FOR REFERENCE ONLY

The Nottingham Trent University
Library & Information Services
SHORT LOAN COLLECTION

| Date | Time | Date | Time |
|---|---|---|---|
| 12 JUL 2012 XXXXXXX | RGF | | |

Please return this item to the issuing library.
Fines are payable for late return.

THIS ITEM MAY NOT BE RENEWED

Short Loan 01

40 0692905 9

ProQuest Number: 10183226

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10183226

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

# INTEGRATED FAULT TOLERANCE FOR PACKET-SWITCHED NETWORKS

ROBIN HOTCHKISS

A thesis submitted in partial fulfilment of the requirements of The Nottingham Trent University for the degree of Doctor of Philosophy

Department of Electrical and Electronic Engineering
The Nottingham Trent University
Burton Street
Nottingham
United Kingdom
NG1 4BU

October 2000

## ABSTRACT

A novel VLSI hardware packet routing switch, integrating fault tolerant mechanisms, has been developed. The device can be used to construct an embedded parallel or distributed processing architecture.

A review of packet switched techniques has been carried out, leading to a high-level specification for a next generation packet switched network. From this specification, an eight-link router-switch has been implemented in a programmable logic device, which significantly enhances the primary routing features compared to earlier devices.

The distributed fault tolerance features were implemented in two stages. The first stage is concerned with critical interruption of the link stability, which has been proven through simulation and hardware verification. The first stage features detected and localised the effects of network failure, while supplying features to reduce link activity where required. The second deals with deadlock detection and recovery, which investigated a detection method that minimised time and false detections irrespective of network traffic. The investigative second stage has been verified by simulation.

This work successfully culminated in the production of a fault tolerant hardware routing switch. All basic routing features of the device operation have been proven through simulation and hardware testing. Simulations have been used to subject the device to a range of extended workloads for confidence of operation. The router-switch supports the network specification for a fault tolerant network, based on a distributed mechanism, allowing a linear scaling factor for tolerance to failure as the network is modified. This maximises network availability in the presence of faults. Key features of the design were compared and contrasted with the current state of the art in the literature.

## ACKNOWLEDGEMENTS

I would like to thank the following people for all the help and support they have provided over the duration of my studies. My supervisors Prof. BC O'Neill and Dr. S Clark; my colleagues and friends Dr. GC Coulson, Dr. RH Day, Mr D Downes, Mr JH Ng, Mr. C Oswald, Dr. RM Ranson, and Dr. KL Wong; all my family and other friends, but especially to Mr C Dixon, Mr & Mrs Kaveney-Davis, Mr R Harris, Mr D Moore, and Mrs D Scholey; Mr A Whitehouse who gave me the reference to get me here in the first place.

Finally, I would like to give special thanks to Mr D Corcoran who taught me a great deal in the years of our friendship and was always present with a word of encouragement.

# TABLE OF CONTENTS

## List of Acronyms

| | |
|---|---|
| ASIC | Application Specific Integrated Circuit |
| CAN | Controller Area Network (serial bus-based communication system) |
| CRC | Cyclic Redundancy Check |
| CSP | Communicating Sequential Processes |
| DSP | digital signal processing |
| FIFO | First In First Out |
| HDL | Hardware Description Language |
| IEEE | Institute of Electrical and Electronic Engineers |
| IP | Intellectual Property |
| ISO | International Standards Organisation |
| LAN | Local Area Network |
| Mb/s | Megabits per second |
| MB/s | Megabytes per second |
| MIMD | Multiple Instruction, Multiple Data |
| NRZ-M | Non-Return to Zero Mark (also known as NRZ-I – Invert) |
| NTR08 | A dynamic packet routing device, which resulted from earlier research |
| NTR-FTM08 | The dynamic packet routing device produced by this research |
| NTR-M04 | A dynamic packet routing device, which resulted from earlier research |
| OSI | Open Systems Interconnect |
| PLD | Programmable Logic Device |
| SAN | Small Area Network |
| SIMD | Single Instruction Multiple Data |
| UART | Universal Asynchronous Receiver Transmitter |
| USB | Universal Serial Bus |
| VHDL | Very high speed integrated circuit Hardware Description Language |
| VLIW | Very Long Instruction Word |
| VLSI | Very Large Scale Integration |
| WAN | Wide Area Network |
| X-OFF | Transmit off (also referred to as STOP) |
| X-ON | Transmit on (also referred to as GO) |
| XOR | exclusive OR |

# List of Figures

# List of Tables

# 1 Introduction

The research detailed within this thesis is the investigation and development of a communication network for use in a distributed memory, message-passing architecture. The architecture under consideration is constructed using one or more multiport, dynamic, packet-routing switches, which have evolved from earlier communication systems research [1, 2]. The network is targeted towards small to medium scale embedded parallel or distributed processing systems. The primary aim of the work was to improve the flexibility and reliability of the network compared to previous implementations from the earlier research.

## 1.1 Parallel Concepts

Multiprocessor systems have evolved a great deal since their inception. The impetus of their development was initially, and continues to be in part, the desire for increased performance over the limits set by the technology of the time. In the early years of computing, the cost effectiveness of such systems meant that only large organisations could afford the associated development costs. Meanwhile, the relative ease of development of single processor systems, in comparison to that of parallel systems, reduced costs and allowed a much shorter time to market, which made them much more accessible to a wider market. The shorter design cycle time and lower costs have resulted in a continued bias of single processor systems in the mainstream computing market, compared to the development of multiprocessor systems. This bias of 'single processor system' parts has, in recent years, encouraged research using these low-cost commercial devices in parallel systems [3, 4, 5, 6]. While these solutions will never produce results comparable to custom parallel processor systems, their utilisation provided great improvements over single processor systems. Using microprocessors that were designed for the single processor market has ensured lower development costs while keeping pace with the state of the art technology especially for small scale parallel systems.

Despite many parallel systems adopting the use of microprocessors designed for single processor systems, constant pressure on the manufactures of microprocessors to supply devices with the greatest performance has seen the adoption of parallel processing techniques in the processing core. Pipeline techniques were integrated first into mainstream processors in the mid to late 1980's. Examples of these were the INTEL 386/486 processors [7]. This was followed by super-scalar architectures that allowed

multiple instruction execution in one cycle, for example the Motorola Power PC family [8]. More recently, Single Instruction Multiple Data (SIMD) units have been integrated to improve the performance of graphic intensive tasks. SIMD is one of four architectural definitions of processing systems that define the operation of parallel machines as described by Flynn [9]. A recent example of utilising parallel structures, such as the SIMD, is the AMD Athlon processor [10]. Another modern example of the ingress of parallel techniques in the commercial market of processing devices, is the introduction of Very Long Instruction Word (VLIW) processors. Examples of these are the Crusoe from Transmeta [11], and the digital signal processing (DSP) device, TigerSHARC devices from Analog Devices [12]. A VLIW processor contains a number of parallel execution units. Each instruction word of a VLIW processor contains many sub-instructions that are executed by one of the execution units in the same cycle, theoretically similar to vector processor systems of the 1970's and early 1980's. While these processors are not new, until recently they have remained primarily in the domain of research.

The main aim of multiprocessing or distributed processing is to share a task over a set of resources. This provides many possible advantages, such as exploiting the natural parallelism within the task for increased performance, modularisation for cost efficiency and simplified repair procedures, or replication of system parts for fault tolerance through redundancy. The system designer must select architectural attributes that suit the application needs, and it must be ensured that the architecture is flexible enough to cope with any possible modification that may arise through the development process. Selecting an inappropriate architecture can place a strain on the communication medium, which could reduce network performance dramatically and ultimately could cause a critical failure. It is therefore obvious that the features of the communication medium and the protocols, which are used for the passage of information between the co-operating parts, play a major role in the flexibility and capabilities of any parallel system.

Since the earliest implementations of parallel machines, researchers have continued to attempt to quantify effects of network efficiency. The analysis of network efficiency is a highly complex problem, and as such, must be simplified by concentrating on specific architectures and operating modes [5, 13, 14, 15]. Such work has provided an insight to the advantages of certain architectural features which has enabled improvements on older systems to be made or has helped provide the knowledge for the development of new systems to keep pace with ever-changing application requirements.

### 1.1.1 Parallel Architectures

The two main classes of multiprocessor systems are shared memory and distributed memory architectures. Early shared memory architectures typically used a single high-speed bus to interface to a unified memory area. This allowed fast transfer of information between co-operating parties as the memory could be accessed by either party with very little latency. Although the shared memory architecture provided an obvious ability to 'share' information between the entities that were co-operating on a task, there were some inherent complications of such architectures, such as memory access arbitration, task synchronisation, and producer-consumer data sharing. In addition to this, scaling of these architectures was problematic. As entities were added to the system, the share of the bandwidth of the high-speed bus was proportionally reduced, this formed a natural bottleneck at the memory interface. Early distributed memory architectures maintained isolated data areas, which could only be accessed by a single system entity. Distributed memory architectures primarily relied on data transfers via a global communication network, which were passed in the form of messages. The nature of the message passing technique instantly solved many problems faced by shared memory systems, as formally proven by Hoare [16]. Such techniques also provided better scope for system scaling, which was controlled ultimately by the structure of the communication medium. However, the pragmatics of message passing increased the latencies of the data transfer when compared to the shared memory architecture, thus limiting performance benefits.

The distinct communication aspects of these two architectures saw different application strategies develop that defined the levels at which parallelism was implemented. The higher data transfer latencies of the distributed memory architecture encouraged coarser divisions in the task for parallel execution, whereas the shared memory architecture operated more effectively with finer divisions. These strategies further divided the use of the architectures to suitable applications, but it seems apparent that the evolution of both architectures used lessons learnt by the other. Much work was concentrated on connectivity and efficiency to increase performance and ability to scale. An example of this in early development of shared memory systems was segmented memory. This modification enabled concurrent memory operations, which enabled an increase in system size, but it only eased the bottleneck and did not solve the problem. Distributed memory systems saw the development of new topologies and protocols that reduced message latencies and improved reliability. Although many parallel systems used

the distributed memory architecture, it also began to find success in distributed processing where low latencies and high reliability were a high priority in system design.

Development continued to push the performance capabilities of the parallel systems, which have seen the creation of specialist systems formed as hybrids of these first architectures. Systems, such as the Cray T3D [3], have maintained a logically shared memory while possessing a distributed memory architecture. A high-speed interconnect, which uses a popular distributed memory network structure, moves the shared data around the system in a manner similar to cache or virtual memory operations of recent mainstream processor systems. In this way, the Cray machines have equalled the ability to scale of message passing distributed architectures, while maintaining the illusion of a shared memory organisation to the software designer. Although such systems as the Cray provide the market with 'high-end' products, there is a diverse field of distributed and parallel processing over a range of applications and requirements, from the distributed processing systems seen in transport applications, to specialist video editing systems, to the supercomputers used in many highly mathematical research areas. The smaller scale distributed and parallel systems possess other priorities than just raw performance capabilities, such as flexibility, efficiency, reliability, and cost. The work described here targets small to medium scale systems where the aspects associated to these priorities formed directional guides.

## 1.2 Effects of Technological Advancements in Digital Systems

Improvements in semiconductor electronics have produced increased capabilities in both Application Specific Integrated Circuits (ASICs) and programmable logic devices (PLDs), which has seen a change in roles for these devices. Previously, custom circuit design of complex circuits demanded resources that could only be supplied by ASICs. Programmable Logic Devices (PLD) were not capable of more than simple circuits for decoding or interfacing between ASICs. Additionally, the costs of ASIC implementation were relatively high, which restricted the development of custom parts. Although ASICs have maintained a hold on high volume and high performance products, many low-to-mid scale applications are now realised with PLDs. The improved PLDs are more cost effective in low volume production, and development cycle times are much lower when compared to ASIC designs. This has encouraged manufactures to use PLDs as development devices and occasionally first role-out devices for an early market advantage.

Continued improvement of silicon fabrication techniques and reduction in die features sizes have seen increasing integration. While improvements have been made to the packaging types that hold very large scale integrated (VLSI) devices, the physical problems of distributing the signals from a highly integrated device create a pin-out bottleneck. This is compounded by the move from 8-bit data paths in the 1980's to 64-bit devices in today's products. Maintaining several of these high-order parallel connections places impossible demands on the device packaging. This problem of connection capabilities has produced a renewed interest in serial communications. By replacing parallel with serial interconnects a high number of logical connections may be supplied, while maintaining a low number of physical wire connections. Additionally, improvements in technology have enabled increased performance levels for serial communications, extending the range from low bandwidth applications with extended transmission length. Recent examples of communication systems that are serial based are Universal Serial Bus (USB) [17] and IEEE Std. 1364 (FIREWIRE) [18]. Both of these systems maintain a logical bus operation, in which the medium is shared between all communicating devices but the number of data connections and maximum cable length are limited.

In conjunction with the renewed interest of serial communication, a proliferation of switched-based communication systems is also evident. The success of switched wide area networks (WANs) and the large system data throughput that are possible with these architectures have raised the interest in switch-based intra-system communications. Recently, a number of forums have been created to develop new router-switch I/O communication architectures; examples include Rapid I/O [19] and Next Generation I/O (NGIO) [20] (which was superseded by InfiniBand [21]). Further discussion of these systems is beyond the scope of this work, but they aim to use the advantages of switched communication in a similar manner to this work.

Integration has increased use of reusable design methodologies, which are mostly based on hardware description languages (HDLs). As the scale of digital designs increase, it is not cost effective to repeat the design cycle required for common component. Intellectual Property (IP) is a growing market. By utilising IP and integrating many of the application components into one device, development times can be reduced, which in turn reduces costs. In addition to this, IP can increase reliability as the functional blocks are proven design parts, which can reduce verification time. Unfortunately, the amount of

design effort to integrate an IP design module can be equal to the design effort of a total development of the functionality [22].

## 1.3 Aims of this Research

The work described in this thesis is related to the switched communication network of parallel and distributed systems. Previous research in this area produced dynamic packet switches, which could be used to construct efficient, low-latency networks for use in a distributed memory, message-passing architecture [1, 2]. Tolerance to network failure in these earlier systems was limited and was based on a centralised monitoring and intervention solution. While this solution provided a range of useful features, it possessed many weaknesses. As the resultant network systems were targeted towards board-level or rack-level systems, reliability was predicted to be high, which made the utilised fault tolerance methodology acceptable. Previous work also saw a technology transition from gate array technology to programmable logic, and a design entry change from schematics to a hardware description language. With these transitions came many new challenges.

The primary aim of this work was to improve the flexibility and reliability of the network compared to the previous implementations. The secondary aims were to optimise the basic router-switch design for operation in the targeted technology and entry method with the lessons learnt from earlier work. A short list of requirements that encompass these aims follows:

- Define and supply physical fault detection and recovery procedures

  The network must possess an integrated detection method for basic faults, which will activate a predefined recovery mechanism that will allow the remainder of the network to operate as normal. Features are required that supply the user with confidence that in any state of the network a packet may be injected and will not cause critical failure.

- Maintain or improve same level of support of routing methodologies

  The design specifications made in earlier research work had to be reviewed and followed to maintain the ideals of the system, improving where flaws were discovered.

- Minimise routing overheads

Cross-switch latencies had to be low to minimise the effects on performance as the network scaled. The architecture of the target technology (PLD) requires the design to operate with many levels of pipelining to optimise the routing operation.

- Improve on bi-directional data transfer bandwidth utilisation

A review of low level link protocol was required to analyse the optimal requirements for use in packet switched network where there is a high proportion of bi-directional traffic.

## 1.4 Breakdown of Thesis

A description of the thesis is given chapter by chapter:

- **Chapter 2 : Background Techniques, Methodology and Terminology**

This chapter begins with an introduction to the Open System Interconnection model, which is used in the thesis to help describe the network methodologies and techniques referred to in the thesis. It continues to introduce the techniques, methods and terminology of packet switched networks.

- **Chapter 3 : Review of Earlier Research**

A critical breakdown and discussion of earlier research work is provided, which covers research completed by previous members of the research group and some comparisons with other contemporary devices.

- **Chapter 4 : Design Discussion**

An analysis of the review of the previous chapter is presented, from which the salient design features for the next generation of network are evolved. This is followed by a high level description of implementation details for the features for the switch, around which the network is based. The router-switch description is divided into three distinct parts for implementation; namely the skeletal switch, and two stages of development of fault tolerant features.

- **Chapter 5 : Detailed Router-Switch Design**

This chapter is broken into three sections. The first section defines, in detail, the skeletal router-switch design, which has been defined in the previous chapter. The latter two sections define the implementation of the fault tolerant aspects of the design, which was split into two distinct parts in the previous chapter.

- **Chapter 6 : Design Synthesis & Verification**

The device implementation and operating limits are defined. A description of the verification is described, and the results from a number of operational tests are provided.

- **Chapter 7 : Discussion, Conclusions and Further** Work

This chapter discussed the final design and the results that have been derived from verification. Finally, areas of work that still need to be addressed following this research are presented.

# 2 Background Techniques, Methodology and Terminology

The main aim of multiprocessing or distributed processing is to share a task over a set of resources. In the target area of this research, the aim of task distribution is to exploit the natural parallelism within an application for three reasons, they are: to improve performance or efficiency, to modularise a design for simplified maintenance, or to supply fault tolerance. The interconnection medium is an area that heavily affects the behaviour and efficiency of all parallel systems. The features of the communication medium and the protocols, which are used for the passage of information between the co-operating parts, play a major role in the flexibility and capabilities of any parallel system. The work described by this report concentrates on a distributed memory architecture. Data transfer is achieved using a switched communication network that provides a point to point connection between all processing nodes. 'Switched network' is a high level term which implies a global communication system constructed of switching nodes. As data arrives at a node it is routed to one of the other outputs of the node, that is, the node acts as a switch to form the correct data path.

This work was a progression from earlier research [1, 2]. To help clarify the techniques used in this earlier work, an overview of methodologies and terms will be provided in term of the International Standards Organisation (ISO) Open System Interconnection (OSI) reference model [23]. The OSI is a seven-layer model, which is used to help modularise the aspects of communication system to simplify their description. The description of communication techniques and methodologies will be followed up in chapter 3, with a review of their use in previous research on switched, parallel networks.

## 2.1 The ISO Open Systems Interconnection Model

The ISO OSI model is constructed of seven layers, which are used to help modularise communication systems. The layers of the model are presented from the viewpoint of connection-mode transmission, starting from the interface to the physical medium. As information is transmitted, it moves down through the layers, being modified at each transition. Conversely, as information is received, it moves up through the layers, reversing the modification made as it was transmitted. The modifications at each layer differ, but normally it includes adding information and formatting to the information to allow it to be delivered to the correct destination. It is often difficult to define which part

of a system falls into which layer, as implementation often results in features related to two neighbouring layers.

The layers of the OSI model include:

- Application layer (layer 7)
- Presentation layer (layer 6)
- Session layer (layer 5)
- Transport layer (layer 4)
- Network layer (layer 3)
- Data-link layer (layer 2)
- Physical layer (layer 1)

The work detailed here relates to aspects of the bottom four layers, and therefore they will be used as a point of reference for the discussion of the reviewed systems.

The physical layer regards all the information related to the physical transmission medium. In those regards, it covers aspects such as: connectors, cables, signal encoding, voltage levels, noise margins, data rates, and signalling rates.

The data-link layer interfaces the raw transmission facility and produces an error free medium, over which data can be passed. This is achieved by introducing the formatting of the data stream in to transmission units, on which error detection and error recovery can be based. Additionally, basic arbitration of the medium is specified, and in the case of point-to-point systems, the operation of the data flow control normally would be included.

The purpose of the network layer is to provide an end to end communication circuit. It defines features used for tasks such as routing, switching, and types of interconnection.

The transport layer is normally the domain of software, as its task is to alter the data to be suitable for the particular terminal equipment, that is, independent of the network and type of service. In point-to-point, packet switched systems this includes packetisation of messages and sharing of the data path with multiple data streams. Message-level error checking can be implemented at this level also.

## 2.2 Review of the Methodologies of Switched Communication

This section briefly introduces the concepts and methodologies that have been used in the devices, which are further documented in the historical review in the following chapter.

### 2.2.1 Physical Link Layer Considerations

This section presents some physical connection formats, data rates and synchronisation techniques that have been used in a number of prominent point-to-point switched systems.

