit recent technical developments of highly-integrated, low-power interconnect technology implemented in high-volume commodity VLSI[1] processors, and uses the simplifications in encodings and protocols resulting from the use of relatively reliable media over relatively short distances.

The standard defines point-to-point links, both physical and logically. The links operate in speed ranges from 10 to 100 MBits/sec in the case of DS-Links and up to one GBit/s for the High Speed (HS) Links. The HS-Links have not been used in the studies presented here, thus the description here will not contain further information on these links.

[1] Very Large Scale Integration

The standard describes a four-layer protocol which the hardware must implement in order to comply with the standard. They are the signal, token, exchange, and packet layers.

The Signal Layer The signal layer defines the interpretation of bits. The DS-Link has two signals in each direction. One containing the data signal, and the other containing the strobe signal. This encoding ensures that, within each bit, there is an edge in either the data signal or the strobe signal, as shown in figure 3.1. The edge within each bit enables the receiver to decode the signal without knowing the transmit rate.

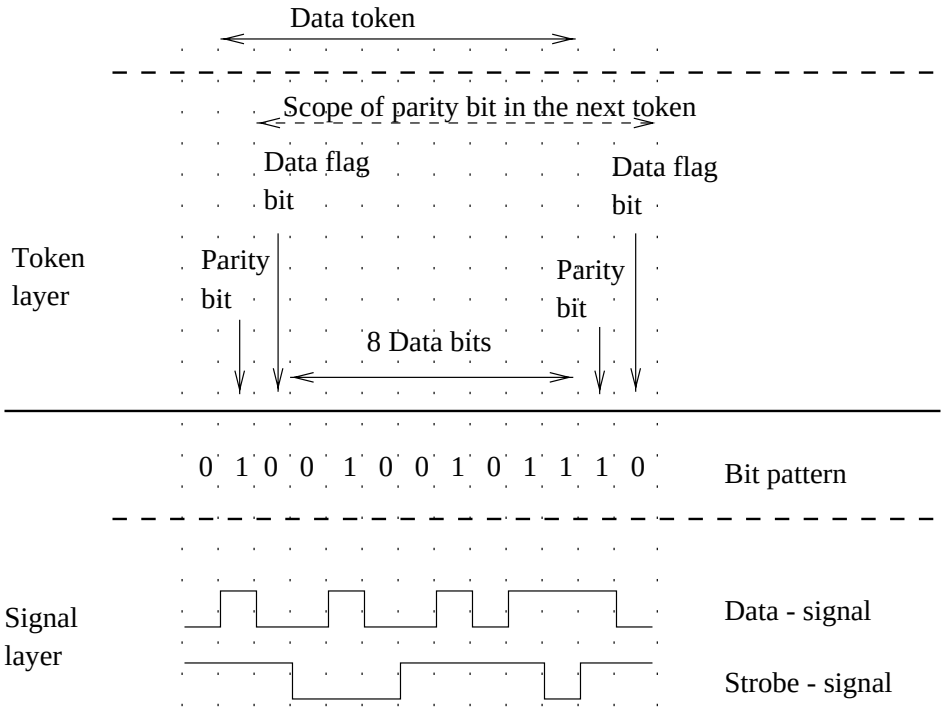


Figure 3.1: The signal and token layer.

The Token Layer The token provides the smallest usable unit of information. There are both control and data tokens. The control tokens are used for control purposes in the upper layers. A token starts with a parity bit, followed by data/control bit. Thereafter, come the data bits of the token, see figure 3.1. Data tokens have eight data bits, while control tokens have only two. Odd parity checking is used for bit error detection. The parity bit covers the previous token's data bits and the current token's data/control bit.

The Exchange Layer This layer describes the procedure for exchanging tokens between nodes. After a link has been started, it starts sending data alignment tokens (idles); a control token. This token is always sent in the absence of other tokens. This allows parity and disconnect errors to be detected at all times.

A flow control scheme is defined in the exchange layer to guard against data loss due to an overflow of the receiving buffer. The flow control is implemented such that when the receiver has room for eight more tokens in its internal buffer, it sends a flow control token to the sender. The sender can thus keep track of whether or not it can send the next token. Consequently data can be queued, but not lost.

The exchange layer also defines the behaviour of the link in case of a reset, a disconnect error, and a parity error. In the first two cases, the link is stopped, and brought to a ready state. From this state, the link can safely be started again. In the case of a parity error, the link output will be stopped and a disconnect error will be detected at the other end. Both links should then be reset. In case of either error, the error flag on the link is raised.

The Packet Layer This upper layer of the IEEE 1355 protocol defines the exchange of packets between nodes. A packet consists of a header, a payload, and an end-of-packet token. The header contains the address of the hardware destination processor, and if applicable also software process identifier. There are two end-of-packet tokens, one indicating a normal end of the packet, the other indicating an unexpected end of packet. There are no limits to the number of tokens per packet in the IEEE 1355 protocol.

3.2 The Clos Topology

This section provides a description of the network topology used in both the Macramé test-bed and the GPMIMD machine.

A switch takes n inputs and provides m outputs; n and m are in most implementations the same. The main feature of a non-blocking switch is that an incoming signal can be switched to any of the output ports without interfering with other signals crossing the switch at the same time. Using one or more switches, a network can be formed upon which terminal nodes can be placed. Depending on how the switches are interconnected, different topologies can be built. One way of connecting switches is by placing them in

a grid and connecting the links to their nearest neighbours. An example of a grid is shown in figure 3.2. A more complicated scheme is used in the Clos topology. The Clos[2] topology was developed by Charles Clos in 1952 [17]. An example of a Clos network is shown in figure 3.3.

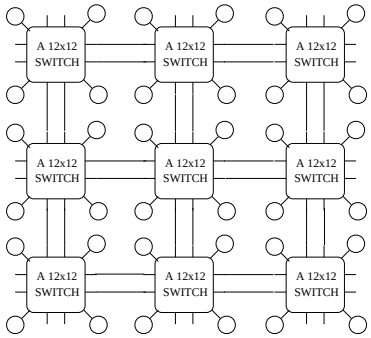


Figure 3.2: An example of a grid network using switches with 12 links. Four of the links on each switch, are used for terminal nodes (shown as circles connected to the switch).

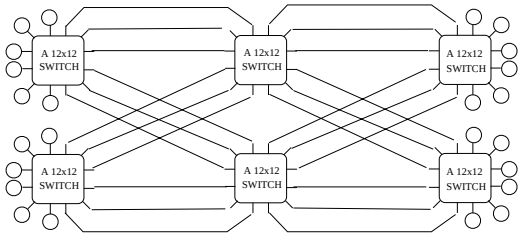


Figure 3.3: An example of a Clos network with 24 terminal nodes and three stages. As in the example above, any terminal node can communicate with any other terminal node.

The Clos topology is non-blocking. That is, any two non-busy nodes, can communicate regardless of the state of the rest of the nodes and network. Another advantage of the Clos topology is that, in order to pass from any one node to another, a packet needs, at maximum, to go through S switches, where S is the number of stages in the network. This can be compared to, for example, a two dimensional $M \times M$ grid network where there can be up to $2M - 1$ switches to go through. The distance, in term of switches, is therefore more uniform for the Clos topology than for the grid topology. In the case of the networks used for the emulation studies, S equals three.

[2] Also known as multi-stage and folded Clos.

3.3 The Macramé Test-bed

Macramé is defined by Webster's New World Dictionary [18] as:

> **mac·ra·mé** *(makґre-mā ´),* **n.** *[Turkish maqramah, serviette, towel < Arabic migramah, a veil], a coarse fringe or lace of thread or cord knotted in designs, etc.*

Having seen the test-bed fully wired up, the name, Macramé, is quite fitting. The Macramé test-bed is part of the Macramé ESPRIT project 8603 [19]. The test-bed was designed, built, tested, and used at CERN.

The test-bed was designed [20] to investigate and demonstrate a number of aspects of network handling and usage, e.g. to construct very large networks with different topologies, and for each topology, to measure the performance of the network, i.e. latencies and throughput, as a function of packet length and traffic patterns. Furthermore, it was designed to investigate traffic patterns corresponding to those found in a variety of application areas, both commercial and in the field of high-energy physics.

The work on this test-bed described in this thesis has been to investigate network traffic, for the ATLAS level-2 trigger. In the following section, the test-bed will be described, starting with the C104 switch, followed by the traffic generating and intelligent nodes, and finally the configuration and the software of the test-bed.

The C104 Crossbar Switch The C104, see figure 3.4, is a asynchronous 32-way dynamic packet switch with DS-Links. In addition to its thirty-two data links, the C104

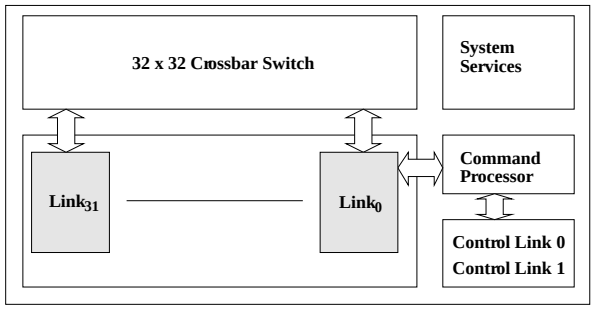


Figure 3.4: Diagram of the C104 switch.

has two control links. All links are DS-Links. Thirty-two packets can be transmitted through the switch at any one time.

Interval labelling is used to route packets through the switch. In this technique, each output link is assigned a range, or interval, of device labels corresponding to physical devices that are accessible via that link. Each device has an unique label associated to it. When a packet enters a C104, the device label contained in the header is compared to the device intervals. The output link, with a interval containing the device label, is chosen to route the packet out of the switch.

For some network configurations, there might be several routes a packet can take to reach its destination. The C104 switch implements grouped adaptive routing to allow a choice to be taken, as to which way to route the packet. Output links can be grouped, so that packets will be routed to the first free link in the group. Grouped links have the same interval labels.

The C104 switch uses worm-hole routing. With this routing technique, the switch takes a routing decision as soon as a packet header has entered the C104. This routing decision leads to the creation of a circuit in the switch which is closed as soon as the end-of-packet token leaves the switch. As a consequence of this routing technique, a packet may be in transit through several switches at the same time, see figure 3.5, thus minimising the transmit latency of the packet through the network. The header of a packet is deleted by the C104 switch before it leaves the switching network.

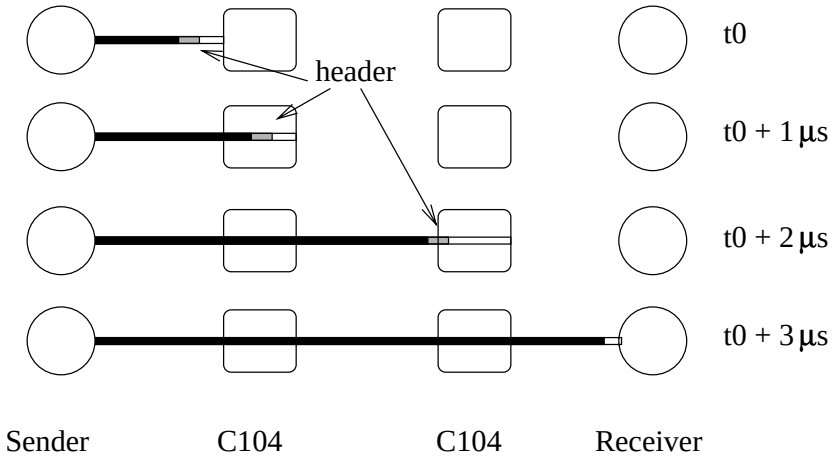


Figure 3.5: Worm-hole routing.

The performance of the C104 has been previously bench-marked [21]. The elapsed time for a packet to be routed through the C104 has been measured to be approximately one microsecond.

The Traffic Generating Node The traffic generating node is a DS-Link data source. It can be programmed to generate network traffic. The node consist of a controller, traffic pattern memory, and a DS-Link. A global clock exists on the test-bed such that all the nodes are synchronised.

The traffic pattern is stored in memory as a list of packet descriptors. Each descriptor consists of three entries:

- Delay
- Packet Length
- Header; one to four bytes

The traffic is pre-programmed into the on-board memory. The controller reads the traffic descriptors from the memory, and then feeds the packet to the DS-Link. When the node receives a packet the size of the packet is added to register storing the total volume of data received. The packet is then discarded. The information of the amount of data sent and received is kept, such that after a run the average receive and transmit rate can be obtained.

The number of packet descriptors that can be stored in memory depends on the number of bytes in the header, see table 3.1. When the controller has gone through all

Header length [Bytes]	No. of packet descriptors
1	8190
2	6552
3	5460
4	4680

Table 3.1: Number of packet descriptors.

the descriptors in memory it will start over from the top. This "wrap around" means that in order for the nodes to stay synchronised, they must all wrap around at the same time.

The delay values in the descriptors are relative to the time when the previous packet was scheduled to be sent. If a packet cannot be dispatched at its proper time, a timer is used to measure the delay from when the packet should have been sent, to when it is

sent. The delay of the next packet is reduced by the amount accumulated; if the result is negative the packet is dispatched immediately; if the result is positive the timer is reset to zero, and the packet is dispatched after the corrected time (delay - amount accumulated) has passed. See figure 3.6 for an example of the mechanism. This procedure enables

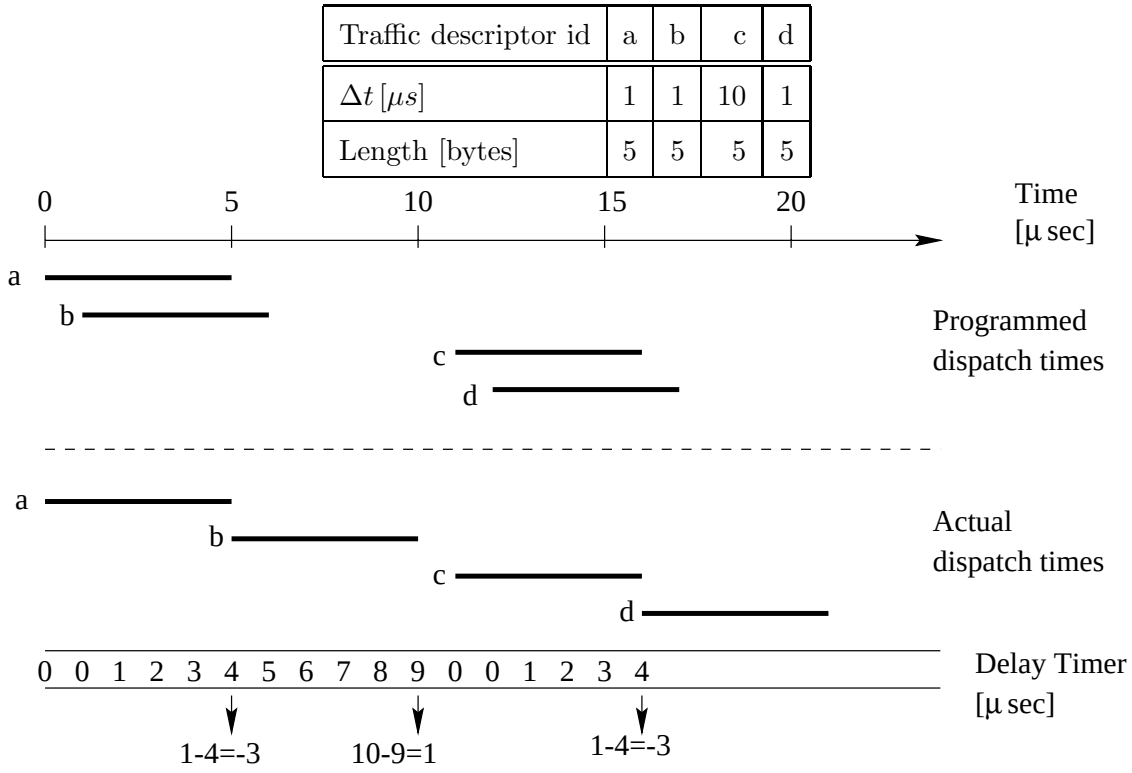


Figure 3.6: An example of how a traffic generating node tries to keep in synchronisation with the other traffic nodes. The link speed has for the figure been assumed to be 1 bytes a micro second.

the nodes to stay synchronous, as long as this timer does not reach its maximum value corresponding to 32.8 ms. The large capacity of this timer ensures that the nodes can deal with transient hot-spots in the network.

There is no maximum packet length and the effective link speed for messages above 400 bytes is 10.0 MBytes/s in the case of unidirectional use and 9.5 MBytes/s [22] for bidirectional use.

The Timing Node The purpose of the timing node is to measure the latency distribution across the network when the network is subjected to the traffic from the traffic

generating nodes.

The timing nodes work in pairs with one node sending trace packets to the other node. When the second timing node receives a trace packet, it calculates the latency from the time of the global clock and the time-stamp in the trace packet. The latency is stored in a latency histogram. Therefore the time evolution of the latency distribution is not available for later analysis.

Trace packets are sent at a regular but infrequent interval (to avoid disturbing the traffic pattern defined by the traffic nodes). The length of the trace packets, is constant and defined at start-up.

Configuration The configuration of the test-bed, as used for the emulation studies, is a 512 node Clos. See figure 3.7. Sixteen traffic generating nodes are connected to each

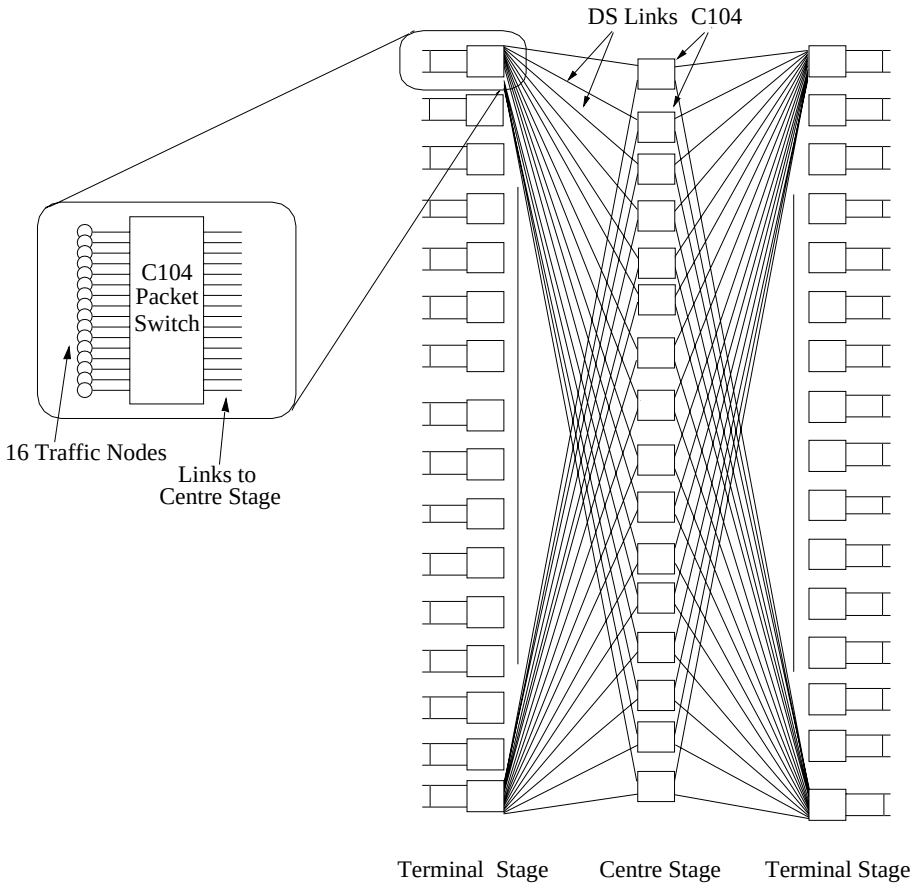


Figure 3.7: Diagram of a 512 node Clos network.

terminal stage switch. Each terminal stage switch is connected to each of the centre stage

switches by one link.

Two timing nodes has been placed on two different terminal stage switches. Macramé modules are constructed with sixteen traffic nodes connected to a C104 switch. To attach a timing node, one of the centre stage links has to be used. See figure 3.8. To remedy this

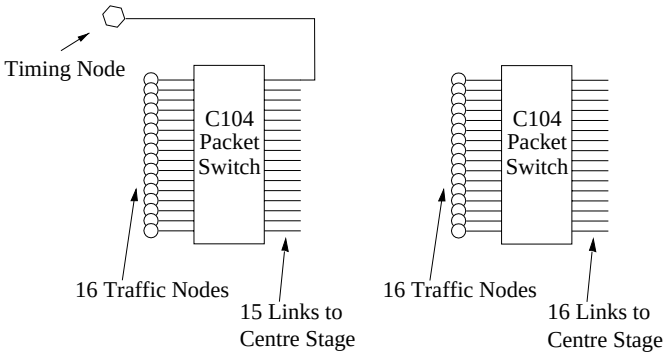


Figure 3.8: A terminal stage switch with and without a timing node.

situation, only fourteen of the traffic generating nodes are used during the emulation. This leaves full bandwidth through the Clos for all active nodes, traffic nodes and timing nodes alike.

3.4 The GPMIMD Machine

Computers can be divided into four categories [23] based on how many instructions and data streams they have. The traditional von Neumann machine, as we see it in personal computers etc., is single-instruction single-data (SISD); i.e. a CPU works on one set of data. The vector machine, is single-instruction multiple-data (SIMD). Here, the CPU takes a set of data and performs the same instructions on all of them at the same time, e.g. the Pentium MMX. The third category, multiple-instruction single-data (MISD) is mentioned here for completeness. This would involve multiple processors applying different instructions to a single input stream; this hypothetical possibility is generally deemed impractical. Finally, we have the multiple-instructions multiple-data (MIMD) machine where multiple instructions are performed in parallel on multiple data streams. These machines are usually referred to as parallel computers. The General Purpose Multiple-Instructions Multiple-Data machine, GPMIMD, is a member of this family.

This section will describe the GPMIMD machine used for the last part of the emula-

tion studies. First, a overview of the machine is given, then the processor units, T9000 transputers, will be described. Finally, the application software production environment will be illustrated.

The GPMIMD machine consists of sixty-four T9000 transputers which are fully inter connected via four independent Clos networks. Each Clos network consists of twelve C104s, four in the centre stage and eight in the final stage. There are thirty-two external DS-Links, which can be used to feed data into the machine. As a consequence of these external links, eight of the transputers are not placed directly on the three-stage Clos network. Rather, they have been placed behind one more C104, see figure 3.9. This

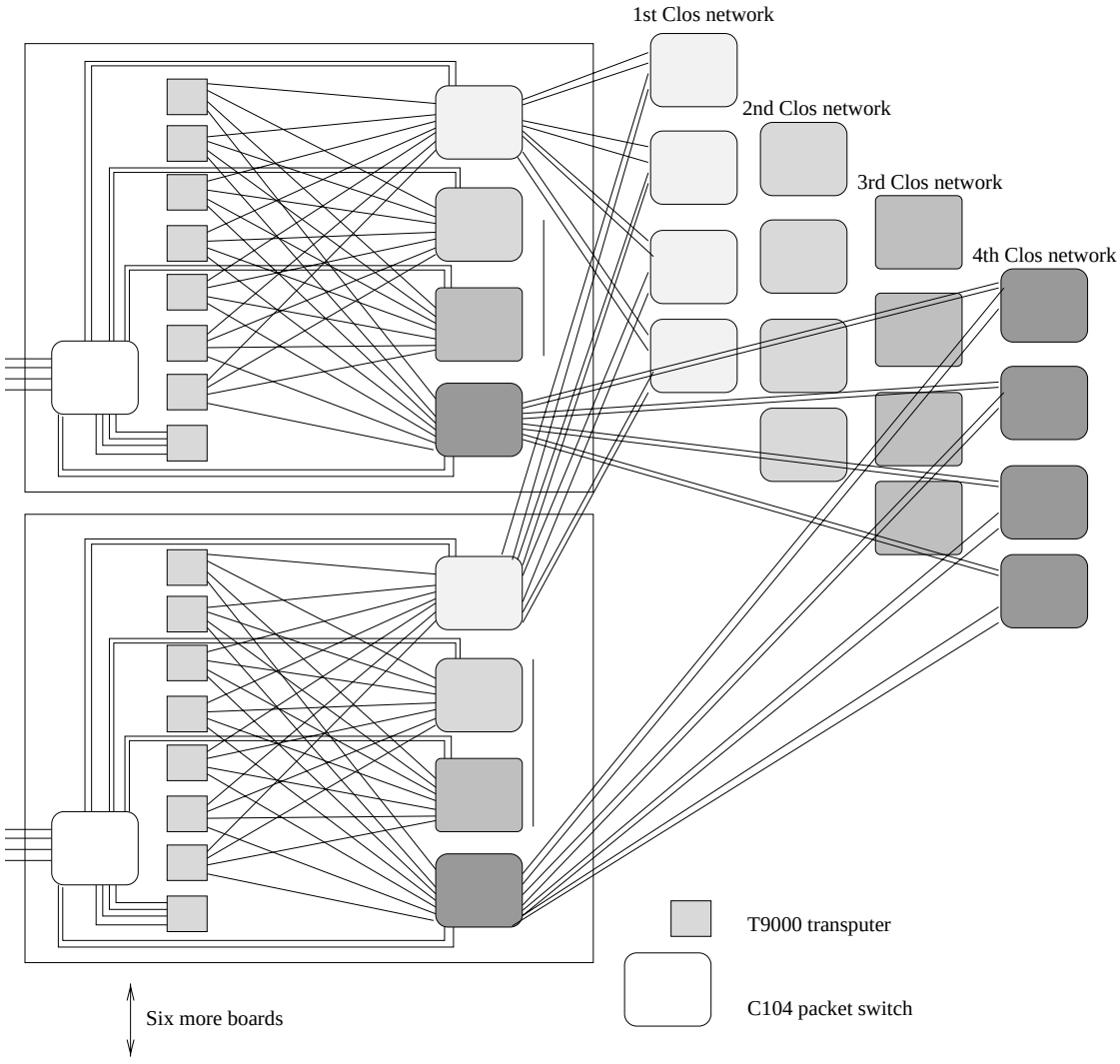


Figure 3.9: Sketch of the interconnectivity of the components in the GPMIMD machine.

means that, whereas fifty-six of the transputers have to send their packets through three C104 switches, eight (one on each board) must go through one more switch.

The T9000 Transputer The T9000 transputer is a complete microcomputer on a single VLSI chip. Figure 3.10 shows a diagram of the T9000 transputer.

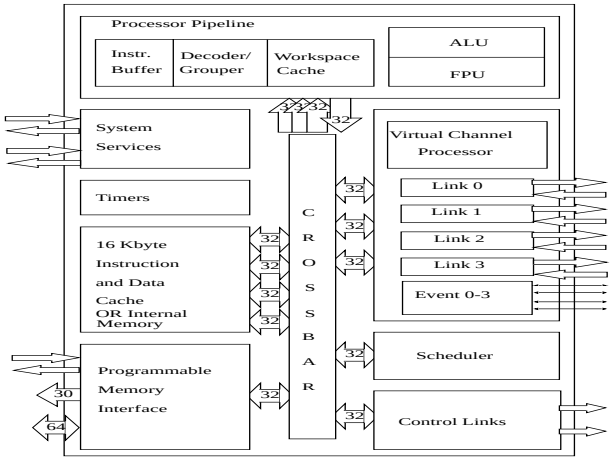


Figure 3.10: Diagram of the T9000 transputer.

The transputers main components are a processing unit, memory, the link unit, and a scheduler. The processor is 32-bit pipelined, and is supported by a 64-bit Floating Point Unit (FPU). The clock speed of the transputer is 20 MHz. The transputer has sixteen kilo-bytes of on-chip memory.

The T9000 transputers have four DS-Links allowing them to be connected together, either directly or through the C104 packet-routing switch. Two additional DS-Links are used for control. Each link has its own link engine, freeing the processor unit from dealing with the I/O. The link engine performs the link initialisation, link speed setting and error reporting as specified by the contents of the link registers in the configuration space. It is also responsible for implementing the four layered DS-Link protocol, i.e. converting bytes of data into packets, packets into tokens incl. parity bits etc., and finally tokens into data-strobe signals. The link engine also connects its link to the virtual channel processor.

For small transputer networks with header length equal to one byte, i.e. up to 256 nodes, the unidirectional link speed is 9.5 Mbytes/s and 8.7 Mbytes/s for bidirectional use [24]. The degradation of link speed from the raw DS-Link speed of 10 Mbytes/s is due

to an extra protocol layer handled by the transputer. This protocol layer ensures that each message is acknowledged, and that each message is split into packets of thirty-two bytes.

Virtual Channel Processor Communication between T9000 processes is done via virtual channels. Just as several processes can be running on a processor, with the appropriate scheduling, more communication channels can use the same physical link. The Virtual Channel Processor (VCP) of the T9000 transputer is a hardware communications processor which multiplexes the virtual channels onto the physical link. The VCP is also responsible for splitting a message into thirty-two bytes packets before they are given over to the link engine of the physical link.

The Hardware Scheduler The scheduler is in charge of handling the concurrent processes on the transputer. Only a few registers are used to store the state of the current process. This, as well as the scheduler being implemented in dedicated hardware, makes the process switching fast. The context switch time for the transputer has been measured to be 1.9 μs [21].

Application Software The manufacturer of the T9000 transputer, SGS-Thomson, provides a tool set which includes all the normal components needed to prepare software like compilers, both for Occam and a modified C language, a linker, and a special "make" program. They also provide more network specific tools that check and compile network descriptions. The tool set user guide [25] provides more information.

The Occam language [26] is a procedural language which is well suited for programming transputers. It supports inherently both parallelism and I/O in a very straightforward way. It does not, however, support dynamic data structures and, except for simple multidimensional arrays, no user-defined data structures exist. With regard to the I/O, there is extensive support for user-defined protocols.

When designing programs in Occam with multiple interacting processes, the subject of object-oriented programming immediately arises. The similarities between object-oriented design and Occam design is very simple to follow once the connection between an object and a process has been established. A process in Occam can be described by a finite state machine, as can an object in a object-oriented design. The flow diagrams

known from object-oriented design, can be used to document the communication between the processes in Occam. This paradigm has been used to document the emulation on the GPMIMD machine in section 7.1.