Physical connections in point-to-point networks are defined by the type and requirements of the target application. For example, serial connections were chosen for early system interconnections as single wire signalling could be driven over reasonable distances at rates that were acceptable for the target technology and application, at a low cost. Common transmission rates of these early systems were in the order megabits per second (Mb/s). For the early applications that needed higher transmission rates, parallel connections were chosen, where multiple bits were transmitted together. However, the device architectures were limited by high pin count for such interconnections by the integrated packaging technology. A solution for this limitation was the use of special encoding techniques that allowed a single set of interconnections for full duplex operation, that is concurrent transfers in both directions [24]. Other later systems moved to fibre optics for improved performance, but this demanded integrated optical interfaces or parallel output from the communicating device to the optical transceiver [25].

Recently, a trend has been to move back towards serial formats, even with intra-system communication, where the demand for performance commonly resulted in parallel interconnections. The improvements in data signalling has resulted in high transfer rates whilst using minimal interconnection resources. The return to serial communication formats has been helped by the availability of low-cost, high-speed driving circuits that will drive signals over extended distances. Such is the interest in the latest signalling technology that it has also been integrated into large scale PLDs, which are targeted towards communication applications [26].

As parallel systems are constructed from parts that operate autonomously, the physical medium must include some method to allow the communication to be recoverable or synchronised from the message source to the destination. In small parallel systems, this

may be achieved by a single global clocking system, but even this is not a simple solution. Maintaining low amounts of skew as the clock signal is distributed becomes difficult as number of connections and distances increase. Therefore, global clocking systems normally are limited to board-level solutions. For larger parallel systems and distributed systems, each individual unit normally operates on a local clock, to which the incoming data stream must be synchronised.

A common solution to clock distribution is to send the transmitter clock with the data stream either by extra signal wires, which is implemented in some parallel interconnection systems, or by data encoding, which is favoured in serial interconnection systems. Signal skew between the data and clock signal is a concern for those systems that employ extra signal wires, which can result in very expensive cables or severe cable length limitations. Systems that encode the clock into the data stream suffer less from skew, but must employ additional circuitry that can recover the clock from the encoding as a useable signal. One technique utilises a phase-locked-loop (PLL), which demands frequent transitions on which the PLL can lock. Some examples of encoding used for this purpose are bit stuffing as used in CAN bus system [27] or Manchester encoding as used in Ethernet [28]. Additionally, the predictable format of bit streams such as the bit stuffing or Manchester encoding format are often used as a method of error detection. Unfortunately, some implementations can be complicated by the encoding mechanism. Research into the CAN bus implementation of bit stuffing has shown the technique to be detrimental to fault detection that were employed in the protocol [29].

An alternative approach to using data stream transitions for clock recovery is technique that has been used with a parallel interconnection system [30]. The physical link was nine bits wide, which transferred one byte and a control bit in each transmission cycle. A non-return to zero inversion (NRZ-I) code was used, which toggles the signal level on the transmission of a logic '1', and leaves the signal stable on the transmission of a logic '0'. In itself this form of encoding did not provide a minimum number of transitions, but the parallel nature and token encoding of the protocol ensured that each unit of data possesses at least one bit set high. This in turn ensured that at least one transition occurred per unit of information. The guaranteed transition could then be used to trigger latching circuits to recover the information. The signalling rate of NRZ-I is variable, with a maximum of the data rate. This is advantageous as lower signalling rates reduces noise and lowers power requirements. The parallel interconnection system using this method of

data recovery required high quality, matched physical cabling to minimise skew across the parallel signalling lines.

Another example of using transitions to encode the transmitter clock, is the data strobe technique where two signalling wires are used, one called the data wire and one called the strobe wire [31]. Transitions on the data line are caused by change in the binary value of the data. Transitions on the strobe line occur when no transitions occur on the data line. These transitions result in the encoding of the transmitter clock within the data stream, which can be regenerated at the receiver by a logical XOR operation of the two signals. Unfortunately the data strobe technique also required accurately matched signal wires, as too much signal skew between the signal lines would make data recovery impossible.

One final method of data stream synchronisation is over-sampling, which is a tried and tested technique that has been used in a number of systems. The most common implementation of over-sampling is the serial communication device in most desktop computers, namely the RS-232 driven universal asynchronous receiver transmitter (UART). The receiver circuits of such systems sample the data stream at a rate many times higher than the signalling rate, which allows them to synchronise to the bit stream. Basic sampling theory states that a data stream should be sampled at greater than twice the fundamental frequency signalling rate [32]. Many systems use a higher sampling rate to ensure correct sampling and to provide suitable tolerance to signal skew, for example the RS-232 UART used 16 times over-sampling. Figure 2-1 shows the technique utilising five-times over-sampling, highlighting the areas when the data is sampled. It is evident that the disadvantage of this technique is the increased rate at which the receiver circuits must operate, although this can be altered to compromise between skew tolerance and operating frequency.

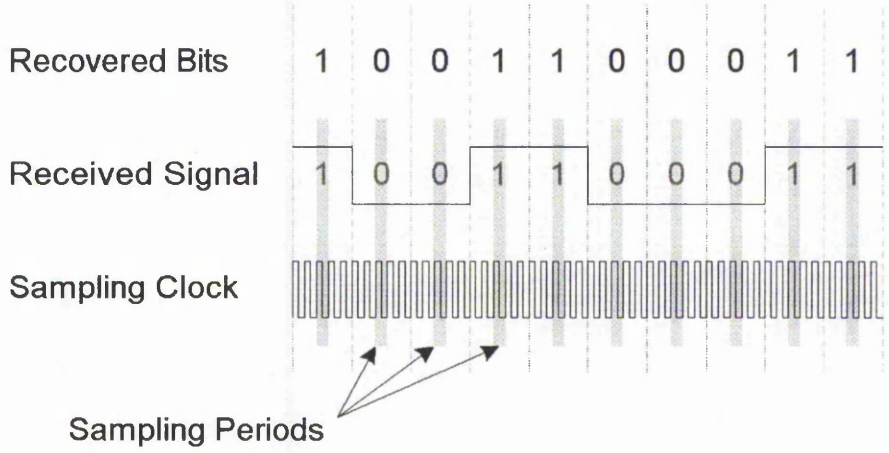



**Figure 2-1 : An example of five times over-sampling**

### 2.2.2 Data-Link Layer Considerations

The data-link layer interfaces the raw transmission facility and produces an error free medium, over which data can be passed. This is achieved by introducing the formatting of the data stream in to transmission units, on which error detection and error recovery can be based. Additionally, basic arbitration of the medium is specified, and in the case of point-to-point systems, the operation of the data flow control will be included.

#### 2.2.2.1 Token Definitions

The data link layer formats the data stream so that it is suitable for transmission over the physical medium. Thus, the formatting is dependent on the physical layer. Network control information and data often are transmitted over the same medium in point-to-point systems, which relies on a low volume of control signalling. Thus, the token format also may be required to differentiate between control and data.

As highlighted previously, some parallel systems may use a parallel format for the physical interconnection. Such systems may use the width of the link for a data unit [30], alternatively multiple transfers of the parallel link may be used [33]. Most serial systems, however, format the data stream in such a way to aid recovery at the receiver by encapsulating the data in extra bits, which is known as tokenising, or framing.

#### 2.2.2.2 Flow Control

To provide a system in which data transmission between producer and consumer occurs, a form of flow control or handshaking is required. This should ensure that link bandwidth is not wasted due to buffer underrun, and data is not lost due to buffer overflow.

There are many different schemes to provide this functionality, which are matched to network structure and application. As data is normally divided into frames or tokens, flow control mechanisms must operate at this level. Hence, flow control mechanisms are often described as part of data-link layer.

The method, with which flow control information is passed between communicating parts, depends on the physical layer of the system. One solution is to use the data path to relay this information as separate tokens or as part of the data frame. Alternatively, additional signal wires may be used to reduce the overhead on the data stream.

This section details two types of flow control mechanisms: permission based systems and credit based systems. Permission based systems are most prevalent in point-to-point systems, but earlier work was based on credit based system, thus both system are presented.

**Permission based flow control**

Permission based systems inhibit or permit the transmission of data based on the status of the receiving node. These mechanisms are also known as STOP/GO or X-ON/X-OFF mechanisms. In all the implementations of permission based flow control mechanisms, buffer overrun and underrun are the primary areas of concern. This is due to the controlling factor of the mechanism, which is the time it takes the transmitter to react to the flow control signalling. The reaction time is related to the signalling rate and them maximum transmission distances, which both must be limited to define buffering requirements of a workable system.

There are three areas where permission-based flow control regulates buffering requirements; the third is only relevant to systems where the flow control mechanism uses data bandwidth. The three regulatory clauses are:

1. There must enough buffering to ensure that any data in transit can be stored for a period until the flow control has taken effect following a STOP.
2. There must be enough buffering to store enough data to ensure starvation does not occur in forwarding the packet following a transition of flow control status from STOP to GO

3. There must be additional buffering to ensure the flow control mechanism does not use the majority of the link bandwidth (hysteresis)

Figure 2-2 shows the operation of an example link, which is governed by a STOP/GO flow control mechanism that uses separate tokens on the data path for control signalling. The figure highlights the areas, to where the three regulations, defined above, relate. Clause 1 is the only imperative rule, as it relates to the amount of tokens that can be accepted after the assertion of the STOP flow control. Without this rule the link is prone to buffer overrun, and is only valid for a predefined transmission length and data rate. It can be seen by the simple example of Figure 2-2 that even with a transmission time of approximately one token, the buffering require an extra four tokens over the limit that generates the STOP command.

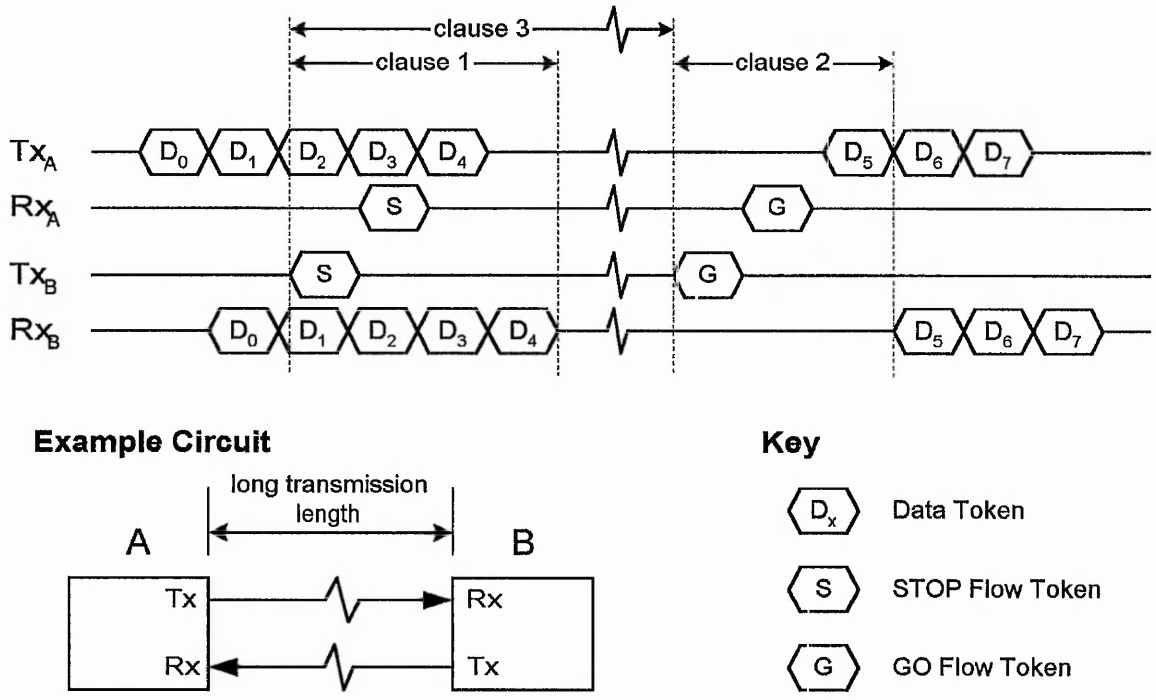



**Figure 2-2 : Depiction of the STOP/GO flow control mechanism with the important features indicated that must be regulated for correct operation**

Clause 2 is required to prevent buffer underrun that could occur from the latency of the flow control mechanism. In this example, if node B was forwarding the data it was receiving, it must have sent the GO token before it possesses less than four tokens. If this is not adhered to, the forwarding bandwidth would be wasted while node B was waiting for data from node A.

Clause 3 is only necessary to prevent the flow control consuming too much reverse path bandwidth, and as such, it is more difficult to depict. Clause 3 defines the hysteresis of the mechanism between the STOP and GO states; thus, it defines the reaction of the mechanism to the flow of data out of the receiver FIFO. The reaction to network performance of this value would be dependent on traffic patterns, but optimisation can be chosen to minimise the effects of the mechanism in worse case scenarios.

Figure 2-3 shows the implementation of buffering required for clause 1, 2 and 3 in the representation of the receiver FIFO used for the permission based mechanism. It shows how the clauses map out onto the hardware, and how the state of the buffering controls the mechanism.

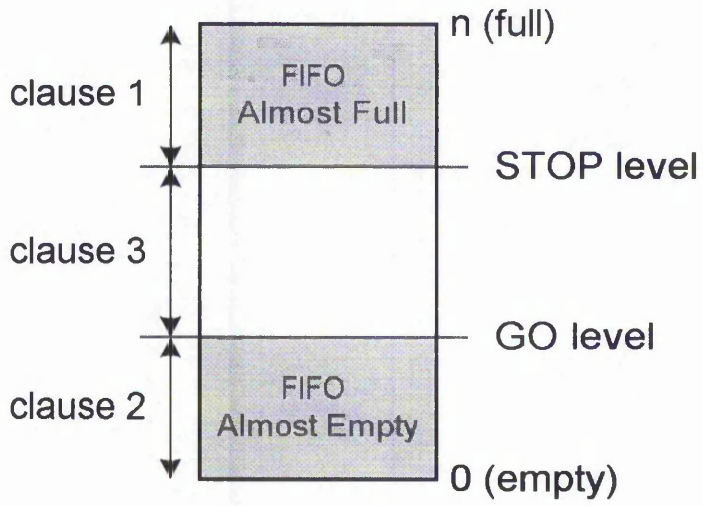



**Figure 2-3 : Representation of the buffering required for the STOP/GO flow control mechanism**

Clauses 1 and 2 can be presented as a simple equation in term of the transmission distance (metres), signalling rate (bits per second), bits per data token, bits per flow control token, propagation speed (metres per second) and total delay of the interface circuits (seconds). The bits per unit defines the ratio of data and control bits in the data stream and therefore converts the total number of bits in transit to total number of tokens. The propagation speed relates to the speed of a signal travelling down the medium. The logic delay of the interface circuits should include the worse case time to generate the control signal, and the worse case time for it to take effect after its receipt. The transmission of the flow control and data tokens only play a role if the data path is used, and relate to the worse case delay of the arrival of control information. Equation 1 calculates the minimum buffering for clause 1 and 2. Note that the distance/propagation

speed, which provides the delay of the cable, is doubled. For clause 1 this is due to the fact that at the point in time when the flow control is triggered the medium may possess back to back data. The assertion of STOP will take the same delay to propagate to the receiver, thus the number of back-to-back data tokens will double from the time the STOP was generated. Clause 2 requires it for the opposite reason, as it will take twice the propagation delay before the first tokens arrive after the assertion of GO.

$$\text{Total extra delay} = \text{Logic Delay} + \text{Tx of Flow Control} + \text{Tx of data token}$$

$$\text{Buff}_{\text{min}} = \frac{\text{Signalling rate}.\left(\left(\frac{\text{Distance}}{\text{Propagation speed}}\right).2 + \text{Total extra delay}\right)}{\text{Bits per unit}}$$

**Equation 1 : Equation for minimum buffering requirements of clause 1 and 2 for permission based flow control**

**Credit based systems**

With a credit based flow control system, the receiver passes a credit to the transmitter, which allows a predefined amount of data can be transmitted. Without credit transmission data is inhibited. Credit allocation is based on receiver buffering status, where the next credit is sent when enough buffering is free for the next set of data units. The number of tokens that may be sent per credit, which will be referred to as a flow group, and how many credits the transmitter can hold is defined by the implementation. Obviously, as the size of the flow group increases, so do the buffering requirements. Thus, in this respect, smaller flow groups are preferred. The smallest flow group would be a single token, however, as the right-hand diagram of Figure 2-4 shows, small flow groups can cause loss of bandwidth when used with large transmission distances or high signalling rates. The gaps between data tokens on the diagram are losses of link bandwidth. Therefore, as the flow group increases in size, the mechanism gains more resilience to longer transmission lengths and higher signalling rates.

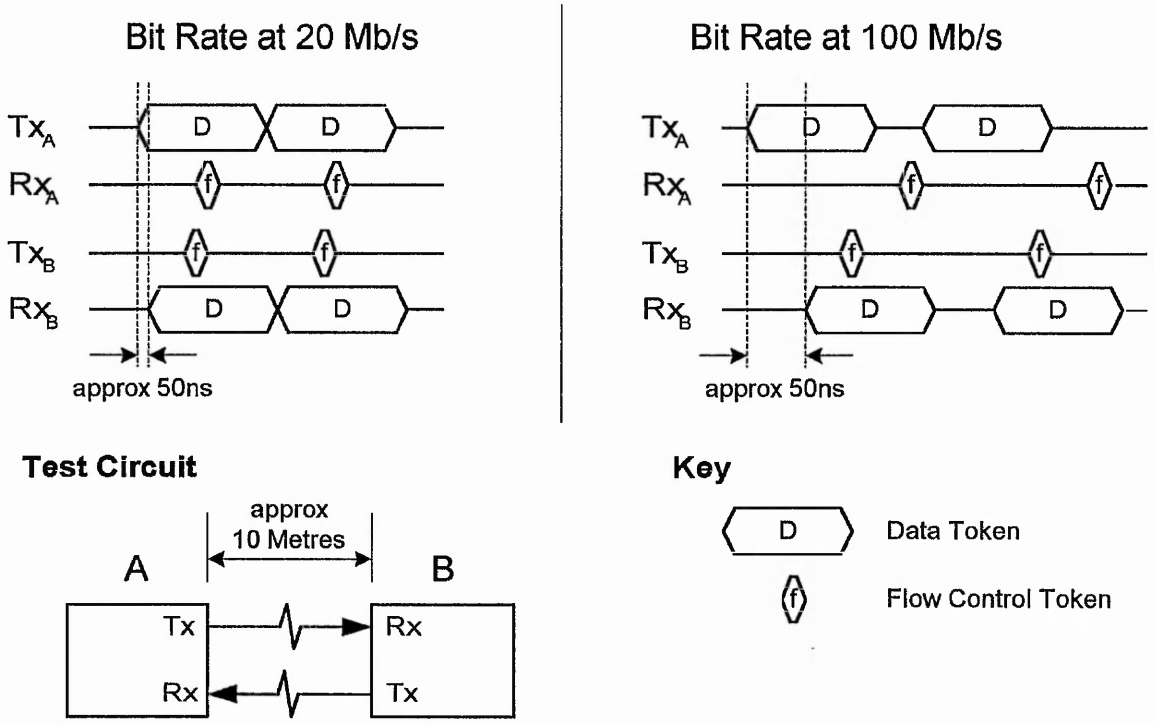



**Figure 2-4 : Depiction of the operation of a credit based flow control mechanism over 10 metres at 20 Mb/s (left) and operation over 10 metres at 100 Mb/s (right) showing the appropriate transmission delays**

The advantage of the credit based mechanism is that FIFO overrun can never occur in a fault free environment, as the flow control tokens must be received for further data to be sent. However, the credit-based systems are penalised by bi-directional data transfers, as a link that transmits back-to-back data, will use a predefined amount of bandwidth in the opposite direction for the flow control signalling.

The size of the flow group for any given application can be infinitely extended to improve resilience to speed and distance and bi-directional transfers, but buffering resources must be sufficient to cope with the incoming data, which normally provides a cost-effective limit.

#### 2.2.2.3 Data Link Fault Tolerance

The data link layer is the first layer of the protocol that can perform data and format checking for fault detection. As the layer defines the smallest unit of data, fault detection mechanisms operate at this level. Examples of detection mechanisms are parity checking, data encoding validation, link connection validation, and synchronisation checking through encoding or framing checks.

There are a number of different types of parity checks, but they all work on the same premise. A single bit is added to each unit of data, which is set to a value dependent on the type of parity. Mark and space parity set the bit to logic '1' and '0' respectively, where odd and even parity set the bit to make the total number of logic '1's in the data unit as odd or even. The use of parity was prolific in earlier communication protocols as line reliability was low, and it was a cost-effective solution for error detection. The worth of parity as a fault detection mechanism in modern systems is questionable. Improvements in reliability have reduced bit error rates that the earlier systems suffered. Additionally, the single bit overhead per byte has become excessive in systems that transfer large volumes of data. As such, systems that operate on blocks of information provide a lower overhead and are therefore a more reasonable solution in these situations.

Other methods of error detection operate on failure of the data stream to conform to some form of data encoding or the loss of signalling. Systems that operate on encoding define an alphabet of characters or sequences that are valid, and inverting this premise forms the method of error detection. While using encoding as a sole fault detection mechanism would be insufficient in most applications, used with other techniques they can be useful to supply extra confidence to data link integrity. Synchronisation checks using encoding or framing checks also falls in to the same category as encoding validation as it uses the expected format of a data stream to conform to a predefined form. Loss of signalling is used as a detection method for disconnection or critical failure of the communicating node. Communication channel initialisation, and connection validation are used often as a higher level of point-to-point check for the protocol, which form definable states that aid error detection.

### 2.2.3 Network Layer Considerations

The network layer provides the functionality for an end-to-end connection, defining such features as routing, switching and a range of fault tolerance features.

#### 2.2.3.1 Switched Architectures

All communication systems may use varying types of topologies. Dally [34] defines topology as the interconnection graph of the network. In point-to-point networks, the different topologies provide distinct system characteristics in regards to types of data traffic patterns, raw throughput and fault tolerance. These features must be considered when the topology is selected for an application.

Topologies for switched, point-to-point networks fall into one of two main categories, regular or irregular. The structure of regular topologies conform to definable structure, such as the 2D mesh or hypercube shown in Figure 2-5. Much work has been undertaken to characterise these types of topologies, one example is [35]. The use of regular networks is common due to their predictable structure. This is their primary advantage, as it can be used to help optimise the routing algorithms of the lower layers of the protocol. However, not all applications suit regular topologies, and consequently, their use can be impractical.

The structure of irregular topologies are not predefined and may take any form, where interconnections are only limited by the devices, with which the networks are constructed. By using switching devices with large numbers of ports, the number of permutations of different network structures increases, and it can reduce the number of switches between any two communicating nodes, which in turn lowers overheads. The irregular nature of the topology forces more simple routing schemes, and involves more complex system design requirements to optimise a system to the application. Figure 2-5 also shows an example of a possible irregular network constructed with a number of multiport switches.

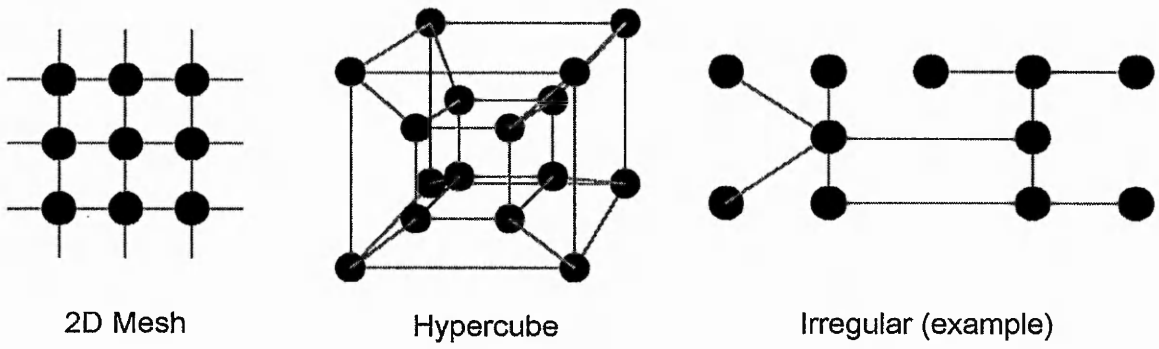



**Figure 2-5 : Depiction of the interconnection structure of the 2D mesh and hypercube topologies, and an example irregular network.**

#### 2.2.3.2 Connection Methodologies

The connection methodology that is used within a network can be defined by the type of message and the method with which packets are connected and switched.

**Types of Connection – Unicast / Multicast / Broadcast**

A message can be described as being transmitted from a single source, to be received by a single or multiple destinations. If a message possesses only one destination,

it is said to be unicast. Message that are sent from one source to many destinations are said to be multicast, or broadcasting if sent to all destinations in the system. From hereon, multicasting will be used as a general term to encompass multicast and broadcast messaging. This description of the type of connection can also be used to define the type of message.

It is difficult to place the feature of connection/message types in any specific layer, especially with regards to multicasting in point-to-point systems. In systems that do not support multicast communications in the lower layers, it is still possible to implement such features in the higher layers. Bus based systems can implement multicasting easily, as all devices receive every message it is just a matter of the lower-levels actioning the receipt. For example, with bus architectures such as ETHERNET [28] or CAN [36] (Controller Area Network), the multicasting implementation would fit better in the data-link layer, as the decision of whether the message is accepted is relative to token level definitions of the protocols. In contrast, the network resources of point-to-point systems are disjoint for maximum communication concurrency, and a message only uses certain network resources if they lie in the route between source and destination. This greatly complicates multicast solutions.

The first versions of multicast algorithms in point-to-point parallel networks were software controlled, based on message retransmission in the upper layers of the protocol. Initially the source was given the task to send all the replicated messages, but such solutions took up valuable processing resources, which led to hardware implementations of the replication [37, 38]. This did not solve the intrinsic problem with the technique, the bottleneck of the single message source, especially if the source only possessed a single link to the network. This led to the development of a technique called tree-based multicasting, where the task of message replication was distributed over the network and completed in parallel, which improved all aspects of message replication and delivery [39]. The primary aspect of all the earlier multicasting techniques was that the nodes were using simple unicast switching to achieve their goal. This allowed the systems to maintain all the advantages that had evolved from earlier work in unicast networks.

Concurrent to the latter unicast-based solutions, multicast switched-based systems were developed [40, 41] that aimed to improve on the efficiency over the node-based systems by automatic replication of the messages on switching. Despite more complex

implementation problems, switched-based multicast algorithms have been shown to be an effective solution for irregular topologies.

**Methods of Switching**

A method of switching relates to the manner the resources of the network are allocated and released, as data is moved from source to destination(s). The earliest systems used a technique called circuit switching, which formed a path of reserved resources from source to destination before data transmission was carried out. The main advantage of circuit switching was the guaranteed bandwidth once a connection was made. The disadvantage, however, was that other connections could not be made until the resources were relinquished. This meant a loss of network throughput. Reduction of the maximum message size lowered the rate of message blocking. Therefore, a compromise between message blocking and connection overheads had to be made to obtain the optimum network efficiency.

As the size of networks steadily increased a more co-operative form of sharing network resources was needed. This brought about the packet switched network, where connections in the networks were localised which increased the communication concurrency. There are many flavours of packet-switching techniques, but the majority of them are based on three main connection methodologies [42]:

- Store-forward;
- wormhole;
- virtual cut-through.

*Store-forward switching* - The store-forward methodology receives and stores the message in its entirety at each switching node before connection resources are allocated. This technique suits larger networks, or networks with a high volume of traffic, as switching can be optimised with predefined sizes of data blocks. As messages are disjoint from the data path, resource sharing can be optimised. If a message is stalled waiting for connection resources, its disconnection from the data path allows subsequent messages arriving on the same input to bypass the stalled message. In addition to the performance benefits of improved sharing, the buffering of the entire message allows error checking to be used to enable the removal, or localised retransmission, of bad messages. However, store-forward switching demands a large amount of buffering resources and message

latencies can be high. The latency of any message is directly proportional to the message size and the number of switching devices between source and destination, plus delays due to volume of network traffic.

*Wormhole switching* - Switches using the wormhole methodology connect messages across the router-switch as soon as enough routing information has been received and the connection resources are available; minimising switching latencies. In contrast to the store-forward methodology, wormhole switching can be implemented with minimal buffering resources at each node as the message effectively is distributed across the network. Minimal message latencies and minimal buffer requirements are the primary advantages of this methodology. However, as a message can span many hops, blocked messages can utilise substantial proportions of localised routing resources, which can reduce network efficiency as the network approaches saturation. In addition, without precautions, networks using the wormhole methodology may be prone to deadlock. Deadlock is discussed in more depth later, but can be simply described as the network state, in which a number of messages stall and prevent delivery of one another due to resource contention through cyclic dependency.

*Virtual cut-through switching* - There have been many variations of the virtual cut-through routing methodology, but the primary aspect of this technique is the ability to operate as either wormhole or store and forward system. This provides the advantages of low latency communications but possesses enough buffering to ensure that blocked messages can be stored to free up localised routing resources. More advanced versions allow subsequent messages to bypass the blocked messages by their complete removal from the normal message path. Such techniques increase the complexity and buffering requirements of the switching fabric and may impose upper limits on message or packet sizes.

As stated above, the majority of switched networks operate based on an implementation of one of these three techniques. However, more recently, research has been undertaken to emulate high-levels of network protocol, by utilising a hybrid of switching techniques, which attempt to supply guaranteed quality of service [43, 44]. Quality of service is a term used when referring to the ability to guarantee predefined levels of network use in terms of bandwidth and reliability. It is an aspect of network communication that has become prevalent with the increased interest in multimedia applications. Further discussion of these techniques is beyond the scope of this thesis.

**Virtual Channels**

Virtual channels allow the multiplexing of one or more messages over a single physical link such that at a high level it appears that many physical channels exist. This can be implemented at many levels from token level to message level. Virtual channels have been shown to substantially increase the utilised throughput of a network [45]. Normally, in a hardware implementation, there are a number of virtual channels associated with each point-to-point connection. Each virtual channel possesses buffering that allows a decoupled operation with regard to the other channels associated to the same physical interconnection. This results in each virtual channel possessing its own control and connection status. Dally [45] analogises adding virtual channels within a switched network to adding extra lanes to a town traffic system. A network without virtual channels is the equivalent to a town with single lane roads. Thus, any obstacle in such a system would block all traffic attempting to use the same route. By increasing the number of lanes, that is virtual channels, traffic can bypass blockages that may form. In addition to such benefits, virtual channels have been used to implement more efficient deadlock free routing algorithms [46]. Deadlock will be discussed further, in section 2.2.3.7. Dally also highlights the increased router-switch complexities required for a hardware implementation of virtual channels. He highlights that virtual channels increase the number of connections to the core of the switch, which in turn complicates the control and the switching matrix.

#### 2.2.3.3 Packet Format

As highlighted in section 2.2.1, messages can be divided into smaller units to provide for better resource sharing across networks. The majority of packet switched protocols mainly follow a format similar to that shown in Figure 2-6. Normally the packet has two or three parts. The first part is the header, which contains information relative to the route through the network and possibly other control information relative to the network such as packet length, type, or priority. The second part contains the payload of the message, which was formatted by the preceding protocol layers. The final part is optional and could be a packet delimiter or contain information relative to an error detection mechanism. More specific details of the format of the packet depend on the protocol, for example the size and meaning of the header, type of packet delimitation, or type of embedded error checks.


**Figure 2-6 : Generic packet format used as a basis for the majority of protocols**

#### 2.2.3.4 Adaptive Routing Techniques

Within an embedded switched point-to-point network the topology is normally fixed, with only rare applications that may add or subtract from the core architecture. In such a network, which is non-adaptive, all possible routes must be predefined to allow passage of information from source to destination(s). Networks such as these are called 'oblivious', as the route that the messages take is oblivious to faults or to the dynamic traffic workload. By supplying flexibility to the routes that a message may take, the messages may *adapt* to the state of the network. Implementing adaptively can be complicated by concerns of deadlock and non-delivery. Deadlock is a network state in which messages inhibit the progression of one another, and ultimately it can grind the whole network to a halt. The other danger is that the adaptive features may unacceptably extend the transmission time of messages as it attempts to circumvent fault or hot spots. The extreme of this condition results in messages that never reach their destination hence the term non-delivery. This is sometime referred to message livelock.

Out of order delivery is an additional problem in adaptive packet switched networks. As the amount of adaptivity increases, the probability of packets arriving out-of-order also increases. The probability of this increases further with larger buffer sizes in the nodes, and therefore it is a concern in many adaptive networks. This can be solved with restrictive measures, such as packet level acknowledgements, which prohibits transmission until the arrival of previous packet has been confirmed [47]. Alternatively, control information can be sent with each packet, which allows correct ordering at the receiving node [48].

Adaptive routing may be implemented in varying levels, which is reflected in the effects in the implementation of the architecture of the switch. Many systems have been devised, which normally fall under the categories of minimally adaptive, partially adaptive or fully adaptive.

Recall that oblivious routing algorithms contain one route between all source and destination pairs. Minimally adaptive algorithms extend the number of possible routes, by

a finite amount, along which the message may travel to reach its destination. Group adaptive routing [49] is an example of a minimal adaptive algorithm. This technique allows a number of point-to-point connections within the network to be logically associated with one another. Thus, if a message requires the resources of a point-to-point connection that has been defined as part of any given group, it may use any of those its associated connections to reach its destination. In this way, group adaptive routing attempts to distribute messages over many links to maximise link bandwidth, which has been shown to improve the cross-sectional bandwidth of the network [50]. This in turn simplified network implementations where more bandwidth was required in localised areas of the network to ease bottlenecks. This method only allows the redirection of a message where configuration has made it possible, and therefore a tight control on the routes that a message may take is maintained. Such control on the routes provides some adaptivity without interfering with any measures that may have been taken to tackle network deadlock, but at the price of extra physical links. In addition to these benefits, grouping also could provide a small level of fault tolerance. By supplying *n* alternative physical routes by grouping, the network can tolerate failure *n-1* faults between a given source and destination.

Partially adaptive algorithms attempt to supply more flexibility than minimally adaptive systems. This is achieved by supplying extended routing freedom but with some method of limitation to ensure that deadlock or non-delivery does not occur. Virtual channels are often employed to prevent deadlock, as in systems devised by Dally [51], Duato [52] and Bolding [53]. Such systems use the virtual channels as escape paths from deadlock, where a number of channels allow unrestricted progressive routing, but the remaining channels operate on a deterministic algorithm that could be proven as free from deadlock or non-delivery. Other partially adaptive systems restrict the routing decisions to ensure deadlock cannot occur, as with the turn model suggested by Glass [54]. These mechanisms will be discussed further in their relationship to deadlock in section 2.2.3.7.

Research has shown that partial adaptive algorithms utilise network throughput more effectively than equivalent oblivious systems [55]. In addition to improving performance, the adaptivity supplies certain levels of fault tolerance as the dynamic nature of the routes allow faults to be bypassed. However, many adaptive algorithms use topology information in their implementation, which restricts their use.

More recently, the limitations of partially adaptive algorithms have been criticised as being wasteful, as the restriction of routing decisions or reservation of resources may prohibit the full use of all possible routes for the sake of deadlock freedom. Fully adaptive algorithms removal all routing restriction from the decision processes, which allows for the full resources of the network to be utilised. Recent systems by Pinkston [56], Martinez and Lopez [57] have devised deadlock detection mechanisms, which then trigger recovery mechanisms. With the use of a detection and removal technique that incurs little operational overhead, it has been shown that the lost bandwidth due to routing restrictions can be recovered. These mechanisms will be discussed further in their relationship to deadlock in section 2.2.3.7.

#### 2.2.3.5 Routing Decisions

For the packets of a message to reach the correct location through a communication network, they must carry sufficient information for the transfer within the packet header. There are a number of different methods, with which the information can be formatted, allowing the routing device to decide which connection should be made. At the highest level, this can be divided into source routing and network routing. Source routing requires the source node to inject each packet into the network with a header that dictates the route through the network by some form of encoding. Conversely, in network routing the source injects each message with the identifier of the output and route taken is decided by the network. For switched embedded parallel or distributed systems, source routing is used in the majority of systems as a preference, as static topologies and source routing requires lower implementation resource requirements.

In all of the networks that have been review for this work, routing information is sent as part of the data stream, which compromises the use of data bandwidth for minimal physical resource requirements. It is interesting to note, however, that some more specialist, high-performance parallel systems provided routing information by separate resources to dedicate the data stream for maximum performance; for example the CRAY T3D [3].

To described the most prevalent methods for route decoding of source-routed networks five addressing modes have been identified, which are used to encode the path through the network. They are:

- Physical (Absolute) addressing [1],
- logical addressing [2],
- interval addressing [35],
- relative addressing [58],
- graph addressing [53].

Figure 2-7 will be used as a point of reference for the description of the first four of the addressing modes, as it depicts a simple connection structure of a theoretical switched network.

**Physical (absolute) Addressing**

Physical addressing is the most basic method, with which a message can be supplied with a route through a network. The size of the header information is directly proportional to the number of switches between the source and destination nodes, as each unit of information contained within the message header relates to a routing decision at each switch. For physical addressing the value of each unit of information directly relates to the destination port number of the relative switch. For example, for a message to travel between node 1 and node 2 in Figure 2-7, the routing header must contain one unit of information with a value of {1}. However, for a message to travel between node 1 and node 5, the header must contain two units of information, namely {2, 3}. This technique infers that each unit of information is useless once it has been used as the message traverses the network. For this reason, this unit of information is normally removed from the message. This technique of removing the routing information is called header stripping.

The mechanism for physical addressing decoding is easily implemented in hardware and requires only a small amount of resources. Additionally the routing decision can be made very quickly. Such advantages have made this technique popular in small switched-networks. Hardware support of multicast is possible with this form of addressing, although its implementation is limiting for the protocol. By using the routing header as a bit vector for desired outputs, the functionality can be achieved, but once the specification sets the size of header, the size of router-switch is also limited. Additionally, the definition of a large header would be detrimental to routing performance.

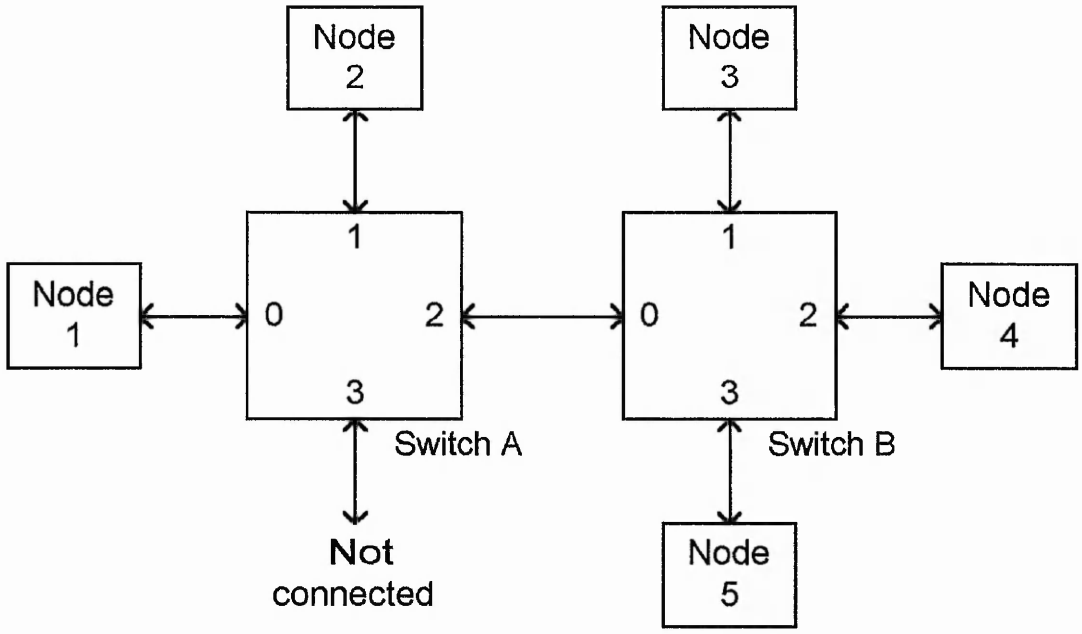



**Figure 2-7 : Depiction of a theoretical network showing five communicating nodes connected via two, four-port switches**

The limiting feature of this methodology is the number of intermediate switches between source and destination, as each router-switch requires its own piece of routing information. Networks with a high number of switches between communicating nodes would demand routing information that would make the system impractical.

**Logical Addressing**

Logical addressing increases the level of encoding that is implemented in the units of the routing header, and as such, more information can be inferred by each unit of header information. Through the configuration of the network, each unit of routing header can specify a single path of definite length through the network, which lowers the routing header overhead in router-switch networks when compared to physical addressed systems. Table 2-1 presents the configuration that would be required for the example network of Figure 2-7. Using this configuration, a message may travel between any source and destination by using only one unit of routing header information. For example, a message travelling from node 1 to node 2 requires a single unit header of {2}, and a message travelling from node 1 to node 5 requires a single unit header of {5}. It must be also noted that specific headers may also be classed as invalid if the destinations are not in use. Reaction to these invalid headers must then be defined for a complete decoding strategy.