3.5 Controlling DS-Link Devices

The DS-Link devices, the T9000 transputers and the C104 packet switch, are controlled and programmed via two control links: a control-up and a control-down link. For example, the control links on the C104 switch allow access to the status of the network even though the normal data network may be in error. Commands can be sent as packets along the control network from the host controller to access status registers within the devices.

The control network can also be used by DS-Link devices to report errors back to the host controller. An error message is transmitted as a single packet which includes an error code to identify the error.

Control links can be connected in a daisy chain to each other or C104 switches can be used to form tree structured control networks. The thirty-two data links of the C104 are identical to the control links, hence control packets can be routed through the data links to form a control fan-out. Figure 3.11 shows a control fan-out structure which contains two separate daisy chains, each of the daisy chains is called a "subnetwork." The first C104 on the control chain is partitioned into a control partition and a data partition. The control partition is responsible for routing control packets into the correct subnetwork, the data partition can route packets from the main data network. The control down of the device is connected into the control partition, to allow control packets to flow into the data links of the switch. It is possible to use all the links of a C104 to perform control link routing.

A control fan-out provides more tolerance to faults within the control network and packets from the host controller have to pass through fewer devices than for a control network consisting of a single daisy chain.

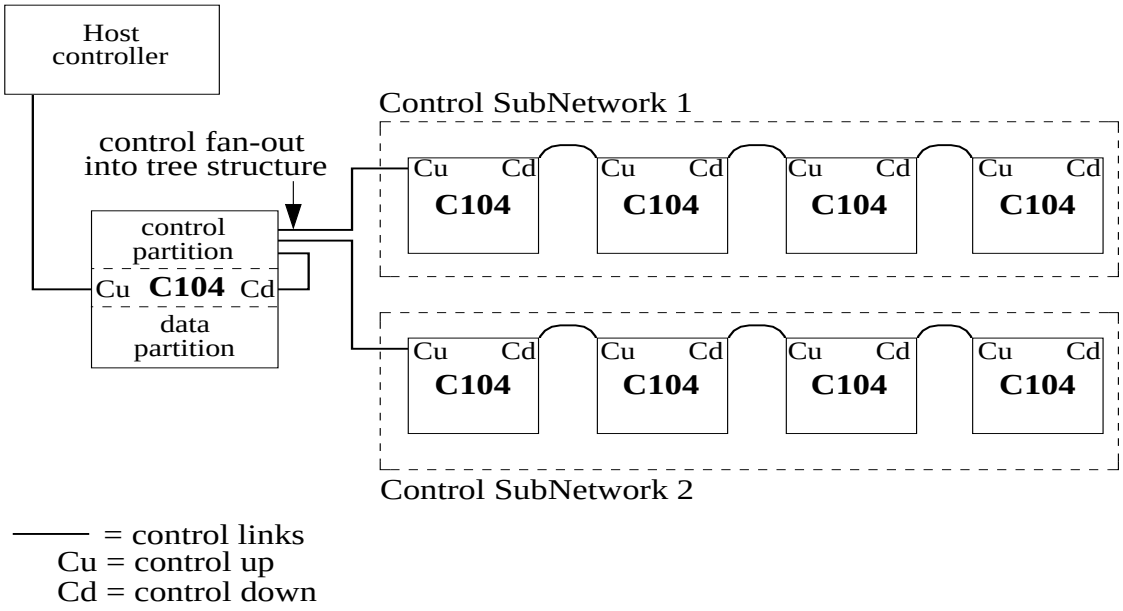


Figure 3.11: The control-chain fanout and the corresponding tree structure.

3.6 The Host Systems

To control the network, access is needed to the control network. This access is provided through the host system. The host system provides the interface between the user's computer and the reset, data and control link to the network. The control link gives access to the control network. The reset link enables the user to reset the whole system in one action. The data link provides access to the data network for down loading code into transputers.

Two host systems exist: An SGS-Thomson B103 Ethernet to DS-Link interface using a Sun workstation (SunOS or Solaris), and an FPGA based PCI-DS-Link interface using a PC running Linux [27]. The B103 system has been used for the emulation, and most of the Netprobe development, see chapter 4. The Linux/FPGA host is being used for developing the ARCHES HS-Link test-bed [28].

To download code, configure and test the network etc., two programs are available; Irun[3] and Netprobe. The basic part of Irun is a server, through which the user or another program can interact with the DS-Link network. On top of this server level, Irun has

[3]On the Linux/FPGA system this software package is known as the Control Software.

facilities for configuring the network and down loading transputer code. Netprobe is a test and diagnostic software package for DS-Link networks. Netprobe interacts with the network through Irun (or the Control Software in case of the Linux/FPGA host system). Netprobe is the work of the author and is detailed in the following chapter.

The two host systems are shown in figure 3.12.

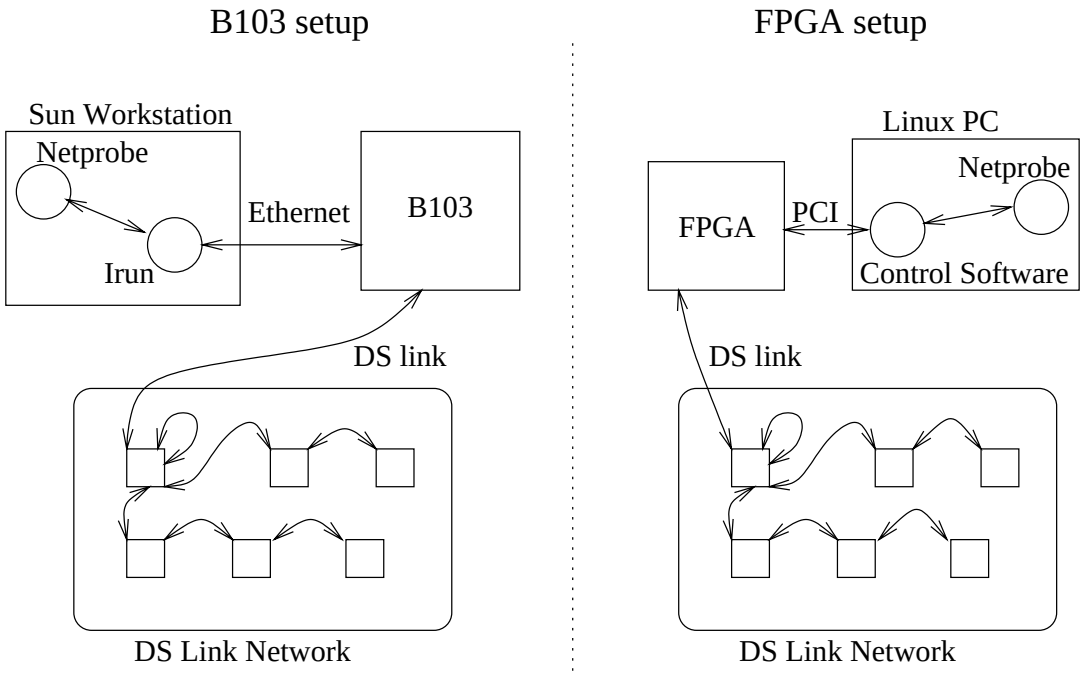


Figure 3.12: An overview of the host systems including software.

Chapter 4

Netprobe - A Data-Strobe Link Debugging Tool

Given the familiar scenario in high energy physics of thousands or more cables connected here and there in a way not logical to the eye, try asking the question: "How long would it be until someone noticed and repaired the damage if two cable ends were swapped around?" There are several possibilities: either the system would keep running, but data would be corrupted; or the system would halt immediately or some time later. In all case, it can be difficult to find the two cable ends which were swapped around without help from some debugging tool.

With DS-Link networks, and especially Macramé, the problem is similar. When wiring up Macramé to a 512 Clos network, there are about 500 cables, each with two ends which has to be plugged in by hand. Therefore, at the construction of the network, a tool for checking the configuration is invaluable.

Netprobe was designed to be able to help in situations like the one mentioned above, as well as debugging modules in the initial construction phase. The work presented in this chapter is the author's.

4.1 Overview

Netprobe provides the following functionality:

- access to the registers in all network devices.

- spy on the network i.e. report to the user the devices in a given physical network and their interconnection. The results may be reported in tabular form or in a network definition language (NDL) file. A NDL file describes a network in terms of its devices, the devices' link connections and how the user wants the configuration space of each individual device set. An example of an NDL file for a small network is given in appendix B.

- verify the network i.e. verify that a physical network matches a given configuration specified by the user. The configuration specified by the user may be in tabular form or in the NDL language.

- configure the network, i.e. start the network, and set the configuration space of each individual device according to the input file. The input file is either a NDL file or its binary version, the network initialisation file.

The DS-Link debug environment prior to the development of Netprobe consisted of two separate software programs; Clink and T9spy. The interface of Clink has been reused in Netprobe after changing the underlying variable structure. Most commands in Clink have been enhanced during the production of Netprobe, e.g. online help. A debugged version of the depth-first algorithm in T9spy has been reused in the standard spying routine in Netprobe. The procedures for spotting and verifying link connections are new, as well as the network verifying and configurations routines.

The following sections describe the spying, verifying and configuring commands within Netprobe. Following this, the performance is evaluated. Finally, a section describes the work done for supporting HS-Link devices. For a user manual on Netprobe, see the bibliography [29].

Debugging is very much an interactive process and Netprobe has been designed with this in mind. The available Netprobe commands can be split into three different classes: 1) general commands; to control access to network's host, get help, etc., 2) commands to access single devices and 3) commands for probing the whole DS-Link network. These areas are outlined in the following three sections and the individual commands are described briefly in appendix A.

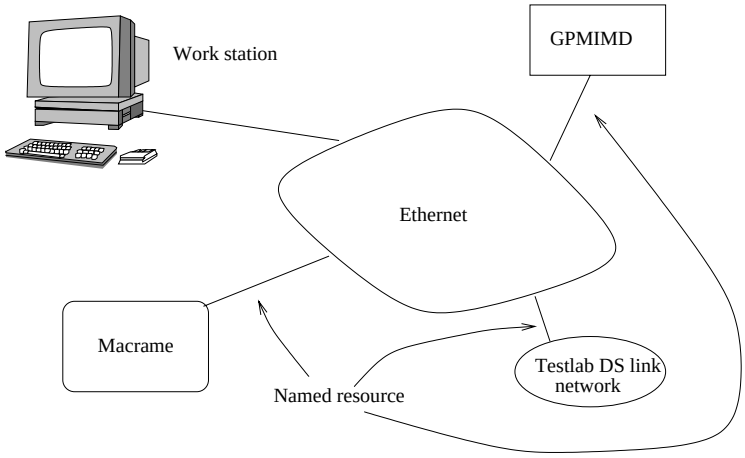


Figure 4.1: Accessing DS-Link networks from Netprobe.

General Commands A DS-Link network host is represented by a single named resource. To access a DS-Link network, the user must specify which host is to be accessed. This is performed using the connect command within Netprobe. It is possible to disconnect from one host and connect to another within the same Netprobe session. If requested, all the interactions Netprobe has with a DS-Link network can be logged to a file. This logging of information can be started, stopped and restarted throughout the Netprobe session. The level of information returned by the commands accessing the DS-Link network can be specified. By default, only error messages are reported; if requested, more information is given — for example, the content of all the packets sent into the network along the control chain can be obtained.

Local Commands The majority of Netprobe commands only access one DS-Link device. These include the ability to start, reset and identify a device. Specifically for the T9000 Transputer, there are commands to boot, reboot, stop and start the processor. It is also possible to read and write to the memory of the T9000. There are also commands to access the configuration registers of any DS-Link device in the control chain. This includes the ability to dump a specified set of these registers using a single command.

Global Commands Assuming Netprobe is connected to a network, the user can obtain a list of the devices known to Netprobe. Commands also exist to perform a hard reset of the network and to halt all the T9000 processors. The user can run a T9000 application

on the processors in the network or just configure the network. Another set of global commands has been created to spy on and verify the configuration of the network.

4.2 General Principles behind Spying and Verifying

To understand how the spying on networks, and later, how verifying of networks is performed, a few principles of a general DS-Link device must be explained.

In terms of spying or verifying the generalised DS-Link device consists of:

- Two control DS-Links, see section 3.5.
- A number of data DS-Links. Four for the T9000 transputer and thirty-two for the C104 packet switch.
- A configuration space containing the following registers:
  - DeviceId (read only), specifies the device identification code, i.e. which type of device this is e.g. a C104 switch.
  - DeviceRevision (read only), specifies the revision of the device
  - ErrorCode (read only), specifies the error state of the device. There are four general error codes, including "no errors" and additionally four error codes per data link, e.g. "Parity or disconnect error on link 2."
  - The DSLinkPLL which is used to set the master clock link speed.
  - The ConfigComplete, which is used to signal to the device that it can start operation since the configuration is complete.

A generalised DS-Link consists of:

- A command register. The following commands can be issued: reset and start link, send parity error, and reset output. A reset output command sets the link output low.
- A mode register. Three bits of this register are used to specify the speed of this link. This is done by specifying by how much the master clock link speed should be divided. Three more bits are used to:
  - Localize errors, i.e. when set do not report errors to the control unit.

  - Disable data alignment token, i.e. idles. This can be used to provoke disconnect errors.
  - Ignore disconnects, when set the link will ignore disconnect errors on the link. This is useful when resetting both ends of a link.

- A status register. The following status bits exist:
  - Error, flags that a parity or disconnect error has occurred on the link.
  - Started, flags that the link is started, i.e. sending idles.
  - Reset output completed, flags that the reset of the output on the link has completed.
  - Parity error, flags that a parity error has occurred.
  - Disconnect error, flags that a disconnect error has occurred.
  - Token received, flags that a token has been received.

Furthermore, a C104 data link consist of interval registers used for routing packets. See also figure 4.2, showing a sketch of a general DS-Link device.

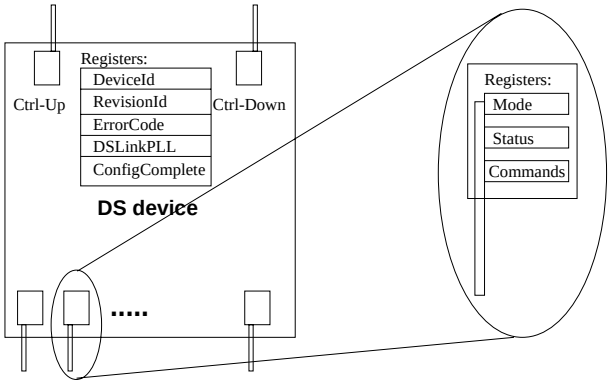


Figure 4.2: A general DS-Link device.

Knowing how the link status register interact with the commands issued to the link is critical to understanding the spying.

- Starting a link causes the data alignment tokens, i.e. idles, to be sent over the link. This, in turn, causes the receiving end of the link to set the status bit “Token Received” in the status register.

- Forcing a parity error on a link, causes tokens with the wrong parity to be sent. The receiving link will raise the error flag and the parity error flag of the status registers, it will then disconnect. This causes the disconnect error flag on both sides to be raised.

The control-up links starts automatically upon receiving a token. Without this behaviour, the host would not be able to get in contact with the next device. No other DS-Link starts itself in this way. It is therefore possible to identify data links used as part of the control chain. The identification process of links which leads to a control subnetwork is detailed in the flow diagram in figure 4.3.

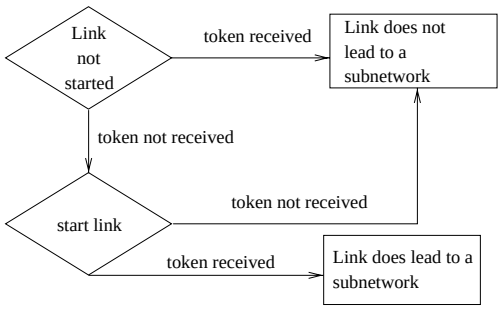


Figure 4.3: Identifying a data link leading to a subnetwork.

Back-Checking for Connections When looking for connections between two data links, we use the following strategy to give us confidence that there is a physical connection between the links in question. After starting a link (let us call it link A), we find a link which now (and not before) has received tokens (let us call it link B). If link B has not already been paired with a link, we do the following "back check": reset both links, start link B and check that link A receives tokens, start link A and check that link B receives tokens. After this test, the link is running normally; an error is then forced at each end of the link and the spy verifies that the corresponding disconnect errors are discovered at the appropriate end of the connection. The back checking scheme is sketched in figure 4.4.

Having given the principles behind the spying on and verifying of DS-Link networks, the next sections describe the algorithms used to perform the spying and verifying.

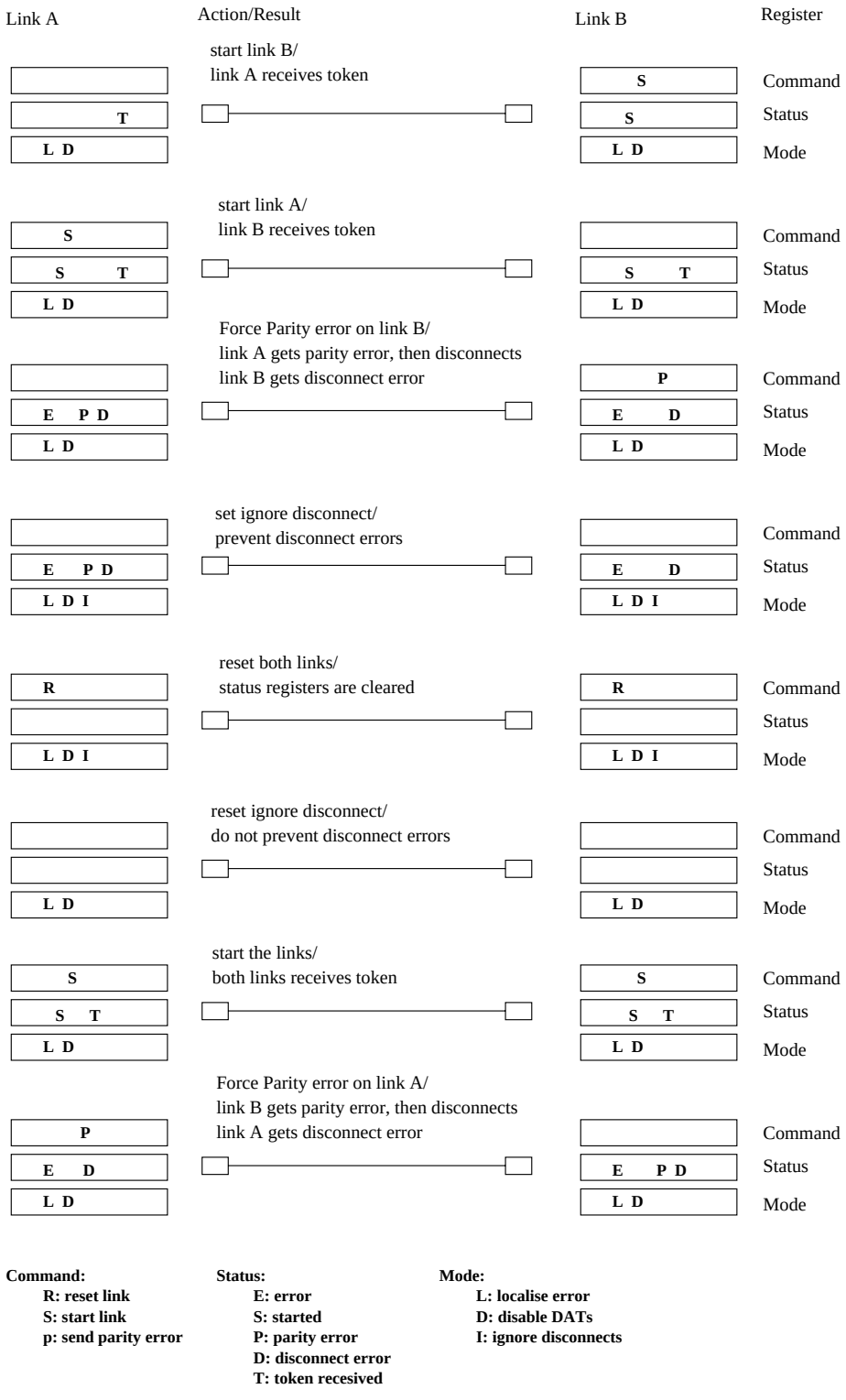


Figure 4.4: Back-checking for a connection.

4.3 Spying on a DS-Link Network

The spy command reports to the user the devices in a given physical network and their interconnection.

The default algorithm used to spy on the network is a two step algorithm. First, all the devices in the DS control chain are started in depth-first order, see figure 4.5; this is

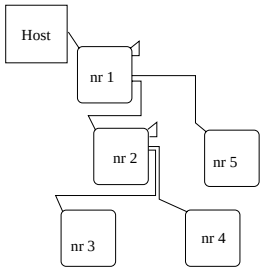


Figure 4.5: Depth first ordered device numbering.

performed by the `WORM` procedure shown in figure 4.7. The `WORM` procedure also uses the `ADD NEW DEVICE` procedure which essentially starts the next device in the control chain and all its links. After all devices in the network are discovered and started, the data link connections in the network are investigated using the `FIND CONNECTIONS` procedure, also shown in figure 4.7.

This default algorithm execution time scales as $N \times L$, where L is the number of links within a system and N is the number of devices in a system; i.e. every link must be started and then each device is checked for a status change.

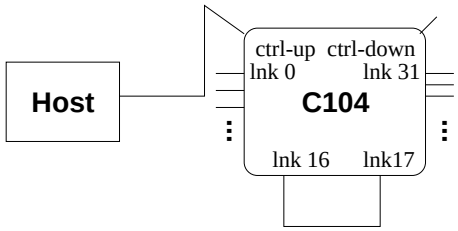


Figure 4.6: Small DS-Link network, consisting of a single C104 switch.

Applying the algorithm to the network in figure 4.6 will result in the following actions. The control link to the first device will be started so that the device and revision registers can be read. Knowing that this device is a C104, all its thirty-two data links are started

and checked for connections to a control subnetwork, see figure 4.3 — none of the data links will be identified as being part of a control fanout. The control-down link of the C104 will be started and by reading and comparing link status registers Netprobe will conclude that no more devices exist in the network. One by one, the data links are then asked to send idles with wrong parity. After link number 16 has done this the device error register will contain an error saying that link number 17 has received a parity error. As a consequence of this, the "back-check" algorithm, see figure 4.4, is started on those two links. The "back-check" will succeed, and link no. 16 will be known to be connected to link no. 17.

The Alternative Algorithm An alternative algorithm can be used to spy on the network. In this algorithm, the network devices and their interconnections are explored simultaneously. A `WORM` procedure starts each device in the network, for each device it starts all links and checks for a status change on any link in the network. The algorithm execution time scales as L^2, i.e. every link must be started and then every link in the network must be checked for a status change. In general, this alternative algorithm is slower since N << L, however, on small (less than approximately 100 links total) systems it can be faster than the default algorithm; see also the performance discussion section 4.6. This alternative algorithm is shown in figure 4.8.

Applying this algorithm to the network from figure 4.6 gives the following result. First, the device is started and identified like in the standard algorithm. Then, all data links are started one by one, and all known link status registers are read to check for a possible control fanout or data link connection. Netprobe will, in this way, find the connection between links no. 16 and no. 17. Going on to explore the rest of the control network, the control-down link of the C104 is started. Since this causes no change in any link status registers, Netprobe concludes that the network consists of only one device.

Spy Result Format Having successfully discovered the topology and interconnection of the network, the spy outputs the information in tabular form. If the spying had to be aborted due to an irrecoverable error, e.g. a control link not able to start, the output will only contain the part of the network which could be spied on. In this case, the spy function will also report the problems it found.

By default, the spy command produces a table of the format shown in table 4.1. In

```
SPY
  WORM
  FIND CONNECTIONS
---------------------------
WORM
  ADD NEW DEVICE
  Start the control-down link
  If this caused a status change to a link known to the spy
                 (i.e.  if control fan-out is present)
    For all links leading to subnetworks (connected to control-up links)
      If current device is an C104
        Configure routing values (for control packets) in current C104
        Run WORM on subnetwork
      If current device is a C104, reconfigure routing values
    End for
  Else if the control-down link has received a token
                 (i.e.  it is connected to a control-up link)
    Run WORM on device connected to the control-down link
  End if
---------------------------
ADD NEW DEVICE
  Start next device on control chain
  Identify device by reading the DeviceId and DeviceRevision registers
  For all data links
    Start link
    Test if link leads to a subnetwork
  End for
---------------------------
FIND CONNECTIONS
  For all devices
    For all links
      Put link into error
      Read error register on all devices to find corresponding
                 error on other end of link
      Verify possible connections in both directions (back-check)
    End for
  End for
```

Figure 4.7: Standard spy algorithm.

```
WORM
  ADD NEW DEVICE
  Start the control-down link
  If this caused a status change to a link known to the spy
              (i.e.  control fan-out is present)
    For all links connected to subnetworks (connected to control-up links)
      If current device is a C104
        Configure routing values inside current C104
        Run WORM on subnetwork
      If device is a C104, reconfigure it
    End for
  Else if the control-down link has received a token
              (i.e.  it is connected to a control-up link)
    Run WORM on device connected to control-down link
  End if
---------------------------
ADD NEW DEVICE
  Start next device on control chain
  Identify device by reading the DeviceId and DeviceRevision registers
  For all data links
    Start link
    If this caused a status change to a link known
              to the spy there is a possible connection
      Check that there is a connection in both directions (back-check),
              if yes, record the connection
  End for
```

Figure 4.8: Alternative spy algorithm.

cases where information is needed, the "View All Status" table can be requested, see table 4.2. The spy table contains information on all devices discovered in the network control chain. The device type and revision are displayed and the link's connection is displayed, i.e. whether they are empty "....", connected to the host "HOST" or connected to another link "0:17" (meaning this link is connected to link 17 on device 0). Ctrl0 and Ctrl1 refer to the control-up link and control-down link, respectively.

SPY							
Device	Revision	Ctrl0	Ctrl1	Link0	Link1	Link2	Link3
0: C104	BetaB02	HOST	....	....	....	....	....
+ 4				....	....	....	....
+ 8				....	....	....	....
+ 12				....	....	....	....
+ 16				0:17	0:16	....	....
+ 20				....	....	....	....
+ 24				....	....	....	....
+ 28				....	....	....	....

Table 4.1: Example tabular output from spy.

View All Status								
Device	Revision	Error	Ctrl0	Ctrl1	Link0	Link1	Link2	Link3
0: C104	BetaB02	None	-S---T	-S----	-S----	-S----	-S----	-S----
+ 4					-S----	-S----	-S----	-S----
+ 8					-S----	-S----	-S----	-S----
+ 12					-S----	-S----	-S----	-S----
+ 16					-S---T	-S---T	-S----	-S----
+ 20					-S----	-S----	-S----	-S----
+ 24					-S----	-S----	-S----	-S----
+ 28					-S----	-S----	-S----	-S----

Table 4.2: Example of a "View All Status" table from spy.

The "View All Status," table 4.2, shows the status of the devices and the links. The Device and Revision columns are the same as for the spy table. The Error column displays the last read value of the device error register. Due to the way the connections are checked, it is quite likely that some, or all, of these have an error listed.

It is possible to have the spy produce a network description language (NDL) file representing the network. Labelling information for the data network cannot be obtained from spying on the network. This information must be added by the user before the NDL file can be used e.g. to configure the network.

The speed of the data and control link while they are being spied on can be specified. The default link speed is 10 Mbit/s.

4.4 Verifying a DS-Link Network

The difference between spying on and verifying a network can be characterised as the difference between entering a maze with the aim of making a map versus venturing inside with a pre-existing map with the intention of verifying it. This conceptual difference results in a substantial improvement in performance, see section 4.6.

The verify command takes a network description (NDL) file or a previous spy output table and verifies that a physical network matches the description contained within the NDL file or the previous spy output table.

In order for the verify command to be able to parse the network description, it must conform to either the NDL format or to the format for the spy tables.

Parsing a spy table is rather trivial compared to parsing an NDL file. Here, a more complex parsing/interpreting algorithm needed to be implemented in order to handle the Network Description Language. This language supports variables, procedures and multidimensional arrays like most high-level programming languages. Such a parser existed already at Sintef, one of our partners in ARCHES. We were given access to their source code for this. It was written in C++ while Netprobe has been kept in C. To ease future maintenance, the parser was rewritten in C.

Before a physical network can be verified, the network description is checked for consistency (both NDL input and spy table input). This includes:

- a check that all devices have a consistent type, revision and number of links, e.g. that a device defined to be a C104 has thirty-two links.
- a single control-up link must be connected to the host control link.
- if there are T9000s in the network, then the host data link must be connected.

- all control-up links must be connected.
- a link must be connected in both directions (this is automatically ensured in the NDL file).

The verify command also explores the control network in depth-first ordering of devices in the same way as the spy command. If the verify command confirms that the input is consistent it starts to explore the network, one device at a time. The device is started, identified, configured with routing information if required and all connected links are started. Once the entire network has been started the suspected connections are back checked in the same manner as used for the spy algorithms.

VERIFIER							
Device	Revision	Ctrl0	Ctrl1	Link0	Link1	Link2	Link3	OK
0:T9000	BetaB02	HOST	....	....	0: 2	0: 1	HOST	

Table 4.3: Successfully verified network.

The verify table is basically the spy table with some additional information. This extra information informs the user whether or not the information in the input file was successfully verified. Table 4.3 shows the scenario where everything in the input configuration was found to be present in the physical network. In table 4.4, we see an example

VERIFIER							
Device	Revision	Ctrl0	Ctrl1	Link0	Link1	Link2	Link3	OK
0:T9000	BetaB02	HOST	....	....	-0: 2	-0: 1	HOST	*

Table 4.4: A missing connection.

of a network which contains the correct device, but the connection has not been found. This has been marked with a minus sign in front of the connections. The OK column has been marked with an asterisk "*" indicating that this device was not identical to the input configuration. The values in the verify table are those provided by the user as an input configuration, apart from the device and revision column, which contain the

actual device and revision obtained from the network. Asterisk signs are used where the physical network does not match the input configuration. If the input format provided is an NDL file the output from the verify will still be the format shown within table 4.3.

VERIFIER							
Device	Revision	Ctrl0	Ctrl1	Link0	Link1	Link2	Link3	OK
* 0: C104*	* BetaB02	* HOST	* 1:C0	- HOST	- 1: 0	- 2: 0	- 1: 1	*
* 1: ??? *	* ???	* 0:C1	* 1:31	- 0: 1	- 0: 3	*	*	*
+ 4				*	*	*	*	
+ 8				*	*	*	*	
+ 12				*	*	*	*	
+ 16				- 2: 1	- 2: 3	*	*	
+ 20				*	*	*	*	
+ 24				*	*	*	*	
+ 28				- EDGE	*	- 2:C0	- 1:C1	
* 2: ??? *	* ???	* 1:30	*	- 0: 2	- 1:16	*	- 1:17	*

Table 4.5: No devices are verified correctly.

Table 4.5 gives an example where nothing matched the input configuration exactly. Looking at the first two columns one sees that only the first device was identified, even though the asterisk shows that it was not found to be a T9000 (only four links are listed in the table) but a C104 revision BetaB02. The verify could not identify any of the other devices. The device which has been started was not what we expected, and no other devices could be started; no verification could be done of any link connections.

4.5 Configuring a DS-Link Network

The configure command allows the user to configure a pure C104 network described by an NDL description or a DS-Link network (including T9000s) described by a Network Initialisation File (NIF). A NIF file is a binary representation of an NDL file which contains the same information. With the Irun software, configuration can be performed via a NIF file, Netprobe uses the same algorithm to configure from this format. The configure command using NDL is new and gives useful information on the state of devices in the case of an error configuring the network; no errors are reported when configuring from a NIF file. This is not a limitation since NIF is produced from NDL, so any user who produces a NIF file will already have an NDL description of the network. The motivation

for not enhancing the configuration from NDL to be able to configure T9000 networks, was that the T9000 transputer had by then been phased out of production.

A tabular spy output representation of the network cannot be used to configure the network. This is because it does not contain the labelling information required by the switches in the network. In general a user produces a description of the network in NDL. This can be written by the user or generated automatically by the Netprobe tool, see section 4.3.

If the configuration file is an NDL file, the configure command starts devices in the network in a similar fashion to the verify command, i.e. depth-first through the tree control structure. For the configure, all configuration registers must be initialised, which is not the case for the verify procedure. The following list provides further details on the information and commands C104 switches receive during configuration:

- the C104 is started and identified,
- the link speed is set,
- all registers containing labelling information are cleared,
- all labelling information is entered into the appropriate registers,
- all information required for adaptive routing and universal routing is entered into the appropriate registers,
- a configuration complete command is sent to the C104 which signifies that the configuration registers are configured,
- all connected links are started.

Having configured and started the network, all the data links which are connected are checked for the following: that they have started correctly, have received tokens, and are not in error.

4.6 Performance and Testing

All Netprobe functions have been tested on multiple large networks, including the 122 device (58 C104s and 64 T9000s) GPMIMD machine and the 1024 node Macramé C104

network (65 C104s). Its functionality has also been adapted to meet the requirements of the users of these large systems.

Performance measurements have been made which allow a comparison of different algorithms (spy versions, verify and network configuration) and host interfaces. Results are shown in table 4.6 and 4.7 for the GPMIMD machine and a test system of two C104s

	Std. Spy	Alt. Spy	Verify	Configure from NDL
Orig. B103	11341 sec	15927 sec	384 sec	NA
New B103	1494 sec	2140 sec	334 sec	NA
Linux/FPGA	NA	NA	NA	NA

Table 4.6: Performance measurements on the GPMIMD machine.

	Std. Spy	Alt. Spy	Verify	Configure from NDL
Orig. B103	70 sec	52 sec	7 sec	8 sec
New B103	20 sec	15 sec	6 sec	7 sec
Linux/FPGA	4 sec	4 sec	3 sec	4 sec

Table 4.7: Performance measurements on two C104s switches.

respectively. Configure from NDL and the FPGA host do not support T9000s and hence the GPMIMD machine, these are shown as NA (not applicable) in the table.

Results for the B103 system are for two versions of the software running on the Sun workstation and the B103. The new version, also the work of the author, optimises performance by buffering and grouping commands before they are sent over Ethernet. The difference in results indicate the severe bottleneck introduced by sending individual commands to the network over Ethernet.

The results from the two C104 network shows that for small systems the performance of the alternative spy algorithm is competitive.

4.7 High Speed Link Support

After finishing the work on Netprobe, other people at CERN were designing and building a test-bed similar to the Macramé test-bed. This test-bed [28] for ARCHES is a thirty-two

node machine being built using High-Speed (HS) link technology. The HS-Link technology is also part of the IEEE 1355 standard and uses a link speed of one Gbit/sec. This technology is of particular interest for building scalable Ethernet switches and routers. The main aim of this test-bed is to run different traffic patterns, in much the same way as on Macramé, but with eight to ten times higher link speeds.

A new router, the RCube [30], has been developed for switching these links. The issue of how to deal with configuring and testing this test-bed arose. The RCubes used for the test-bed have been equipped with a controller using DS control links. These boards can be connected via their control links to any DS-Link control network just like the C104 and the T9000. It was therefore feasible, as well as interesting, to enhance Netprobe to handle this router. The basic enhancement was to allow the Netprobe to recognise the content of the DeviceId and DeviceRevision registers of the RCube as being an RCube. The primary aim, though, was to enable an automatic configuration from an NDL file since the Irun software, written prior to the existence of the RCube, does not support RCube configuration. This has been done and includes support for reading from and writing to the registers of this device.

4.8 Conclusion

It can be concluded that the work on Netprobe dramatically improved the debugging facilities for DS-Link networks, both in the sense of increased functionality, improved performance and user-friendliness — the last having been particularly improved by collecting the debugging facilities inside one program and providing online help-pages.

Netprobe has been crucial in the development and use of the Macramé 1024 node DS-Link network. With the extension to the HS-Link RCube device, Netprobe is playing a similar role in the ARCHES test-bed work.

The general usability of Netprobe has also been positively demonstrated by its use in the Demonstrator B's DS-Link vertical slice [15] in ATLAS and in a system consisting of some twenty T9000 transputers and two C104 switches, used as the Level 2 trigger in L3 [31, 32, 33], a large general purpose high energy physics experiment at the CERN LEP accelerator. For the maintenance of the L3 trigger, which runs continuously, Netprobe eased significantly problem-spotting in comparison to earlier debug facilities.

Chapter 5

Inputs to the Emulation

In the two previous chapters, the emphasis has been on the DS-Link technology which is used for the emulation. Before describing the emulation procedures and results, the input to the emulation is discussed.

The main purpose of this chapter is to describe the construction of data files used as input to the Macramé Test-bed and the GPMIMD machine which match the expected data distribution both spatial and temporal in ATLAS.

The main topics of this chapter are the trigger menus, the benchmarks of the trigger algorithms, the configuration of the sub-detectors and, finally, the simulated event scenarios which were used for driving the emulations. The author's work was defining the content of these event scenarios.

The input parameters for the emulation originate from many subgroups of the ATLAS experiment. In preparation for the different modelling activities during the "demonstrator project," two internal ATLAS DAQ notes [34, 35] were prepared. Their basic aim, was to collect all the important numbers and so establish a set of well-defined assumptions from which to work. In the following chapter, the parameters used in the emulation are summarised.

5.1 Trigger Menus

A very important constraint on the level-2 trigger is that it should be very flexible in terms of the physics processes it selects. This flexibility is envisaged to be implemented by having, from run to run, a configurable trigger menu. The trigger menu consists of a

set of constraints on the trigger items. Sometimes this flexibility is referred to as "the programmable trigger."

The trigger item is an abstraction of the physics we are interested in saving to tape for later analysis. One trigger item, for instance, a μ with a transverse momentum (p_t) greater than twenty GeV/c, could correspond to some very interesting physics, e.g., $H \rightarrow WW \rightarrow l\nu jj$. A trigger menu consists of a list of signatures, also known as menu items, covering the physics programme in ATLAS.

The first level trigger menu itemises the conditions for acceptance of events at the first level trigger and represents the distribution of trigger types expected. Thus, the trigger menu is the physics input to the second level trigger. The estimated total event rate from the first level trigger at low luminosity is 37.5 kHz. To allow for uncertainty in the level-1 rate estimates and to allow increased rates for interesting channels, the level-2 trigger must be designed for a 75 kHz input rate and be scalable to 100 kHz [36]. The trigger rate comes mainly from well known low-p_t processes rather than from the interesting physics processes. The signature of the physics processes determines the trigger menu, but not the distribution or frequency of data which the trigger sees.

In the following the trigger items and trigger menus which are currently being considered for the final system will be described.

The trigger community anticipates using the following five components in the trigger items: jets, electro-magnetic showers, muons, the missing transverse energy and the energy sum from the first level trigger.

The ATLAS detector will contain four main subsystems, the magnet, the calorimeter, the inner detector and the muon spectrometer. The calorimeter consists of two subdetectors: the electro-magnetic calorimeter (ECAL) and the hadron calorimeter (HCAL). Of the inner detectors, data from the Transition Radiation Tracker (TRT) and the Semi-Conductor Tracking (SCT) detectors are available to the level-2 trigger. From the muon spectrometer, data from the Monitored Drift Tube (MDT) chambers and from the Resistive Plate Chambers (RPC) are available. The data from the MDT are high precision, while the RPC provides more coarse information mainly used for triggering purposes.

The second level trigger uses full-granularity, full-precision data from the detectors. In order to cut down on the load of the trigger networks as well as processors, only data

from regions which the first level trigger found to contain interesting information will be transferred and processed. The main function of the second level trigger, therefore, becomes verifying the level-1 triggers.

Upon accepting an event the level-1 trigger provides information to the second level trigger's supervisor. This information includes information on the position in η and ϕ of the Regions of Interest (RoIs) and the level-1 selection criteria. Based on this information the supervisor can determine the type of the RoIs and which read-out buffers (ROBs) contain interesting data. For certain cases of the RoI type, the level-2 supervisor decides not to read out all the sub-detectors. For the jet RoI type, only the ROBs in the RoI region for the calorimeter are read out; for the electro-magnetic showers, all ROBs in RoI region except for those in the muon detector are read. If the RoI type is that of a muon, all ROBs in the cone specified by the RoI are read, see figure 5.1.

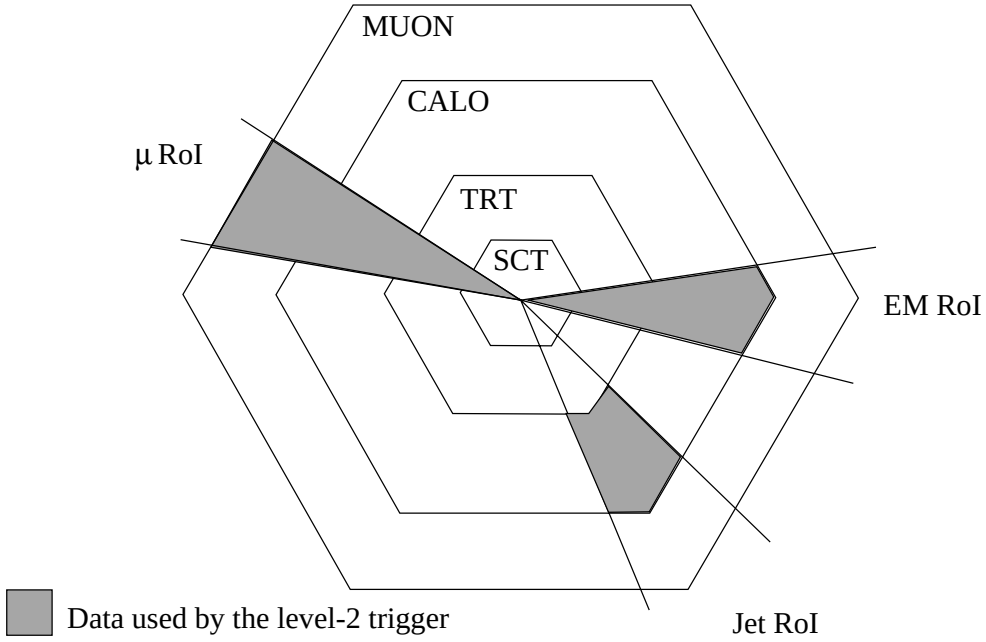


Figure 5.1: Correspondence between RoI types and sub-detectors.

The LHC will run at a range of luminosities, starting around $10^{33}\text{cm}^{-2}\text{s}^{-1}$ and building towards the design luminosity of $10^{34}\text{cm}^{-2}\text{s}^{-1}$. The beam energy remains the same, hence the high-p_t physics remains constant. The number of p-p interactions per bunch crossing rises with luminosity so the events look different due to the "pile-up" of several

low transverse momentum events. This results in higher occupancies in the tracking detector. At the low intensity runs in the start, ATLAS will concentrate on B-physics, while during the high intensity runs the focus will be on high-p_t physics. This means that we must have, at least, two trigger menus.

The trigger menus can be expressed in two different ways, inclusively and exclusively. The inclusive form is more compact and readable, but for the various simulation/emulation studies, they do not contain enough information. To emulate the trigger, we must know how many RoIs there are and their types in order to find the data rates through the network and to estimate the processing power needed. The trigger menus can be configured as either "minimal" or "extended." In the first case, only the level-1 trigger RoIs are included; in the extended case, the secondary RoIs are also used. For the emulation studies, the high-luminosity, extended menu have been used. A full listing of this trigger menu has been included in appendix C.

Another important set of assumptions for the emulation is the selection strategy used by the trigger. The selection strategy and the order of processing will be strongly linked in any efficient trigger system. For architecture B, it is natural not to do the selection until all features of all the regions of interest have been determined (in parallel). In the emulation, this selection strategy has been used.

5.2 Algorithm Benchmarks

During the emulation, no real physics algorithms run in the processors, but instead the processors wait for an amount of time which corresponds to the amount of CPU time which the physics algorithm would take; therefore the time involved in the individual processing steps is an important input parameter. These are taken from algorithm benchmarks as described below; see table 5.1 for a summary. The "physics" algorithm execution time in the global processor has been estimated as 150 μs per event.

In architecture B, the different feature extraction algorithms are executed on processors where there is only access to a specific type of detector data. Depending on the RoI-type given by the level-1 trigger, some detector within the RoI might be disregarded altogether, as mentioned earlier.

RoI type	Sub-detector			
	SCT	TRT	CALO	MUON
Jet	—	—	100 μs	—
Em	500 μs	310 μs	100 μs	—
μ	500 μs	590 μs	100 μs	100 μs

Table 5.1: Extrapolated algorithm times based on 500 MHz processors

The principal features of the algorithms are given here. The track finding used to extract features from the silicon tracker and transition radiation tracker is performed by using histograming for the initial pattern recognition. Then, a fit is done for all the combinations of hits consistent with the bins above threshold in the histogram. The feature extraction for the calorimeter is a calculation of the size and position of the cluster area. The feature extraction for the muon spectrometer is performed by combining the hits into a track.

5.3 Configuration

The parametrisation of the sub-detector data used by the level-2 trigger will be summarised first in this section. Following this, the trigger system itself will be summarised.

Detector System	Sub-detector	# ROBs
Inner Detector	SCT	256
	TRT	512
Calorimeter	ECAL	432
	HCAL	48
Muon Spectrometer	MDT	192
	RPC	22
Total:		1462

Table 5.2: The number of ROBs for the different sub-detectors.

Upon an accept from the level-1 trigger, each sub-detector feeds the read-out buffers (ROB) with data via the front-end electronics. The number of buffers for the individual sub-detectors is detailed in table 5.2. The RoI information for an event is expected to be contained in about 200 bytes. All the data in all the ROBs are available to the level-2 trigger. The amount of data per buffer depends not only on the detector but also on the type of RoI; see table 5.3. From each local processor participating in the event, 150 Bytes, which includes the feature information, are sent to the global processor.

Detector System	Sub-detector	Jet RoI	em RoI	μ RoI
Inner Detector	SCT	—	1000	1000
	TRT	—	740	740
Calorimeter	ECAL	60	1300	1300
	HCAL	890	890	890
Muon Spectrometer	MDT	—	—	600
	RPC	—	—	100

Table 5.3: Size in bytes of data in a ROB for a given RoI type.

Having gone through the most important detector parameters, from the point of view of my measurements of the level-2 trigger, we now turn to the parameters of the trigger itself.

Architecture B has five processor farms; four local and one global. Figure 5.2 shows a diagram of the architecture marking the ROBs, FeX etc. The precise size of the processor farms is an unknown at present, since it depends on the processing requirements as well as the technology available at the time of construction. The aim of the "Paper Model" [37] is to allow a quick investigation of different parameters and options in the second level trigger. It consists of a simple model implemented on a spreadsheet. The input to this model is the trigger menu and various technology parameters. Assuming 500 MIPS per processor and the current trigger algorithms, the "Paper Model" has estimated the size of the farms; these numbers are quoted in Table 5.4. It must be stressed, though, that the algorithms are in a very early stage of development and that, accordingly, the processing power needed has a very high level of uncertainty associated with it.

Initial emulation measurements had shown that a maximum output on the traffic

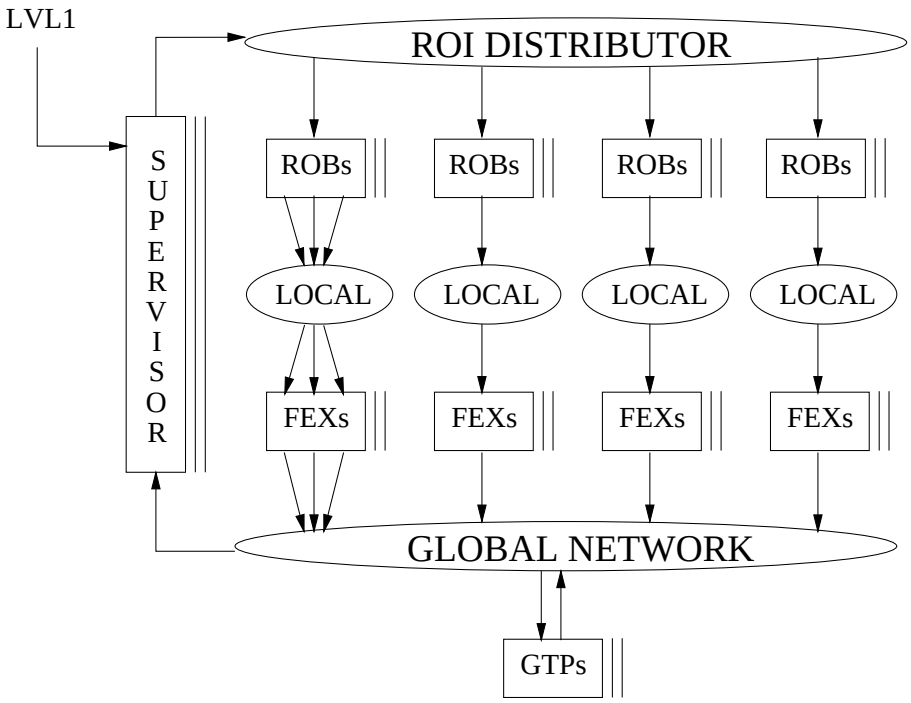


Figure 5.2: Diagram of architecture B.

nodes with this kind of traffic was limited to 8 MBytes/s. To ensure that the data fragments could be accepted by the receiving traffic nodes at a 100 kHz event rate, we adjusted the farm size such that, the output rate would be around 8 MBytes/s at an event rate of 120-130 kHz. In Table 5.4, the farm sizes used for the emulation are shown; the number of MIPS needed per processor for current algorithms are also shown.

5.4 Drive Files

The contents of the drive file can be read as a trace or log of the event flow through the level-2 trigger in terms of how much data enters the system and how long the data is being processed at the various stages. The task of the "drive file" became to guide the emulation on all the things that the emulation itself does not do, e.g. to put data in the read-out-buffers or to process the data. Therefore, the drive file had to contain information that in the real system would come after a level-1 trigger, i.e. RoI information from level-1 and data from the sub-detectors. Since we do not perform real feature extraction or any other physics algorithms, Monte-Carlo data is not required; but instead, the amount of

Farm	Paper Model		Emulation	
	Farm size	MIPS	Farm size	MIPS
Global	18	500	12	750
SCT	37	500	80	231
TRT	41	500	118	174
Calorimeter	74	500	286	129
Muon Spectrometer	5	500	10	250

Table 5.4: Farm sizes for high-luminosity/extended trigger menu. The MIPS quoted under the emulation is calculated using the adjusted farm size and the current processing estimates.

data to be transferred.

The drive files are generated during computer modelling [38] and provide input to the emulation. The computer modelling of the second level trigger implemented in Simdaq++ takes the trigger menu as input and uses a random number generator to create a set of events in the second level trigger. This set of events can be dumbed to an ASCII file. This facility has been used to generate the drive file used as input for the emulation. This ensures that the emulation, the computer modelling and the "Paper Modelling" start with common input parameters. The parameters listed previously in this chapter are the parameters which have been fed into the computer model. Some of these, like the algorithm bench marks, are averages over a distribution. The averages rather than the distribution have been used in the conventional modelling, and therefore also in the emulation, because it is less complex. In the conventional modelling, the trigger item for the current event is picked from the trigger menu. Each menu item has a probability to be picked in proportion to its frequency. The menu item specifies the number of RoIs and their individual types.

To drive the emulation studies, the following information must be provided for all events:

- for each RoI in the given event:
  - type of RoI (electro-magnetic, jet or muon), this is used to determine which sub-detectors should be involved (see table 5.1) and how many FeXs must be allocated to this RoI.
  - for each feature in the current RoI:
    - number of ROBs involved, this information is used to tell the fex in charge of this feature how many data fragments to collect before the processing can begin.
    - for each ROB:

      ROB identifier, used instead of performing the (η, ϕ) to ROB mapping in the supervisor.

An example of the format of the drive file is given below, the content is not real values.

```
....
<EVENT> 3 2 0 1000 150 1 1000
<ROI> 0 jet 2.400000 0.000000 14 0 1
<FEX> cal 150 111 2
<SUBFEX> emcal 6
<ROB> 376 92 10 78 <\ROB>
<ROB> 377 92 10 78 <\ROB>
<ROB> 391 92 10 78 <\ROB>
<ROB> 392 92 10 78 <\ROB>
<ROB> 393 92 10 78 <\ROB>
<ROB> 407 92 10 78 <\ROB>
<\SUBFEX>
<SUBFEX> hadcal 2
<ROB> 40 922 10 44 <\ROB>
<ROB> 47 922 10 44 <\ROB>
<\SUBFEX>
<\FEX>
  <\ROI>
<ROI> 1 jet -3.200000 0.392699 0 1 1
<FEX> cal 150 111 2
<SUBFEX> emcal 5
<ROB> 0 92 10 78 <\ROB>
<ROB> 1 92 10 78 <\ROB>
```

```
<ROB> 8 92 10 78 <\ROB>
<ROB> 9 92 10 78 <\ROB>
<ROB> 10 92 10 78 <\ROB>
<\SUBFEX>
<SUBFEX> hadcal 2
<ROB> 0 922 10 44 <\ROB>
<ROB> 1 922 10 44 <\ROB>
<\SUBFEX>
<\FEX>
  <\ROI>
<\EVENT>

<EVENT> 4 2 0 1000 150 1 1000
....
```

The `<EVENT> 3 2 0 1000 150 1 1000` tells that event number 3 has 2 RoIs, that the level-2 decision is "no", that the size of level-2 output is 1000 bytes, and that 150 μs is used for the global processing step. The last two numbers are not used in the local-global architecture. The following line containing `<ROI> 0 jet 2.400000 0.000000 14 0 1` says that the first RoI is of type jet, the following four numbers are the η, ϕ coordinate and index respectively. The last number says that only one FeX is needed to handle this RoI. The third line `<FEX> cal 150 111 2` gives the sub-detector, here the calorimeter, the FeX output of 150 bytes, the feature extraction time of 111 μs and that data fragments from two set of ROBs are expected. The following line `<SUBFEX> emcal 6` says that the first set of ROBs is the electro-magnetic calorimeter ROBs, and it contains 6 fragments. `<ROB> 376 92 10 78 <\ROB>` says that ROB number 376 must send 92 bytes to a FeX. The last two numbers are extraction and pre-process time respectively.

The drive file generated for the emulation contained 140,000 events corresponding to 1.4 seconds at the 100 kHz event rate for the second level trigger. Due to memory size constraints in the traffic generating nodes and the transputers, not all the emulation studies have been run on the complete event sample.

From the drive files the event fragment distributions for the four local farms and the global farm can be extracted. The distributions are shown in figure 5.3. From the distribution corresponding to the SCT local farm, one can see that the average number of ROBs sending data to one SCT feature extractor is four. For the calorimeter the average is about fourteen, this includes contributions from both hadron and electro-magnetic

ROBs.

Component	Rate [kHz]	Component	Rate [kHz]
SCT ROB	aver. 1.81 max 2.79	SCT FeX	1.436
TRT ROB	aver. 1.79 max 2.33	TRT FeX	0.974
ECAL ROB	aver. 6.24 max 7.59	CAL FeX	0.823
HCAL ROB	aver. 14.83 max 16.49	MUON FeX	3.321
MDT ROB	aver. 0.39 max .67	GTP	8.333
RPC ROB	aver. 2.57 max 7.48		

Table 5.5: Component rates obtained from the drive file.

The hitrate of the components can also be obtained from the drive files. These rates are shown in table 5.5. The rates have been compiled as follows:

ROBs For each ROB; count how many times it is hit per second.

FeXs For each sub-detector; count number of RoIs in sub-detector per second; divide by number of FeXs.

GTP With an event rate of 100 kHz and twelve global trigger processors the hit rate becomes $100\,\mathrm{kHz}/12 = 8.3\,\mathrm{kHz}$.

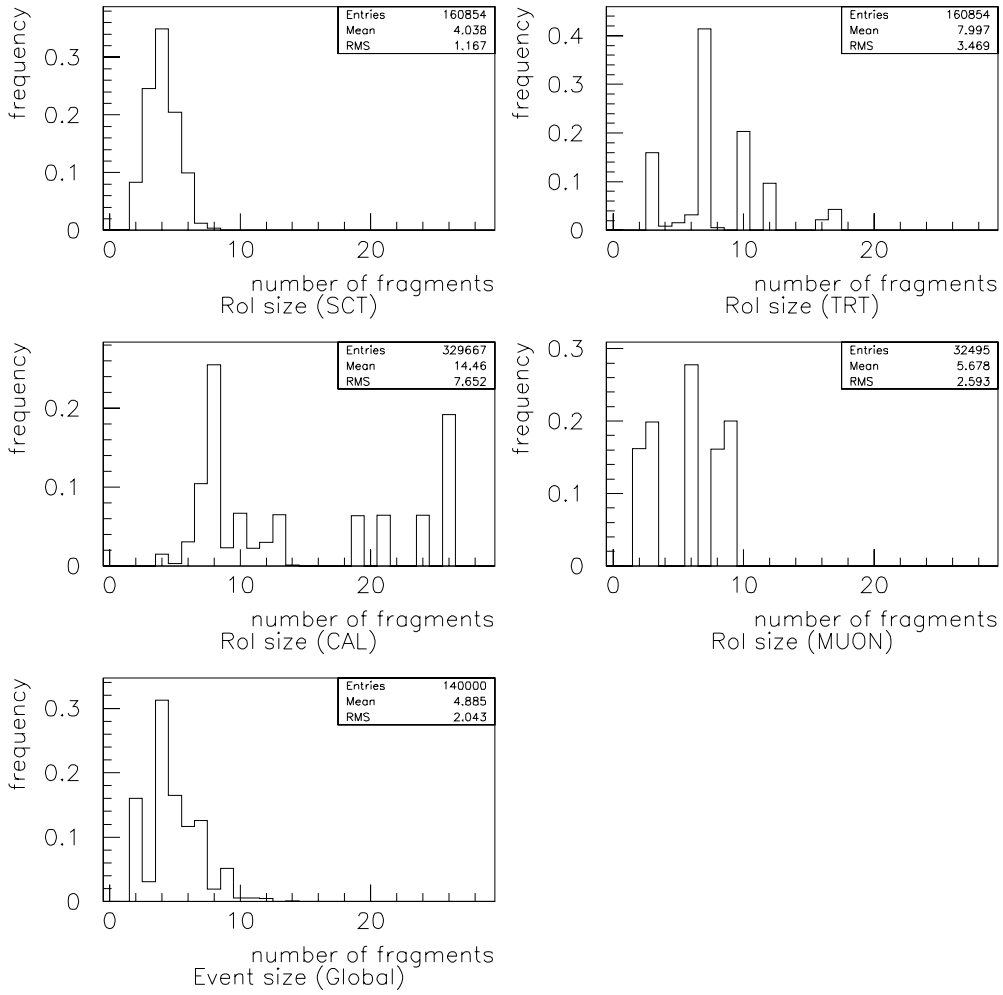


Figure 5.3: The event fragment distribution from the five networks.

Chapter 6

Emulation on the Macramé Test-bed

This chapter describes the Macramé test-bed and the results obtained by applying ATLAS level-2 trigger traffic to it.

The test-bed consists of traffic generating nodes, timing nodes and a DS-Link network. For the measurements described here, the test-bed was configured with 512 traffic generating nodes, and two timing nodes connected as a Clos network.

The aim of this chapter is to present the constraints and possibilities of the Macramé test-bed, the measurements performed, the results obtained and finally the understanding which came out of the study. The measurements and results presented in this chapter are the authors work.

6.1 Constraints and Possibilities

Macramé is a network test-bed, not a parallel computer. This gives some practical limitations to what is possible and what is not. The traffic generating nodes send data out into the network according to traffic descriptors. Each node looks in its memory, to determine when, according to the global clock, it should send a packet with a specified size and destination. The total amount of data sent and received by each node is logged such that, at the end of the emulation run, knowledge of the transmit and receive rates can be obtained, i.e. simple averages. The traffic generating nodes have no ability to act upon either the content, or the number of packets they receive. A packet once received is

discarded. A timing node sends trace packets with a specified frequency and length, to a second timing node. The receiving timing node calculates the latency of the trace packet by comparing the time stamp in the packet to the global clock. The packet descriptors of the timing nodes can only be specified once, i.e. with a specific interval, a certain length packet is always sent to the same destination. For more detail on the Macramé test-bed see chapter 3.

Macramé allows each node to independently deliver a predetermined set of packets to the network and can as such be used to model ATLAS level-2 architecture B traffic where data is pushed through the network. The size of the test-bed is comparable to the size of the networks needed in the trigger.

6.2 Implementation Details

Having covered the main constraints of the test-bed and what was emulated as a result, the focus is now turned toward the details of the emulation on the Macramé test-bed. First, the assumptions made will be summarised, this is followed by a description of the measurements performed.