**Table 2-1 : Configuration details for switches A (left) and B (right) in the network as depicted in Figure 2-7**

| Switch A | | Switch B | |
|---|---|---|---|
| Header Unit Information | Destination Output | Header Unit Information | Destination Output |
| 0 | invalid | 0 | invalid |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 2 | 0 |
| 3 | 2 | 3 | 1 |
| 4 | 2 | 4 | 2 |
| 5 | 2 | 5 | 3 |
| 6 | invalid | 6 | invalid |
| 7 | invalid | 7 | invalid |

In the example provided here, the configuration tables, the switches, and the number of destinations are much simpler than would be found in most applications. To supply enough routing possibilities for a reasonable sized network two options exist. The first option is to possess configuration tables that are the same size as the total number of individual destinations. This demands a high amount of hardware resource at each switch. The second option is to introduce header stripping as an optional feature on some parts of the network. In this way, each unit of header information may be used to route within an area of the network, or cluster. Stripping options can be configured at each router-switch that is associated to each header (or destination port) to create these areas. As the message crosses between one cluster and another, the first unit of header is removed. Thus, multiple clusters may be used to create any size of network, only limited by the similar constraints associated with physical addressing.

Logical labelling provides more practical support for multicast messages, which can be implemented using the same method of header decoding, although hardware resource requirements increase. The large amount of hardware resources is the main drawback to logical addressing. The configuration tables at each router-switch node must be of a reasonable size to provide an adequate size of network, even with optional stripping. In addition to the resource consideration, the decoding of the routing header is more complex when compared to the physical addressing and the network must be configured before the system can be used efficiently.

**Interval Addressing**

Interval addressing allows a less flexible but similar routing operation to logical addressing. By the use of network configuration, interval addressing is used to define message routes through a network. To use this technique effectively, a holistic view of the network must be used. Network nodes that are local to each other with respect to the physical topology are provided with adjacent address numbers. In this manner, the network is constructed such that at any point in the network paths will lead to a range of destinations. Thus, by labelling each output at all switches with a range of address intervals, all the valid paths for the network can be defined. In this manner, a single unit of routing information can be used by many if not all switches in the network. The manner in which the intervals operate may vary slightly from one implementation to the next, but there are two rules for interval decoding that must be followed to ensure legal operation. They are:

1. Interval blocks must be disjoint and sequential.
2. Addresses that do not fall into any specific interval must be dealt with in a predefined manner.

Table 2-2 shows an example configuration for interval decoding for use with the theoretical network of Figure 2-7. As stated before, complete routes may be defined by a single unit of header information, thus, for the basic network of the example its operation would be identical to that if logical addressing was used. Thus, a message travelling from node 1 to node 2 requires a single unit header of {2}, and a message travelling from node 1 to node 5 requires a single unit header of {5}. Additionally, intervals of invalid headers within interval addressed networks are marked and must dealt with, as described of the illegal headers within logical addressed networks.

**Table 2-2 : Interval configuration for switches A (left) and B (right) in the network as depicted in Figure 2-7**

| Switch A | | Switch B | |
|---|---|---|---|
| Interval Limit | Destination Output | Interval Limit | Destination Output |
| 0 | invalid | 0 | invalid |
| 1 | 0 | 2 | 0 |
| 2 | 1 | 3 | 1 |
| 5 | 2 | 4 | 2 |
| 7 | invalid | 5 | 3 |
| | | 7 | Invalid |

Interval addressing possesses all the advantages of logical addressing with the exception that it is not suitable for switched based multicast routing. However, as the configuration tables of the logical and interval addressing schemes show, less hardware resources are required for interval addressing than for logical addressing. The number of intervals required for each router-switch is normally a small number above the number of ports. For example, a commercial routing device, the STC104, possesses 32 ports and only 37 intervals that can be used to address a network of 64k destinations before optional stripping is used [59].

**Relative Addressing**

Relative addressing operates with a similar technique to physical addressing with the exception that the decoding of the unit of routing header is effected by the input, at which the message arrived. In practice, the unit of header is normally a signed binary value, and the destination is produced by summing the header to the value of the input. As with physical addressing, each unit of header information is only valid for one router-switch along the route and therefore must be removed after its use. This technique was designed to allow a simple calculation of the return route of any possible routes within any network topology. Due to the relational method of this encoding, the return route may be calculated by inverting the order of the headers and inverting the sign of each header. Unfortunately, relative addressing inhibits the use of any dynamic adaptive routing techniques.

Returning to the example network of Figure 2-7, a message from node 1 to node two would require a single unit of header information with a value of {1}. To make the return path the value would be {-1}. For a message from node 1 to node 5, a two unit

header would be require of {2, 3}, with a return path of {-3, -2}. Table 2-3 contains all the valid headers for a four-port router-switch that uses relative addressing. Values that do not conform to those listed would be considered illegal, and recovery procedures would be required.

**Table 2-3 : Valid routing headers for relative addressing for a four-port switch**

| Incoming Link ID | Required Destination Link ID | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | 1 | 2 | 3 |
| 1 | -1 | 0 | 1 | 2 |
| 2 | -2 | -1 | 0 | 1 |
| 3 | -3 | -2 | -1 | 0 |

**Graph Addressing**

The routing information used in graph addressed switches is presented in the form of an offset relative to the current location in the network. Thus, this type of routing demands that such networks must be connected in a specific network topology, such as a 2D mesh network. The routing information is modified as each translation towards the destination is made.

Knowing the topology permits a more intelligent manner of routing algorithms to be implemented in hardware, which can supply improved network bandwidth utilisation and supply fault tolerance by routing messages round faults. However, the restriction of network topology, which provides these improved routing strategies, does limit the flexibility and application of the resulting network. In addition, as the routes taken by messages are controlled by hardware routing algorithms and network loading, only retransmission based multicast messages can be implemented.

#### 2.2.3.6 Fault tolerance – Error detection and recovery

It is a primary aim of a communication system to supply an error free method to transfer information from one place to another. The aim of a network layer error detection mechanism is to flag errors on a packet basis. This normally utilises a block checking technique. Mechanisms, such as the cyclic redundancy check (CRC), are a popular for block error detection at the network layer [60]. CRC are calculations that are based on a polynomial equation, on which the data of the message is applied. The result is sent as part of the message, such that when the whole message has arrived at its destination, the same calculation may be carried out for comparison with the received value. While such

systems are not infallible, the length of the polynomial can be extended to reduce the probability of not detecting an error to an acceptable level for the application environment. The detection of an error usually triggers a recovery mechanism, which may include a request for the retransmission of the data at fault or inhibit the transmission of a packet acknowledgement. Such mechanisms are not within the scope of this work, and therefore will not be addressed further.

#### 2.2.3.7 Deadlock

Deadlock can be simply described as the network state, in which a number of messages stall and prevent delivery of one another due to resource contention through cyclic dependency. That is, two or more messages cannot proceed as the required resources are held by one another. Figure 2-8 shows a simple example of a single cycle deadlock. Much work has been carried out to analyse deadlocks in switched networks, providing an insight into how complex the subject area is [61, 62].

Much work has been undertaken to combat deadlock in router-switched networks. Lopez [57] highlighted three strategies for handling deadlock: prevention, avoidance and recovery.

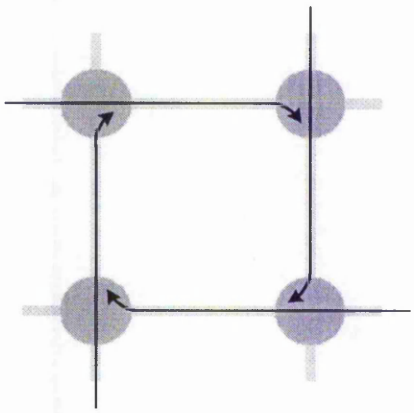


**Figure 2-8 : Depiction of deadlock in a four node system**

**Deadlock Prevention**

Prevention ensures deadlock freedom by reserving all the resources from source to destination before transmission is undertaken. This functionality implies some form of circuit-switching technique. Pipelined circuit switching was devised to provide a fault tolerant version of a circuit-switched technique [63], which decouples the path set-up and data transmission stages. This allowed adaptivity, as the routes could be backtracked to

route around faults. Once the path was fully connected, a path acknowledgement was returned via the reserved path to initiate the transmission of data. However, pipelined circuit switching was a very conservative mechanism that was comparable to the techniques that initially encouraged the move from circuit switching to packet switching. A subsequent mechanism attempted to amalgamate the benefits of pipelined circuit switching and wormhole routing, which was called Scouting routing [64]. Scouting routing decoupled path set-up and data transmission, but the path acknowledgement was generated on a per hop basis and the time of generation was determined by a network variable. Thus, the network could operate as a wormhole switched network or pipelined circuit switching network, dependent on the variable. The technique was proposed to operate with a range of delays that are dependent on routing decisions, which would supply enough flexibility to backtrack where required.

**Deadlock Avoidance**

Avoidance techniques use strict rules in the routing algorithms to ensure deadlock-generating cycles do not form. There are many algorithms for regular topologies, but irregular topologies are less predictable and therefore the majority of systems operate a derivative of the same technique. In irregular topologies, source routing networks know all possible routes and therefore the avoidance algorithm is used to define these predefined routes. Regular topologies may use oblivious or partially adaptive routing algorithms, thus the avoidance algorithms normally are part of the network fabric as part of the routing decisions.

The implementations of avoidance mechanisms on irregular networks predominately are based on a technique that uses a tree structure of valid routes. If all messages use the predefined messages, then cycles should not form. Autonet [40], a LAN replacement for ETHERNET that was devised by Digital Equipment Corporation (DEC), used a tree based methodology called up/down routing for unicast and multicast communications. A message was said to be routed 'up' if the transition made the message move closer to the root node of the spanning tree. Conversely, a message that was routed 'down' was said to be moving away from the root node. To ensure deadlock freedom, the message had to adhere to a routing rule that: "A legal route must traverse zero or more links in the up direction followed by zero or more links in the down direction." By forcing the order, in which the transitions are made towards the destination, cycles are prevented and therefore deadlock is impossible. Unfortunately, this restriction may force the

message to take a non-minimal path, possessing more router-switch transitions than is necessary. This has stimulated research into improving this algorithm such that minimal route can be taken advantage of, whilst maintaining deadlock freedom [65].

Regular networks are predictable in their interconnection graphs, thus the techniques to provide deadlock avoidance has continually evolved. An early solution was similar to the up/down routing algorithm of the irregular network, where the message followed a priority based routing algorithm, called dimension order routing. The routing rule for dimension order routing was as follows (paraphrased from Dally [51]):

> *A packet is routed along each dimension until it reaches a node whose address in dimension **d** matches the address of the packet destination node in the same dimension. If the addresses match, then the packet continues to route in the next lower dimension where the current channel address and the destination address differ. This continues until the packet reaches its destination*

Although dimension ordered routing took a minimal path from source to destination and guaranteed deadlock freedom, the limited path did not consider the dynamic state of the network. Thus, network resources could be wasted and network faults could not be circumnavigated. This encouraged the development of partially adaptive routing algorithms. One of the earliest was the turn model proposed by Glass [54] for multidimensional mesh and cube networks. The fundamental concept was to prohibit turns that would cause cycles to be formed, which was an evolution from the dimension ordered routing algorithm where a maximum number of turns were prohibited. By inhibiting only one turn per possible cycle, it was possible to supply more adaptivity while preventing the formation of cycles.

Other methods of partially adaptive, deadlock free algorithms utilised virtual channels to progressively restrict routing options, to prevent cycles forming [34]. A number of virtual channels are associated with each physical link, and virtual channels are ordered, similarly to the dimensions in dimension ordered routing. Thus, routing is restricted to visit channels in decreasing order to eliminate cycles and therefore prevent deadlock.

The distinct disadvantage of preventive algorithms is loss of network throughput, which is caused by the routing limitation. Additionally, many of the techniques are prone to failure if the data stream introduces bit errors.

**Deadlock Detection and Recovery**

In recent years, deadlock recovery methods have been shown to be suitable replacements for earlier preventative and avoidance strategies. Pinkston [66] provided simulation data showing the performance advantage recovery based systems have over a number of selected avoidance techniques. The work on these recovery mechanisms declare that recovery techniques can improve on avoidance and preventive techniques as the networks can be used more effectively and the cost of the recovery mechanism on the performance of the network is low if it is executed infrequently. They further state that the probability of deadlock in a fully adaptive network only approaches unacceptable levels as network approaches saturation, which can be limited by controlling the volume of traffic. However, he notes that the efficiency of the detection and recovery algorithm plays an important role, and in some situations, avoidance will provide better results. This is because the loss of bandwidth through resolution can be greater than the overall bandwidth restrictions of preventative and avoidance techniques. Other research undertaken by Martinez [67] suggested a restricted message injection technique to prevent the network from becoming saturated. This maintains a low deadlock probability, thus allowing routing restrictions to be relaxed, with confidence that the recovery-based system will operate efficiently.

All the systems used for deadlock detection, which have been reviewed, utilise some form of time related function. Deadlocks are flagged when data transfer has not taken place, or delivery has not been completed in a predefined time-period. The most basic systems work purely the on the progress of data through the network and the time-period. If no progress is made within a specified time limit, the message is flagged as locked, and the recovery procedure is executed [68, 69]. While these systems can provide guaranteed detection of deadlock conditions, thus ultimately providing the functionality required, they must be optimised to suit the network parameters; such as, network topology/diameter/throughput, and packet sizes. Without optimisation, time-based mechanisms loose efficiency either by using time-outs that are too short or too long. Short time-out periods cause higher numbers of false detections, which forces wasteful use of

resolution procedures. Conversely, long time-out periods allow the networks to stall for long periods, which lowers network availability.

Lopez and Martinez devised mechanisms that attempted to minimise the number of false detections whilst identifying true deadlock conditions as quickly as possible, without dependency on network traffic patterns or workloads [57]. This is achieved by the identification of the root of the cyclic dependency of the deadlock tree and execution of the recovery mechanism on that node. The root of a cyclic dependency tree is a single node, where if the message is removed from the cycle all other message will be able to continue along their route unhindered. Although Lopez stated that "a deadlock mechanism should propagate information along the branches of the tree of blocked messages indicating there is no deadlock", he maintained that information provided by the flow-control mechanism was adequate for deadlock detection. In practice, his deadlock detection mechanism used 'data flow' status flags and timers that were local to each switch. The deadlock recovery mechanism was invoked when the timer, which being monitored by the message marked as root, elapsed. Lopez used a single premise to identify a root, which was: "A message shall only be flagged as a possible root of a deadlock cycle if the message blocks whilst waiting for resources occupied by a non-blocked message that later becomes blocked". While this mechanism still relied on the expiry of a time-out, the action of minimising the number of messages flagged for recovery, improved the effectiveness of the mechanism to that of basic time-outs. . Their results showed that with the timers set to an optimum threshold to suit predefined network traffic, low false detection rates could be maintained.

Pinkston [70] quantifies three procedures for deadlock recovery: deflective, progressive and regressive. Deflective methods encompass the systems that route a message out of the dependency cycle effectively moving it away from its destination. Martinez [67] uses an example of deflective recovery, where selected messages are redirected to the processing node local to the router-switch. Progressive methods are similar to deflective methods, where messages are removed from the cycle, but the selected message still moves towards its destination. Pinkston presents his mechanism, DISHA [68], as progressive. This mechanism implements a by-pass route and extra buffering at each router-switch, which supplies a minimal deadlock-free path to the destination. Finally, regressive methods remove the selected message from the dependency tree by deletion of the messages from the network. Following deletion, the messages must be transmitted again by the source.

### 2.2.4 Transport Layer Considerations

The transport layer would not be normally associated with a hardware mechanism. However, certain aspects of the transport layer affect the lower levels such that it cannot be ignored. These aspects are the segmentation of the data to suit the lower level, message level error detection. Segmentation of the data, or packetisation, is a common theme in the majority of communication protocols. In earlier point-to-point systems, the devices would establish a connection between source and destination, after which the entire message would be transferred. Modern systems maintain the connection based system, although the physical connection has been replaced by virtual connections. The abstraction of the middle layers gives the appearance that a permanent connection exists to the higher layers. In reality, many devices may be sharing the same network resources, and thus the communication must be split into smaller parts of a predefined size to suit the network. The aim of a transport layer error detection mechanism is similar to that of the network layer error detection procedures, but the detection mechanism is carried out on the message as a whole. Again, as in the network layer, the transport layer normally utilises a block checking technique.

# 3 Review of Earlier Research

A critical breakdown and discussion of earlier research work is provided, which covers research completed by previous members of the research group and some comparisons with other contemporary devices. These topics will be address in terms of the lower three layers of OSI model; the transport layer of these systems will not be addressed.

## 3.1 Introduction to Earlier Systems

To aid the reader, this section provides a brief overview of the systems under review, which discusses grounds for development and target applications.

### 3.1.1 NTR08

The first router-switch design was the NTR08 [1]. The device was an eight-port dynamic packet routing switch targeted for use in building parallel processing networks using the Transputer microprocessor. The research concentrated on solving a communication bottleneck suffered by the first generation transputers. Transputers were produced by INMOS and later by SGS-Thompson, they first appeared on the market in 1985. They employed integrated serial communication links that enabled them to be used as building blocks for constructing parallel systems. This serial communication link was based on an over-sampled asynchronous protocol that will be referred to as the 'OS link' from hereon [71].

Transputers were revolutionary for that period, due to their architecture and operational methodology, which followed the communicating sequential processes (CSP) model as expounded by Hoare [16]. Simply, the CSP model operates on the premise that any task can be broken down into smaller tasks (or processes) that synchronise and pass information in the form of messages. Transputers supported the CSP model in hardware, which meant that the first programming language, OCCAM, formally complied with the methodology and systems could be mathematically proven. Sharing data by self-synchronising channels, as specified by CSP, removed the need for extra mechanisms for mutual exclusion, such as monitors and semaphores, thus reducing software requirements. In the transputer family the OS links directly supported the self-synchronising channels of the software model, and provided one-to-one logical to physical mappings for off-chip communication channels. As such, at the user-level, there was no difference in

communication between local or remote processes, which meant that a system could be executed on one or more transputers without further software development.

The main design premise of the transputer family was the minimisation of the board-level component count, as it increased reliability and reduced production costs. As such, many transputer family members integrated standard parts with the transputer communication system. These devices allowed the construction of systems that were much simpler in structure and therefore much easier to design and build. The OS links provided an efficient system to connect transputers, like building blocks, to create systems of any size. INMOS claimed that any system could be expressed using the range of transputer products by localising closely related tasks to overcome physical communication limitations. Using such a localised design methodology theoretically supplied a flexible distributed processing topology where processing power could be supplied to areas that required it.

Using task localisation with the transputer was very efficient in many applications, particularly those that were naturally parallel, but many other application areas required more inter-processor connections than were available. As system complexities and system sizes increased, interconnection requirements also became more elaborate, which could not be supported by the simple logical to physical one-to-one mapping communication methodology of the transputer. Early research into the physical limitations of the transputer brought about two solutions; programmable crossbar switches and software routing mechanisms. INMOS developed a programmable crossbar switch the C004 [72], which was designed to be used statically. In the standard operation of networks constructed with the C004, the connection topology was set before running a statically configured program. Independent research was carried out using the C004 for dynamic switching [73, 74]. This demanded high system complexity to ensure communication integrity, which made such solutions impractical in most applications. The other solution of using software routing algorithms utilised processing time to redirect messages towards their destination, such that many CSP channels could be multiplexed on a physical link [75]. For networks that contained low volumes of messages, this solution was acceptable but as the message workload increased, the routing began to affect the efficiency of the network to unacceptable levels. Clearly, these two methods for message distribution were problematic.

Research resulting in the NTR08 device proved the feasibility of networks constructed with a multi-channel hardware message routing device, which combined ideas from both earlier solutions. This was demonstrated to provide more efficient implementations than many software-based equivalents [76].

### 3.1.2 ICR-C416

The ICR-C416 [77] was a commercial development based on the research of the NTR08. The device is an industrial standard, sixteen-port hardware packet routing switch, which possessed all the routing features developed from the NTR08 research.

### 3.1.3 NTR-M04

Development of the NTR-M04 [2] followed the completion of the NTR08. The NTR-M04 was a prototype four-port packet routing switch that investigated hardware support for multicast connections.

The NTR08 proved successful in improving network efficiency for the first generation transputer networks. During the period of the development of the NTR08, the second-generation transputer, the T9000, and C104 packet routing switch were produced. Unfortunately, there was limited interest in the T9000 and as support for the first-generation transputer dwindled, the use of both generations declined. Initially the NTR-M04 was targeted towards the second generation of transputers, which was based on the DS link protocol [80]. Its main aims were to improve certain design aspects of the NTR08. During the early stages of project development, a decision was made to take a step away from transputer systems and concentrate on improving the routing devices for the target applications of embedded and distributed processing. This decision produced a modified list of aims, which included protocol modifications, hardware support of multicast messaging, review of device configuration, change of design entry techniques, change of target technology and preliminary investigation of embedded fault tolerant features.