Assumptions The time distribution between events is assumed to be constant and the timing of the individual packets associated with a given event is assumed to be synchronised, i.e., all ROBs participating in a given event are instructed to send the data at the same time according to the global clock. In the global network, the same assumption is applied to the FeXs. This strict synchronisation will not occur in ATLAS but, due to various smoothing mechanisms in the data acquisition system, it is probably closer to reality than a random timing distribution.

The sizes of the ROB output quoted in chapter 5 table 5.3 is only the detector data. For the emulation on Macramé, thirty-two bytes have been added to take into account the routing information etc. needed by the level-2 protocol, such as e.g. which global trigger processor is assigned to this event.

Some networks could be completely emulated within the 512 nodes available. Where the total requirement, i.e. ROBs plus processors, exceeded 512, only a fraction of the network could be mapped onto Macramé. For the TRT, a total of 630 nodes were required for the full system. In the cases where it was not possible to emulate the full system,

we chose to emulate the first N per cent of the network, where N is the largest fraction possible to emulate. In the case of the TRT network N equal eighty. The implication of this approach is that if an RoI is within the first N per cent it is fully emulated, if it straddles the border only some of the data packets are sent, and if the whole RoI is outside the N per cent, it is completely discarded.

Generation of Traffic Descriptors For each traffic generating node a program extracts from the drive file the following information for each packet to be sent:

- the time between packets
- the length of the packet
- the address of the receiving node

As there is memory for only 6552 packet descriptors per node, the number of events which the traffic nodes cycle over in the emulation varies, depending on how often data from the given source are needed. Table 6.1 gives the number of events for each subnetwork emulated. Precautions were taken to ensure that the descriptor lists for all traffic generating nodes wrap around at the same time.

Processor, i.e. receiver, assignment is made using round-robin scheduling. For the global network emulation, the FeXs to be used as sources are also assigned by a round-robin scheduler per sub-detector.

The Basic Measurements At initialisation, traffic nodes are assigned as receivers or sources and loaded with the traffic descriptors generated from the drive file. After initialisation, all the traffic nodes are started simultaneously and start sending packets according to their packet descriptors. When the traffic nodes have been sending for some time, such that a steady state is reached, the timing nodes start sending their trace packets. The process continues until of the order of 100,000 trace packets have been transmitted.

As already mentioned, each traffic node monitors how many bytes per second it transmits and receives. The timing nodes only record the latency of incoming trace packets. The timing node can discriminate between data-packets coming from traffic nodes and trace-packets coming from another timing node. Upon receiving a trace packet,

it observes what time it is, and from the time-stamp contained in the trace packet, it calculates the latency of the packet. This will be referred to as the "single packet latency."

As the network approaches saturation, the traffic nodes cannot send their packets at the specified time intervals. Two possibilities arise corresponding to a transient and a permanent state. In both cases, the traffic node knows that it has a back-log of packets to send. In the first case, it manages to make up for the lost time by sending the next few packets back to back. In the second case, the node cannot catch up, and the synchronisation between the nodes is irreversibly lost once a buffer counter inside a traffic node overflows. In this case, the "offered" and "accepted" data rates will differ.

6.3 Configurations Emulated

The configuration information for each run can be divided in two: how many components are being emulated and how they are mapped onto the hardware.

Network	% done	Sources	Sinks	# events
SCT	100 %	256 ROBs	80 FeXs	101,442
TRT	80 %	414 ROBs	94 FeXs	105,729
Calo	66 %	286 Ecal ROBs 32 Hcal ROBs	190 FeXs	38,171
Calo	50 %	216 Ecal ROBs 24 Hcal ROBs	143 FeXs	38,171
Muon	100 %	192 MDT ROBs 22 RPC ROBs	10 FeXs	67,863
Global	100 %	80 SCT FeXs 117 TRT FeXs 288 Calo FeXs 10 Muon FeXs	12 GTPs	108,874

Table 6.1: Configuration of emulated networks.

A number of emulation runs were performed; each involving only one level-2 trigger network. Table 6.1, gives the number of components emulated for the individual networks. Initially, sixty-six per cent of the calorimeter network was emulated. Due to

severe problems with the achieved throughput, later runs were performed with a different component to node mapping. For these future runs, only fifty per cent of the network could be accommodated on Macramé.

Two basic mappings, the "grouped" and the "distributed" were implemented. In the grouped case, the first sixteen sources are placed on traffic nodes connected to the same final-stage switch, the following sixteen are placed on the traffic nodes on the next final-stage switch and so on, until there are no more sources, thereafter we simply continue with the destination components. For the distributed mapping we take the first thirty-two[1] components and place each one onto the first traffic node on all the final-stage switches, then continue with the next thirty-two components which are placed on the second traffic node on all the final-stage switch, etc. until all the components have been placed, see figure 6.1.

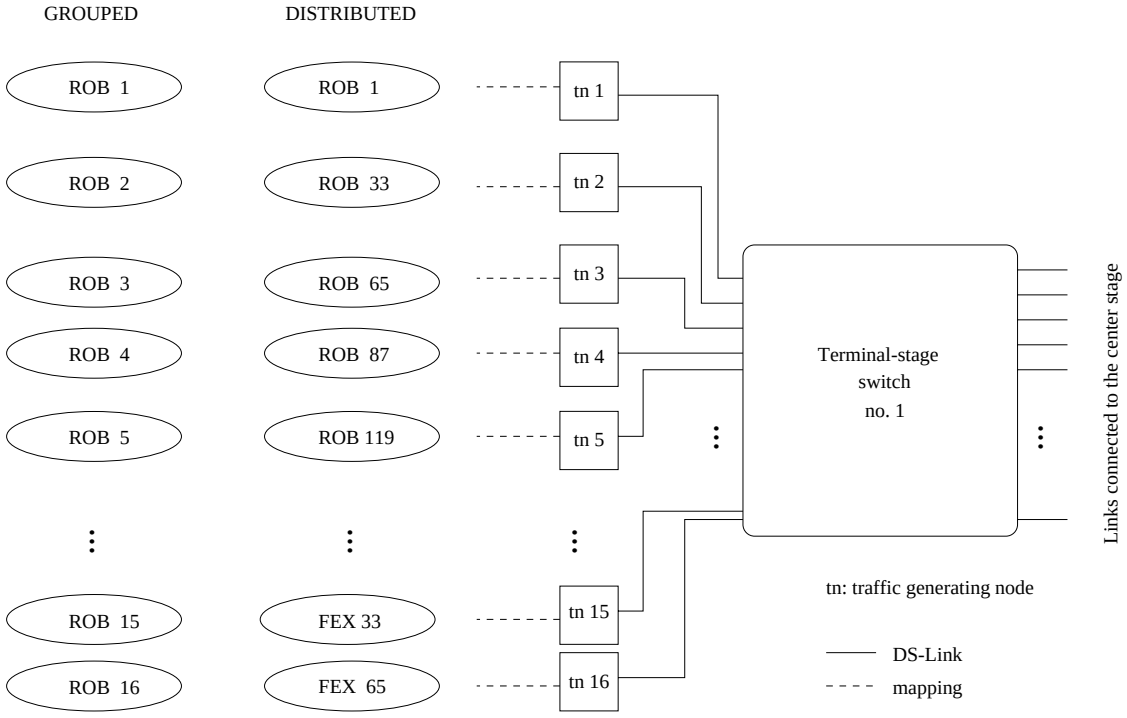


Figure 6.1: The two different mappings, grouped and distributed for terminal-stage switch number 1 in the case of the SCT sub-detector. See Figure 3.7 for a diagram of the Clos network.

[1]There are thirty-two final-stage switches in the Clos network.

6.4 Description of Results

This section describes the results obtained and conclusions reached from performing the measurements described above and the analysis of the output. The main measurements are the network throughput and trace packet latency distribution as a function of event rate.

Distributed versus Grouped Mapping Figure 6.2 shows the network throughput versus attempted event rate for both the distributed and the grouped mapping for the TRT local network. The throughput scales linearly with the event rate as long as the

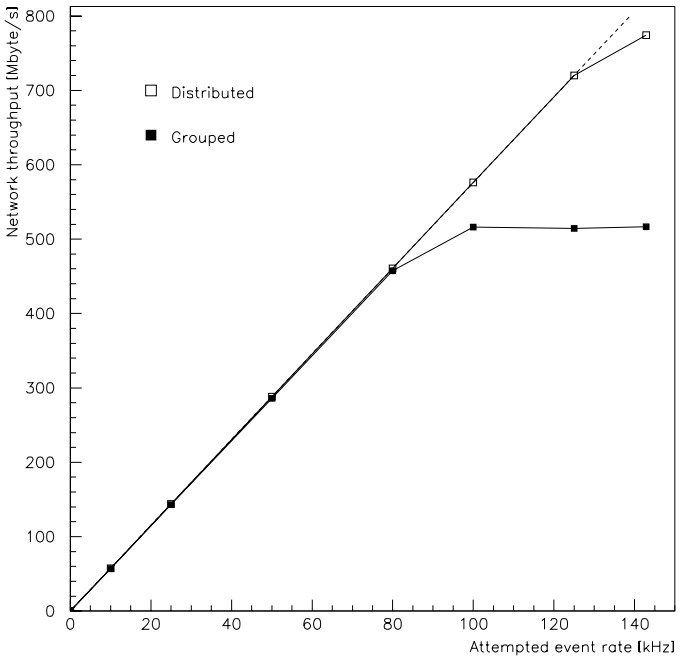


Figure 6.2: Network throughput for 80 % of the TRT network; distributed mapping vs. grouped.

attempted event rate is achieved. Where the rate falls below this straight line, saturation has occurred and the event rate cannot be sustained. The event rate achieved by the network is the maximum rate before the onset of saturation. It can be seen that the emulation of the network achieved an event rate of 125 kHz in the distributed case, but only 80 kHz with the grouped mapping.

In the figure 6.3, a latency plot at 10 kHz event rate for the TRT network is shown. The latency for a trace packet is measured from the time the timing node is instructed to send a packet to the time it is received. This includes any time queued in the timing node awaiting access to the network.

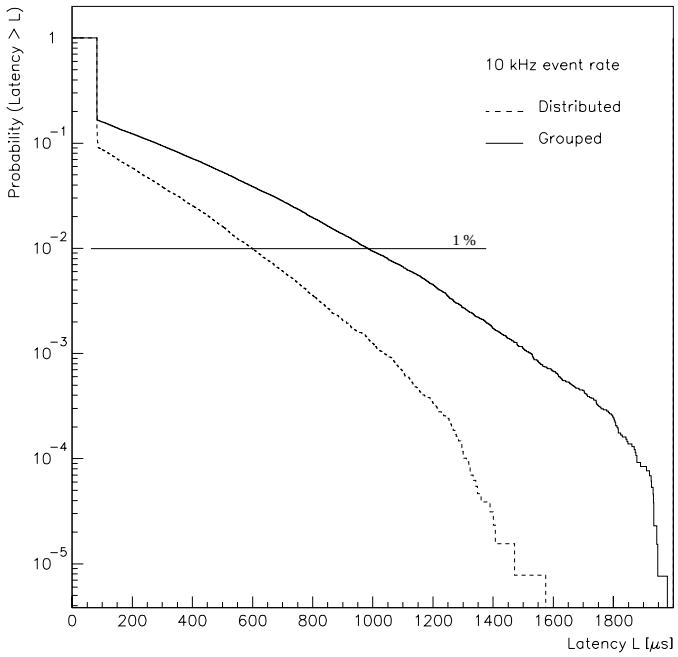


Figure 6.3: Single packet latency distribution for 80 % of the TRT network; distributed mapping vs. grouped.

Here, the probability that the latency is greater than a certain time is plotted. Note that the latency is lower for the distributed mapping, e.g., for the distributed case, 99 per cent of the packets have a latency less than 600 μs, the number for the grouped case is 975 μs.

When comparing the distributed versus the grouped mapping, it was observed that for both the single packet latency and the achieved event rate, the distributed mapping gave a better performance, independent of the frequency or the emulated network, see Table 6.2. We have used $l_{1\%}^{10\text{kHz}}$ as a measure of the latency where $l_{1\%}^{10\text{kHz}}$ is defined as the time for which only one per cent of the packets exceed this latency at an event rate of 10 kHz.

Network	Event Rate [kHz]		$l_{1\%}^{10\text{kHz}}$ [μs]	
	Grouped	Distributed	Grouped	Distributed
SCT	80	125	599	408
TRT	80	125	984	600
Calorimeter	25	25	945	830
Muon	100	142	358	264
Global	100	125	103	74

Table 6.2: Achieved event rates for the various network for both the grouped and the distributed mapping.

A network hot-spot occurs in the grouped case when packets from up to sixteen consecutive events are heading for destination nodes connected to the same switch, in the distributed case the packets are funnelled to a different terminal-stage switch for subsequent events.

The improvement can thus be attributed to a reduction in congestion in the terminal stage switch for the receivers. It must therefore be concluded that a distributed mapping of the emulated components onto the network nodes is preferable to the grouped mapping in terms of network performance. The results quoted in the rest of this chapter have all been obtained with distributed mapping.

In figure 6.4, the achieved network throughput as a function of the second level event rate is shown for all the emulated networks. One observes that the 512 Clos network could sustain the traffic emulating the global, the SCT and 80 % of the TRT network up to 125 kHz and the muon network even up to 142 kHz. The Clos network could, however, only deal with an event rate of 25 kHz for 66 % of the calorimeter network.

Optimising the Hadron Calorimeter ROB Mapping From the "paper modelling" [37], it was known that the HCAL read-out buffers were heavily loaded at 100 kHz and we expected the output DS-Link to saturate at around 60 kHz event rate. The deterioration in performance above 25 kHz was not expected and needed explanation.

In order to try to overcome the poor performance, we tried two additional configurations. The basic idea was to enhance the HCAL ROB interface to the network. First, an increased link speed was emulated by using three traffic nodes to send the data for

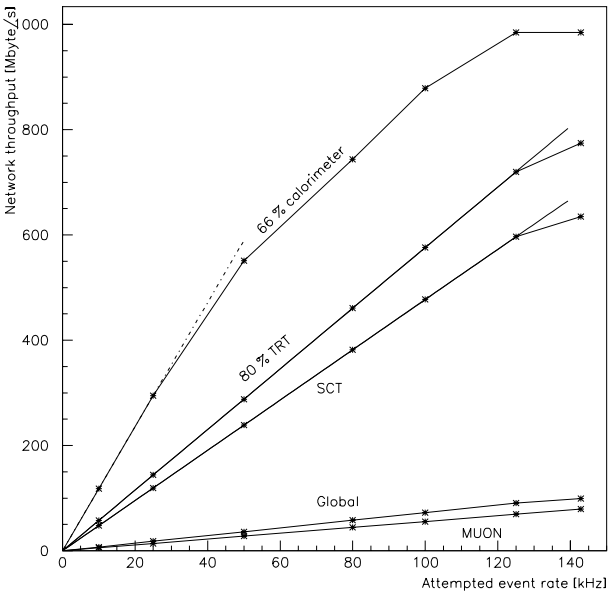


Figure 6.4: Throughput for the distributed runs.

one HCAL ROB, where for an event each traffic node sends one third of the data. Since they all fire at the same time, this scenario is called "Salvo". In the other scenario, three traffic nodes per HCAL ROB were also used, but here the traffic nodes took turns to send an event, i.e. one traffic node sends all the data for one event, but only participates in one third of the data transfers. This scenario we call "Round Robin." All three configurations (basic, Salvo and Round-Robin) were run with 50% of the calorimeter being emulated.

Measurements on the three configurations are shown in Figure 6.5. The top row shows the transmit rate as a function of the traffic node number, the second row shows the corresponding hit rates; all the plots correspond to a 10 kHz event rate. The hit rate is defined as the number of RoI data fragments sent per ROB per second. The spikes are the HCAL ROBs, the ECAL ROBs appear as the steady "background".

The first column *a*) contains data where the calorimeter has been emulated, with a distributed mapping. Here it can be seen from the transmit rate that, by scaling up to 100 kHz, that the network will not be able to sustain such an event rate, since this would demand a transmit rate of 13-15 MBytes/second out of the HCAL ROBs well beyond the

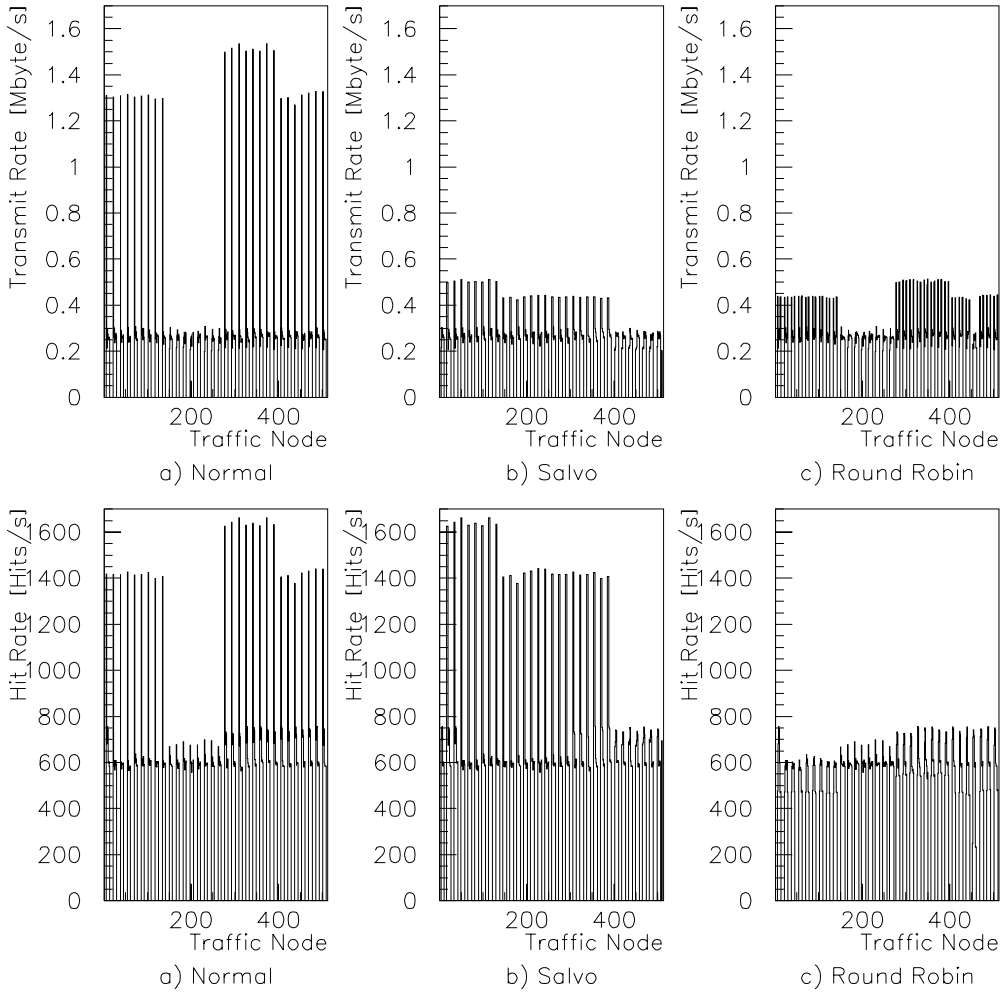


Figure 6.5: The first row shows the transmit rate for the calorimeter ROBs at 10 kHz event rate, the second row shows the corresponding hit rate. The first column *a*) is from emulating half the calorimeter using the normal distributed mapping. The second *b*) and third *c*) column is for the "salvo" and "round-robin" scenario respectively. The spikes in the histogram corresponds to the HCAL ROBs, the "background" are the ECAL ROBs. The rearrangement of the spikes along the node axis is because the exact mapping was not the same for the emulations, but the overall distributed mapping was still applied.

theoretically available link speed for data of 9.5 MBytes/s, (this is why the performance was expected to saturate above 60 kHz).

The first approach was to use the "Salvo" scenario described earlier. Looking at the transmit rate in the second column *b*) in Figure 6.5, one observes that the link bandwidth is not a limiting factor even at a 100 kHz event rate. The output link bandwidth is also not the problem, as the number of feature extractors was chosen in section 5.3 such that at 100 kHz event rate the throughput is less than 8 MBytes/s. The third column *c*) shows the transmit and hit rate for the "Round Robin" scenario.

The cause of the problem requires a more detailed study of the data transfers to the FeXs. For the calorimeter data, there are on average 14 event fragments to be collected, with a minimum of 4 and a maximum of 26 fragments, see Figure 5.3. On average, two of the 14 fragments will come from the HCAL ROBs and each contain 890 bytes of data plus the 32 byte protocol header. The rest of the fragments come from the ECAL ROBs and have sizes according to the RoI type, see Table 5.3. From the trigger menus, approximately half the RoIs are jets. Collecting the 14 fragments takes of the order of:

$$(9.5\text{MBytes/s})^{-1} * (2 \text{ fragments} * 922 \text{ B} + 12 \text{ fragments} * 1/2(1332 + 92) \text{ B}) = 1\ ms \quad (6.1)$$

As all fragments try to send at the same time, this implies an average wait-to-send time of 0.5 ms. During this waiting time, a sending node participating in the event is stalled and any subsequent events are queued. This is known as "head-of-line-blocking." The wait-to-send time should be compared to the hit-rate. At 100 kHz event-rate the hadron calorimeter read-out buffer hit rate is 16 kHz giving an average separation for sending of data fragments of only 63 μs. This is clearly much smaller than the average 500 μs wait-to-send time calculated above. However, queued packets for subsequent events will suffer a shorter wait-to-send time when they do access the network since some other fragments for that event will have been transferred already. Measurements on the test-bed show that problems arise in the case of the calorimeter, when the hit separation is less than half the average wait-to-send time.

The Salvo technique does very little to alleviate this problem, as it mainly reduces the time a link is active. The small improvement in performance, see Figure 6.6, is due to the increase in probability of transferring part of the data with a reduced wait-to-send time. The Round-Robin technique reduces the hit-rate for any one HCAL ROB links

by a factor of three and therefore improves the rate limit by a similar factor. With this technique, the network operates up to nearly 80 kHz.

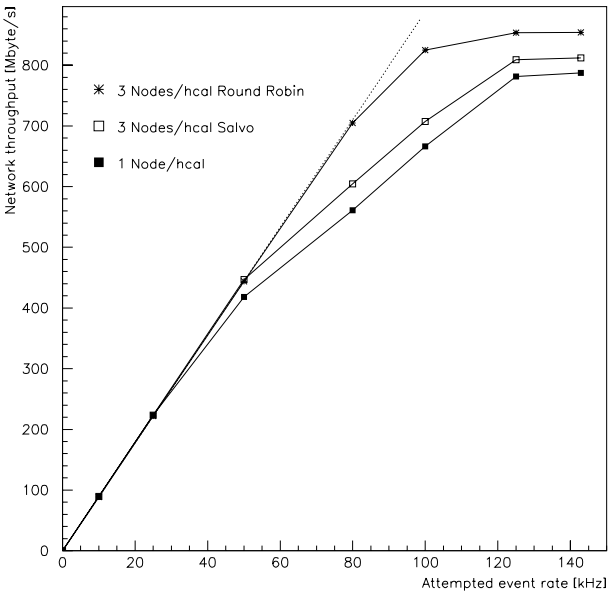


Figure 6.6: Throughput measurement on the three different emulations of half the calorimeter network.

Alternatives to the round-robin technique are:

- Using faster links throughout the network, e.g. gigabit links [39]. This will decrease the wait-to-send time for the buffers since the data are removed faster from the links.
- Structure the data dispatch times so that the wait-to-send time is substantially reduced, i.e. traffic shaping.

The Latency Distributions The latency distributions for the five different networks are shown on the following pages. The latency distribution is shown both at the 100 kHz event rate (80 kHz for the calorimeter) and at the 10 kHz event rate, such that the evolution of the latency distribution can be evaluated. The latency distributions are shown first as a simple histogram with the fraction of events with a given latency on the y-axis. In the second plot, the probability of the latency being greater than L is plotted

against L.

The latency distribution for the SCT network is shown in figure 6.7. The majority of trace packets arrive in a little more than 100 μs. The position of the peak corresponds to the minimum transfer time of a packet in the SCT network:

$$1032B/9.5MBytes/s + 3 \text{ switches} * 1\,\mu s/\text{switches} = 112\,\mu s. \tag{6.2}$$

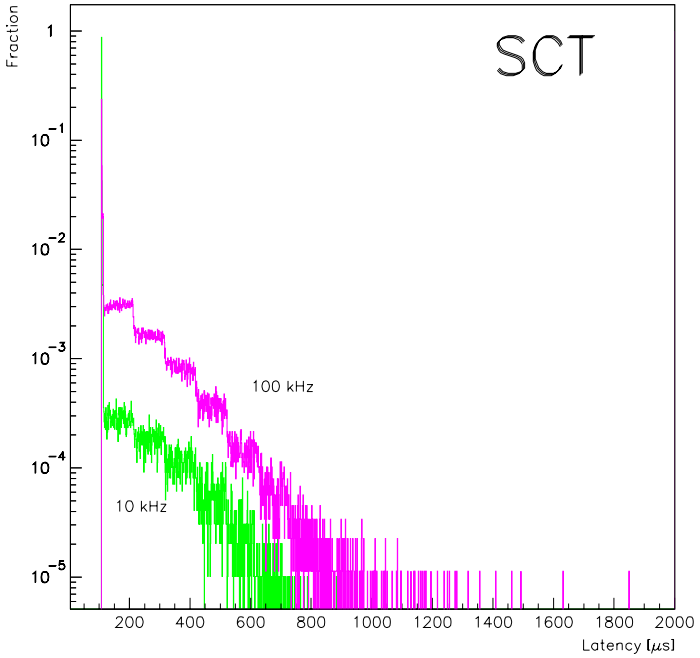
In the case of the latency distribution for the 100 kHz event rate, it is easy to see the step-like nature of the histogram in figure 6.7(a). The width of these steps also corresponds to the packet transfer time. The maximum latency measured is 900 μs for the 10 kHz event rate and 1850 μs for the 100 kHz event rate. From figure 6.7(b), it can be seen that these maxima occur only for one event per hundred thousand. From the same figure, it is also possible to read that one per cent of the trace packets will have a latency greater than 0.4 ms in the case of the 10 kHz run and 0.6 ms in the case of the 100 kHz run.

Figure 6.8 shows the latency distribution of the TRT network. Here, the peak containing most of the entries is moved to 85μs corresponding to the packet size of 782 Bytes. As in the case of the SCT, we can observe steps on the plot in figure 6.8(a). The sharp cut in the 100 kHz TRT distribution at 2 ms is due to the latency measurement limit in the Macramé test-bed.

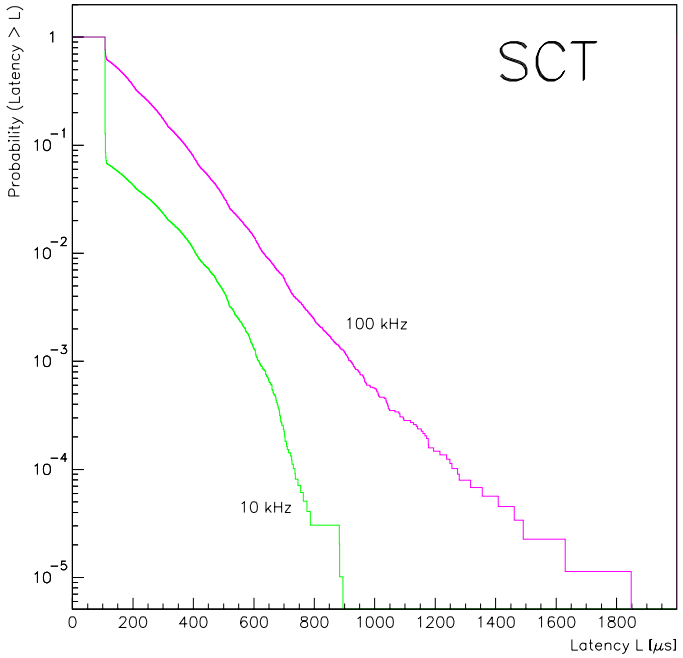
The latency plot for the calorimeter network, shown in figure 6.9(a), differs from the SCT and TRT latency plots in the sense that there are no clear steps. This is due to mixture of data packet sizes (92 Bytes, 922 Bytes and 1332 Bytes). The position of the peak at 97μs corresponds to the size of the trace packet, 922 Bytes. The sharp cut at 2 ms is also seen in the 80 kHz calorimeter distributions.

The latency distribution for the muon network, figure 6.10, can be compared to the latency distribution of the calorimeter, as also there are multiple length data packets. The worst case latency compares, however, more to the SCT local network. This is due to the relative ease with which the Macramé test-bed handles the muon network traffic, compared to the calorimeter local network traffic.

Figure 6.11 shows the latency distributions for the global network. The step like structure on the plot in figure 6.11(a) is back again as the traffic on the global network consists of only 150 bytes packets. The ease at which the test-bed handles the traffic is evident in the short tail on the latency distribution, even in the case of the 100 kHz event rate.

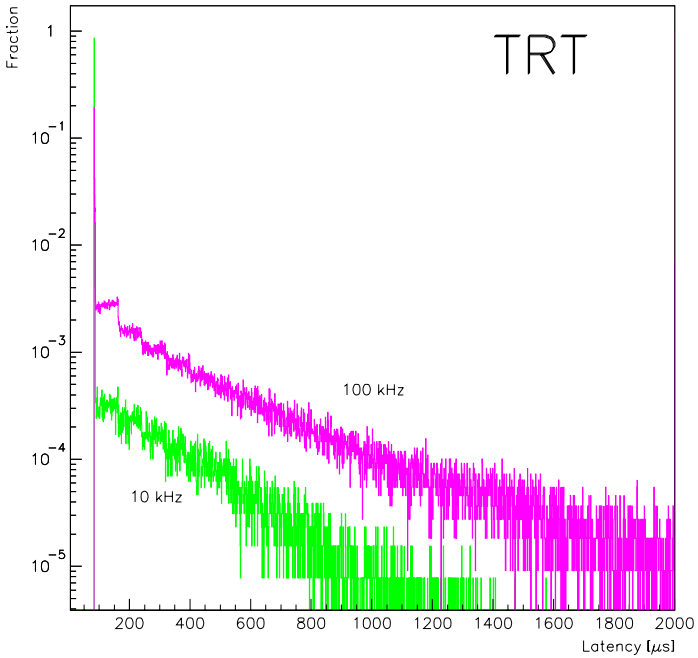


(a) The logarithm of the fraction of events against the single packet latency in μs. Both the 10 kHz and the 100 kHz event rate runs are shown. Notice the very clear step structure in the case of the 100 kHz event rate.

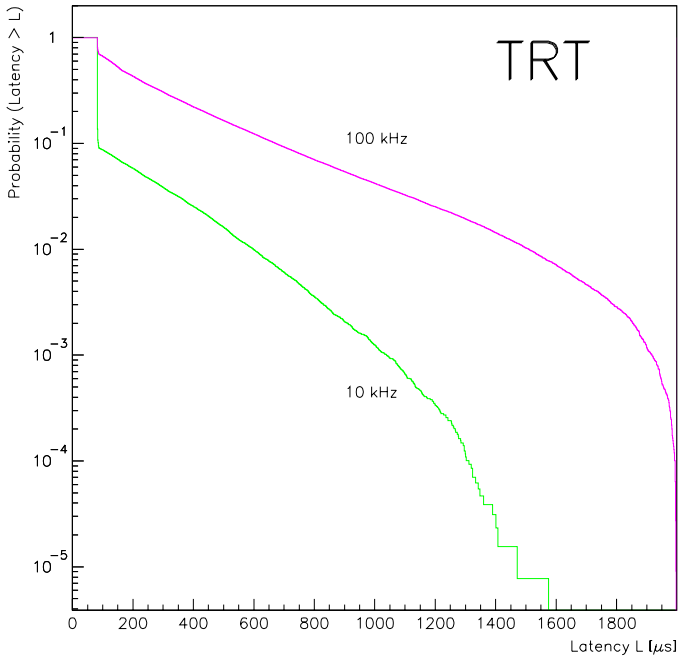


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.7: Latency distribution for the SCT network.

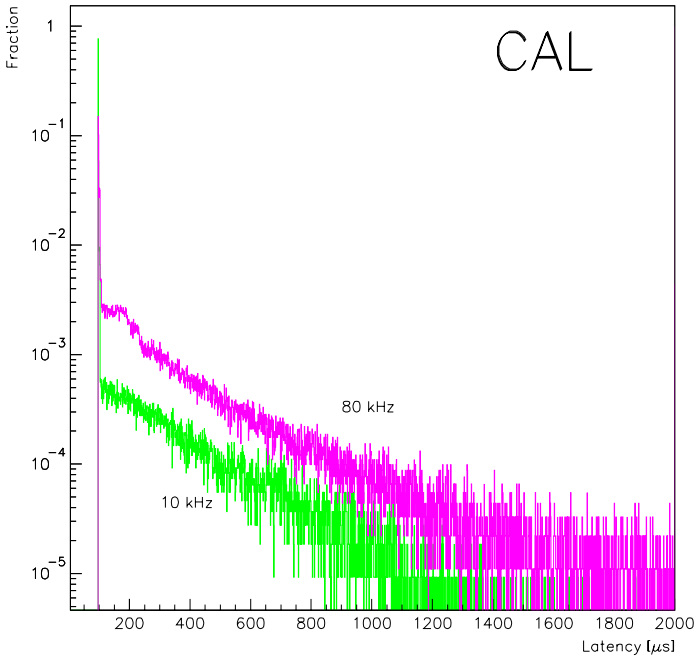


(a) The logarithm of the fraction of events against the single packet latency in μs. Both the 10 kHz and the 100 kHz event rate runs are shown.

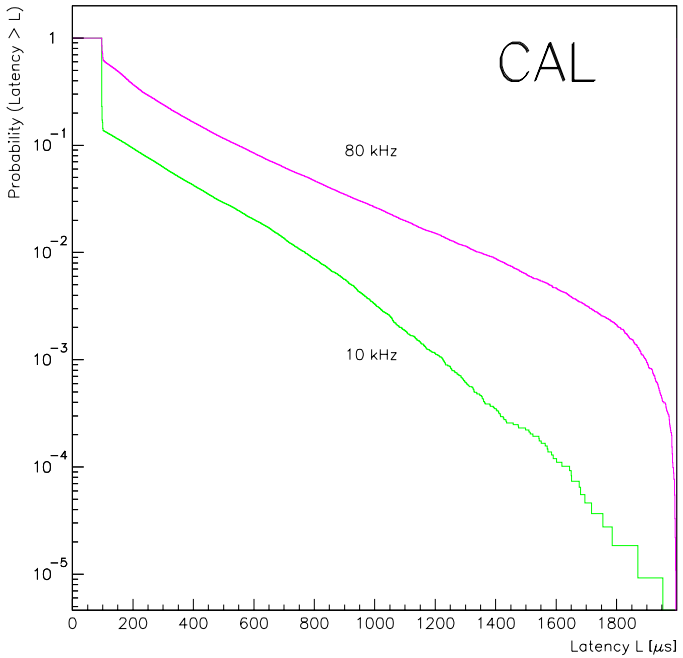


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.8: Latency distribution for the TRT network.

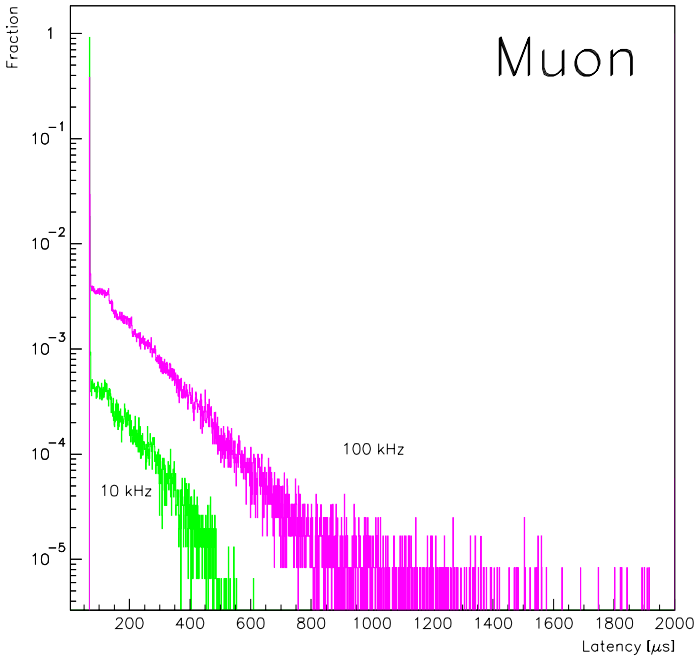


(a) The logarithm of the fraction of events against the single packet latency in μs. The 10 kHz and the 100 kHz event rate runs are both shown.

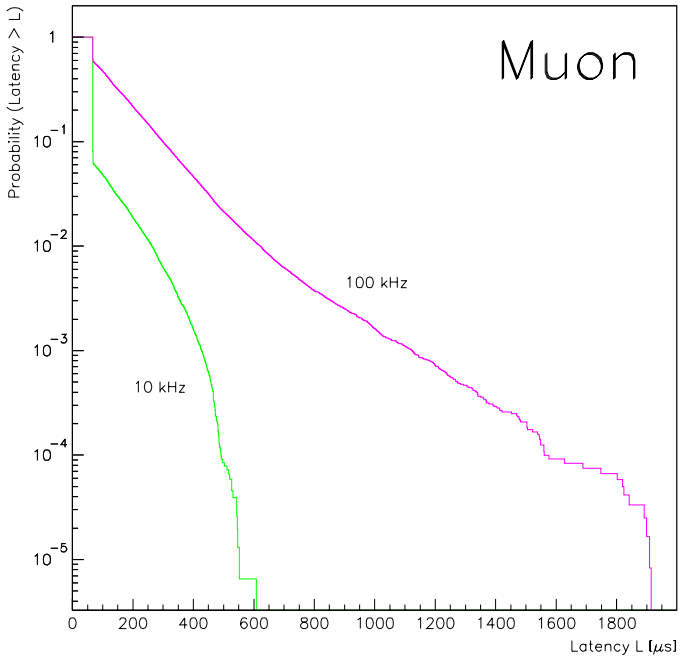


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.9: Latency distribution for the calorimeter network.

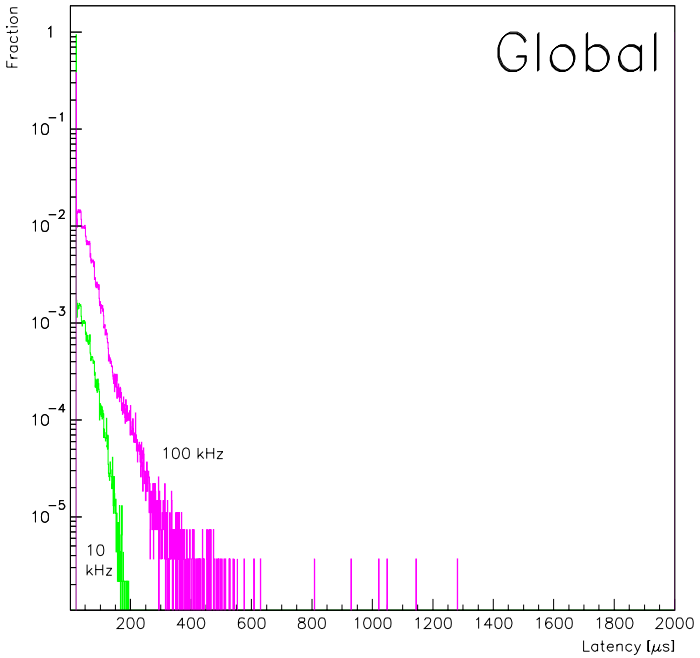


(a) The logarithm of the fraction of events against the single packet latency in μs. Both the 10 kHz and the 100 kHz event rate runs are shown.

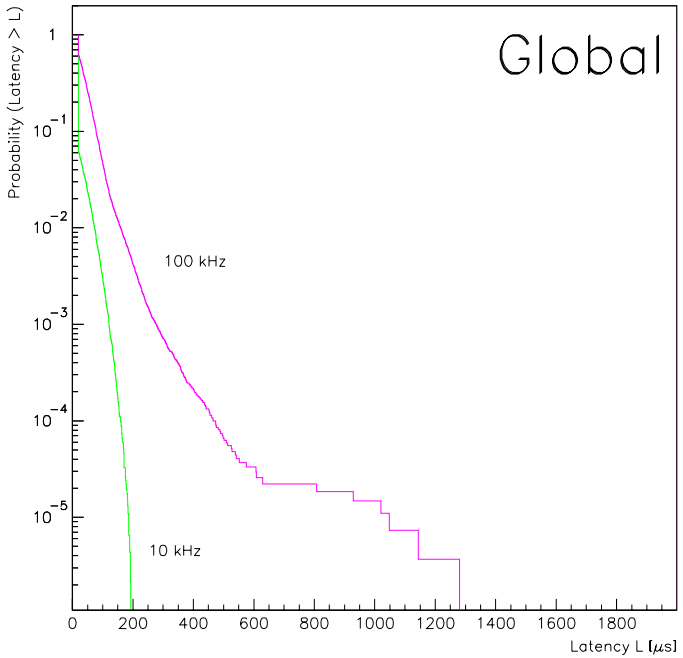


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.10: Latency distribution for the muon network.



(a) The logarithm of the fraction of events against the single packet latency in μs. Both the 10 kHz and the 100 kHz event rate runs are shown.



(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.11: Latency distribution for the global network.

Modelling the Latency Distributions In an attempt to better understand the latency distributions, a simple model has been developed. The latency distributions are measured with trace packets sent from one timing node to another timing node.

The main ideas behind the model are:

- all packets, i.e. both trace and data packets, are queued in the switch until they have arrived at the destination node (this is ensured by the flow-control in the DS-Link network),
- all packets, i.e. both trace and data packets, compete equally for access to the destination node, and
- there is only one effective point of contention in the network[2].

Furthermore the model assumes that:

- all packets, including the trace packets, are of the same size and require a time t_p for transfer across a single link,
- all data packets from an event attempt to access the network simultaneously, and
- there are a maximum of two events headed for the same destination in the network at any point.

Consider a time interval where the first competition time is at time 0, the second at t_p, the third at $2t_p$ etc. At the first competition, there are N data packets; at the second one $N-1$ etc., where N is the total number of data packets for this event. The trace packet, which is applied to the network at a random time, waits for the next opportunity in which it competes equally with all the remaining data packets. The latency of the trace packet is calculated from when it first enters the network until it "wins" a competition; to this is added the nominal transfer time(t_p).

In order to calculate the latency distribution of the trace packets all possible times, with respect to the data packets, for applying the trace packet to the network must be considered. This is done by stepping through all possibilities, adding each result as an entry to the latency histogram. In the cases where the trace packets arrive after all the

[2]This is consistent with the fact that a Clos network is non-blocking.

data packets have arrived at the destination node, the trace packets will not have any competition and their latency will be t_p.

So far we have only taken into account one event size, i.e. the trace packets competing with N data packets. As shown in figure 5.3, the number of data packets in a sub-detector RoI varies from event to event. To take this into account, the histogram calculated above must be recalculated for all possible number of data packets. These histograms are then finally summed up according to their weights.

One overlapping event is taken into account by increasing N by M, the number of packets in the next event, after the average time between events destined for the same processor. Also here, we have used different values of M, and thereafter, done a weighted sum. Figure 6.12 provides a sketch of the model.

This model was compared to the latency measurements. Figure 6.13 to 6.15 shows the modelled single packets latency distributions superimposed on the measured distributions. Good agreement between model and measurement was obtained.

The calorimeter and the muon networks were not modelled in this simple way because they have several different sizes of event fragment packets.

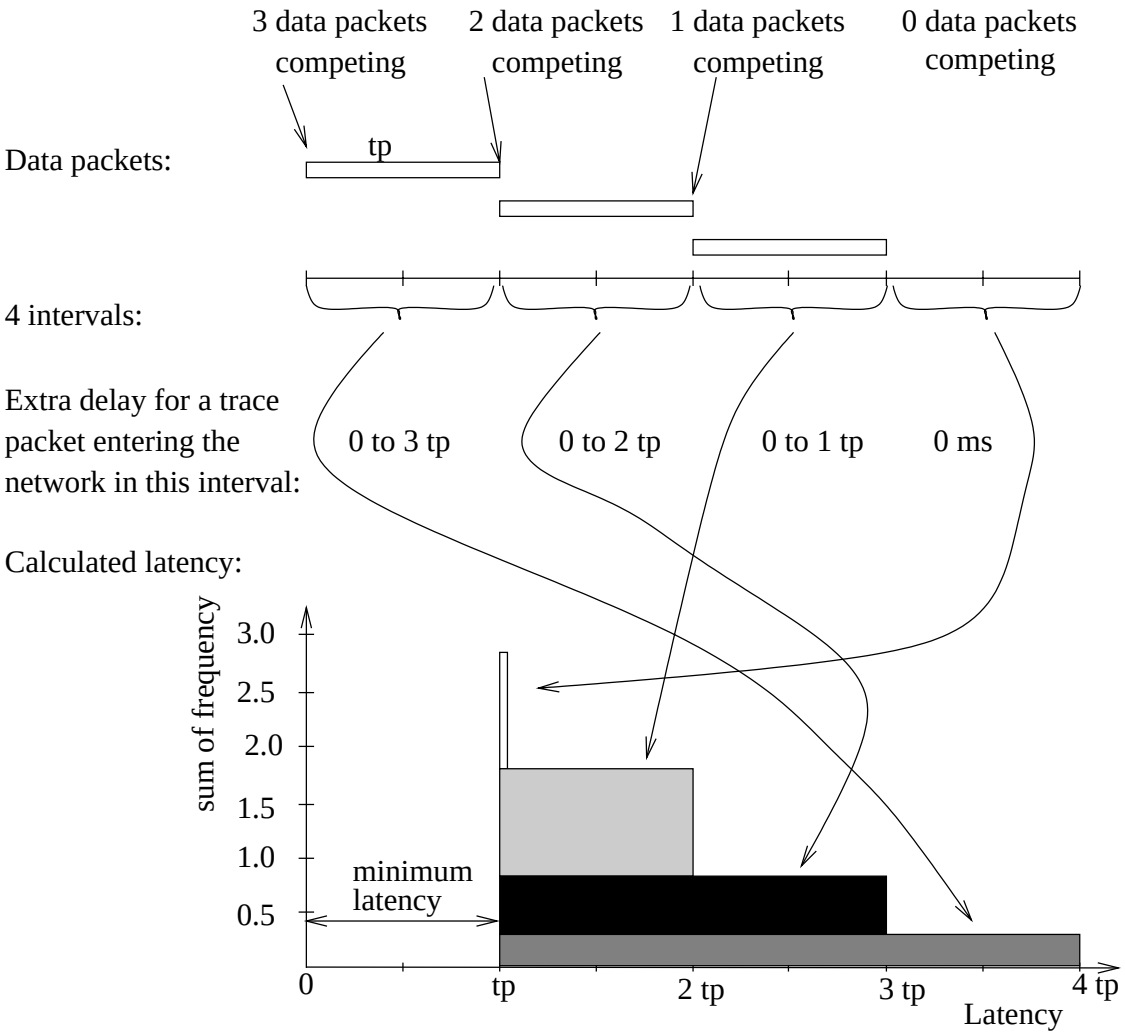
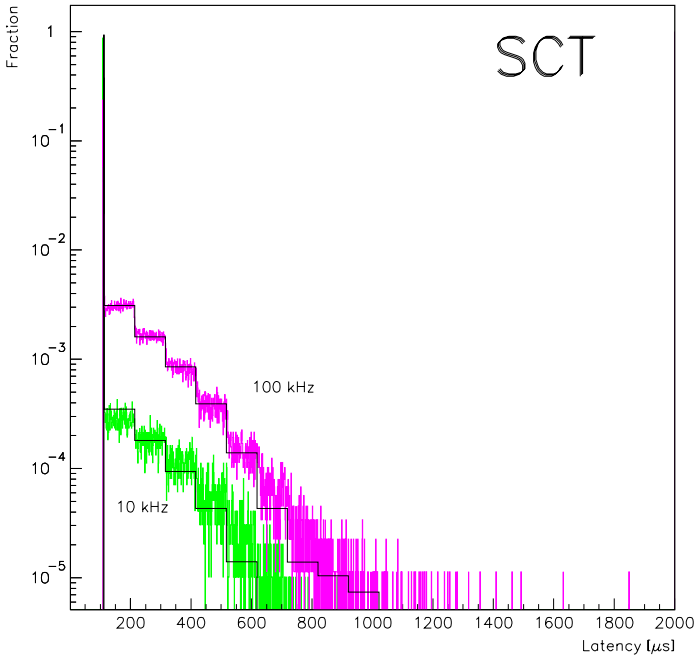
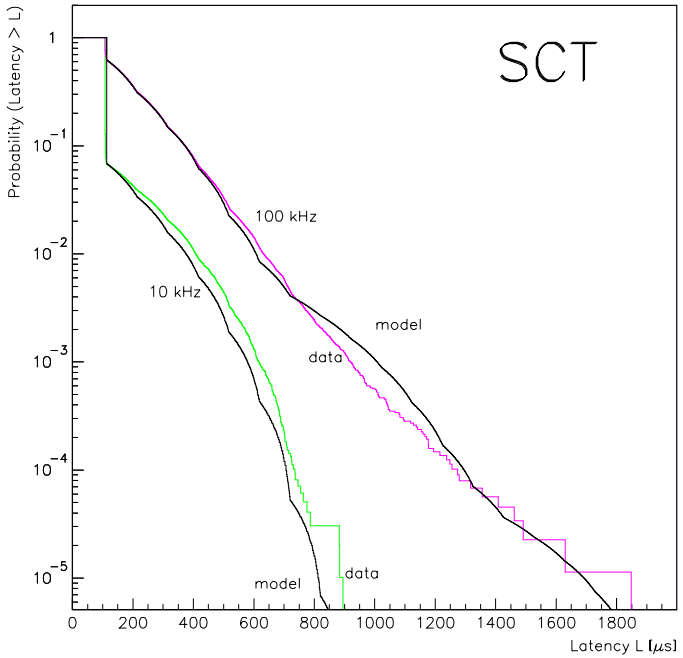


Figure 6.12: Sketch of how the latency model works. In the figure, the event has 3 fragments, and there is no overlapping event. The trace packet can arrive at any time in the four intervals. If it arrives during the last interval, it can be transfered across the network with the minimum latency; giving rise to the first peak. If the trace packet arrives in the first interval, it has 1/4 chance of getting transferred first, second, third or last; giving rise to the base in the calculated latency. The latency histograms are calculated for each possible number of fragments and then summed with the appropriate weights (see Figure 5.3).

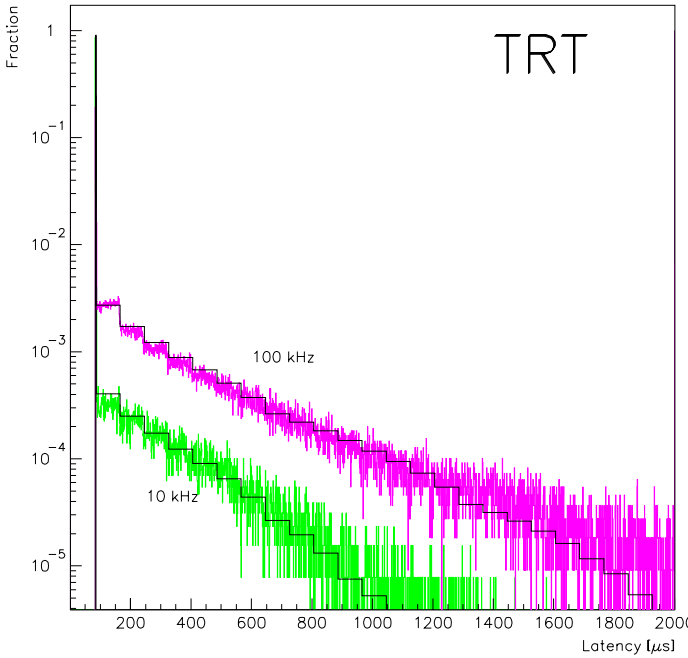


(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.

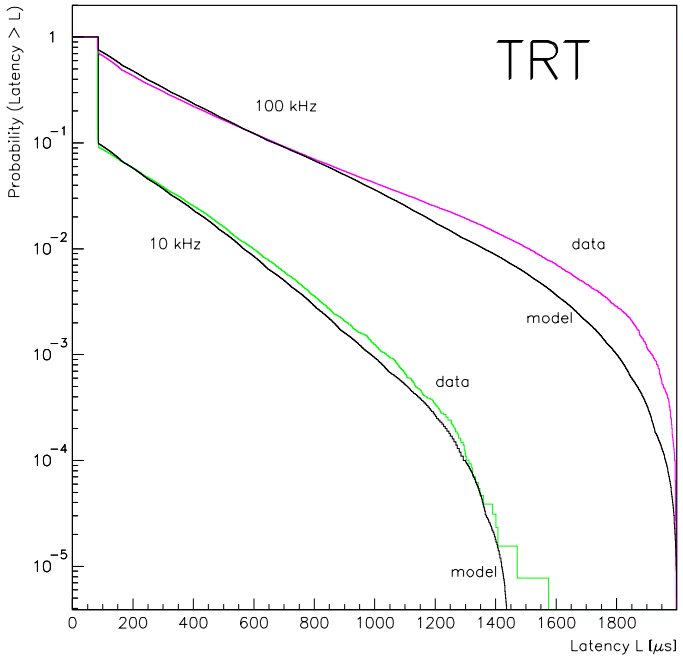


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.13: Model superimposed on latency distribution for the SCT network.

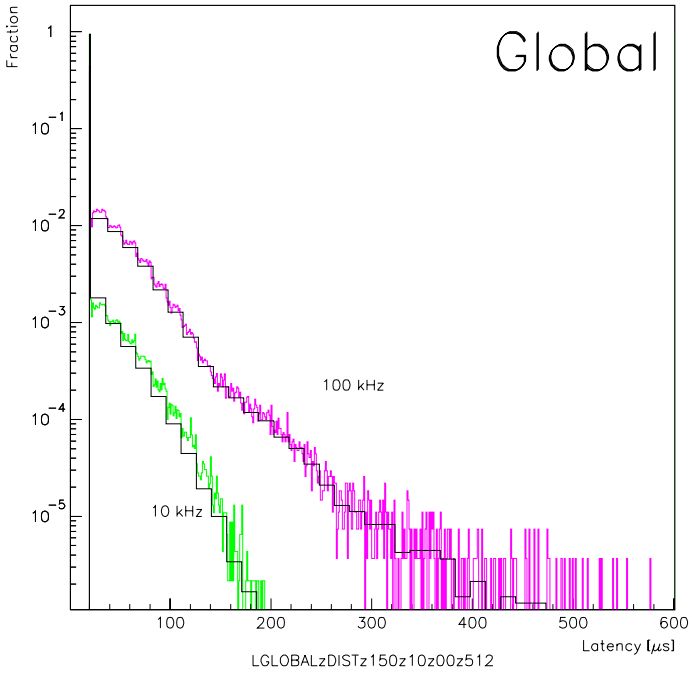


(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.

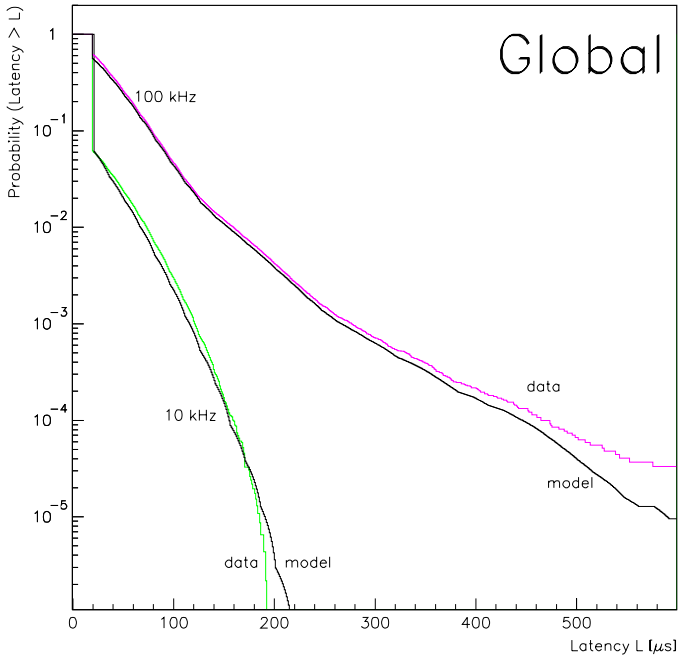


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.14: Model superimposed on latency distribution for the TRT network.



(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.



(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.15: Model superimposed on latency distribution for the global network.

In section 6.2, earlier in this chapter, we argued that the event separation is smoothed but that it will not be completely constant, as we have implemented it on the Macramé test-bed. Rather, it will be somewhere between randomly distributed and constant.

To illustrate the range of possible expectations for the final level-2 trigger system's latency distributions, we have changed the overlapping events in the previous model to appear at a random time rather than at a fixed time. The probability of having overlapping events has been taken from the following distribution:

$$\text{Prob} = \frac{1}{X} \int e^{-t/X} dt$$

where X is the mean time between events. The probability of having more than one overlapping event is small and has been discarded in this model. If taken into account it would raise the small probability of obtaining a large latency.

The model does not estimate the height of the first peak, rather, it calculates the shape of the latency distribution. The measurement and the model has been aligned at the first "step" on the histogram.

The results of this model are compared to the Macramé test-bed results, see figure 6.16 to 6.18. This model is in good agreement with the SCT and TRT results at the 100 kHz event rate. For the 10 kHz event rate, and the 100 kHz event rate in the case of the global network, the model does not agree as well as the first model does.

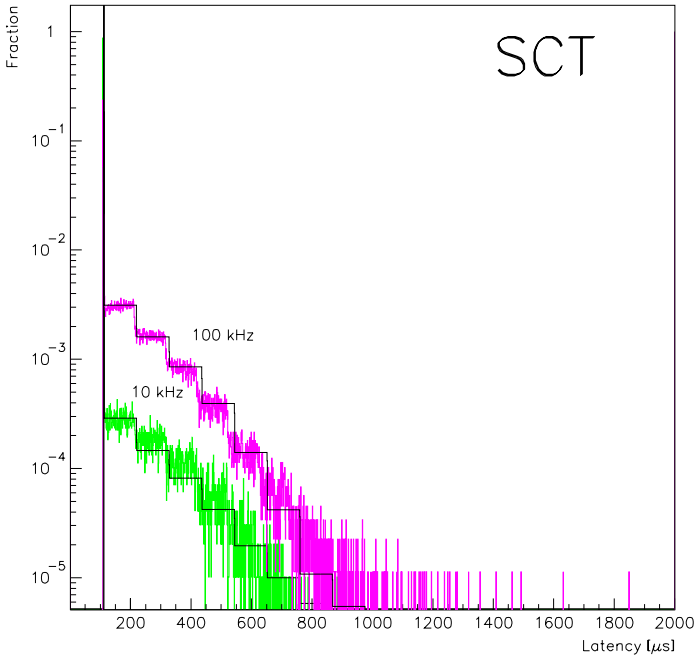
This can be understood in the following way:

10 kHz The random spacing between events can space two events very closely, causing very fierce competition. This results in long tails on the latency distributions. On the Macramé test-bed and in the case of the first model, overlapping events do not occur, resulting in small tails.

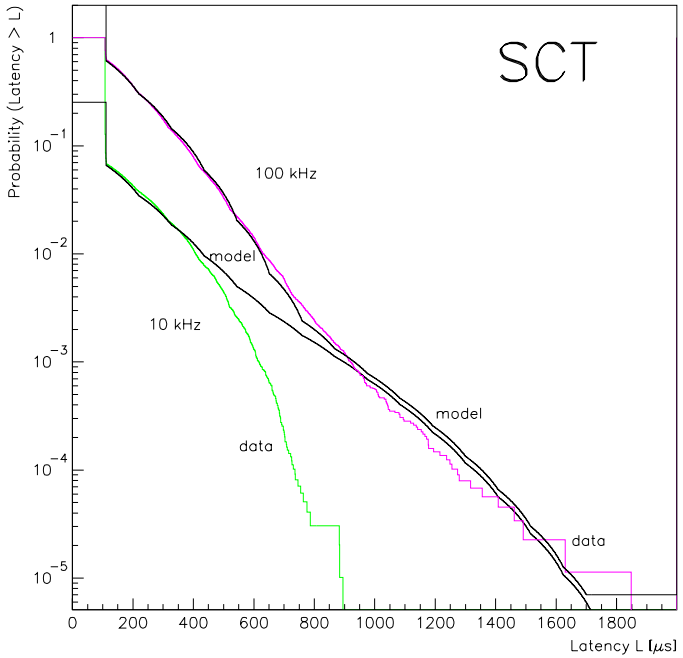
This reasoning also applies to the global network at the 100 kHz event rate, due to the small packet length (150 bytes) compared to the average event spacing.