By moving away from supporting transputer products, the NTR-M04 could customise the network protocol to suit system ideals. The NTR08 had to create a hybrid protocol such that it could operate with transputers. This introduced limitations in the system, as the OS protocol was not designed for such use. Similarly, the DS protocol was initially designed for use with the T9000, which supplied rigid compliance to the lower four layers of the protocol via hardware and configuration software. As the NTR-M04 was

targeted towards a heterogeneous switched environment, the networking protocols were defined with system and router-switch considerations at the fore, which included alterations at all of the lower layers.

Multicast messaging was always problematic in switched networks as they are constructed using a disjoint set of resources and messages only use the resources that are necessary. While this is a disadvantage for multicasting, it allows for concurrent communication increasing the network throughput many times over the basic bandwidth figures. Section 2.2.3.2, highlighted research related to multicast messaging on point-to-point systems. The NTR-M04 aimed to improve the network operation through hardware support of multicast connections, which effectively removed hot spots from the network ingress points over unicast derived solutions.

The review of device configuration targeted features that supplied flexibility in the configuration of the network. The earlier NTR08 possessed a range of comprehensive configuration, monitoring and intervention features that were available through a single connection that was removed from the data network. The resulting consequences of this architectural decision were considered and an alternate solution was presented.

With the progression of semiconductor technology, many systems have been devised to improve the speed of implementation and reusability of proven design structures. The NTR08 was implemented using an accepted design flow that included schematic design entry. At the commencement of the NTR08 project, hardware description languages (HDLs) were starting to become industrially accepted, although tool availability, support and reliability were low. HDLs could be compared to parallel programming languages that allow the definition of system or device operation at many levels of abstraction. This allows the language to be used at more levels of the design flow and quick re-targeting of technologies, but for hardware implementation the technique relies on the abilities of the tool-set for optimisation and correct synthesis. With effective tools, the design cycle of a project can be dramatically speeded up with the use of HDLs when compared to schematic entry techniques. The change in design entry also arrived with a change in target technology. The NTR08 was implemented in a gate array technology, which incurs high non-recurring engineering costs, but low manufacturing costs for high volumes. This made development of niche products, like the router-switch, an expensive exercise. For lower initial costs and small volume products, programmable logic devices would have been more suitable, but early products were too basic to be used

for any reasonably complex task. The last five years has seen significant improvements in this technology, which has seen them move from being used as small decode or interface logic to the main application devices. Whilst the unit cost is relatively high for the larger PLDs, the overall cost saving in prototyping, and in some areas full scale production, has seen them take a large proportion of the ASIC market. The improvements of HDL design tools and PLD technologies allowed a new design flow to be tested within the development cycle of the NTR-M04.

Finally, the NTR-M04 work included a preliminary investigation of embedded fault tolerance. It was recognised that the monitor and intervention techniques of the NTR08, whilst effective, could result in complex and expensive system solutions. It was the aim of the project to highlight areas of development to supply a more resource effective solution.

### 3.1.4 Contemporary Devices

Throughout the development of both projects of the NTR08 and NTR-M04 there was much research into switching networks for parallel and distributed applications. To reflect other aspects of network design in these areas are introduced briefly. They are; the STC104 32 port router-switch [78], the 4+1 port Reliable Router-switch [79], and Myrinet a workstation clustering network system [30, 58].

#### 3.1.4.1 STC104

INMOS (later SGS Thompson) during the period of the development of the NTR08, released the second generation transputer, the T9000, and some time later the STC104 (formerly the IMS C104) packet routing switch. INMOS had recognised the limitation of the localised processing methodology and included the STC104, which allowed a modified communication system structure for their second-generation devices. This saw the move away from dedicated point-to-point systems to switched, virtual point-to-point systems.

The STC104 operated with an upgraded DS link protocol that replaced OS link protocol of the first generation of transputers. The DS link, so called due to the data-strobe data recovery technique, on which it was based, operated using a four-wire, self-synchronising protocol. The DS link was specified to operate at a variable bit rate up to 200 Mb/s [80]. The system protocol was later formalised by the Institute of Electrical and Electronic Engineers into the IEEE 1355 standard [81].

Although the T9000 never saw the popularity of its predecessor, the STC104 maintained a position in the market, which saw it accepted as a device disjoint from Transputer systems, through the IEEE standard.

#### 3.1.4.2 Reliable Router

In addition to the work centred around distributed systems, there were a number of switches that were designed specifically for use in high-performance parallel systems only examples include the Reliable Router [79], Chaos router [82] and the WARRP [25]. The Reliable Router is a typical example of such switches; it saw most of its development in the early to mid 1990's. The target applications of the resulting networks were dissimilar to those constructed with the NTR08 and the C104, but some overlapping of ideals occurred. The primary difference between switches such as the NTR08 and the Reliable Router was one of evolution. As the Reliable Router and its brethren were designed for high performance parallel processing, their functionality was refined, restricted and optimised. The removal of extraneous features refined the devices; for example, they include enough communication links to suit the system, in which they operated. Restrictions were imposed that simplified aspects of their operation, which in turn improved their efficiency.

#### 3.1.4.3 Myrinet

Myrinet [30, 58] was designed to operate as a point-to-point, switched, interconnection system for clusters of workstations, PCs, or single-board computers of irregular or regular topologies. Other systems have been used to cluster workstations, Ethernet being the most obvious example. In this role, the demands on the communication system are large, as they must provide high data rates with low latencies. Myrinet claims to achieve both these aims with built in fault tolerance to detect and isolate faults.

As a commercial product, all components for the Myrinet system are available for purchase. It was first shipped late 1994 and since then has gained wide support and may be used with a number of operating systems. Since this first shipment, the Myrinet system specification has been updated, although the primary aspects remained the same.

## 3.2 Physical Layer

To provide a valid solution for use with transputers, the NTR08 protocol had to be defined to be compatible to the physical asynchronous serial communication system and allow for a scaleable, flexible connection topology. This was the first project challenge, as the protocol was designed for reliable single point-to-point communication. The OS link protocol defined two serial lines, which were used as a co-operating pair that transmit both data and control information to form one full-duplex communication link. The OS link specification was primarily a board-level protocol, or board-to-board connection via a backplane, and as such was designed to operate in a low noise environment. However, the OS link was not limited to board level implementations. For off-board applications, INMOS recommended the use of additional drivers. Recent research based on a CPLD implementation of an OS link has shown that the technology can be used up to 100 metres with suitable driver circuits, even at over double the originally specified bit rates [6]. The asynchronous nature suited the embedded applications based around the communicating sequential process model. The asynchronous serial aspect of the protocol was a popular for the era in which it was developed. It operated at similar bit rate to the RS422 but utilising a point-to-point nature similar to the RS232. The over-sampling technique was favoured as it simplified synchronisation at the receiver. Although in present terms the link bit rate is considered to be slow, the implementation resource requirements are low whilst supplying a very reliable physical layer.

In contrast to the NTR08, the STC104 operated with the DS link protocol. Each link was bi-directional, using two wires for each direction. The line encoding of the data-strobe was introduced in section 2.2.1, which highlighted that the transmitter clock was embedded into the signalling. A negative aspect of this method was that the link had to operate continuously to maintain synchronisation, which could demand large power requirements for large networks, even if communication was infrequent. Additionally, the wire pairs had to be accurately matched for off board transmission, as too much signal skew between the signal lines would make data recovery impossible. The DS links were specified for use at distances up to 1 metre without additional drivers, up to 10 metres with differential drivers and up to 500 metres using fibre optics.

The Reliable Router took a greatly different approach to that used by either the NTR08 or STC104. Since the primary aim of a parallel processing network was the overall speed-up of task execution, the network had to be able to move data as quickly as

possible with least amount of delay at each intermediate step. The Reliable Router achieved this by streamlining the switch to maximise throughput. The links of the Reliable Router was specified to operate at a bit rate of 3.2 Gb/s. The high link transfer rates were achieved using parallel links, as the Reliable Router used twenty-eight separate lines per bi-directional link. These links included eight control lines, sixteen data lines and four clocking lines. Such parallel links were a problem in these types of switches, as the number of available pins on the device packaging was limited. As a high-performance parallel processing system, the physical links of the Reliable Router were defined for board-level or backplane use. The Reliable Router used plesiochronous data recovery [83]. This is where two communicating devices operate with separate nominal free running clocks. The transmitting clock was sent along with the data, and was used to read the received data. The data was then synchronised across the receiver and core clock domains. Specific details of this technique are beyond the scope of this work. In addition to this recovery procedure, the Reliable Router also used bi-directional signalling on a single line to reduce the high pin demanded by the link [24].

The development of the NTR-M04 saw the move away from the transputer protocols. In this way, it was possible to devise a protocol that could use the best features from the techniques that were already known. The over sampled technique of data recovery was proven to be a reliable technique that was suitable for distributed networks from the earlier work of the NTR08. Over-sampling forces a higher sampling rate than the bit rate. Thus, this lowers the bit rate by the multiple of over-sampling rate, which is governed ultimately by the limitations of the implementation technology. To redress this, the NTR-M04 protocol included a second optional signal line per physical link. The second line on the link was optional as, if at reset the second line was disabled, the link would operate with just one line. This differed from previous implementations of parallel links, as bits were not transferred in parallel, but separate data words were sent. Figure 3-1 depicts this operation. The system could be otherwise described as two links co-operating in the transfer of a single message. This had the benefit that the skew tolerance between the lines was much greater than that of the DS protocol, for example. With the DS protocol, it is stated that up to one bit of skew can be tolerated. In contrast, with the NTR-M04 only byte order was imperative, which provided a skew tolerance of half a data token between the two signal wires. This meant that non match cabling could be used, reducing the cost of implementation. However, the cost of implementation in silicon for the

NTR-M04 dual wire link was higher than first predicted. This form of link effectively used four receiver circuits and four transmitter circuits per physical link, plus additional control logic to share the data between the two parts. This aspect of the design is an optional method to double the link bandwidth, but at a higher design cost.

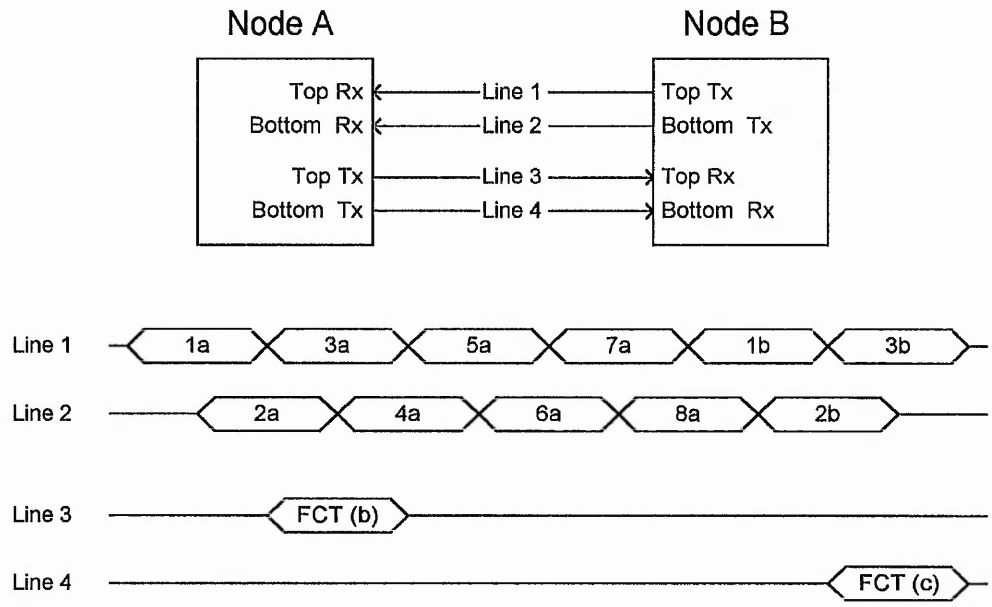



**Figure 3-1 : A diagram of the split link operation of the NTR-M04**

Similar to the Reliable Router, the Myrinet system [30, 58] also utilised a parallel connection over which bits of the same token were transmitted in parallel. The first generation of Myrinet [30] specified a nine-wire, parallel, physical link per direction, which comprised of eight data bits and a type bit. The second generation of the Myrinet [58] system contained two types of interconnect, which only differed slightly in the physical layer of the protocol. The two types were referred to as the SAN (system area network) and LAN (local area network) interconnects. The SAN interconnect was constructed of two, ten-wire physical links per direction, which included an additional dedicated line for permission based flow control. Whereas the LAN physical interconnect matched the specification of the first generation.

The first generation Myrinet protocol was specified with a data rate of 80 MB/s at a maximum transmission length of 25 metres. The second generation was specified at range of operating limitations. LANs are specified to operate at a data rate of 80 MB/s at a maximum length of 25 metres, and at a data rate of 160 MB/s to a maximum length of

10 metres. SANs are specified to operate at either 160 MB/s or 320 MB/s at lengths of up to 10 feet (approx. 3 metres). The alterations to the specification in the physical layer showed a desire for improved raw bandwidth in localised regions. This appears to be an obvious step taken from feed back from customers.

The physical medium of both Myrinet specifications used non-return-to-zero, mark (NRZ-M) encoding. However, the exception to this encoding was the flow control line of the SAN link, which was level encoded as it reflected the status of the receiver buffering. This NRZ-M encoding was covered in section 2.2.1. The receiver operated asynchronously, and a sampling window was triggered with the first detected transition on the link, per token. Each token was guaranteed to possess at least one transition by the valid token definitions, this will be discussed later in section 3.3.1. As the data recovery mechanism was triggered on the first transition per data transfer, the Myrinet specification called for a maximum skew limit of 40% of the character period between the first and last transitions of a token. It was further specified that cable skew should be no greater than 25% of the character period. These requirements equated to a skew limitation of between 781.25 ps and 1.5625 ns for SAN cables, and between 1.5625 ns and 3.125 ns for 25 metre LAN cables.

## 3.3 Data Link Layer

This section covers the data-link layer aspects of the devices and systems that are under review, considering basic units of information, data flow control and fault tolerance.

### 3.3.1 Token Definitions

The NTR08 link had to comply with the asynchronous, serial OS link of the transputer. The asynchronous bit-stream formed tokens that provided a means, to synchronise the data at the receiver. The receiver resynchronised to each token, which resulted in the full amount of skew tolerance provided by the over-sampling technique per token. Two tokens were defined; the *data token* and the *flow control token*. Figure 3-2 shows examples of these tokens. A data token encapsulated eight data bits within an eleven-bit token, and the flow control token was formed with just a start and stop bit. Using these two tokens, the protocol operated using a form of credit-based, back-pressure flow control, where the state of buffers *upstream* controlled the flow of data by propagating control information *downstream*, as described in section 2.2.2.2.

The OS protocol was designed for single point-to-point connections, with a one-to-one mapping of software to physical channels.

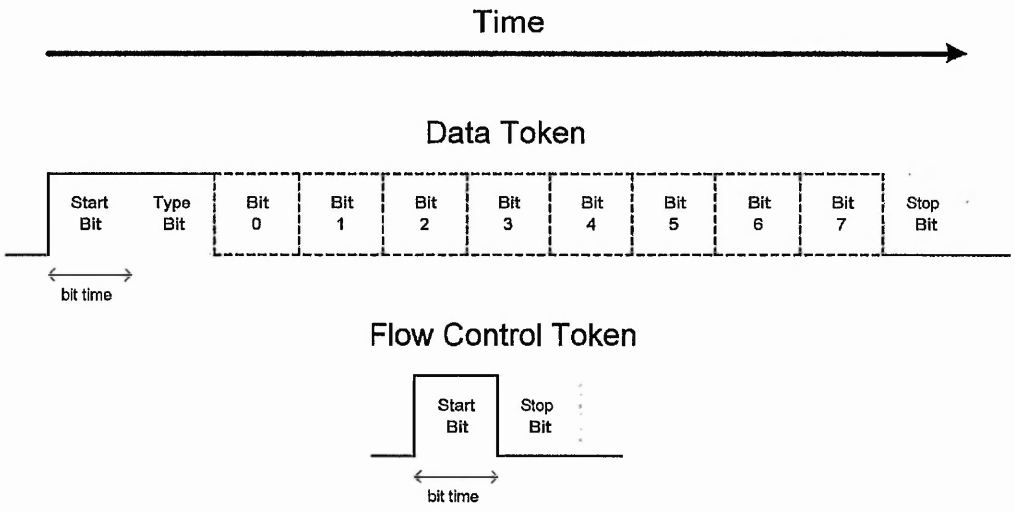



**Figure 3-2 : OS link protocol base tokens**

No fault tolerance aspects were defined as part of the data-link layer of the OS link protocol. However, token framing could be used to check synchronisation, but the NTR08 did not attempt to use this, as the bit error rates of the technology at board level and rack-level solutions were exceptionally low.

The communication methodology of the second-generation transputers saw differences between the OS and DS protocols. The limitations of the OS protocol resulted in the DS protocol supporting physical resource sharing through packetisation. This was implemented at the data link layer by extending the scope of the control tokens. The extended control token was four bits in length, including one control bit, a parity bit and two identifying bits. The data token was constructed of eight data bits, a parity bit and control bit, which resulted in a token one bit smaller than the OS protocol. This was possible with the alteration in the synchronisation technique, which reduced the protocol overhead at the expense of continual transmission to maintain a valid connection. Figure 3-3 shows the format of these two tokens back-to-back. The parity bit was used as a form of low-level fault detection. While the applicability of parity in modern communications is questionable as discussed in section 2.2.2.3, its use in the DS link protocol helped verify link synchronisation, thus making it integral to the solution.

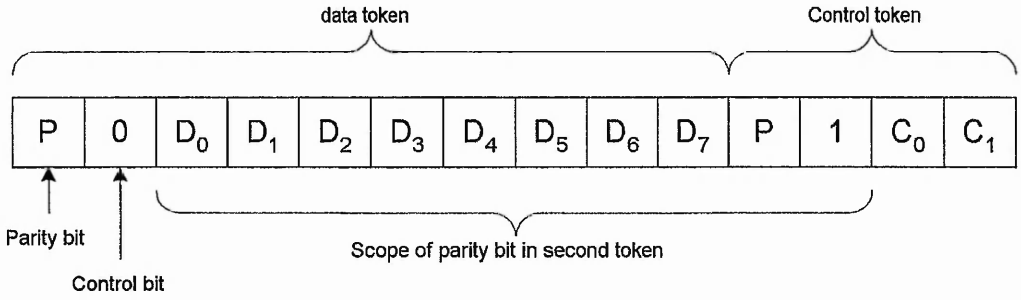



**Figure 3-3 : Format of the data and control tokens of the DS protocol**

The extended format of the DS link control token provided support for synchronisation support, flow control and packetisation, at the expense of increased overheads for control-signalling. Table 3-1 lists all token that were defined for use in the DS link protocol. In an idea similar to the ASCII escape code used in terminal communication, one control token was defined as an escape symbol. Although the escape symbol was only specified for use in one control token, namely the NUL token, this allowed scope for future additions or modifications to the protocol while minimising the control-signalling overhead for the basic control symbols. The NUL token was used to maintain synchronisation while there was no data to transmit.

**Table 3-1 : List of all type of token used in the DS link protocol**

| Token Identifier | Abbreviation | Coding [Parity, Type, LSB … MSB] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Flow control | FCT | P | 1 | 0 | 0 | | | | | | |
| End of packet | EOP | P | 1 | 0 | 1 | | | | | | |
| End of message | EOM | P | 1 | 1 | 0 | | | | | | |
| Escape | ESC | P | 1 | 1 | 1 | | | | | | |
| Null | NUL | ESC | | | | P | 1 | 0 | 0 | | |
| Data | DATA | P | 0 | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |

With the move away from the transputer architecture, the NTR-M04 protocol could be developed from the ground up for virtual point-to-point connections over a scaleable, switched network, unlike the hybrid protocol of the NTR08. This was reflected in the token definitions, which involved aspects of both the DS and OS protocols. The over-sampling, asynchronous features of the NTR-M04 link required a similar formatting to the OS protocol to support data recovery. Figure 3-4 shows the low-level formatting, where the primary differences between the OS protocol and the NTR-M04 is the position and value of the type bit in the token, and the length of the data token.