100 kHz (SCT, TRT) The large packet size compared to the event rate, creates a situation where the two models generate the same result. This is because even with the average event separation, the last packets from one event are not delivered before the next set of packets appears at the bottleneck.

Close overlapping events cause tails on the latency distributions. One can use this to monitor network overload, as traffic overload causes growing latency distributions.

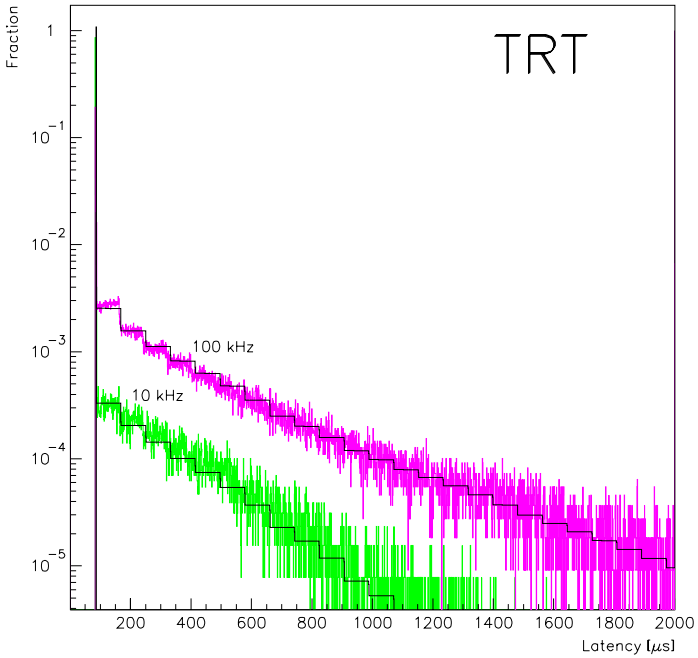


(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.

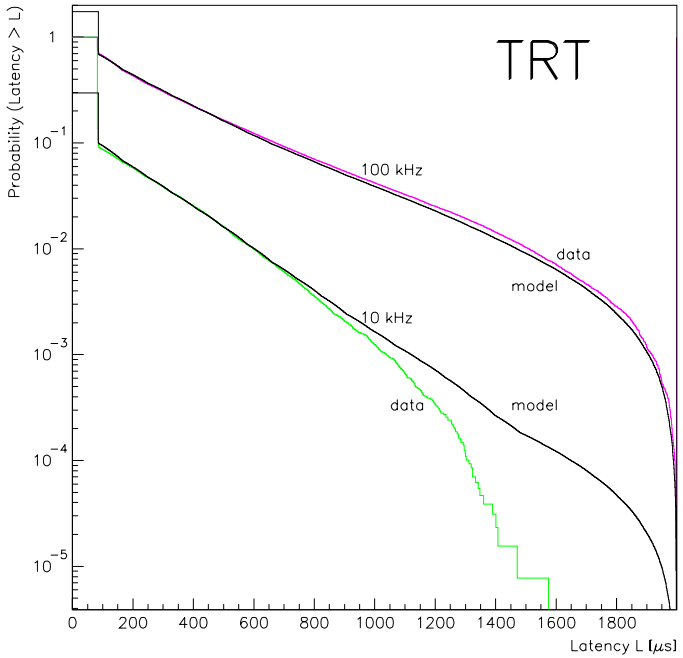


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.16: "Random" model superimposed on the latency distributions for the SCT network.

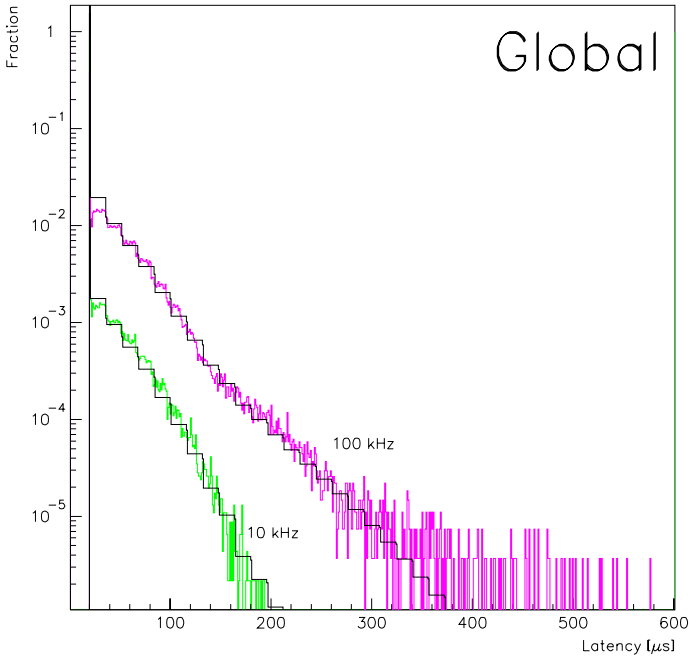


(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.

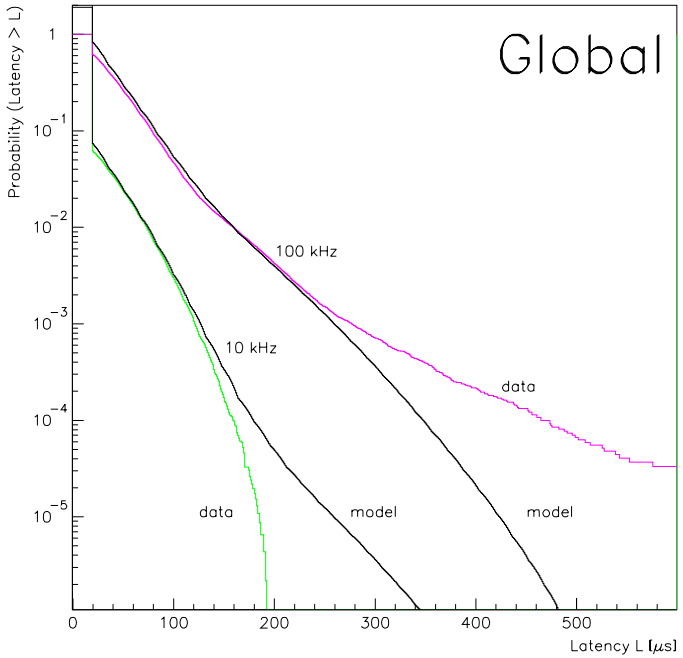


(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.17: "Random" model superimposed on the latency distributions for the TRT network.



(a) The logarithm of the fraction of events against the single packet latency in μs. The stepped lines are the model.



(b) The logarithm of the probability of obtaining a latency greater than L plotted against L.

Figure 6.18: "Random" model superimposed on the latency distributions for the global network.

6.5 Conclusion

The emulation studies of the ATLAS second level trigger, architecture B on the Macramé Test-bed have shown that distributing the sources and sinks evenly over the network rather than grouping them is important for enhancing the usable bandwidth of the network. Emulating the calorimeter network showed that decreasing the hit rate may be necessary to improve the throughput and thereby the sustainable event rate. The latency distribution can be understood by a simple model for how data blocking can occur throughout the switching network.

The local muon and SCT networks and the global network were emulated at full size. Substantial fractions of the TRT (80%) and the calorimeter (50%) networks were emulated, so that only a limited extrapolation is necessary for scaling to full size. The Clos networks have been shown to scale for systematic traffic from 64 nodes up to 512 nodes using Macramé [39], and theoretical considerations of the Clos topology show it should scale further [40].

Finally, it should be noted that the emulation of the calorimeter local network ran at an event rate of almost 80 kHz when three links for the hadron calorimeter read-out buffers were used. The other networks ran with an event rate of 125 kHz.

Chapter 7

Emulation on the GPMIMD Machine

This chapter describes the emulation study of the ATLAS second level trigger performed on the GPMIMD machine. This machine is a parallel computer with sixty-four transputers which are fully interconnected via four Clos networks as described in detail in chapter 3. The advantage of the GPMIMD machine over the Macramé test-bed is that unlike Macramé the nodes are microprocessors and it is possible to programme the individual transputers to act as a component of the ATLAS second level trigger. By programming different transputers in this way, it is possible to get the combined set of processors to behave in a manner similar to the ATLAS second level trigger. The GPMIMD machine is, however, not large enough to emulate the whole second level trigger at once.

The main objective of this study was to design a process model of the second level trigger, map it to the GPMIMD machine, and execute it there. To facilitate identification of bottlenecks etc. the implementation was benchmarked. A very simple model was used to compute latency and throughput for comparison with the measurements. Agreement with the model was rather poor. A detailed investigation of the mismatch was not carried out due to lack of time.

7.1 Design

This section details the design of the second level trigger as it was emulated on the GPMIMD machine. Flow diagrams will be used to explain the interactions between

the components, i.e. who sends what to whom, and if applicable how tasks have been split into independent objects. State diagrams will be used to describe the actions taken within a given object when receiving messages. One component can consist of one or more objects.

First, the overall flow diagram of the system will be described. Next, the flow diagrams of the components, and the state diagrams of the objects will be shown and described.

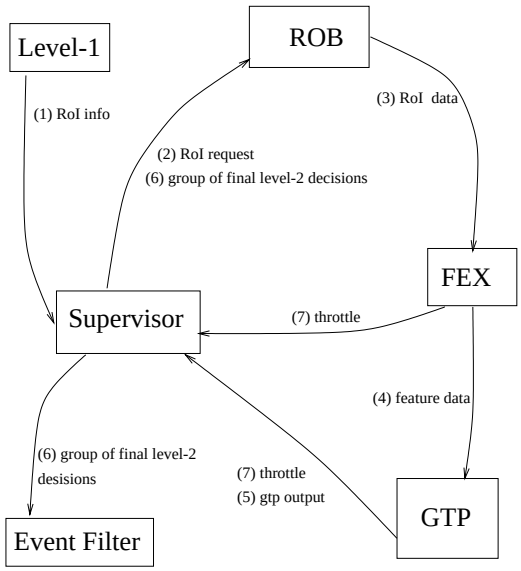


Figure 7.1: Flow diagram of the overall second level trigger system.

Level-2 Trigger Flow Diagram The level-2 trigger flow diagram is shown in figure 7.1. Compared to the sketch of the protocol in chapter 2, figure 2.3, throttle signals are added from the processors to the supervisor. This allows them to tell the supervisor that they have too much work to do. This information permits a prioritised round-robin scheduling of processing power to be implemented. Table 7.1 defines the second level trigger protocol. The numbers in the table correspond to the numbers on the flow diagram.

The Four Main Components The supervisor, the read-out buffer, the feature extractor and the global trigger processor constitute the main components of the level-2 trigger. Here, we will discuss the limitations of our design of these components compared to the final system. In general, these limitations are about eliminating event processing

Message No.	Signal Name	Sender	Receiver	Content
1	RoI information	LVL1	Supervisor	event identifier no. of RoIs list of RoI types RoI data
2	RoI request	Supervisor	ROB	event identifier feature identifier RoI type no. of features no. of ROBs/feature FeX identifier GTP identifier
3	RoI data	ROB	FeX	event identifier feature identifier RoI type no. of features no. of ROBs/feature GTP identifier RoI data
4	Feature data	FeX	GTP	event identifier no. of features feature data
5	GTP output	GTP	Supervisor	event identifier decision GTP output data
6	Level-2 decision	Supervisor	Event Filter ROB	event identifier decision
7	Throttle	Processor	Supervisor	up/down

Table 7.1: The level-2 trigger protocol.

aspects. This is justifiable, as our interest is not in the "number-crunching," but rather in the protocol handling and software design issues.

The emulated supervisor does not convert η, ϕ coordinates and RoI type to read-out buffer identifiers. The information of which read-out buffers are involved in a given event will, instead, be available to the supervisor as static information.

The read-out buffer in the final system, will not only interface to the level-2 trigger but also the event filter and the front-end electronics connected to the detector. For the emulation, only the interface to the second level trigger has been implemented.

Both the emulated feature extractor and the global trigger processors do not perform any physics algorithms. Rather, they are descheduled for a time equivalent to the estimated execution time in a 500 MHz processor; see table 5.1.

Flow Diagrams Figure 7.2 shows the flow diagram for the read-out buffer component. The buffer object receives RoI requests and level-2 decisions from the supervisor and sends RoI data to feature extractors.

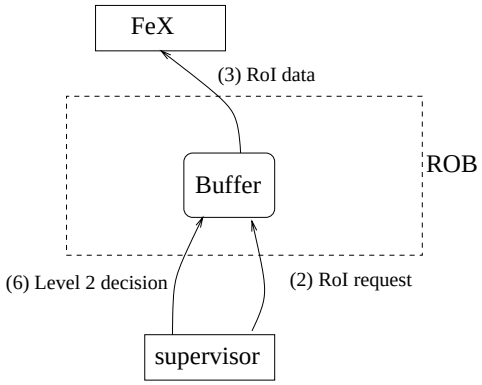
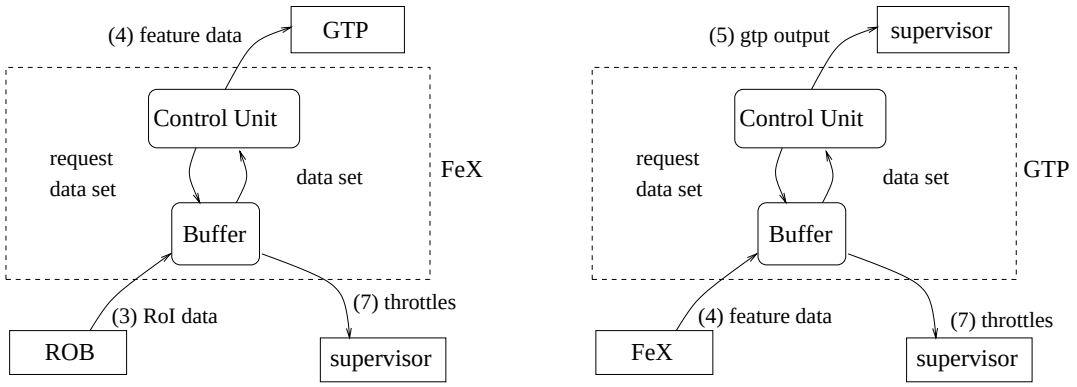


Figure 7.2: Flow diagram of the read-out buffer component.

The feature extractor as well as the global trigger processor component is split into two objects, a buffer object, and a processing object. The buffer object receives the data from the read-out buffer or the FeX and collects the data together. When a complete set of data is assembled, it is transferred to the processing object. The buffering object is also responsible for letting the supervisor know if it cannot handle the load. The task of a processing object is, as the name suggests, to process the data and upon completion send the result to the global trigger processor or supervisor. The flow diagrams for the



(a) The FeX flow diagram.

(b) The GTP flow diagram.

Figure 7.3: Flow diagram of a FeX and a GTP component.

feature extractor and global trigger processor are shown in figure 7.3.

The supervisor component is split up into three separate objects, a Job Issuer (JI), a Job Controller (JC), and a Process Manager (PM). The task of the Job Issuer is to receive the RoI records from the first level trigger, assign processors, and issue RoI requests to the read-out buffers. The task of the Process Managers is to keep track of which processors should be assigned more jobs, and to provide this information to the Job Issuer. The Job Controller receives processed data from the global trigger processors, and decides whether or not the current event should be accepted. This decision is propagated to the read-out buffers and the third level trigger. Figure 7.4 shows the flow diagram for the supervisor.

State Diagrams The following section describes in detail the actions of the individual objects with the help of state diagrams. All the objects can run in parallel on one or more transputers, being synchronised only at points of message passing between objects. The symbol ">" on the state diagrams indicate the start state.

The First Level Trigger The first level trigger object acts as the source for the emulation. Its state diagram, see figure 7.5, contains only one state, the "send data" state. The level-1 trigger object repeats this state, as it tries to send data to the supervisor as fast as possible. The rate will then be determined by the supervisor's capacity, as new data will not be transferred before the old data have been accepted. The shown

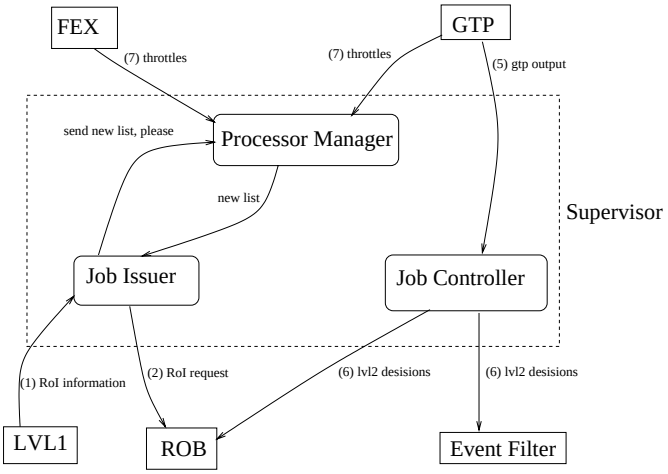


Figure 7.4: Flow diagram of the supervisor component.

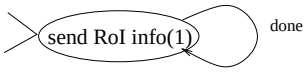


Figure 7.5: State diagram of the first level trigger object.

state diagram is simplified in the send state. Figure 7.6 details the sending state. First, permission to send data is requested. When this has been granted by the receiving process, the data is transfered. Thereafter the process can go back and ask to send more data if it wishes to. For all subsequent shown state diagrams containing a send state, this is what happens in detail.

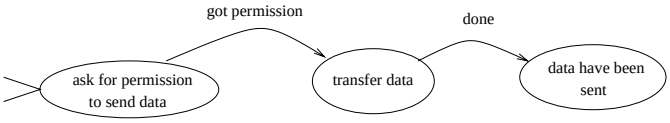


Figure 7.6: State diagram of a process sending data.

The Job Issuer The Job Issuer object receives, as being part of the supervisor component, the RoI records from the first level trigger. Having received a RoI record, the next job is to allocate one or more FeXs according to the content of the RoI record, as well as a GTP for this event. Having sent the RoI requests to the ROBs involved, serially, the Job Issuer object returns and listens for more RoI records. In figure 7.7, the

state diagram of the Job Issuer object is shown. There is no buffering of events inside the Job Issuer.

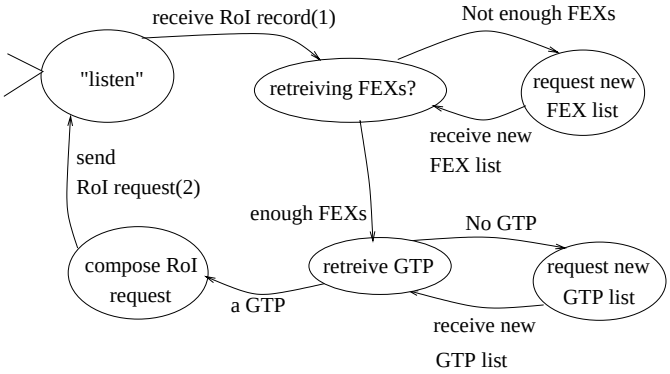


Figure 7.7: State diagram of the Job Issuer object.

The Job Controller The Job Controller object receives recommended level-2 decisions from the global trigger processors, and stores them. When a number of decisions have been collected, the read-out buffers and the event filter are informed. The state diagram for the Job Controller object is shown in figure 7.8.

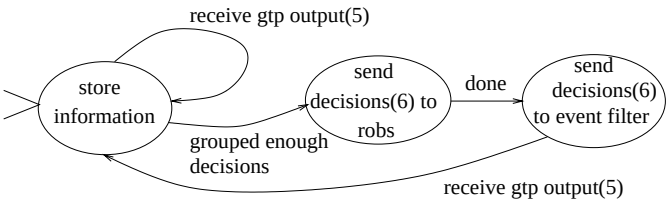


Figure 7.8: State diagram of the Job Controller object.

The Processor Manager The task of the Processor Manager object is to keep track of which processors are available, and upon request, to give this information on to the Job Issuer object. See figure 7.9 for the state diagram of the Processor Manager object.

The Read-Out Buffer The Read-Out Buffer object deals only with the task which is directly linked to the second level trigger, e.g. sending data to feature extractors. The ROB object does not, in any way, emulate getting data from the detector or sending data

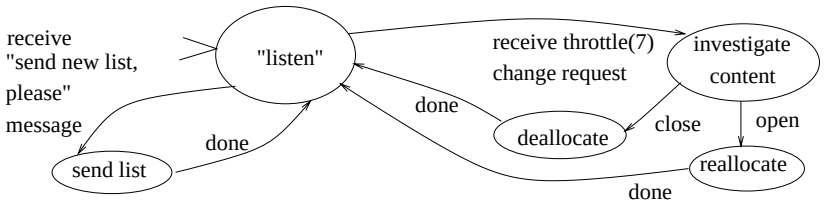


Figure 7.9: State diagram of the Process Manager object.

to the event filter. The state diagram in figure 7.10 shows how the ROB object listens for the "level-2 decision" signal and the "RoI request" signal. In case of receiving the

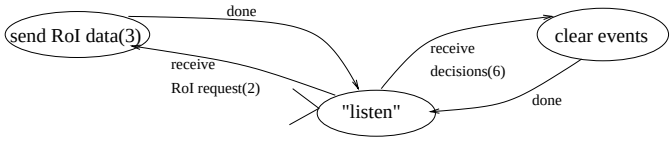


Figure 7.10: State diagram of the read-out buffer object.

first signal, the ROB object pretends to clear the mentioned events; but as no data is really stored this is a "dummy" routine. In the case of the RoI request signal arriving, the ROB sends "dummy" data to the FeX processor specified in the signal. The steering information, e.g. the GTP identifier, is forwarded to the FeX, as well.

The FeX Buffer The FeX buffer object can receive two different signals: a request for more data from the FeX Control object, and RoI data fragments from ROB objects. In the case of the first signal, a flag is raised; in the latter case the RoI data must be stored together with other fragments belonging to the same event and region of interest. Having stored the data, the buffer manager is updated. If no signals are coming in, event data have been requested, and a complete set of event data has been collected in the buffer, then these data are sent to the FeX Control object, and the buffer manager is updated to reflect the new state of the buffer. The state diagram for the FeX buffer object is shown in figure 7.11. In case of space problems in the buffer, the throttle signal is sent from within the "update" state to the processor manager.

The FeX Controller The job of the FeX Controller object, as sketched in figure 7.12 is to request a set of RoI data, then wait to receive the data. Upon receiving the data

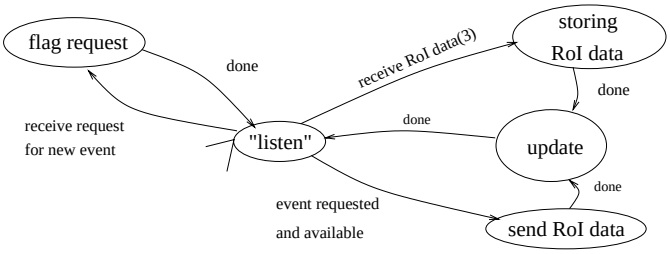


Figure 7.11: State diagram of the FeX buffer object.

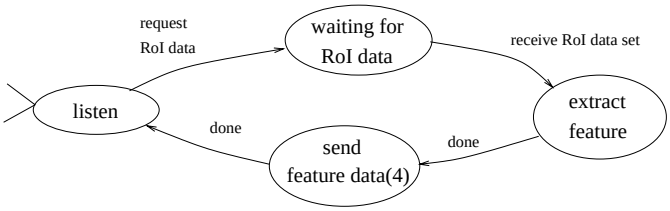


Figure 7.12: State diagram of the FeX controller object.

the FeX Controller object must extract the feature; this is a "dummy" routine during which the object sleeps, i.e. becomes descheduled. Having "extracted" the feature, this feature is sent to the global trigger processor.

The GTP Buffer The state diagram for the global trigger buffer object is shown in figure 7.13. It is very similar to the state diagram for the FeX buffer object presented in figure 7.11. The only real difference is in the type of data it receives, rather than its actions.

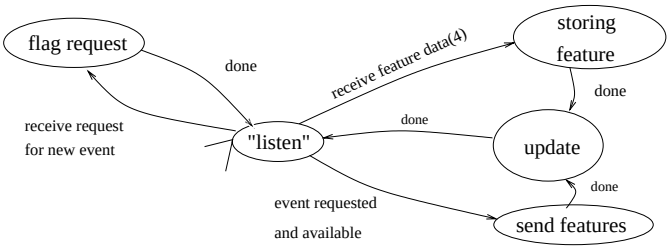


Figure 7.13: State diagram of the GTP buffer object.

The GTP Controller The state diagram for the global trigger controller object is shown in figure 7.14. As with the GTP buffer, the GTP controller is very closely related

to its FeX counterpart.

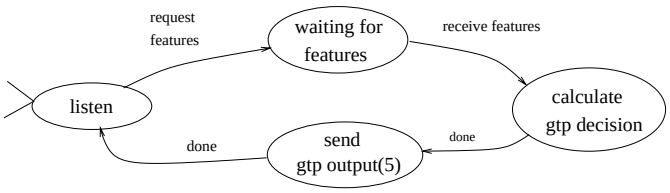


Figure 7.14: State diagram of the GTP controller object.

The Event Filter In the emulation of the second level trigger the Event Filter object acts as a pure sink for the data. This can be seen from the state diagram in figure 7.15, where the "store data" state is a pseudo-state and used for book-keeping rather than emulation purposes.

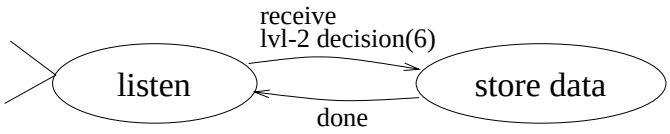


Figure 7.15: State diagram of the event filter object.

7.2 Implementation Issues

This section describes the implementation issues of the level-2 trigger emulation. First the programming language choice and general implementation strategy is explained, then initial benchmarking is discussed. These benchmarks resulted in implementation improvements and additional benchmarks. Finally, a short summary is given of software implementations concerns.

Choice of Programming Language The parallel programming language Occam [26], developed especially for transputers, was chosen for the implementation of the emulation on the GPMIMD machine. The reasons for this choice was that Occam provides an optimal environment for parallel programming of transputers. Occam provides simple constructs for message handling and parallelism; see also section 3.4. The implementation

of the Occam compiler combined with the T9000 transputer ensures both low context switch time, 1.9 μs per context switch, and low message overhead of 2.3 μs [21].

One Object — One Occam Process The implementation of the individual objects and their communication follows from the design. This not entirely obvious, but is mainly due to the choice of programming language, Occam. It allows us to take any of the previously shown state diagrams and implement it once and for all no matter how we later want to place the processes on the nodes in the GPMIMD machine. This means that whether we want the whole emulation to run on a single node or to spread it out over many nodes, does not affect the Occam source code.

```
WHILE TRUE -- forever
  ALT  -- be prepared to handle these two incoming signals

    -- receive level2 decision from Job Controller
    --         using a user defined protocol
    r.lvl2.decision ? num.of.decision.ids::decision.id;
                      num.of.decisions::decision
      -- clear or keep the data
      clear.keep.data( num.of.decision.ids,decision.id,
                       num.of.decisions, decision)

    -- receive RoI request from Job Issuer
    --         using a user defined protocol
    r.RoI.request ? event.id; feature.id; roi.type;num.feature;
                    num.robs.in.roi;fex.id; gtp.id
      SEQ
        Delay(RobDelay)
        -- send RoI data to the appropriate FEX
        --      using a user defined protocol
        s.RoI.data[fex.id] ! event.id; feature.id; roi.type;
                             num.feature; num.robs.in.roi; gtp.id;
                             Output[subD.id][roi.type]::data
```

Figure 7.16: Occam code for the read-out buffer object.

An example of the code corresponding to a state diagram is shown in figure 7.16. This is the Occam code for the ROB object of figure 7.10. The code is for a continuous loop. From the start state there are two possible events which can happen; it can receive one of two messages. The “ALT” construct allows the object to watch two or more conditions

at the same time. Here it watches for the conditions of having received a message. Until one of the messages has arrived nothing happens; as indicated by the state diagram. Having received a message the corresponding action is carried out, and control is placed back in the ALT construct.

The other state diagrams have been implemented using the same technique.

Runtime Procedures In order to be able to measure the latency and rate through the emulation setups, the implementation has been enhanced to start and stop the Occam processes, and print out measurements from them.

The level-1 process is given the task of starting all the processes, while the event filter process is in charge of starting the shut down. These processes were chosen for these tasks because there is only one instance of each and because inherently level-1 is the source of action, and the event filter is the sink.

The first task of the level-1 process is to perform its own initialisation. Thereafter, it sequentially performs a "start dialog" with all the other processes. The "start dialog" consist of the exchange of two messages, a "start command" message from the level-1 process and a "Ok, I start right now" message being returned from the process to the level-1. The process being asked to start must not answer the start dialog before it has performed its own initialisation. When the level-1 process has asked all the other processes to verify that they are ready to start in this way, it starts its own main activity of sending RoI information to the supervisor component.

During the emulation the event filter knows how many events to expect. When having received them all, it waits a short amount of time, e.g. 10 sec. Thereafter it saves to a file, on the UNIX file system, the time-stamps recorded from each received message. Having done its own termination procedure, it dispatches a "stop" message to the level-1 process. The event filter process exits after this communication. The level-1 process is, from here on, responsible for the shut down. In a way similar to the start procedure, the level-1 process conducts a "stop dialog" with each of the remaining processes sequentially. Each process is responsible for saving to disk its output from the emulation during the stop dialog, and afterwards to exit.

All the processes time-stamp incoming messages. After the emulation, these are saved on files on the UNIX file system, as described above. From this information, it is possible

to extract the rate throughput and event latency.