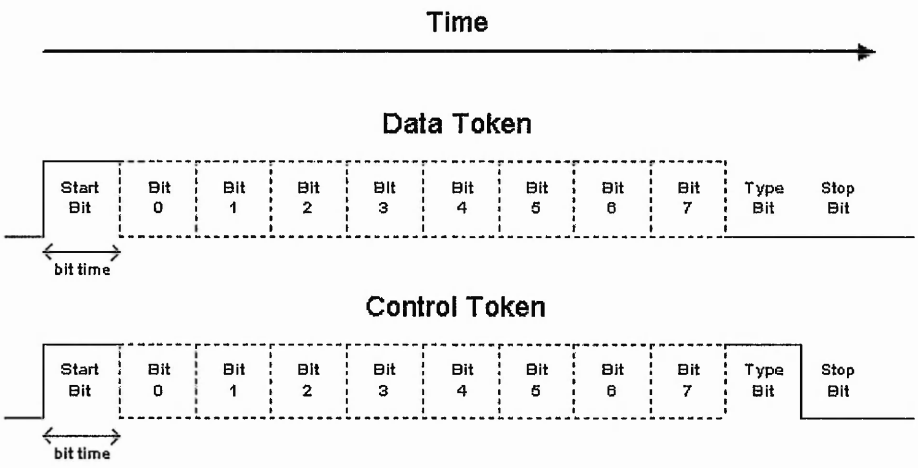



**Figure 3-4 : Token format used in the NTR-M04 protocol**

The eleven-bit control token provided scope for 256 unique tokens. Although it is unlikely that such a number would be used in any protocol, it supplied more flexibility and scope for protocol refinements, without complicating the receiver circuits. The similarities of the NTR-M04 protocol to the DS protocol were primarily related to the control tokens, as both systems were designed for packet based communication. Table 3-2 lists all of the tokens that were defined for use with the NTR-M04 switch. The function of all these tokens will be discussed in section 3.4.

**Table 3-2 : List of all defined types of token used in the network layer of the NTR-M04 protocol**

| Token Identifier | Abbreviation | Coding [LSB ... MSB, Type] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Flow control | FCT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| End of packet | EOP | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| End of message | EOM | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| End of block | EOB | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Forward reset | FRES | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Data | DATA | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | 0 |

Myrinet and the Reliable Router shifted multiple data bits in parallel for high data rates thus, the data formatting had to adhere to that. Myrinet possessed physical links that were nine bits wide. Comparing to the previous serial based systems, this relates to the eight data bits and one control bit, but as the bits are transferred in parallel, no further formatting bits were required. The synchronisation was based on the idea that all tokens of information would contain at least one logic '1' value. Table 3-3 lists all the tokens that

were specified in the first generation of the Myrinet system. The table shows that only the IDLE token contains all zeros, which ensures that each token of information will generate a transition with the NRZ-M encoding. The tokens marked with * were defined in the first Myrinet specification, but they were not implemented in any of the devices.

**Table 3-3 : List of all defined types of token used in the first generation Myrinet protocol**

| Token Identifier | Abbreviation | Coding [Type, MSB … LSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Flow control (tx off) | STOP | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Flow control (tx on) | GO | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Idle | IDLE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Packet gap | GAP | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Forward reset * | FRES | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Backward reset * | BRES | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| Over run alarm * | ORUN | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Probe nnn * | PRB | 0 | 1 | 1 | 0 | 0 | 0 | n | n | n |
| Reply to probe xxxx * | REPL | 0 | 1 | 1 | 1 | 1 | x | x | x | x |
| Data | DATA | 1 | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

In 1998, the second version of the Myrinet specification was released. Table 3-4 shows the modified list of defined tokens after the alterations to the specification. Most obvious is the removal of all the tokens that were not implemented in the first devices. Another omission to the specification of the second generation is the IDLE token. It is supposed that the definition of the token was superfluous, as the format of the IDLE meant no change to the receiver circuits, thus no actual transfer or event to record.

**Table 3-4 : List of all defined types of token used in the second generation Myrinet LAN protocol**

| Token Identifier | Abbreviation | Coding [Type, MSB … LSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Packet gap | GAP | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Flow control (tx on) | GO | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Flow control (tx off) | STOP | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Beat symbol | BEAT | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Data | DATA | 1 | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |

The data format of the Reliable Router also reflected the parallel physical interconnection, but multiple transfers were used to construct a single data unit. Four block transfers, called 'frames', were used to transfer seventy-five bits of information. The seventy-five bits were comprised of sixty-four data bits, eight parity bits and three bits of

type information. Table 3-5 shows the format of the four frames that were transferred as a single unit where 'BPx' are the parity bits for each data word and 'Kind' is the type information. Other areas of the frames include virtual channel information (VCI), the flow control information (Copied kind, Copied VCI and Freed), link status information (U/D and PE bits), and user bits (USR0, USR1). In addition to these fields, each frame was also accompanied with an extra parity bit. The format of the data unit shows the complexity required for very high bandwidth systems, which is in direct opposition to those systems previously addressed. Eight additional bits per frame, making a thirty-two bit overhead per 64 bits of data, is high, but allows the maximum signalling rate for the technology without bandwidth loss for control signalling.

**Table 3-5 : Frame format use for the Reliable Router as one flow control unit**

| Bit Field | 22 | 21 | 20:18 | 17 | 16 | 15:0 |
|---|---|---|---|---|---|---|
| Frame 0 | PE | USR0 | VCI | BP1 | BP0 | Data[15:0] |
| Frame 1 | Copied kind | | Copied VCI | BP3 | BP2 | Data[31:16] |
| Frame 2 | U/D | USR1 | Kind | BP5 | BP4 | Data[47:32] |
| Frame 3 | Freed | | | BP7 | BP6 | Data[63:48] |

### 3.3.2 Flow Control

Prominent examples of flow control were the OS and DS protocols that were used in the first and second generations of transputers [72, 84]. The acknowledgement aspect of the mechanism suits the operation of the transputer as the system model operates on synchronising communicating channels. The target interconnection models and target bit rates of these two protocols are reflected in the specification of the mechanisms. In the OS link model, link usage was highly variable on application, but most would display low workloads, which would result in low amounts of bi-directional traffic. Consequently, the OS protocol used one flow control unit per data unit. In implementation, the flow control token (FCT) was sent soon after the receipt of the start of the incoming data token, which allowed back-to-back data transfers where required. This is depicted in the left-hand diagram of Figure 2-4, which is based on the observation of transmission on circuits implemented in programmable logic, which returned a flow control token after the receipt of 3 bits of a data token.

The specifications of the DS protocol differed slightly from the OS protocol, as the system model used a many-to-one logical to physical channel mapping and was defined to work over distinctively larger distances. The limitation of the earlier protocol was evident

with the high use of software routing algorithms in the first generation transputer systems. With this change in system model came more demands on the communication system. The crux of the demands was the increase of traffic on each link, which would increase the probability of bi-directional traffic and would demand a high data rate to share between all communicating channels. As these systems also used the data path for control signalling, the bi-directional transfers would effectively reduce the available unidirectional data bandwidth in proportion to the data to control signalling ratio. Hence, the DS protocol increased the number of tokens that could be transmitted under the credit of a single flow control token. This reduced flow control signalling, which resulted to an improvement in the bi-directional transfer rates. The term 'flow group' is used to refer to the variable number of tokens within credit-based flow control protocols. The DS protocol defined eight data tokens per flow group, which provides adequate time for the return of the flow control token at the high data rates and larger transmission distances.

As part of the task of the data link layer to supply a low-level data connection, regulation must be included to ensure data is not lost. In point-to-point systems this is provided by a flow control mechanism, which ensures that data is only moved when the receiving party is ready. Section 2.2.2.2 highlighted two main mechanisms that are used in point-to-point systems. All the systems under review use one of these techniques.

The NTR08 implemented a credit based flow control mechanism, which conformed to the OS protocol mechanism of one data token per flow group, which helped to minimise buffering requirements. Both the STC104 and the NTR-M04 also used a credit based mechanism, but with eight tokens per flow group. Larger flow groups provide more tolerance for transmission length vs. speed conditions, which ensure bandwidth is not lost. The DS protocol of the STC104 was designed to operate at up to 200 MB/s and up to transmission distances of 500 metres. Although, larger flow groups minimise bandwidth loss on bi-directional transfer on bi-directional links, the larger flow groups demand a proportionally larger amount of buffering. It is reported that a single path across the STC104 contained seventy tokens of buffering [85], where the equivalent path in the NTR08 contained only three tokens of buffering [1]. The NTR-M04 used twenty-two tokens of buffering; this was the limiting factor of the number of ports on the device [2].

Both Myrinet specifications utilised the permission based flow control mechanism. The first generation and second generation LAN specifications used the data path to pass flow control information with STOP and GO tokens, where the SAN specification used an

extra level-sensitive signal line per direction. Permission based flow control requires certain buffering requirements, which are tied into the maximum transmission distance and speeds. The specification of the first generation Myrinet inferred a receiver buffering capacity of fifty-nine, nine-bit tokens. From Figure 2-2 and Figure 2-3 this is composed of clause 1 and 2 requiring twenty-three tokens, and clause 3 requiring sixteen with a maximum cable length of twenty-five metres at a data rate of 80 MB/s. The Myrinet second-generation specification did not specify the buffering requirements. However, it did indicate that the clause 1 should be calculated to a formula similar to that described in section 2.2.2.2, and clause 2 should be twice the amount of that, which is required to prevent data starvation. This amount of buffering requested for clause 2 seems excessive, as a figure similar to clause 1 should be adequate. The second specification does not dictate the size required for clause 3.

The parallel interface of the Reliable Router transmitted a data unit in four frames, on which the flow control operated. The flow control mechanism was complicated, as aspects of the fault tolerance was integrated into its operation. Each receiver possessed five, sixteen deep, seventy-five bits wide buffers. Five buffers were used as the Reliable Router implemented five virtual channels per physical link. The basic principle of the mechanism operated on a permission to release buffering, as each allocated buffer had to be maintained until the data had been successfully transferred ahead. This technique ensured that retransmission of data could take place in event of failure. This fault tolerant mechanism called the Unique Token Protocol, will be briefly discussed in the subsequent section.

As the flow control mechanism for the NTR08, NTR-M04, STC104 and Myrinet utilised data bandwidth for control signalling, it is possible to analyse the overheads of their operation (the complex nature of the Reliable Router mechanism makes a similar analysis impossible given the available information). It is easy to define the percentage of bandwidth used by flow control in the credit-based systems, as flow control credits must be sent.

The OS protocol defined the transfer of a two-bit flow control token for each eleven bit data token. Thus, in ideal operation of a bi-directional link, control signalling consumes $2/13^{ths}$ (15.4%) of available bandwidth, of which only $8/13^{ths}$ (61.5%) is data. Practical implementations have shown that this figure is very optimistic, in reality the utilisation of the bandwidth can drop to $8/17^{ths}$ (47.1%) [6]. This figure is highly

dependent on implementation, but the granularity of the flow control mechanism does play some part to this inefficiency.

The NTR-M04 uses an eleven bit flow control token, but it was valid for eight, eleven bit data tokens. Thus, $1/9^{ths}$ (11.1%) of the bi-directional bandwidth, which results in the $64/99^{ths}$ (64.6%) used for data. The larger flow group allows better utilisation of physical bandwidth, so the practical figures do closely match the theoretical ones. The difference between the OS protocol and NTR-M04 protocol is low due to the size of the control token; a price paid for control signalling flexibility.

The DS protocol shows a dramatic improvement on both these systems, as the data token is only ten bits in length and the control token is only four bits. Thus, the ideal operation of the flow control would utilise $1/21^{th}$ (4.8%), which results in $16/21^{st}$ (76.2%) utilisation for data. Again, the larger flow group supplies better utilisation of the bandwidth, which maintains these figures in practical implementations.

Permission based system are more difficult to analyse as flow control is only used when traffic conditions demand it. Thus, the best case would be 0% usage of the bandwidth. When the mechanism must operate, it is clause 3, which dictates the behaviour of the flow control mechanism. The first generation of Myrinet defined a value of sixteen to clause 3, which, the specification stated, would make a worse case bandwidth utilisation figure of about 6%. Independent tests carried out earlier in this project showed an average of an 8% improvement of bandwidth utilisation between the NTR-M04 protocol and an equivalent permission based implementation under extreme workload conditions (see section 6.1).

### 3.3.3 Fault Tolerance

The OS protocol of the first-generation transputers did not integrate any form of error detection within the token format of the data stream. The only possible method of providing confidence of the data link was to check the data stream against token format. The NTR08 also omitted link error detection as the reliability of the link was said to be extremely high, which would make any implementation wasteful. The same premise was accepted in the design of the NTR-M04, which saw no implementation of parity or frame checking. In contrast, the STC104 implemented two mechanisms for the detection of disconnection errors and parity errors. As the DS link protocol operated continuously to maintain synchronisation by transmitting data or IDLE tokens, this was used to detect link

disconnection. In implementation, a lack of activity on the incoming transmission line for 850 ns is deemed as a disconnection error. This figure equates to a maximum figure of 170 bits or 17 data tokens at 200 Mb/s. Additionally, as highlighted earlier, the DS protocol utilised odd parity checking, which also aided holistically in synchronisation checking. For the DS link to safely operate using these error detection mechanisms, a controlling state machine was implemented. The mechanism operated with nine states, which covered link initialisation procedures and recovery procedures on error detection. Figure 3-5 shows the controlling state machine for the standardised DS protocol link, which was taken from the IEEE Std. 1355-1995 [81]. The figure shows the transitions from the state of a reset link (Ready), to an active link (Run). It also shows the steps taken to recover from a parity or disconnection error.

A notable characteristic of the DS protocol is the dependence of its operation on link integrity. In many aspects, this can be seen as advantageous, as differing levels can provide confidence in the link operation from physical to network layers. This confidence is vital to a fault tolerant system. However, reference material [86], which appeared after the development of the STC104, presented additional precautions to supply further fault protection. This indicates that there can be problems if one device is expected to supply a *catch all* solution, and as such, systems should be devised holistically.

Unfortunately, the DS controlling state machine was shown to be flawed by a standard that was derived from the IEEE 1355, which is currently under development for use in space applications [87]. These flaws centred round the initialisation procedure, which could result in the state machine locking, thus disabling the link. The derived specification reviewed the IEEE 1355 specification and simplified the mechanism to a six state system, which solved the problems contained in the system.

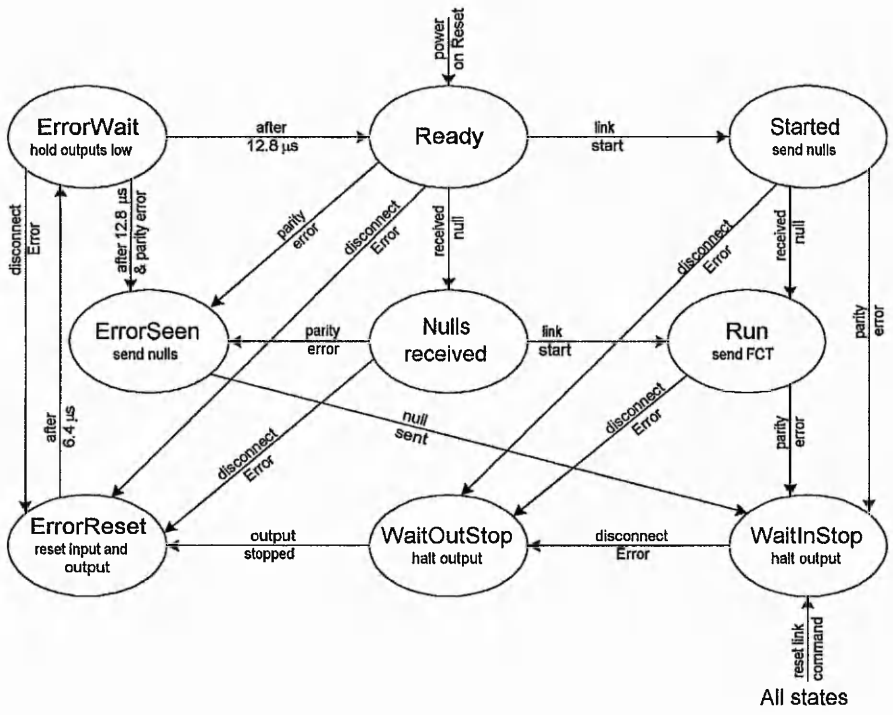



**Figure 3-5 : Link state machine for the IEEE Std. 1355-1995**

Similar to the STC104, the first generation of the Myrinet devices used a time-out feature to detect disconnection errors. In contrast, the specification indicated that the flow control tokens or GAP tokens could be used in the absence of data. The choice between the two would be dependent on the network layer status; that is if a packet was active across the link. This will be clarified further in the next section. Also similar to the STC104, the time-out period for disconnection was sixteen tokens, or 0.2 µs. The Myrinet second generation operated slightly differently. Firstly, an optional token was specified for transmission in the absence of data, namely, the BEAT token. Additionally the GAP for SAN and LAN devices and the flow control tokens for SAN devices could also be transmitted in the absence of data. The specification stated that if the BEAT token was to be used it had to be transmitted approximately every 10 µs, and any time-out period was left to the discretion of the implementation. This was a distinct alteration from the earlier version, as due to the increased data rate, the 10 µs interval for the SAN was equivalent to either 1600 or 3200 tokens, compared to either 800 or 1600 for the LAN.

The Reliable Routers reliance on parity may seem excessive given the comments made in section 2.2.2.3, but the bit rates and synchronisation techniques of this system were pushing technology limits at the time of development. Another technique utilised by

the Reliable Router was the Unique Token Protocol (UTP). This technique operated in co-operation with the flow control system, which utilised a data acknowledgement structure at the data link level. This removed the need for end-to-end acknowledgements, which lowered buffering requirements at the source for retransmission and removed the need of message replication checking at the destination node. As the system confirmed every transfer between switching nodes, the resultant mechanism was distributed and the resources, therefore scaled linearly with the network. In practice, the mechanism injected packets in to the network, and the flow control system ensured that each flow group of the packet had been transferred completely between neighbouring nodes before releasing the resources. This resulted in two copies of each flow group in the network. A special character called the 'token' was injected after the packet, which helped ensure the integrity of the mechanism. Each 'token' was unique, and its arrival at the destination, guaranteed that the message had been completely delivered. If a packet arrived at a node with a failed output, network layer fault tolerance would route round the error. However, if a failure occurred mid-message, each part of the severed packet could continue to its destination, with a modified 'token' to highlight the error. This required that each router needed to store the routing header for the duration a packet was active. The arrival of the severed parts of the packet at the destination with the modified 'token' and other information known about the packet, the message could be reconstructed. The complex nature of this flow control based fault tolerance mechanism minimises the overheads of fault recovery at the cost of performance. The work presented did not show the probability of failure [79], but the high levels of parity and the UTP mechanism indicate that it was significantly high to warrant mechanisms to improve the efficiency of the recovery mechanism. This operates on the same premise as the deadlock detection and recovery mechanisms reviewed in section 2.2.3.7, where the overhead of the detection and recovery mechanisms is less than the penalty imposed by the avoidance routing restrictions.

## 3.4 Network Layer

Within point-to-point switched networks, features such as routing, switching and packet-level fault tolerance features are suited best to the network layer. These aspects of the devices and systems under review are now presented.

### 3.4.1 Switched Architecture

The majority of devices under review were designed to support irregular topologies to provide flexibility in application implementation. The NTR08 and STC104 were developed for use in transputer networks, which saw most systems implemented for distributed processing or control. The NTR-M04 was based on the ideals of the NTR08, and therefore followed suit. Myrinet was also designed for irregular networks, in contrast to the systems like the Reliable Router. The Myrinet system originated as a LAN replacement for high-performance networks of workstations, or clusters. Point-to-point LAN replacements, like Myrinet [88] or the earlier Autonet [40], selected the flexibility of irregular networks as they suited the dynamics of the application. Whereas systems like the Reliable Router were designed to be implemented in a rack system, to which processing problems would be sent. The 4+1 port architecture was a common configuration for systems like the Reliable Router, and as such many were connected in mesh topologies as depicted in Figure 3-6. By fixing the topology, system optimisations could be made by utilising routing and fault tolerance.

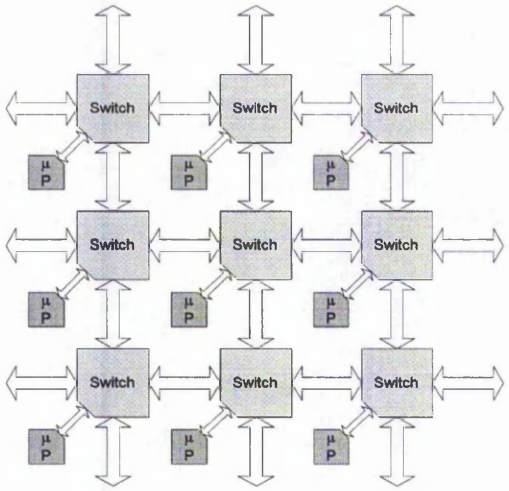



**Figure 3-6 : Prevalent mesh topology used with 4+1 port router-switches for high-performance parallel processing**

### 3.4.2 Connection Methodology

Section 2.2.3.2 highlighted that the majority of early devices for switched networks supported unicast messaging only. The distributed nature of the architecture provided greater message concurrency than shared mediums, but did not supply simple methods to support multicast messaging. The point-to-point CSP methodology of the transputers was based on one-to-one connection model. This coupled with the added complexity and the

software based multicast algorithms, led to hardware support of multicast being omitted from the NTR08 and STC104 specifications. Similarly, the Myrinet systems omitted hardware multicast support, but independent research was undertaken to supply it in firmware within the network interface [37, 89]. The fixed topology of the Reliable Router allowed for more flexible routing algorithms, but prohibited the use of hardware support for multicasting. The NTR-M04 was designed from the outset to support multicast switching, by using multi-destinational routing headers similar to other concurrent research [90]. Stunkel [41] highlighted two connection methodologies for use with multi-destinational headers, namely synchronous and asynchronous:

- Synchronous systems require that the packet be forwarded to all required outputs simultaneously, which can degrade network performance if one or more destination links stall.
- Asynchronous systems allow for separate delivery, but this can increase complexity and increase local buffering requirements while restricting maximum packet sizes.

Although switch-based solutions have been shown to improve multicast messaging, steps must be taken to ensure deadlock allocation is impossible. Deadlock allocation is when two or more messages hold resources that others need to proceed with the connection, as shown in Figure 3-7. As asynchronous systems may break down a multicast connection into multiple cycles, which do not require all outputs to be available, thus allocation deadlock is prevented. Only synchronous systems must implement further preventive methods. As the NTR-M04 supported unlimited packet sizes, it implemented a synchronous multicast mechanism. To prevent allocation deadlock a compromise was selected to only allow one multicast message to be serviced at one time.

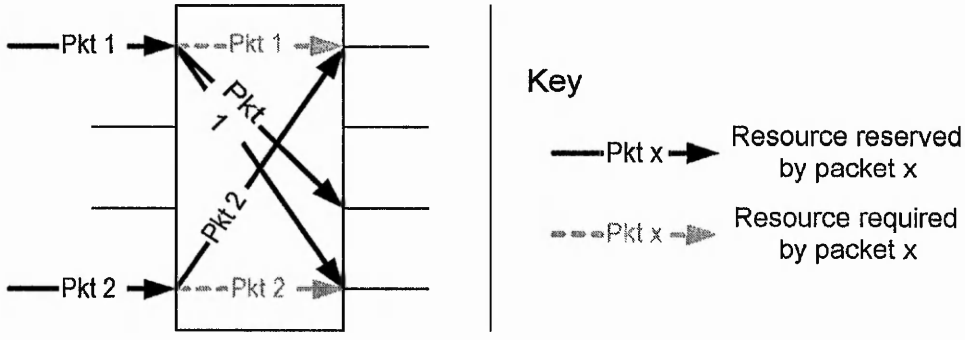



**Figure 3-7 : Depiction of an allocation deadlock scenario**

Wormhole routing has been the most prevalent switching technique in distributed and embedded processing due to the low switching latencies and low buffering requirements. The demand has increased for larger transfer rates, which has modified switching techniques and flow control mechanism that have required larger amounts of buffering. All of the devices under review claimed to implement a wormhole routing methodology, but each implement more buffering than the technique dictates. More recently, research has seen real-time and multimedia considerations factoring into the router-switch design. Interest in wormhole techniques has waned in preference to switching methodologies that can provide a more deterministic operation [43, 91].

The Reliable Router is the only device under review that implemented virtual channels. The 4+1 port architecture of the Reliable Router could easily implement the control for the twenty-five virtual channels, which is still smaller than the thirty-two channels of the STC104. Virtual channels were also popular for deadlock avoidance mechanisms and adaptive routing algorithms of the fixed topology systems. The additional complexity ruled out the use of virtual channels in the NTR08 and NTR-M04.

### 3.4.3 Packet Format

The use of the OS link within a pure transputer application was designed to be an exclusive physical link to software channel mapping. These strict programming ideals of the transputers meant that data, which was transmitted via an OS link, conformed to rigid formatting. For the NTR08 to operate efficiently, it was necessary to create a hybrid of the OS link protocol that would allow for global routing through a scaleable, switched network. As the OS link protocol was based on byte-sized tokens, the hybrid router-switch protocol used the tokens as atomic units. The operation and meaning of a data token depended on the state of the receiving unit when a token arrived.

In normal operation, the device started in the idle state, thus the first token is assumed to be a header token. After the routing header tokens, a single token was used to define the size of the message payload, which was then followed by the payload. Figure 3-8 shows the packet format that was defined. The figure shows that the most significant bit was used to delimit the end of the routing headers and the position of the size byte. This format limited the maximum packet size to 256 bytes. While this limitation was suitable for many control applications, which normally use small packets of information, large data streams could become inefficient.

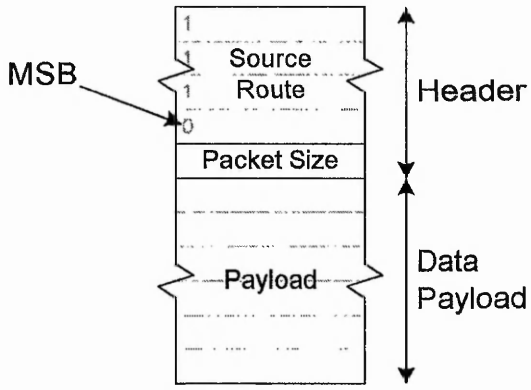



**Figure 3-8 : Packet structure used in the hybrid protocol of the NTR08**

The STC104 operated without regulation on maximum packet or message size. As presented earlier, the DS protocol possessed end of packet, and end of message control tokens. This allowed the STC104 to operate with a packet similar to the generic structure presented in section 2.2.3.3: one or more routing headers followed by a variable sized payload and terminated by a predefined tail, namely, either of the EOM or EOP control tokens. When INMOS created the T9000, they set a packet size limitation to 32 bytes [92], and each packet was acknowledge by the receiving device. This was to supply better resource sharing, minimise buffering requirements and provide data acknowledgement for synchronised process communication. However, this additionally showed a clear aim of targeting control system applications, as the packet size limitation would degrade applications with greater volumes of data signalling. The T9000 enforced stricter transport and network layer rules than the DS protocol. Upon standardisation of the DS protocol, some changes were made to close loopholes within the DS protocol that were covered by the higher layers of the T9000 communication model. This included the reclassification of the packet termination control tokens. The T9000 used the tokens as presented in section 3.3.1. The strict operation of communication was statically defined, where message size and number of packets were known at run time. Thus, any deviation in message format would highlight an error in the transmission medium. The STC104 used the EOM token to terminate messages that were active when the data link layer error detection mechanisms were triggered. Thus network errors were detected and recovery procedures could be taken within the T9000. The IEEE 1355-1995 standard reclassified the end of packet token to the 'normal end of packet' token (EOP_1), and the end of message token to the 'exceptional end of packet' token (EOP_2). Thus, the new network model used multiple

packets to form a message, but the last packet was not specially marked, thus leaving the pragmatics of the message delivery to the high levels of the communication system. This left the operational implementation of the STC104 valid within the standardised protocol. Although this has provided a workable solution, it is not a normal procedure to make modifications to maintain hardware compatibility, as it often can result in less efficient system implementations. Inefficiencies resulting from hardware backward compatibility were evident in the NTR08 protocol, as mentioned previously.

The packet format used by the Reliable Router also conforms to the generic packet format of routing header, payload and tail, as did the DS protocol and the NTR-M04. The unrestricted packet size is the primary motivation for this format. The NTR-M04 supplied three packet termination tokens in contrast the two defined in the IEEE 1355-1995. Following the mechanism of the DS protocol, packet and message formats could be supported with the EOP and EOM tokens. The third termination token, the end of block token, was included to operate at a layer above the involvement to the router-switch; that is, the router-switch was oblivious to this token. This meant that a unicast or multicast connection could be formed that could remain allocated, while blocks of data could be delimited by the communicating entities. The connection that was formed was still temporary, as it still could be relinquished by either the EOP token, or EOM token. This technique effectively altered the connection methodology of the switched network, and it was designed to provide a guaranteed constant bandwidth between any given points in the network. At the time of the development of the NTR-M04 much research was concentrating on quality of service that had stemmed from the increase popularity of multimedia applications. Quality of service permeated many levels of computing and it was a simple premise that a connection must be able to guarantee a predefined level of bandwidth [93]. The implementation of the end of block token in the NTR-M04 protocol was a simple solution for a useful service guarantee. Additionally, as it was not restricted to unicast messaging, the scope of its use in parallel applications; for example, for synchronisation techniques, made the mechanism attractive. Unfortunately, if the feature was used, it could penalise the operation of the rest of the network if special considerations in network topology were not made. By semi-permanently allocating resources in a dynamic network, the advantages of the packet switched network were negated.

Both specifications of Myrinet used a packet format close to the generic definition as defined in section 2.2.3.3, but the packet termination token was not specified as part of

the packet, although the GAP token had to be transmitted to delimit packets. As described earlier, to maintain synchronisation, a constant stream of characters was transmitted, which included the possibility of multiple GAP tokens between packets. Figure 3-9 shows the packet format for both Myrinet specifications. Similar to the NTR08, the most significant bit was used as a delimiter of the header. The GAP tokens are shown in grey to signify that they may appear more than once outside of the packet format. The only difference that separates the first and second-generation specifications is the inclusion of the packet type field, which must conform to the specification and the packet's function. Some examples of packet types are route mapping packets, data packets or special traffic types defined for application use [94].

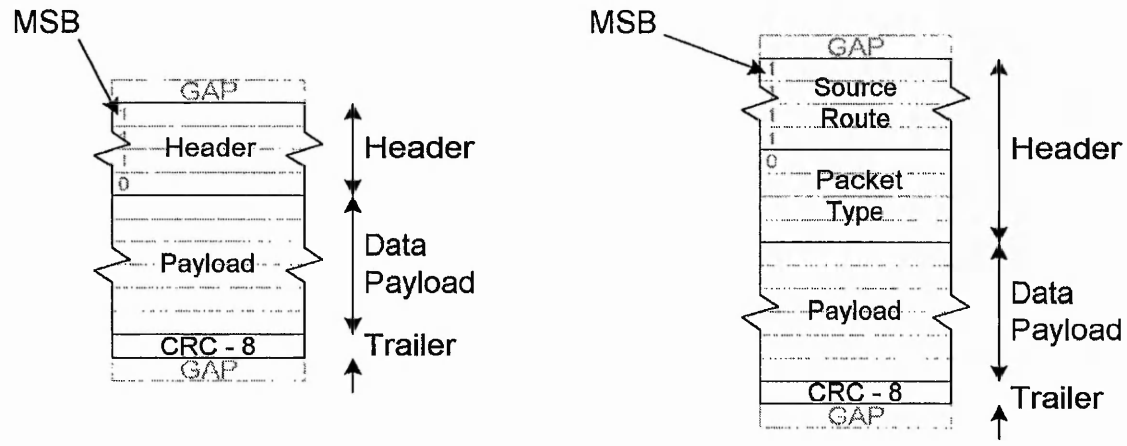



**Figure 3-9 : Packet format for the first generation (left) and second generation (right) of Myrinet**

### 3.4.4 Adaptive routing techniques

The NTR08, STC104, and NTR-M04 were all designed for irregular networks, which limited the choices of adaptive techniques. Thus, each device employed group adaptive. The STC104 also implemented a two stage routing procedure that was designed to distribute the network traffic. Universal routing, as INMOS called it, could include an additional header, which was chosen at random from a predefined range. The header would route the message to an intermediary location in the network, where the random header was stripped. The message could then proceed to its destination. This technique improved network throughput as the load was distributed better. However, the effect of including a random header negated any avoidance technique for deadlock. An expensive solution was suggested [95], which solved the deadlock problem by providing completely separate routing resources for the first stage of routing.

In contrast to the group adaptive algorithm, Myrinet, which also was designed for irregular networks, used firmware self-mapping algorithms to plot paths through the network. While this technique did not react dynamically to the network traffic load, it could update the routing tables as the topology changed through faults or user alterations. The Reliable Router, through its restriction of topologies, implemented three routing algorithms to supply flexibility. In conjunction with virtual channels, dimension-ordered routing, minimally adaptive and partially adaptive routing algorithms were used as part of the routing decision. The minimally adaptive algorithm would take any port that would advance the message, but was oblivious to deadlock concerns. The partially adaptive algorithm was based on the turn model suggested by Glass and Ni [54]. The operation of this algorithm will be covered in the next section.

### 3.4.5 Routing Decisions

The main purpose of the NTR08 was to provide efficient routing utilising the selected gate array technology. The gate array design demanded that gate count and operational speed were primary concerns. Keeping to the ideal of minimal hardware resource requirements, the NTR08 was implemented using physical address decoding. This technique forced automatic header stripping of the source routing information. As the target networks were normally small to medium scale, the extra header byte per router-switch transit was deemed to be an acceptable overhead.

The STC104 was targeted at slightly larger networks, with the T9000 as the processing device. This led to the use of interval routing that could use one or two bytes as a single routing header. This provided up to 65,536 individual possible destinations with a single routing header, plus possible expansion of this with optional header stripping.

The NTR-M04 was realised in a PLD, which possessed embedded memory blocks, which made it feasible to implement logical addressing. However, the routing algorithm automatically stripped the routing byte for unicast messages, thus the device operated with the same overhead as a physically addressed device. Multicast headers, however, operated without header stripping. As the NTR-M04 was a prototype device, only sixteen entries in the look-up table were included. This allowed the proof of concept of the techniques employed.

Myrinet utilised firmware within the network interface to map the network to maintain a valid list of all possible routes. For this to be possible, Myrinet utilised relative

addressing, which is an absolute addressing mode. Relative addressing required a header per switch. The interface would inject a number of 'mapping packets' that possessed different header permutations in to the network hoping to reach another interface. By using relative addressing, the interface could also include the return route with the mapping packet so the receiving node would know how to find the source. By mapping the network, multiple paths could be constructed that used the network in the most effective way. Some work has been undertaken to optimise the mapping algorithms of networks using relative addressing [96].

The Reliable Router utilised a graph routing scheme, as this could supply adaptive routing features over the fixed mesh network. In conjunction with virtual channels, the Reliable Router supplied three disjoint routing techniques, which were a collaboration of previous methods of various earlier devices. The three algorithms, described below, were used to maximise resource utilisation through adaptive routing, but also to supply a method to circumnavigate faults that could have occurred within the network.

In the first algorithm, the Reliable Router attempted to choose a minimal adaptive route for the packet. A minimal adaptive route in this type of network is one that selects any direction that will bring the message closer to its destination. The second algorithm occurred if no such route was available: a specially selected channel was requested that conformed to a dimensional ordered routing algorithm as described in section 2.2.3.7. If the selected dimension ordered routing channel was busy, the packet was stalled and the mechanism returned to check the adaptive routing options in the following routing cycle. The third algorithm was used when the dimension ordered selection was found to be faulty: the packet was then routed to a fault-handling channel. This third method used a partially adaptive algorithm, where any free output could be taken with the exception of the input; that is, a 180-degree turn. After the packet was forwarded to the next router-switch using this method, the packet possibly could be allowed to revert to use the previous method of route selection. The decision on this was dependent on guidelines based on the turn model theory, as described in section 2.2.3.7. Two virtual channels were used for the dimension ordered algorithm, two for the minimally adaptive algorithm, and one for the fault-handling algorithm. This ensured that all the algorithms were logically separate, and the system could be proved as deadlock free.

### 3.4.6 Fault tolerance

The NTR08 did not contain any automatic integrated fault tolerance in the network layer protocol, but fault monitoring and intervention was supplied by a control port. This feature was implemented by an additional block that could be accessed by an additional serial connection, separate from the data path, which conformed to the OS link protocol. The control port allowed a network controller to configure, monitor and control the operation of the device. As the control could be used to monitor and manipulate connections on the device, a basic fault tolerant system was possible. A network implementation could provide a monitoring system that detected stalled messages, which could then use a holistic approach to resolve the problem. The negative aspect of this was there was no simple scaleable solution for the monitoring and recovery system. Firstly, the control port connection was not designed to work in collaboration with other router-switches. Therefore, each router-switch would require a dedicated connection to the controller, as it did not utilise a packet format for the interface. A simple wired-OR and decode and multiplexer system could be used to reduce these requirements. However, as a network scaled, the implementation overheads and mechanism complexity also would increase, which prevented such a solution for a large numbers of switches. Additionally, as a central controller must monitor all devices in the network, as the number of switches increased the reaction time for detection, and intervention would scale proportionally. It must be conceded that the NTR08 was targeted towards small to medium size networks, and therefore this solution was adequate for the specification.

The C104 contained some fault tolerant features, which could be configured to operate automatically. This was regulated by a configuration setting called 'localise error'. By enabling this feature, the link was reset and the active packets were truncated if a data link error was reported. This action allowed corrupted packet to be removed from the network, without affecting the remaining network operation. A configuration setting, which is called 'discard if inactive', is associated with each output, to ensure that packets without active destinations will not remain in the network. Thus, if a packet is destined for an inactive output and the 'discard if inactive' flag is set, the packet is deleted. While this ensures that the network will never deadlock due to undeliverable packets, documentation implies this procedure is irrespective of any grouping configuration. This negates the extra route availability that is supplied by group adaptive routing. Without either of these configuration settings, the device would have to wait until the error was cleared by a

command from the device control port. The control port operated on a similar premise as implemented in the NTR08, but the interface was a daisy-chained connection, which would allow the devices in the network to be connected by a single serial link. The daisy-chained link operated on its own fault tolerant protocol, which allowed each device to be addressed separately for configuration or management procedures [97].

The forward reset token (FRES) of the NTR-M04 was designed to supply a basic recovery mechanism within the switched network. Its function was identical to that of the token suggested for use in the Myrinet system. The FRES was designed for situations of link failure, and was specified to clear the path it traversed of resources. It was sent after a predefined fault detection mechanism was triggered. As the FRES was specified to be transparent to the data link layer, it was not restricted by the flow control mechanism. On recognition of a FRES at the receiver of link, the contents of the buffering were cleared and it was transferred across the crossbar of the router-switch over any connections that were formed with that receiver. By forwarding the token to the relevant transmitters, all of the connections would be released and the FRES would continue to any other connected device. The FRES would continue in this manner until it exited the network or reached a router-switch that was not connected via its arriving link. The effect of the FRES could remove many messages from the network, making it a complicated and a highly regressive recovery mechanism, while the extra resources for its implementation made it an expensive addition.

The Myrinet system specified additional control tokens for use in the fault tolerance mechanism of the first specification. These tokens were never implemented in any Myrinet devices, possibly due to complexity of their intended operation. The extra tokens were: forward reset (FRES), backward reset (BRES), over run alarm (ORUN), probe (PRB) and probe reply (REPL). The FRES was just described as part of the NTR-M04. The BRES was similar to the FRES, but its specified operation was vague. The BRES function would have been initiated in a similar way to the to the FRES. The difference being that when BRES symbol was received, it was reported on its control interface, and the interface control circuit would have initiated a FRES. The overrun alarm symbol was defined to flag a breakdown of the flow control mechanism, from which the STOP/GO may suffer due to loss of flow control data or transmission delay exceeding the predefined limits. The probe and reply symbols would have supplied the functionality for the network status monitoring and control features over the system architecture.

The fault tolerance aspects of the Reliable Router have already been addressed within the description of the routing decisions of section 3.4.5, which involved fault tolerant routing algorithms.

### 3.4.7 Deadlock

The NTR08, STC104 and NTR-M04 used source routing, and therefore had to employ deadlock avoidance techniques to ensure problem free operation, which trade a little network performance. As presented in section 3.4.5, the Reliable Router used three routing algorithms, which when used in the overall routing strategy, also provided a deadlock avoidance mechanism.

Similar to these other systems, the Myrinet system also implemented an avoidance mechanism, namely the up/down routing algorithm, which is recalculated as the topology changes. However, in addition to this technique, Myrinet also operated a form of detection and recovery mechanism. The second generation of the specification stated that a time-out mechanism may be used, but the time-out is in the order of one second in duration. This displayed characteristics of a backup mechanism, for the cases where the avoidance technique failed for some reason.