Process to Node Mapping All processes belonging to one component are mapped onto one transputer node, except for the three supervisor processes, where each is mapped onto one transputer node. In the cases of more than one instance of a component, e.g. multiple ROBs, each instance is mapped onto one transputer node.

The level-1, event filter and supervisor components are mapped onto transputer nodes on the first motherboard in the GPMIMD machine. The ROB components are mapped onto transputer nodes on the second to fifth mother board. The FeX components are mapped onto the fifth and sixth mother board, while the GTP components are mapped onto the last motherboard, see figure 3.9 showing the components of the GPMIMD machine.

This means that communication from the level-1 component and to the event filter component travels through only one C104 switch, while the rest of the communication traverses three C104 switches.

Initial Implementation and Improvements After having verified that the implementation did not crash, deadlock, or perform any other obvious misbehaviour, with some minimal configurations, we benchmarked the individual parts of the system. The motivation behind the benchmarks was to predict latency and throughput in the later setups.

In general, the benchmarking was done at the component level. However, for the supervisor, the benchmarking was done individually for the Job Controller and the Job Issuer. The Process Manager has not been benchmarked, as it is not a critical element for the event throughput. The benchmarking of components and objects corresponds to the process to transputer mapping. Both objects in the feature extractor, as well as the global trigger processor are mapped onto one transputer. The three objects constituting the supervisor are mapped onto three transputers. It is considered important for the comparison between benchmarks and complete loop measurement, that the process to transputer mapping be the same.

The general procedure for preparing the benchmarking of a process, or a set of processes, has been to allocate, in addition, the processes which communicate with the process(es) in question. E.g. for benchmarking the level-1 process, the Job Issuer process

must be there to accept the RoI records. Having allocated the "neighbour" processes, these are trimmed such that they do not do more than is necessary. E.g. the Job Issuer in the above example simply receives the RoI info.

The measurements done during the benchmarking are restricted to two readings of the clock in the receiving process, one at start up, before receiving any messages, and one after the expected number of messages is received. Only the difference is reported. By varying the number of expected events through the system it is possible to calculate the throughput. For the benchmarking done here, the average of five measurements with 500 events, and then five measurements with 1000 events were used.

Value of Parameters The motivation for the work described here is to estimate the performance of the raw (physics independent) protocol. Consequently the trigger execution time in the FeX and the GTP is not important. Therefore, the drive-files, as described in chapter 5, have not been used for the benchmarks. The fragment data size from the ROBs, in message no. 3, is set to 40 bytes, and the GTP data output size in message no. 5 is reduced to 1 byte. The RoI data size in message no. 1, and the feature data size in message no. 4 are the same as stated in chapter 5, respectively 206 and 150 bytes. The number of RoI per event was set to one for the benchmarking of the Job Issuer. Ten events were grouped before information about them was sent (message no. 6, the level-2 decision) to the event filter and the read-out buffers for the benchmarking of the Job Controller, ROB and Event filter.

Process(es)	Inverse event rate [μs]
LVL1	70
JI	88+88*rob
ROB	206
FeX	63+112*rob
GTP	49 + 94*f
JC	122
Event Filter	3

Table 7.2: Initial benchmark results to within a micro second.

The results from the benchmarks are listed in table 7.2. The level-1 trigger can provide

an event rate of 14 kHz to the Job Issuer. The Job Issuer uses 176 μs to receive a message from the level-1 process and transmit a RoI request to a ROB. For each additional ROB which needs a RoI request 88 μs is added to the elapsed time per event. The ROB takes 206 μs to receive a RoI request and send 40 bytes of data to a FeX along with the information in the RoI request message. The feature extractor takes 63 μs plus 112 μs per ROB fragment to handle one event. Here the feature extractor is the combined set of the buffer process and the controller process. Similarly for the global trigger processor. The benchmark result for the GTP is 49 μs plus 94 μs per FEX output. The elapsed time for an event to go through the Job Controller is 122 μs. The benchmark for the Event Filter is 3 μs, i.e. 30 μs per group of 10 events.

Looking closer at the benchmark results, for instance, the 206 μs elapsed time for the read-out buffer appears long, for essentially receiving seven integers and forwarding six of them along with 40 bytes of data. However, in estimating the elapsed time for receiving and sending this information, we must take into account the effective link speed. This speed depends on the packet length and the number of switches the packet must traverse.

Table 7.3 gives the values [41] used later in this section, as well as the asymptotic value for large packets.

Packet Length	Throughput
4 Bytes	$0.42 \pm .05$ MBytes/sec
8 Bytes	$0.79 \pm .05$ MBytes/sec
9 Bytes	$0.87 \pm .05$ MBytes/sec
28 Bytes	$2.3 \pm .2$ MBytes/sec
40 Bytes	$1.96 \pm .03$ MBytes/sec
64 Bytes	$2.7 \pm .2$ MBytes/sec
1000 Bytes	$3.02 \pm .01$ MBytes/sec
$>$ 10000 Bytes	$3.08 \pm .03$ MBytes/sec

Table 7.3: The effective link speed using one virtual channel and traversing three switches in network.

When sending to or from the read-out buffer, one virtual channel is used and the packets go through three switches. An Occam facility to define a protocol has been

used, both when receiving and sending. See figure 7.17 for an example of a user defined protocol. The assumption is, that using this facility would provides the most optimal

```
PROTOCOL RoI.request IS -- request from supervisor to ROBs

        INT;          -- event identifier
        INT;          -- identifier of this feature
        INT;          -- roi type
        INT;          -- the number of features in this event
        INT;          -- the number of ROBs in this RoI
        INT;          -- address of FeX in charge of this feature
        INT:          -- address of GTP in charge of this event
```

Figure 7.17: Occam declaration of the user defined protocol "RoI.request."

message passing. As an integer is four byte long, the message received by the ROB is 28 byte long. The message transmitted consists of $6 * 4\,bytes + 40\,bytes = 64\,bytes$ of data. Using the effective link speed values from table 7.3, the expected elapsed time due to message transfers is

$$\frac{28bytes}{2.3 \pm .2Mbytes/s} + \frac{64bytes}{2.7 \pm .2Mbytes/s} = 36 \pm 3\,\mu s. \tag{7.1}$$

The expected time ($36\,\mu s$) is a small fraction of the measured time ($206\,\mu s$) and, as this was the major part of the ROB object activity, we investigated the cause of this difference.

Having gone carefully through our above calculation including assumptions, we came up with the hypothesis that the implementation of the user defined protocols did not match our expectations. We had assumed that the compiler had bundled the integers into one packet. If no such optimisation had taken place and the integers were sent individually, then different link speeds would have to be used in the calculation. The expected elapsed time due to message transfers would then be

$$\frac{(7+6) * 4bytes}{0.42 \pm .05Mbytes/s} + \frac{40bytes}{1.96 \pm .03Mbytes/s} = 144 \pm^{17}_{13}\,\mu s. \tag{7.2}$$

This would account for 70 per cent of the measured time. Checking the assembler output of the ROB object code from the compiler verified our hypothesis that no optimisation had been performed.

Therefore, we abandoned the use of user defined protocols and performed the optimisations which we had expected from the compiler by hand. Unfortunately this resulted in less maintainable code. The option of changing the compiler and hiding the optimisation there was not available.

The benchmark results from this optimised protocol implementation is shown in table 7.4. Considerable improvement is seen everywhere. For the ROB an improvement of $108 \pm^{20}_{16}\ \mu s$ on the elapsed time for the ROB were expected from our calculations above. The measurement show an improvement of $130\,\mu s$.

Further improvements to the the system were made by replacing a linear search for existing fragments of an event by a unique (modulo 100) slot identifier for each event. The event identifier, and in the case of the FeX the feature identifier, is used to calculate the buffer slot identifier according to the equations shown below.

$$\text{slot.id} = ((\text{event.id} * 10) + \text{feature.id}) \text{ modulo } 100 \tag{7.3}$$

$$\text{slot.id} = \text{event.id modulo } 100 \tag{7.4}$$

The first (eq. 7.3) is used in the feature extractor, and the second (eq. 7.4) in the global trigger processor. The buffer repetition factor (modulo 100) was chosen because it left the space needed for the buffers reasonably small, and provided enough leeway to protect against overwrites. By reducing the search routines to a calculation of a slot identifier, the time to search for the slot is constant. The benchmark results after improving the buffer implementations is shown in table 7.4.

The increase in the constant part of the results can be understood as increased extraction time from the buffers. The decrease of the messages dependent part can be attributed to the improved insertion time. The improvement is not in performance, but in scalability.

The conclusion to be drawn from the above mentioned improvements is that it is important to understand the software, especially the communication overhead, and to be aware of any effects which increases the execution time as the size of the system increases.

Process(es)	Inverse event rate [μs]		
	Original Source	Impr. Protocol	No lin. Search
LVL1	70 μs	51 μs	52 μs
JI	88+88*rob μs	56+15*rob μs	65+11*rob μs
ROB	206 μs	76 μs	68 μs
FeX	63+112*rob μs	60+41*rob μs	108 + 33*rob μs
GTP	49 + 94*f μs	55 +68*f μs	68 + 60*f μs
JC	122 μs	76 μs	77 μs
Event filter	3 μs	3 μs	3 μs

Table 7.4: Benchmark results to within a micro second accuracy.

7.3 Test Measurements and Results

This section describes the two test configurations, as well as the motivation behind them. Having presented the setups, the results of the measurements are shown. In the rest of this chapter the improved implementation, as discussed above, will be used.

Two setups have been chosen for the initial measurements. The pipeline has been chosen because its simplicity. The second setup, known as the "5-4-2 loop," which has five ROBs, four FeXs, and two GTP, has been chosen as a small size implementation of a non-trivial system.

The aim of the setups is to ensure that we understand the throughput and latency. The data sizes from the ROBs has been decreased with respect to the "real system," and the traffic pattern through the system has been made simple and systematic.

The Pipeline Setup The configuration of the "Pipeline" consists of one read-out buffer, one feature extractor and one global trigger processor. For a sketch of the configuration, see figure 7.18. The pipeline setup uses the same parameters as were used during the benchmarking. The number of RoIs per event is set to one for all events, and the number of ROBs hit per RoI is also set to one for all RoIs. The measurements were performed with one thousand events.

The "5-4-2 Loop" Setup The configuration of this larger loop, consists of five read-out buffers, four feature extractors and two global trigger processors, see figure 7.18.

The value of the various variables has been kept the same as with the pipeline setup, except that the number of ROBs participating in each feature was five rather than one. The traffic pattern emerging from these values is such that all five ROBs are requested to participate in all events, each sending 40 bytes of data to the current FeX. The local processors each participate in a quarter of the events, while each global processor participates in half of the events. Here also, the measurements were performed with a thousand events.

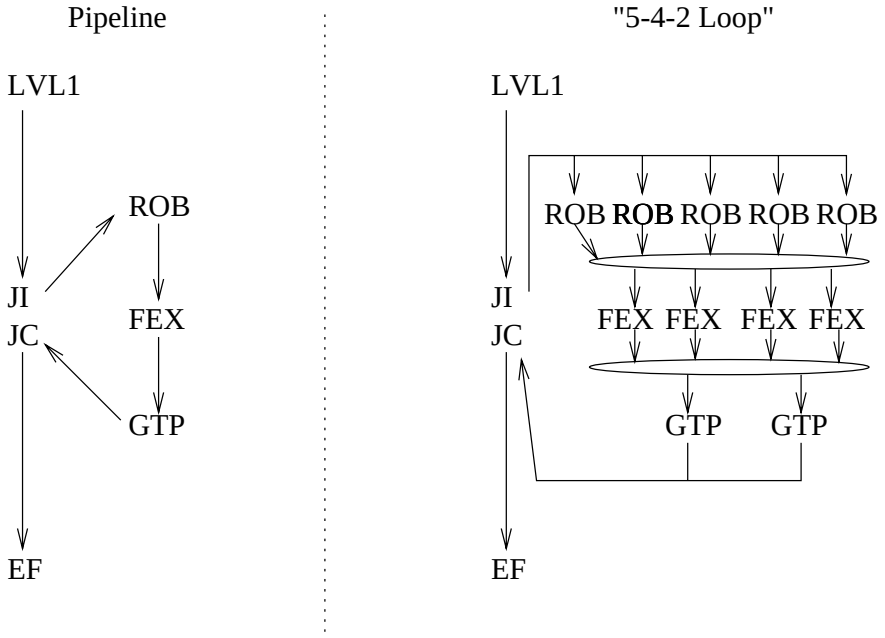


Figure 7.18: Diagrams of the two test setups.

The Results from the above mentioned setups are shown in table 7.5.

Configuration	Latency	Event Rate	Inv. Event Rate
Pipeline	$239.3 \pm .2\,\mu s$	$6.7 \pm .1$ kHz	$149 \pm 2\,\mu s$
"5-4-2 Loop"	$481.6 \pm .5\,\mu s$	$5.6 \pm .1$ kHz	$177 \pm 3\,\mu s$

Table 7.5: Results from the test setups.

The latency is measured from the point at which the RoI information (message no 1)

has arrived at the Job Issuer, to the point where the Job Controller has received the GTP output (message no. 5). The Job Issuer and the Job Controller are both placed on nodes on the same motherboard. Therefore, they share the same clock, and we do not need to worry about asynchronous clocks.

The event rate is measured by subtracting the time-stamp of the first RoI info message (no. 1) from the time-stamp of the last level-2 decision message (no. 6), and dividing this by the number of events. These time-stamps are performed in the Job issuer and the Event Filter processes respectively, which also share the same clock.

Interpreting the Results from the Pipeline First, we will look at the basic numbers. The expected message transfer time over DS-Links can be calculated as in eq. 7.1 and 7.2 using R. Heeley's measurements [41]. The results are shown in table 7.6. The processing time, i.e. the message independent part, for the objects can be calculated by subtracting the message times from the benchmark shown in table 7.4. For example the processing time of the Job Issuer object:

$$P_{JI} = BM_{JI} - (m_2 + m_3) = 11.1 \pm 1.4\,\mu s, \tag{7.5}$$

where BM_{JI} is the benchmark result taken from table 7.4. Table 7.7 shows the values for the processing times. P_{FeX} and P_{GTP} includes the transfer of the message over an internal channel.

No	Signal Name	Time [μs]
m_1	RoI info	$52.7 \pm .3$
m_2	RoI request	12.2 ± 1.1
m_3	RoI data	23.7 ± 1.8
m_4	Feature data	56 ± 2
m_5	GTP output	$10.3 \pm .6$

Table 7.6: Calculated message transfer times.

Object	Time [μs]
P_{JI}	11.1 ± 1.4
P_{ROB}	32 ± 2
P_{FeX}	61 ± 2
P_{GTP}	62 ± 3
P_{JC}	66 ± 2

Table 7.7: Calculated processing times.

The minimum possible latency should be

$$L_{min} = P_{JI} + m_2 + P_{ROB} + m_3 + P_{FeX} + m_4 + P_{GTP} + m_5 = 268 \pm 5\,\mu s \tag{7.6}$$

Already this is in disagreement with the measured latency of 239 μs by 29 μs. In addition, at peak throughput the latency is determined by the latency of the longest stage and its position in the pipeline plus the times of following stages. The latency at peak throughput for an n-stage pipeline with no buffering between stages is given by:

$$\text{Latency} = \max \text{ stage} * t_{\max \text{ stage}} + \sum_{i=1+\max \text{ stage}}^{n} t_i \qquad (7.7)$$

where "t_j" is the time spent in the jth stage of the pipeline, and "max stage" is a stage number such that: $t_{\max \text{ stage}} = \max\{t_j\}$. See the example in figure 7.19.

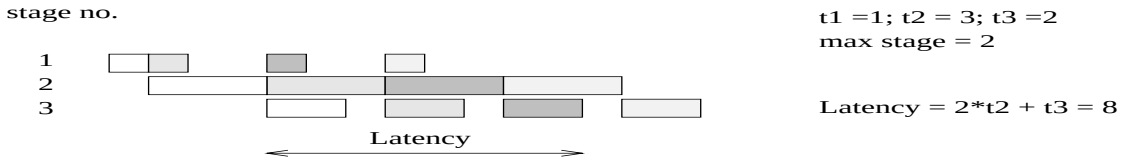


Figure 7.19: Example of latency calculation for a pipeline.

When identifying the independent stages for the pipeline the starting point is the Occam code, as each process runs as a single thread of control. The input and output from a given process, is however, overlapped with the corresponding output and input from the "interacting" processes. Choosing to consider the input and the processing as one independent stage, we get the situation outlined in figure 7.20. The FeX and the

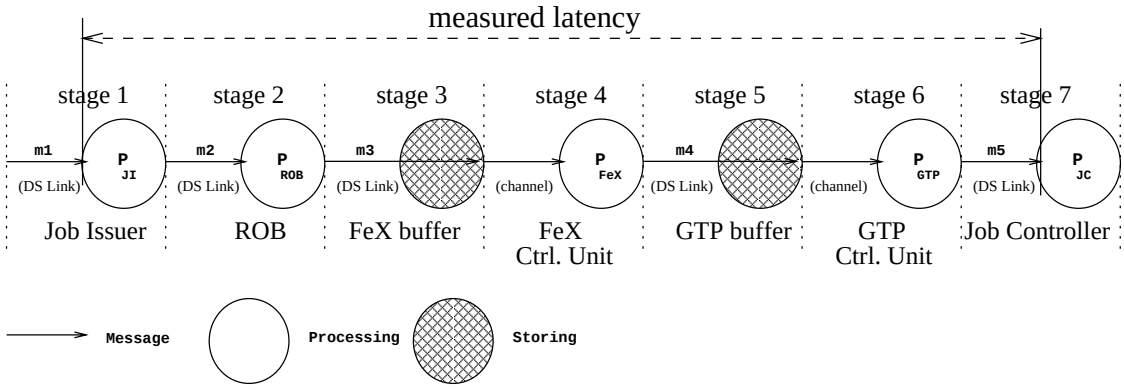


Figure 7.20: Stage definition of the pipeline.

GTP processes differs from the rest, in that they use channel communication from the buffer process to the control unit process within the same transputer, whereas all the other communications are over DS-Links between different transputers. For the buffers,

no processing time is used in the further calculation, and for the ctrl units the message transfer times over the transputer internal channel are not specified but kept as part of P_{FeX} and P_{GTP}. This is because the FeX (and the GTP) has been benchmarked as one unit rather than two. The time for each stage is given by table 7.8

Stage	Time component	Value
t_1	$m_1 + P_{JI}$	$63.8 \pm 1.7\,\mu s$
t_2	$m_2 + P_{ROB}$	$44.2 \pm 2.3\,\mu s$
t_3	m_3	$23.7 \pm 1.8\,\mu s$
t_4	P_{FeX}	$61 \pm 2\,\mu s$
t_5	m_4	$56 \pm 2\,\mu s$
t_6	P_{GTP}	$62 \pm 3\,\mu s$
t_7	$m_5 + P_{JC}$	$76.3 \pm 2.1\,\mu s$

Table 7.8: The definition of the stages and their times.

Assuming the t_7 is not the limiting time as P_{JC} is outside the loop, the time is either t_1 (but m_1 is outside the loop) in which case the latency is 269 or is t_4 at $61\,\mu s$ giving a latency of $4 * 61\,\mu s + 56\,\mu s + 62\,\mu s + 10\,\mu s = 372\,\mu s$. It does not matter at this stage, as both are in disagreement.

To get agreement between the measured latency and the measured times in each process it would be necessary to increase all message times by about 20% or some by larger amounts (note increasing m_1 and m_2 by 20% makes P_{JI} negative).

Turning to the throughput measurement, the inverse event rate is $149\,\mu s$. Throughput is determined by slowest element in the pipeline and this would indicate that the slowest element takes $149\,\mu s$. The assumption has been made that the processes shown in figure 7.20 are running independently. However, in the level-2 demonstrator project [15], P. Maley observed that processes which should have been running in parallel ran sequentially - it required very careful appraisal of the code to verify this. The longest period in one processor is the FEX stage at $141\,\mu s$ in reasonable agreement with the $149\,\mu s$ measured when network delays are added (switches have to be traversed both into and out of the FeX and this adds to the measured process time).

Due to insufficient time and the lack of the necessary equipment it was not possible

to determine the actual message passing times or the precise operation of the code in the processors.

In conclusion, the throughput measured in the pipeline is consistent with the FeX processor running its processes sequentially and providing the dominant delay in the system. The latency measured for the system is inconsistent with the pipeline model and only matches the single event latency if calculated message passing times are too low by about 20%.

Interpreting the Results from the "5-4-2 loop" The minimum latency for an event in the "5-4-2" loop setup should be

$$L_{min} = P_{JI} + 5m_2 + P_{ROB} + xm_3 + P_{FeX} + m_4 + P_{GTP} + m_5, \tag{7.8}$$

where x is $5 - 4\frac{m_2}{m_3}$ since $m_2 < m_3$. With $x = 2.9$, L_{min} becomes $363 \pm 6\,\mu s$. This is significantly lower than the measured latency at peak throughput of $482\,\mu s$. This means that the events are being held back by bottleneck(s) internal to the setup.

Stage	Time component	Value
t_1	$m_1 + P_{JI}$	$63.8 \pm 1.7\,\mu s$
t_2	$5m_2 + P_{ROB}$	$93 \pm 3\,\mu s$
t_3	$5m_3$	$118 \pm 4\,\mu s$
t_4	P_{FeX}	$61 \pm 2\,\mu s$
t_5	m_4	$56 \pm 2\,\mu s$
t_6	P_{GTP}	$62 \pm 3\,\mu s$
t_7	$m_5 + P_{JC}$	$76.3 \pm 2.1\,\mu s$

Table 7.9: The definition of the stages and their times.

Trying to estimate a more realistic latency of the events table 7.9 provides the time for the individual stages in a manner similar to table 7.8. Stage three, the fan-in of the ROB data, is the longest with $118\,\mu s$. As all five ROBs were used for this stage there is no work load sharing. The latency according to the model explained earlier is

$$L = 3 * t_3 + \sum_{i=4}^{7} t_i - P_{JC} = 543 \pm 8\,\mu s. \tag{7.9}$$

This estimate of the latency is in approximate agreement with the measured latency of $482\,\mu s$.

In an effort to estimate the inverse throughput the work sharing between the four FeXs and the two GTPs must be taken into account. The benchmark of the FeX is $(rob * 33 \pm 1\,\mu s + 108 \pm 1\,\mu s)$ according to table 7.4. The message independent part of the benchmark, i.e. the $108\,\mu s$ must, when locating the slowest element, be divided by the number of FeXs. The message part is not shared between the ROBs therefore that part is unchanged. The benchmark for the FeX then becomes:

$$5ROBs * (33 \pm 1\,\mu s) + \frac{(108 \pm 1\,\mu s)}{4FeXs} = 192 \pm 6\,\mu s. \tag{7.10}$$

The two GTPs share the events among them, so also here the benchmark has to be modified. The GTP benchmark then becomes:

$$1feature * (60 \pm 1\,\mu s) + \frac{(68 \pm 1\,\mu s)}{2GTPs} = 94 \pm 2\,\mu s. \tag{7.11}$$

From this it is evident that the FeX becomes the slowest part (with the Job Issuer as the second slowest with $5ROBs * (11 \pm 1\,\mu s) + (65 \pm 1\,\mu s) = 120 \pm 6\,\mu s$). The inverse throughput was measured to be $177 \pm 3\mu s$ in reasonable agreement with the calculated inverse throughput, as given by the FeX benchmark.

In conclusion, the throughput measured in the "5-4-2 loop" is also here consistent with the FeX processor running its processes sequentially and providing the dominant delay in the system. The latency measured for the system is consistent within 11 per cent with the pipeline model.

7.4 SCT Measurement and Results

The previous measurement showed us that the protocol and process model of the second level trigger works, i.e. can handle events correctly etc. Now, an attempt is made to emulate something closer to a realistic system.

The number of components in the full level-2 trigger is much larger than the number of available processors for the emulation. According to chapter 5, there will be 1462 ROBs, see table 5.2, and 506 processors, see table 5.4. Of the 64 processors in GPMIMD there are, however, 8 transputers which are connected to the network via one more layer of C104s, see chapter 3.4 and one of the transputers is dead. This leaves 55 processors for

the emulation. The choice, therefore, lies in either: trying to emulate the whole trigger by adding many processes onto each transputer, or: emulating only a fraction of the trigger by only placing one or two processes per transputer. The first solution will leave us without the possibility to measure latency and throughput, as many processes will be forced to occur in sequence rather than in parallel.

Knowing that we can emulate about two percent of the complete level-2 trigger, the next question is: "Which two per cent?". There are two basic possibilities, to emulate two per cent of every sub-detector, or to emulate a larger fraction of one sub-detector. The first solution will give an unrealistic RoI collection, as very few ROBs per sub-detector will be emulated. The feature collection in the global trigger processor will be reasonably realistic for one of the RoIs, the rest of the RoIs of the event will however most likely not be emulated. The feature collection will therefore also not be realistic. By choosing the possibility of emulating only one sub-detector, at least the RoI collection in the feature extractor will be fairly realistic. The feature collection in the global trigger processors will, however be unrealistic — as features from only one sub-detector will reach it.

The SCT sub-detector was chosen for the emulation on the grounds that the SCT is not too large (256 ROBs, 80 FeX) in terms of components, it acts on half the RoIs, and from measurements on Macramé, it is known as a "typical" sub-detector. The TRT (512 ROBs, 118 FeXs) and the Calorimeter (480 ROBs, 286 FeXs) are both very large. The muon spectrometer is small (214 ROBs, 10 FeXs) and acts only on 10 % of the RoIs.

As we can emulate only about one eighth of the SCT local farm, edge effects result from RoIs not being fully within the emulated part. These effects must be small enough that the throughput can be scaled to the full SCT subfarm. The scaled throughput can then be compared to the event rate expected in the SCT. The expected rate can be extracted from the drive file.

Description of the SCT Setup In determining how large a fraction of the SCT local farm we can emulate, the placement of processes must be considered carefully. With 55 transputers we can emulate about sixteen per cent of the SCT ($55\,\text{processors}/(256\,\text{ROB} + 80\,\text{FeX} + 12\,\text{GTP}) = 16\%$). The first five transputers on the first mother board are used for the supervisor (three), the level-1 trigger (one) and the event filter (one). This leaves 50 transputers to be shared between the SCT ROBs, the SCT FeX, and the GTP. To

ensure that all messages have to pass through the same number of C104 switches, the supervisor processor can not be on the same mother board as ROBs or GTPs, and FeXs can not share mother boards with ROBs and GTPs. To meet these requirements we chose to emulate a higher proportion of SCT ROBs than SCT FeXs.

The chosen configuration is 39 SCT ROBs (15 %), 9 SCT FeXs (11%) and 2 GTPs (16%). They have been placed as shown in figure 7.21.

mother board: \ transputer slot:	1	2	3	4	5	6	7	8
1		**L1**	**EF**	**JI**	**JC**	**PM**	**F**	**F**
2		**F**	**F**	**F**	**F**	**F**	**F**	**F**
3		**R**	**R**	**R**	**R**	**R**	**R**	**R**
4		**R**	**R**	**R**	**R**	**R**	**R**	**R**
5		**R**	**R**	**R**	**R**	**R**	**R**	**R**
6		**R**	**R**	**R**	**R**	**R**	**R**	**R**
7		**R**	**R**	**R**	**R**	**R**	**R**	**R**
8		**G**	**G**	**R**	**R**	**R**	**R**	†

L1: level-1
EF: Event Filter
JI: Job Issuer
JC: Job Controller
PM: Processor Manager
R: SCT ROB
F: SCT FeX
G: GTP
empty
† **dead**

Figure 7.21: Placement of the processes for the SCT configuration.

The thirty-nine SCT ROBs must be chosen from a cone originating from the centre of ATLAS, to ensure that RoIs have a possibility to be completely within the emulated section. A RoI in the SCT hits between two and eight ROBs, see figure 5.3. A RoI should therefore have a good possibility to be completely within the emulated cone. The cone chosen has a direction towards the cavern ceiling, and is defined by $\eta \in [-0.8; 0.8]$ and $\phi \in [5.24; 1.05]$

We chose a subset of the events in the drive file where all ROBs from at least one RoI hit the thirty-nine SCT ROBs emulated. The number of events chosen depends on how many events can be stored on the transputers. Table 7.10 lists essential numbers from the subset of events from the drive file.

Event Characteristics	
# Events emulated	2166
$ROBs$/feature	3.32
$ROBs$/event	3.76
$RoIs$/event	1.13

Table 7.10: Characteristics of the chosen events.

The results from the emulation of the SCT local farm are shown in table 7.11 and the latency distribution is shown in figure 7.22.

Configuration	Latency	Event Rate	Inv. Event Rate
SCT setup	$4284 \pm 23\,\mu s$	$1.8 \pm .1$ kHz	$569 \pm 4\,\mu s$

Table 7.11: Results from the SCT emulation.

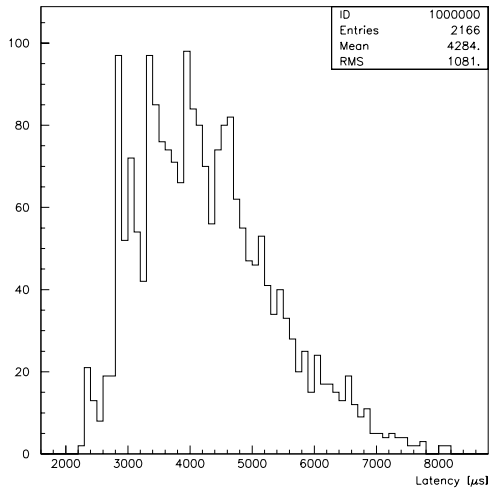


Figure 7.22: The latency distribution for the SCT emulation.

Interpreting the Results from the SCT Setup The average latency of the events through this system is more difficult to estimate than for the test setups, as there are many parallel paths in this setup. The minimum latency can, however, be estimated

by adding up the time spent in each component. Referring to figure 7.20 the minimum latency is:

$$\text{Latency}_{min} = P_{JI} + m_2 + P_{ROB} + m_3 * \frac{\#ROB}{RoI} + P_{FeX} + m_4 * \frac{\#RoI}{event} + P_{GTP} + m_5 \quad (7.12)$$

where m_2 and m_4 are unchanged from table 7.6. The values of m_3 and m_5 are determined according to equations 7.13 and 7.14.

$$m_3 = \frac{1000bytes}{3.02 \pm .01 Mbytes/s} - \frac{40bytes}{1.96 \pm .03 Mbytes/s} = 311 \pm 4\,\mu s. \quad (7.13)$$

$$m_5 = \frac{1000bytes}{3.02 \pm .01 Mbytes/s} = 331 \pm 1\,\mu s. \quad (7.14)$$

P_{JI} and P_{ROB} are unchanged from table 7.7. To P_{FeX} and P_{GTP}, the processing time is added, $500\mu s$ and $150\mu s$ respectively. This amounts to a minimum latency of:

$$\begin{aligned} \text{Latency}_{min} &= 11.1 \pm 1.4\,\mu s + 12.2 \pm 1.1\,\mu s + 32 \pm 2\,\mu s + \\ &\quad 311 \pm 4\,\mu s * 3.32 + (500 + (61 \pm 1))\,\mu s + 56 \pm 2\,\mu s * 1.13 + \\ &\quad (150 + (62 \pm 3))\,\mu s + 311 \pm 1\,\mu s \\ &= 2235 \pm 9\,\mu s \end{aligned} \quad (7.15)$$

The minimum latency measured is $2.3ms$ in good agreement with that calculated.

The throughput is determined by the slowest element in the setup. To take into account the parallelism in the system, the processing and message part must be divided by the available capacity.

The slowest element in the setup is the GTP control unit, with

$$P_{GTP}/2 + m_5 = (150 + 62)\mu s/2 + 311\mu s = 417 \pm 4\,\mu s. \quad (7.16)$$

Thus, we expect the SCT emulation to have a throughput of 2.4 kHz ($1/417\mu s$). However, the throughput was measured to be 1.8 kHz.

In an effort to find the bottleneck experimentally various parameters in the setup was varied. By this procedure we found that by halving the amount of data being sent from the ROBs to the FeXs the inverse throughput increased to $435\mu s$; consistent with the Job Controller being the bottleneck. Further investigations of the trace files confirm that the Job Controller is the bottleneck under these conditions.

For the combined level-2 trigger system to operate at 100 kHz the various bottlenecks, starting with the Job Controller, must be eliminated. Getting the scale factor, needed

for the emulation to deal with the full rate for all sub-detectors, will need full-sized simulation/Monte Carlo rather than just saying: "we were capable at running at 1.8 kHz, therefore we just need a system 55 times larger."

The simple implementation of the supervisor in the emulation with its low performance is not a problem for the complete system, as the supervisor has already received special attention. In 1996 a prototype hardware realisation of the supervisor was running at CERN [42, 15]. This prototype shows that a realisation of the supervisor is possible, and therefore not a major concern of this emulation.

In conclusion, the minimum latency measured was in good agreement with the expected. The throughput model did however not show good agreement, and would need much work before it could be used reliably for large systems.

7.5 Conclusion

The emulation of the second level trigger on the 64 node GPMIMD machine shows that the protocol is robust against variations in the size of the setup and the traffic.

The measurements on the two small test setups, showed that simple models for the throughput and the latency gave a reasonable approximation to the measured parameters. The SCT emulation showed that for more complex setups these simple models breaks down, at least for the throughput. Here, more sophisticated simulation work is necessary. Using the variable parameters in the emulation it is, however, possible to locate bottlenecks and in other respects try to understand the performance.

To summarise the performance of the emulation setups: the pipeline and the small setup achieved event rates of 6.7 kHz and 5.6 kHz respectively, while the SCT setup achieved 1.8 kHz — seven times less than required for the 15 % of the SCT emulated.

The most significant points when designing and implementing the trigger system is that the system must use parallel processing wherever possible and that the software must be scalable.

The final system must operate with many processes running in parallel on both the same and different processors. It will be important to ensure that these processes do indeed run in parallel and are not constrained to operate sequentially.

The software must be fully scalable. Any part of the system where the processing time increases with system size is very likely to create a bottleneck in the full size system.

Chapter 8

Conclusion

This thesis has presented the results of the emulation for one possible architecture of the ATLAS level-2 trigger system, and the produced network diagnostic software, Netprobe. The level-2 trigger will be a vital part of the ATLAS experiment. To increase our knowledge of this subsystem and to help future design decisions the system was emulated in two complementary ways: the pure network aspects on the Macramé test-bed and software and protocol issues on the GPMIMD machine. Both used point-to-point DS-Link technology. To facilitate the use of large DS-Link networks, a reliable diagnostic software tool was needed to ensure the correct configuration/wiring etc. For this use, Netprobe was developed.

The construction of Netprobe improved the facilities, performance and user friendliness of working with DS-Link networks. Netprobe played a crucial role for the work on the Macramé test-bed. Currently, it is being used in the ARCHES test-bed and in the L3 LEP experiment's second level trigger.

Experience with large networks has demonstrated the absolute necessity of this kind of software utility if network problems are to be located and fixed quickly, as is required in a dead-time-sensitive environment such as a high energy physics experiment.

The emulation of the individual networks of the ATLAS second level trigger local-global architecture on the Macramé test-bed showed a variety of things. Distributing the sources and sinks over the switching network makes a sixty per cent difference in the network throughput in the case of the SCT, and increases the achievable event rate by

up to fifty-six per cent.

The 512 node Clos network could sustain traffic in excess of a 100 kHz event rate from the SCT, from 80 % of the TRT, from the complete muon spectrometer, and from the global network. Each level-2 network was emulated separately. The emulation of the calorimeter local network achieved an event rate of 80 kHz by using three traffic nodes per hadron calorimeter read-out buffer.

The shape of the single packet latency distribution across the SCT, the TRT and the global network was understood by a simple queueing model. The model was not applicable to the other networks due to its use of a constant packet size.

The emulation of the second level trigger on the 64-node GPMIMD machine shows that the protocol is robust with regard to various sizes of the setup and the traffic.

Measurements on two small test setups, showed that simple models for the throughput and the latency gave a reasonable approximation to the measured parameters. Fifteen per cent of the SCT was emulated, showing that for larger setups these simple models break down. Using the variable parameters in the emulation it was, however, possible to locate bottlenecks and in other respects try to understand the performance.

The pipeline and the small setup (5 ROBs, 4 FeXs, 2 GTPs) achieved event rates of 6.7 kHz and 5.6 kHz respectively. The emulation of fifteen per cent of the SCT achieved 1.8 kHz, or seven times less than required for the final SCT system.

The most significant points when designing and implementing the trigger system can be summarised as:

- The final system must operate with many processes running in parallel on both the same and different processors. It will be important to ensure that these processes do indeed run in parallel and are not constrained to operate sequentially.

- The software must be fully scalable. Any part of the system where the processing time increases with system size is very likely to create a bottleneck in the full size system.

- The network/processing nodes must be easily configurable and diagnosable, such as to limit down time due to reboots and hardware debugging.

- The assignment of processing nodes and read-out buffers to the network interfaces must utilise a distributed approach, such as to ensure an efficient usage of the network.

The DS-Link technology has currently not found a market which will guarantee its availability in the long term. The ATLAS level-2 trigger community is not expected to make a choice as to which technology to use until June 2001. However, even if the technology studied in this thesis will not be available or optimal, we have still shown that point-to-point technology networks are a feasible solution to the network problem.

Appendix A

The Netprobe Commands

A.1 General Commands

connect

Name connect - connects to a DS-Link host

Synopsis connect <host>

Description The program attempts to connect to the host specified.

disconnect

Name disconnect - disconnects Netprobe from a DS-Link network host.

Synopsis disconnect

Description If Netprobe is connected to a DS-Link network host it closes the connection to the host. If it is not connected, it does nothing.

dbg

Name dbg - turn on debug mode

Synopsis dbg

Description Makes the calls to the network maximally verbose. Can only be called after a connection to a network has succeeded.

info

Name info - turn on info mode

Synopsis info

Description Makes the calls to the network more verbose. Can only be called after a connection to a network has succeeded.

starttrace

Name starttrace - Start tracing the calls to the DS-Link network

Synopsis starttrace <filename>

Description Start the tracing of all calls to the DS-Link network. The trace is appended to the file specified.

stoptrace

Name stoptrace - Stop tracing the calls to the DS-Link network

Synopsis stoptrace

Description Stop the tracing of all calls to the DS-Link network.

quit

Name quit - quit Netprobe

Synopsis quit, Quit, Q or q

Description If Netprobe is still connected to a network it disconnects before it exits.

help

Name help - online help

Synopsis help [<command name>]

Description If no command name is given a help page is displayed, if a command name has been specified a help page on this specific command is given.

See also the registerhelp command section A.1.

registerhelp

Name registerhelp - get help on register content etc.

Synopsis registerhelp [<registername>]

Description Gives online help on all DS-Link registers, if no register name is specified a list of the configuration registers is given.

See also the help command A.1.

A.2 Local Commands

start

Name start - send start command to next device

Synopsis start [<headerin> <headerout>]

Description Netprobe sends a start command to the next device with either the header specified or the default headers. Default "headerin" is deviceId, default "headerout" is (0x8000+deviceId).

When Netprobe is connected to a network, the first device can be started by the start command. Before one can start the next device, the control chain must be started. If the devices are daisy chained, it is adequate to start the control down link, this can be done by "cpoke clnkc 1 s." Now, the next device can be started. On the other hand, if the devices are connected with a control fan-out the appropriate STC104 links must be started and the interval registers must be set.

If the user wish to start the whole network automatically, this can be done by issuing the spy, verify or configure command, see section A.3 for further details.

device

Name device <device_no> - select current device

Synopsis device <device_no>

Description This command is used to select a new device, which must be started.

identify

Name identify - send identify command

Synopsis identify

Description Prints the identity of the device in decimal. The correspondence between device id and device type is summarised in table A.1.

Device Id	Device Type	Revision
300	T9000	Alpha
301	T9000	Beta
302	T9000	Gamma
320	C100	—
340	C100	—
384	C104	—
13	RCube	—

Table A.1: Correspondence between device id and device type.

reset

Name reset - send reset command

Synopsis reset <level>

Description Reset the device according to the level specified. See table A.2 for details on the reset levels.

Level	Description
0	hardware reset
1	labelled control network
2	configured network
3	booted network (only T9000)

Table A.2: Reset levels.

cpeek

Name cpeek - send cpeek command, print return value

Synopsis cpeek {<register-name> [<interval number>] [<link>] | <address> }

Description This command reads the specified register in the configuration space.

See also The online help pages which can be accessed through the command "register-help" see section A.1 for further details.

cpoke

Name cpoke - send cpoke command

Synopsis cpoke {<register-name> [<interval number>] [<link>]
{<bit-pattern>|<value>} | <address> <value>}

Description This command writes the specified value into the specified register in the configuration space.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

peek

Name peek - send t9000 peek command, print returned value

This command is only supported in the Irun/B103 environment.

Synopsis peek <address>

Description Reads the specified memory address of the T9000.

poke

Name poke - send t9000 poke command

This command is only supported in the Irun/B103 environment.

Synopsis poke <address> <value>

Description Writes into the memory of the T9000

systemservicereg

Name systemservicereg - cpeek all system service registers

Synopsis systemservicereg

Description Cpeek all system service registers.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

ctrllnkreg

Name ctrllnkreg - cpeek all control link registers

Synopsis ctrllnkreg [<link>]

Description Cpeek the mode and status registers for the specified control link.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

datalnkreg

Name datalnkreg - cpeek all data link registers

Synopsis datalnkreg [<link>]

Description Cpeek the data mode and status registers for the specified data link. If no link is specified all data links are listed.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

intervalreg

Name intervalreg - cpeek all interval registers

Synopsis intervalreg [<link>]

Description Cpeek all 36 interval registers for the specified data link. If no link is specified all data links are listed.

See also The online help pages which can be accessed through the command "register-help¡" see section A.1 for further details.

packetreg

Name packetreg - cpeek the mode and the command packet registers

Synopsis packetreg [<link>]

Description Cpeek both the command packet register and the mode packet register for the specified data link. If no link is specified all data links are listed.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

randomreg

Name randomreg - cpeek all randomize registers

Synopsis randomreg [<link>]

Description Cpeek all randomize registers for the specified data link. If no link is specified all data links are listed.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

allpacketreg

Name allpacketreg - cpeek all packet registers

Synopsis allpacketreg [<link>]

Description Cpeek all randomize registers, all interval registers, the packect command and mode registers for the specified data link. If no link is specified all data links are listed.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

allreg

Name allreg - cpeek all registers

Synopsis allreg [<link>]

Description Cpeek all the configuration registers.

See also The online help pages which can be accessed through the command "register-help," see section A.1 for further details.

recovererror

Name recovererror - send recover from link error

Synopsis recovererror

Description Restores the protocol after a link error in the control system.

boot

Name boot - send t9000 boot command

This command is only supported in the Irun/B103 environment.

Synopsis boot <length> <address>

Description A boot sequence is started. The length variable is the length of the boot code to be loaded and the address variable indicates where in the memory the boot code is written.

bootdata

Name bootdata - send t9000 boot data command

This command is only supported in the Irun/B103 environment.

Synopsis bootdata <int1> <int2> <int3> <int4>

Description Send boot data to the T9000. The data will be written to the address specified by a previous boot command.

reboot

Name reboot - send T9000 reboot from read only memory (ROM) command

This command is only supported in the Irun/B103 environment.

Synopsis reboot

Description The T9000 reboots from ROM.

run

Name run - send t9000 run command

This command is only supported in the Irun/B103 environment.

Synopsis run <Wptr> <Iptr>

Description The run command causes the processor to start executing with a workspace pointer (Wptr) and an instruction pointer (Iptr).

stop

Name stop - send t9000 stop command

This command is only supported in the Irun/B103 environment.

Synopsis stop

Description Stop the processor "cleanly" so that register values are preserved for debugging.

A.3 Global Commands

hardreset

Name hardreset - hardware reset network

Synopsis hardreset

Description Performs a hardreset on all devices and links in the network.

labelled

Name labelled - obtain the labelling of the network

Synopsis labelled [< numdevices >]

Description Netprobe asks "numdevices" to identify themselves. If the device responds, it's labelling information is requested. If no number of devices is specified, Netprobe simply ask devices to identify themselves until it fails. This failure will be reported to the screen in the same way as any other error message from the network.

This command can be used when Netprobe is accessing a network which has already been labelled.

list

Name list - return current device information

Synopsis list

Description Displays a list of the devices Netprobe is aware of. For each device the following is shown: the type of the device, the headers in and out of the device and the current state of the device. The state information of the device is described in table A.3.

spy

Name spy - spy the network

Synopsis spy [options]

State	Description
Labelled	There is access to the device.
Booting	A boot command has been sent to this device.
Running	A reboot or a run command has been sent to this device.
Stopping	A stop command has been sent to this device.

Table A.3: The state of a device.

Description The spy command explores the configuration of the network.

Options

SI	display information
ST {filename}	saves the final table to a file
VS	print status of network after each new device
FS	print final status of network
C100	give details on C100 connections
V	print status of network as a list, default is a table
CS {speed}	specify link speed for control links
DS {speed}	specify link speed for data links
SLOW	use the slow algorithm
GNDL {filename}	generate an ndl file
ROM	read and display Htram rom information (SGS-Thomson Htrams only)

verify

Name verify - verify the network

Synopsis verify <verifyfile> [options]

Description The verify command takes a network description (NDL) file or a previous spy output table and checks if the network contains what is specified in the verify file.

Options

NDL	the verify file is an NDL file and not a spy table
STRICT	verify the network strictly i.e. also check that unused links are in fact not used

IE	ignore the edges
SI	display information
VS	print status of network after each new device
FS	print final status of network
V	print status of network as a list, default is a table
CS {speed}	specify link speed for control links
DS {speed}	specify link speed for data links
GNDL {filename}	generate an ndl file

configure

Name configure - configure the network according to a description within a file

Synopsis configure <filename> [options]

Description This command enables the user to configure a network according to an NIF or NDL file. Networks containing T9000s in the control chain can only be configured from a NIF file.

Options

SI	display information
SEED {seed}	seed for random number generator. It is only used when the random registers are set.

See also The verify command.

btl

Name btl - load the network

This command is only supported in the Irun/B103 environment.

Synopsis btl <bootfile>

Description Load the network as specified in the bootfile.

See also The serve command section A.3.

serve

Name serve - serve on disconnect

This command is only supported in the Irun/B103 environment.

Synopsis serve

Description Serve starts serving the network as soon as Netprobe disconnects from the network. This command is needed when the user wants to start running an application on the T9000s.

See also The btl command section A.3.

halt

Name halt - halt processors

This command is only supported in the Irun/B103 environment.

Synopsis halt

Description Halt all the processors in the network.

Appendix B

Example NDL description

The following code is an example of an NDL file. It describes a single C104 switch (labelled "Switch" below) which has three links connected to terminal nodes. The connections to the nodes are described as "edges" in the NDL. See figure B.1 for a diagram of the network.

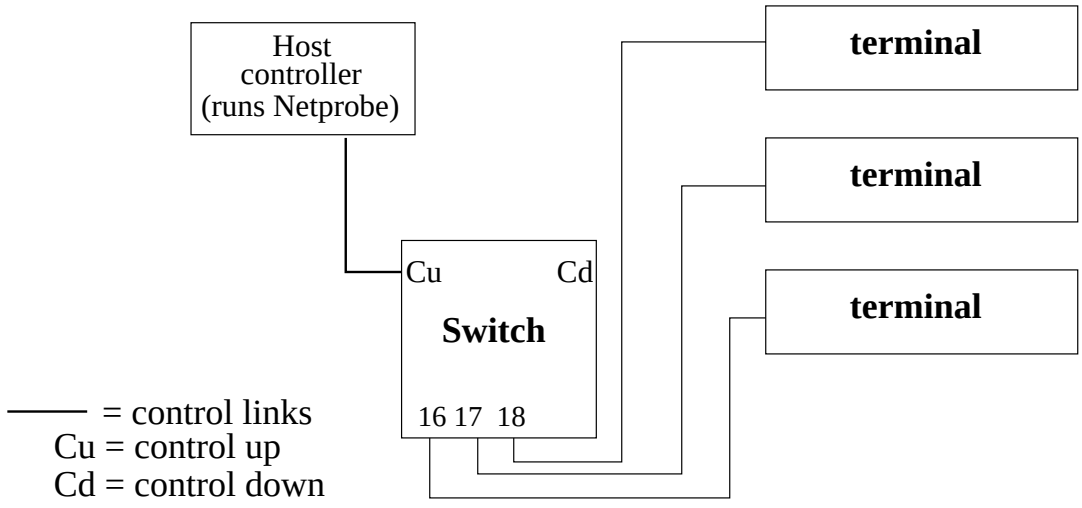


Figure B.1: Diagram of the network.

```
-- Declaration of the connection to the host
CONTROLPORT host:
ARC HostCLink:

[1]NODE Switch:
[3]EDGE terminal:
NETWORK
  DO
    -- Set all default link speeds
    SET DEFAULT (link.speed.multiply  := 10)
    SET DEFAULT (link.speed.divide    := [1])
    SET DEFAULT (control.speed.divide := [2])

    -- Definition of the types of the nodes used in this network
    SET Switch[0] (type := "C104")
    SET Switch[0] (link.speed.multiply  := 20)

    -- Connections between devices
    CONNECT Switch[0][control.up] TO host[control] WITH HostCLink
    CONNECT Switch[0][link][16] TO terminal[0]
    SET Switch[0] (delete[16] := TRUE)
    CONNECT Switch[0][link][17] TO terminal[1]
    SET Switch[0] (delete[17] := TRUE)
    CONNECT Switch[0][link][18] TO terminal[2]
    SET Switch[0] (delete[18] := TRUE)

    -- Interval labelling
    SET Switch[0] (interval.separator[0] := 0)
    SET Switch[0] (link.select[0] := 16)
    SET Switch[0] (interval.separator[1] := 1)
    SET Switch[0] (link.select[1] := 17)
    SET Switch[0] (interval.separator[2] := 2)
    SET Switch[0] (link.select[2] := 18)

:
```

Appendix C

Trigger Menu

The following listing is of the high-luminosity trigger menu for the first level trigger. For each level-1 (LVL1) trigger item, a set of level-1.5 (LVL1.5) trigger items is shown when appropriate. The LVL1 trigger items show primary RoIs. The LVL1.5 trigger items also display the secondary RoIs. The level-2 supervisor gets information about the RoIs associated with the LVL1.5 trigger item.

The component of the trigger items are to be interpreted the following way. MUx, is a muon with a transverse momentum greater than x GeV/c. EMx, is a electron or a photon with a transverse energy greater than x GeV. Jx, is a jet with a transverse momentum greater than x GeV/c. If the component is followed by an I, then the particle or jet must also be isolated.

		Inclusive	Exclusive (Hz)
LVL1	MU6 + EM20I	3000	3000
Total exclusive			3000
LVL1	MU6 + MU6	1000	-
LVL1.5	MU6 + MU6 + J40	1000	328
LVL1.5	MU6 + MU6 + J40 + J40	222	188
LVL1.5	MU6 + MU6 + J40 + J40 + J40	34	34
LVL1.5	MU6 + MU6 + EM20I	300	200
LVL1.5	MU6 + MU6 + EM20I + EM20I	50	50
LVL1.5	MU6 + MU6 + MU6	200	200
Total exclusive			1000
LVL1	MU20	4000	-

LVL1.5	MU20 + J40	4000	2811
LVL1.5	MU20 + J40 + J40	889	752
LVL1.5	MU20 + J40 + J40 + J40	137	121
LVL1.5	MU20 + J40 + J40 + J40 + J40	16	16
LVL1.5	MU20 + EM20I	300	280
LVL1.5	MU20 + EM20I + EM20I	20	20
Total exclusive			4000
LVL1	EM30I	20000	-
LVL1.5	EM30I + J40	20000	12223
LVL1.5	EM30I + J40 + J40	4444	3760
LVL1.5	EM30I + J40 + J40 + J40	684	604
LVL1.5	EM30I + J40 + J40 + J40 + J40	80	80
LVL1.5	EM30I + EM10	3333	3333
Total exclusive			20000
LVL1	EM20I + EM20I	4000	-
LVL1.5	EM20I + EM20I + J40	4000	1511
LVL1.5	EM20I + EM20I + J40 + J40	889	752
LVL1.5	EM20I + EM20I + J40 + J40 + J40	137	137
LVL1.5	EM20I + EM20I + EM10	1600	1600
Total exclusive			4000
LVL1	J150	3000	-
LVL1.5	J150 + J40	3000	2183
LVL1.5	J150 + J40 + J40	667	565
LVL1.5	J150 + J40 + J40 + J40	102	102
Total exclusive			2850
LVL1	ME100	1000	850
LVL1.5	ME100 + J150	150	150
Total exclusive			1000
LVL1	misc. prescaled	5000	-
LVL1.5	J40 + J40	5000	4227
LVL1.5	J40 + J40 + J40	773	682

LVL1.5	J40 + J40 + J40 + J40	91	91
Total exclusive			5000
Total LVL1 rate		40850 Hz	
Total LVL1.5 exclusive rate			40850 Hz

Appendix D

Associated Publications

- R.W. Dobinson, S. Haas, R. Heeley, N.A.H. Madsen, B. Martin, J.A. Strong and D.A. Thornley. *Evaluation of network performance for triggering using a large switch.* Contributed paper to CHEP98, Illinois, Chicago. August 1998.

- R.W. Dobinson, R. Heeley, N.A.H. Madsen, B. Martin and D.A. Thornley. *Netprobe — D 1.4.1 Test/diagnostic software and Documentation for IEEE 1355 DS link networks.* June 1997. `http://home.cern.ch/~madsenn/netbook.ps`

- M. Zhu, D.A. Thornley, J. Pech, B. Martin, N.A.H. Madsen, R. Heeley, S. Haas, R.W. Dobinson and C.R. Anderson. *Realisation and Performance of IEEE 1355 DS and HS Link Based, High Speed, Low Latency Packet Switching Networks.* IEEE TRANSACTIONS ON NUCLEAR SCIENCE p. 1849-1853 Volume 45 no 4. August 1998. `http://www.cern.ch/HSI/dshs/`

Bibliography

[1] ATLAS Collaboration. *ATLAS Letter of Intent for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*. CERN, 1992. CERN/LHCC/92-4.

[2] ATLAS Collaboration. *ATLAS Technical Proposal*. CERN, December 15th 1994. CERN/LHCC/94-43.

[3] Run II handbook, 1998. `http://www-bd.fnal.gov/lug/runII_handbook/RunII_index.html`.

[4] Particle Data Group. *Particle Physics Booklet*. Springer, July 1998.

[5] ATLAS Inner Detector Community. *Inner Detector, Technical Design Report*, volume 1. CERN, April 30th 1997. CERN/LHCC/97-16.

[6] ATLAS Inner Detector Community. *Inner Detector, Technical Design Report*, volume 2. CERN, April 30th 1997. CERN/LHCC/97-17.

[7] ATLAS Collaboration. *Calorimeter Performance, Technical Design Report*. CERN, January 13th 1997. CERN/LHCC/96-40.

[8] ATLAS LARG Unit. *Liquid Argon Calorimeter, Technical Design Report*. CERN, December 15th 1996. CERN/LHCC/96-41.

[9] ATLAS/Tile Calorimeter Collaboration. *Tile Calorimeter, Technical Design Report*. CERN, December 15th 1996. CERN/LHCC/96-42.

[10] ATLAS Muon Collaboration. *ATLAS Muon Spectrometer, Technical Design Report*. CERN, June 5th 1997. CERN/LHCC/97-22.

[11] The LHC Study Group. *THE LARGE HADRON COLLIDER, Conceptual Design Report*. CERN, 1995. CERN/AC/95-05 (LHC).

[12] ATLAS Level-1 Trigger Group. *Level-1 Trigger, Technical Design Report*. CERN, June 24th 1998. CERN/LHCC 98-14.

[13] ATLAS Collaboration. *ATLAS DAQ, EF, LVL2 and DCS, Technical Progress Report*. CERN, June 30th 1998. CERN/LHCC 98-16.

[14] ATLAS/Trigger Performance Group. *ATLAS Trigger Performance, Status Report*. CERN, August 25th 1998. CERN/LHCC 98-15.

[15] P. Maley et al. A local-global implementation of a vertical slice of the ATLAS second level trigger. ATLAS Internal Note; DAQ-No-081, January 1998.

[16] Standard for Hetrogenious Inter-Connect (HIC). Low-Cost Low-Latency Scalable Serial Interconnect for Parallel System Construction, 1995. `http://www.1355-association.org/`.

[17] C. Clos. A study of non-blocking switching networks. *Bell Syst. Tech. J*, 32:406–424, 1953.

[18] D.B. Guralnik et al. ed. *Webster's New World Dictionary of the american language*. The world publishing company, college edition, 1964.

[19] Macramè Esprit project 8603. `http://www.pact.srf.ac.uk/macrame/welcome.html`.

[20] B. Martin et al. Realization of a 1000-node high-speed packet switching network, Sept 1995. `http://preprints.cern.ch/cern/hypertext/ECP-95-016/Martin2.html`.

[21] R. Heeley. *Real Time HEP Applications using T9000 transputers, Links and Switches*. PhD thesis, University of Liverpool, October 1996.

[22] D.A. Thornley. Private discussion.

[23] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. on Computers*, vol. C-21:948–960, Sept. 1972.

[24] M.D. May P.W. Thomsen and P.H. Welch, editors. *Networks, Routers and Transputers: Function, Performance, and Applications*, volume 32 of *TRANSPUTERS AND OCCAM ENGINEERING SERIES*. IOS Press, 1993. ISBN 905199 129 0.

[25] SGS-THOMSON. T9000 occam 2 toolset user guide, May 1994.

[26] J. Galletly. *occam 2*. Pitman, 1990.

[27] M. Boosten. Replacing the B103 with a Linux PC board. Private discussion.

[28] CERN. Deliverable D 2.2.1, Report on the performance on an Rcube switch fabric, 1999. ARCHES, Esprit Project 20693.

[29] R.W. Dobinson, R. Heeley, N.A.H. Madsen, B. Martin, and D.A. Thornley. *Netprobe — D 1.4.1 Test/diagnostic software and Documentation for IEEE 1355 DS link networks*. CERN, 1997. ARCHES, Esprit Project 20693, `http://home.cern.ch/~madsenn/netbook.ps`.

[30] Rcube specification, February 1997. `http://cao-vlsi.ibp.fr/mpc/library/index.gb.html`.

[31] M. Acciarri et al. The construction of the L3 experiment. *Nucl. Instr. and Meth.*, A(289):35–102, 1990.

[32] T. Angelov et al. Performance of the central L3 data acquisition system. *Nucl. Instr. and Meth.*, A(306):536, 1991.

[33] J.J. Blaising et al. Performance of the L3 second level trigger implemented for the LEP II with the SGS Thomson C104 packet switch. In *Beaune 97, Xth IEEE Real Time Conference*, pages 45–50, 1997.

[34] R. Bock and P. LeDu. Detector and readout specifications, and buffer-RoI relations, for the level-2 trigger demonstrator program. ATLAS Internal Note; DAQ-No-062, Jan 27 1997.

[35] S. George, J.R. Hubbard, and J.C. Vermuelen. Input parameters for modelling the ATLAS second level trigger. ATLAS Internal Note; DAQ-No-070, June 12 1997.

[36] Level-2 requirements group. ATLAS level-2 trigger user requirements document. ATLAS Internal Note; DAQ-No-079, November 1997.

[37] M. Dobson, S. George, J.A. Strong, and J.C. Vermeulen. Paper models of the ATLAS second level trigger. ATLAS Internal Note; DAQ-No-113, June 1998.

[38] J.C. Vermuelen. Simdaq documentation. Private communication.

[39] M. Zhu, D.A. Thornley, J. Pech, B. Martin, N.A.H. Madsen, R. Heeley, S. Haas, R.W. Dobinson, and C.R. Anderson. Realisation and performance of IEEE 1355 DS and HS link based, high speed, low latency packet switching networks. *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, 45(4):1849–1853, August 1998. `http://www.cern.ch/HSI/dshs/` Presented at RT 97, Beaune, 22-26th September 1997.

[40] A. Klein. Interconnection network for universal message-passing system. In *Proceedings of Esprit Conference '91*, pages 336–351, Nov. 1991.

[41] R. Heeley. Private discussion.

[42] Robert B. Blair et al. The ATLAS level2 trigger supervisor. In *Second Workshop on Electronics for LHC Experiments*, page 191. CERN, October 1996. CERN/LHCC/96-39.