# 4 Design Discussion

The earlier research supplied a number of development aims, which were used as a guide at the start of this research. A primary aim was to ensure the same routing methodology was maintained or improved. Earlier devices have been shown to supply effective solutions for embedded parallel and distributed systems, and this had to be maintained. For such an effective system, communication switching and protocol overheads must be as low as possible. The earlier research showed that bi-directional data transfers were a weak aspect of the physical link. To improve data bandwidth utilisation at the data link layer, the protocol had to be reworked. Fault tolerance was another area, towards which the research was targeted. A review of techniques and concepts of fault tolerance were required, targeting the types of faults, the main areas that help maintain operability in the presence of faults, and fault recovery techniques.

This chapter will describe the features, resulting from the review, which were chosen for implementation in a new router-switch device. This is then followed by a high-level functional description of this device, the NTR-FTM08 emphasising features targeted for implementation.

## 4.1 Lessons Learnt from Earlier Research

Considering the physical link, an increase in bandwidth over the previous implementations was desired but within a reasonable implementation cost. Parallel links, while providing greater data rates, as with Myrinet, can require expensive multi-core cables for meeting the low skew requirements, in addition to increased device pin count. These concerns were tackled by the NTR-M04 by increasing the granularity of the parallelism of the links. However, the logic implementation costs were higher than predicted. Although the implementation consequences of a single serial line result in lower bandwidth figures, transmission distances and implementation costs make it the most cost-effective solution. It has shown that the over-sampling technique was capable of double the bandwidth figures of previous implementations in the technology targeted for this work. Additionally, over-sampling has been shown to be a resilient recovery technique even at these higher signalling speeds and large distances, with the latest differential drivers [6]. The data-strobe technique used by the IEEE Std. 1355-1995 initially appears as an attractive system, but the requirements of skew tolerance and wire count could increase implementation costs for similar reasons to the parallel implementations.

The review of the control signalling of the previous protocols showed a distinct improvement of the DS protocol over the earlier over-sampling protocols, which was mainly due to the token formats. Reducing the data token to ten bits and the control characters to four bits resulted in a theoretical improvement of 11.6% over the protocol used by the NTR-M04 and a 14.7% over the NTR08 packet data rate. Further modifying token format to improve the control bit / data bit ratio would improve data signalling utilisation of the link bandwidth. Unfortunately, if an over-sampling technique was utilised, increasing the number of bits per token would have a detrimental effect on the skew tolerance, which would reduce the signalling speed and transmission distance capabilities of the system. The current eleven-bit tokens of the NTR08 and NTR-M04 allow the sampling circuits to recover at each token; with three times over-sampling this results in the skew tolerance being a third of a bit over the token. Thus, if the token was extended this $1/3^{rd}$ of a bit tolerance would be stretched over the new greater length of token.

The data line based flow control mechanism also was shown to degrade the data rate of the signal on the co-operative full duplex link of the earlier systems. In contrast, the SAN specification of Myrinet used a separate signal line to overcome this, which resulted in a theoretical maximum data bandwidth utilisation when flow control was not in effect. While the earlier implementation of Myrinet, which also used the data line for flow control signalling, claimed a maximum worse case loss of only 6% of data bandwidth due to the permission based flow control. Although, the transition from credit-based control to permission-based control produces problems with buffer overrun and confidence in link validity, the figures show the adoption of such a technique would match or reduce control signalling relative to message patterns.

Aiming to maintain or improve on the earlier routing methodology limits the design choices to be made at the network layer. A major network feature of earlier work was the support of irregular networks, which indicated the use of some form of source routing. Additionally, the NTR-M04 supplied hardware support for multicast messaging; thus, this feature also had to be maintained. To support multicasting, the review of addressing modes indicated that either physical or logical addressing had to be used. A physical addressing based implementation would limit the number of ports per switch, which would be detrimental to the scaling of the network, plus it would possess the negative aspects of the routing overheads associated with it. Although the target PLD technology currently

limits the size of the device, by choosing a physical addressing mode for future implementations, limitations would also be imposed. Logical addressing would remove this limitation, but it possesses other implementation problems. Analysis of the NTR-M04 implementation, which also used logical addressing, showed that the main bottleneck in connection latency was the decoding, queuing and allocation mechanisms, due to the limited resources in the target technology. Additionally, the NTR-M04 allowed only one multicast connection-request to be serviced at once, which prevented allocation deadlock. An ideal solution would be to have a decoding unit, for each input, which could then be queued and allocated in parallel, with the queuing technique ensuring freedom of allocation deadlock. This would form a natural pipeline for the servicing of messages. A logical addressed system would demand a massive amount of resources to implement this, whereas a physical addressing scheme would require a relatively small amount. The target architecture for this work, resulting in the NTR-FTM08 design, possesses a large amount of embedded memory. This could make a parallel implementation of logical addressing possible, but the whole design must be considered, as the router-switch also requires significant levels of data buffering. The amount of buffering that is required, depends on the flow control mechanism and physical link requirements.

The most prevalent form of connection methodology in earlier systems was a buffered form of wormhole routing, best suited to small, embedded control and distributed systems as the switching latencies are minimised. The introduction of multimedia and other real-time data streams has encouraged the investigation of other methodologies. The review showed that the buffered wormhole methodology is still the most appropriate technique for the target applications.

The choices for adaptivity within source routed irregular networks are limited. The use of group adaptive routing in the past has shown that this technique can be used to implement a flexible system, which can supply varying amounts of bandwidth under control of the application implementation. The group-adaptive-routing implementation of the NTR08 operated at the connection-request queuing stage, as did the NTR-M04. As the act of queuing in the earlier implementations effectively fixed the output, this reduced the effectiveness of the adaptivity as the status of the output could change over the time that the packet was waiting. By delaying the grouping decision later, more adaptiveness may be maintained.

The monitoring and intervention features of the control port, as implemented in the NTR08 and STC104, supplied all the features that would be required to support fault tolerance in software. The STC104 also possessed some ability to react to faults autonomously with configuration settings. While a monitoring and intervention technique can supply all the functionality required, the interfacing control possesses the problems. Firstly, the controlling system must be centralised to maintain a unified and co-operative mechanism, which if it fails removes all system fault tolerance, termed a 'single point failure'. Additionally, a centralised controller may not scale efficiently. For example, as highlighted earlier, the physical interface of the NTR08 does not lend well to a centralised control system. While the implementation is improved in the STC104, the operation the system is still prone to single point failure. The implementation of the NTR-M04 further improved on this by using the data network to supply more flexibility to the control network. Therefore, the reliability of the data network can also be attributed to the control network. However, if the network operates on a monitoring and intervention technique, the use of the data network may not be feasible, through either critical network failure or loss of network performance through excessive network-control related communications. Despite this, the advantages of using the data network show the basis for an improved system.

In an improved connection model, the centralised monitoring and intervention mechanism would demand extra resources within the system. The controlling mechanism would increase in size and complexity as the network scaled, which would reduce reaction time to faults. Additionally, the system would have to be specifically design for each system, and would not be transferable between one system and another. Although the operation of the unique token protocol of the Reliable Router was complex, the distributed nature of the mechanism allowed the system to scale linearly with the network, without additional software overheads. The autonomous features of the STC104 also introduced a rapid response to faults, but recovery was limited to packet truncation or deletion. While this solution would demand extra functionality in the higher layers for message delivery verification, it also ensured that a message would not block the operation of the network and prevent critical failure.

All the devices under review utilised some form of avoidance technique as the primary mechanism against deadlock. This is the most popular technique for use with virtual cut-through or wormhole routing devices. It is only recently that other devices have

turned to recovery techniques [57, 70, 98], and returned to developing preventive [43] algorithms. Avoidance techniques are most prevalent in source routed based devices, for irregular networks, as the techniques require no implementation considerations in the router-switch architecture and therefore are the cheapest to implement. Unfortunately, avoidance techniques can be prone to errors, as corruption of the data stream, on which the mechanism operates, could circumnavigate the protection offered. This implies that a detection and recovery mechanism should be used as either a primary mechanism, or a last resort mechanism. The cheapest detection solution is a simple time out mechanism, although the time-out is normally very large to ensure heavy network traffic does not falsely trigger the recovery mechanism. Such a solution is suitable for a last resort mechanism, as suggested in the Myrinet system which implements an avoidance technique as the primary mechanism. Although in this case the deadlock recovery mechanism would be triggered infrequently, the long time-out, on which it is based, could cause critical application failure. A system that relies wholly on time for deadlock detection cannot maintain implementation flexibility and provide a workable solution. To maintain implementation flexibility and to remove false deadlock detection, the detection mechanism needs to propagate status information along the branches of the tree structure, as stated by Lopez [57]. Folkestad [62] suggests such a technique in his work on deadlock probabilities, which describes a system that propagates deadlock notification messages around the network containing the ID of the source router-switch. He states a notification message should be sent back towards the tail of the blocked message and along any blocked routes that form part of the dependency tree. If the source router-switch receives a deadlock notification message with its own ID then a cyclic route has been detected and a deadlock condition is flagged. There are two obvious problems with such a system. The first being cyclic non-deadlock conditions, as described by Pinkston [66]. Non-deadlock cycles can exist in systems where messages have many possible routes created by the functionality of the routing architecture. Examples of this are multiple paths from virtual channels or grouped physical channels. Cyclic paths would return the deadlock notification message to the originating node while an 'exit path' still exists, and therefore a false deadlock detection occurs. The second problem is based on the router-switch ID. As the notification message must carry the router-switch ID, the router-switch architecture must provide extra hardware resource to allow injection and extraction of small control messages, separate to the data path. In addition, the physical aspect of forcing each router-switch to identify itself ultimately limits the network size. If a deadlock condition occurs

due to the formation of a cyclic dependency, it is reasonable to use the cycle for detection as Folkestad suggested. However, to overcome the problems of Folkstad's solution, information needs to be passed both up and down the dependency tree.

## 4.2 Basic Router-switch Definition

### 4.2.1 Basic Router-switch Physical Layer Protocol Description

The over-sampling technique, which was used in the earlier NTR08 and NTR-M04 devices, was selected for the synchronisation technique for the physical link. The technique has been shown to operate reliably with the target application area, and recent research has shown that it possesses scope for higher signalling rates and greater transmission distances than has been previously implemented. The link will maintain the two wire, full duplex format as implemented in the NTR08 operating with TTL logic levels at board level. However, differential signalling was targeted for off-board connections up to a range of 100 metres. The link has been targeted to operate at above 30 Mb/s.

### 4.2.2 Basic Router-switch Data Link Layer Protocol Description

The physical layer specification of the NTR-FTM08 was similar to the NTR08 and NTR-M04 for data recovery. Table 4-1 lists all the tokens that were defined for use with the basic router-switch model. This specification has taken the eight bit control token format from the NTR-M04, but the bit ordering from the NTR08.

**Table 4-1 : Definition of the basic tokens for the NTR-FT08**

| Token Identifier | Abbreviation | Coding [Type, LSB ... MSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Permit transmission | XON | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Inhibit transmission | XOFF | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| End of message | EOM | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| End of packet | EOP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Data | DATA | 1 | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |

Note that the end of block token of the NTR-M04 protocol has been omitted. The value of this token was questionable, however, as its operation was transparent to the switched network, it could be included later. Two new types of control token are defined, namely data path visible and data path invisible. All tokens that are invisible to the data path are localised to a single point to point link. These tokens have the LSB set to a value of '0'. Visible tokens have the LSB set to a value of '1'. This simplifies control logic and provides an easy method to ensure backward compatibility for future specifications.

The data link layer of the NTR-FTM08 operates on a permission-based flow control mechanism that utilises the data path for control signalling, as discussed in section 2.2.2.2. The flow control tokens possess a higher priority than the data tokens, thus worse case transmission delay for a flow control would be a whole data token. To conform to the physical layer specification, the mechanism must be able to operate at a maximum range of 100 metres and at bit rates above 30 Mb/s. To supply additional leeway the operational parameters will be calculated for 110 metres and 50 Mb/s. Clauses 1 and 2 need a value of at least nine buffer entries, and on the recommendation of Myrinet, clause three should be at least twelve buffer entries.

The basic router-switch definition possesses no error detection or recovery mechanism at the data link layer.

### 4.2.3 Basic Router-switch Network Layer Protocol Description

As with the previous devices, the NTR-FTM08 is targeted for use in irregular and regular networks. This will include hardware support for unicast and multicast messaging based on a buffered wormhole routing algorithm. Routing adaptivity will be supplied by group adaptive routing, which will be configured via a configuration port that is accessible via the data path. The NTR-FTM08 will use the same message format as the NTR-M04, which specifies a message being divided by multiple packets. Individual packets are not limited in size by the specification.

To support multicast messaging, logical addressing is the most suitable technique. However, to supply enough entries for the entire address range would demand a high amount of resources. In addition to this, the majority of messages will not require multicasting abilities. For these reasons, multiple addressing modes are implemented. Interval addressing is implemented as it allows for efficient routing after configuration at a low hardware overhead. Physical addressing is also included so that a default routing mechanism is provided irrespective of configuration. Physical addressing performs unconditional header stripping, as required, but additionally, optional header stripping can be configured on an output basis for logical and interval addressing via the configuration port.

Table 4-2 details the format of the legal routing headers. The table shows that the routing headers are divided into four parts, three for the three different addressing modes and one part that should be used for connection to the configuration port. The upper two

bits are used for decoding of the headers, which provides sixty-four valid entries for the three addressing modes. Thus without stripping, sixty-four unicast and sixty-four multicast message destinations can be defined. Although sixty-four entries for the physical decoding mode seems wasteful for an eight port device, this allows for the construction of larger devices whilst maintaining the protocol format.

**Table 4-2 : Header format for addressing modes**

| Type of addressing | Coding [Type, LSB ... MSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Interval addressed | 1 | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | 0 | 0 |
| Logical addressed | 1 | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | 0 | 1 |
| Physical addressed | 1 | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | 1 | 0 |
| Connect to the configuration port | 1 | X | X | X | X | X | X | 1 | 1 |

The interval addressing mode includes a total of ten definable intervals. This number supplies an interval for each output plus an additional two for definition flexibility. The logical addressing mode operates as defined in section 2.2.3.5, but includes slightly modified operation for multicast packets. For such packets, the configuration is defined as a multicast connection-group. Thus if any packet arrives at the switch, it is forwarded to all the outputs defined by the configuration unless the input port (or a group associated link) is included in the connection-group, where it is to be omitted from the connection. Thus multicast packets used to synchronise a number of processes require only one logical address definition, which all processes may use, thus reducing the requirement for multicast message headers.

In adherence to the operation of logical and interval addressing, the network layer of the basic device specifies that messages with unrecognised message headers should be flushed. This will ensure that the network will remain operational in the presence of illegal messages. No other error detection or error recovery is defined for fault tolerance in the basic definition of the NTR-FTM08.

### 4.2.4 Other Router-switch Features and Operation

As stated previously, the configuration port for the NTR-FTM08 should be accessible via the data path. The configuration port should contain all the registers for addressing, stripping and grouping, these are:

- eleven, five-bit registers for interval port information;
- ten, six-bit registers for interval limit information;
- a sixty-four entry, ten-bit look-up table for logical addressing;
- eight, three-bit registers for grouping information;
- one, eight-bit register for stripping information.

## 4.3 Stage One Development Features

The stage one developments of the router-switch definition are based around the premise of fault detection and recovery through localisation. Fault localisation restricts the effects of hardware failure to the failed links and supports a local method of recovery that prevents the failure from effecting neighbouring network entities, similar to the 'localise error' feature of the STC104. This should ensure that any given messages that are injected into the network would not cause the system to deadlock through physical failure. Additionally, features are included that supply a verifiable connection, but with optional mechanism that will automatically inhibit wasteful verification handshaking procedures in periods of extended packet inactivity.

### 4.3.1 Stage One Physical Layer Protocol Enhancements

There are no modifications to the physical layer, although over-voltage and over-current protection mechanisms could be included to suit application requirements.

### 4.3.2 Stage One Data Link Layer Protocol Enhancements

Stage one of the fault tolerance modification includes two additional tokens. Table 4-3 provides the details of these two control tokens, the addition of which provides link initialisation, and link failure isolation support for the communication protocol. The 'connection request' token was included to provide functionality for link initialisation, but also operates as a link reset token. The 'bad end of packet' token was included for corrupt message truncation for failure isolation, similar to the 'exceptional end of packet' token of

the IEEE Std. 1355-1995. However, by introducing an extra termination token, full support of the message format can be maintained, which was a mechanism that was compromised by the standardisation of the DS protocol to the IEEE Std. 1355-1995.

**Table 4-3 : Definition of the extra control tokens for stage one of the fault tolerance mechanism for the NTR-FT08**

| Token Identifier | Abbreviation | Coding [Type, LSB … MSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Connection Request | CONREQ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Bad end of packet | BEOP | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Fault detection and isolation mechanisms of the state one modifications operate based on a state machine shown in Figure 4-1. By defining legal link activity for each state, any non-conforming activity flags an error. The state machine shows that all errors result in the state returning to a single state, namely the 'reset' state. This ensures that each error state attempts to recover by link initialisation. The two steady states of the mechanism are 'asleep' and 'awake'. Whilst in the 'awake' state the link operates as a standard flow controlled point-to-point link. As transmission of data tokens is inhibited in both the 'waking' and 'asleep' states, any receipt of data in these states will return the link into the 'reset' state. The link is also forced into the 'reset' state if a kick-start request is made whilst in the 'asleep' state; the action of this event will be addressed below.

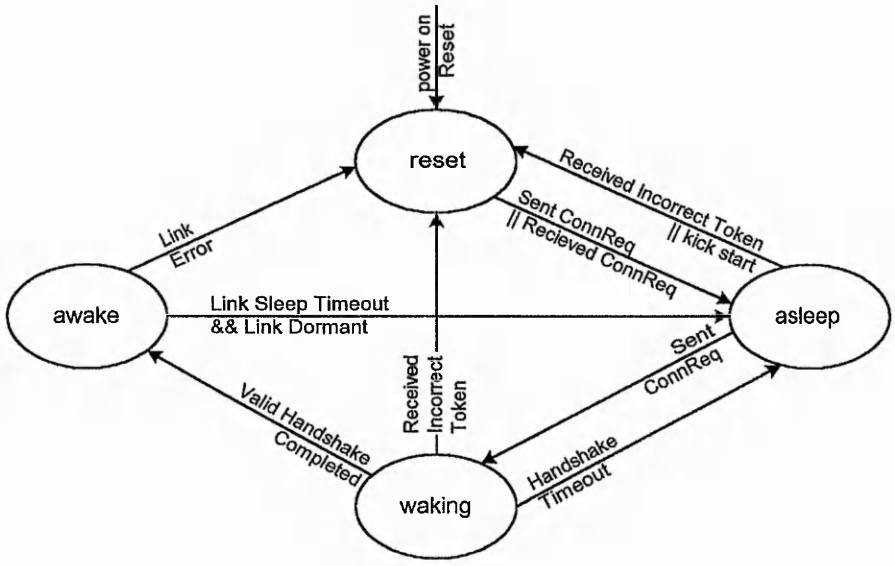



**Figure 4-1 : Finite state machine for the NTR-FTM08 link status**

A failure in the 'awake' state is detected in one of four ways, as described in the four paragraphs below. These four methods concentrate on critical failures of the associated interconnection, and not errors in the bit stream.

The first failure detection is based on synchronisation. As previously described, the format of each token includes a logic '1' start and a logic '0' stop bit. If the sampling circuits detect an incorrect frame, the synchronisation error is flagged.

The second method of failure detection is the monitoring of the receiver FIFO for an overflow condition. Although the flow control mechanism should prevent such a situation from occurring, a buffer overflow indicates the corruption of the data link or its operation outside of its specified parameters.

The third trigger for failure detection is a link activity time-out. Each side of the point-to-point link monitors the link for activity, and if no activity occurs within a predefined period, the connection is said to be lost and the error is flagged.

The final error detection method is triggered on the receipt of a CONREQ token while the link is in the 'awake' state. As recovery of an error state involves reinitialising the link, CONREQ token will be transmitted. Therefore, the receipt of a CONREQ token indicates an error condition at the other side of the point-to-point link, which thus invalidates the current link status.

A similar disconnection feature to the third trigger of failure detection was implemented in the IEEE Std. 1355-1995 standard, which was fundamentally supported within the protocol, as constant transmission was required to maintain synchronisation. While this enabled the implementation of the disconnection error, continual data transmission was unattractive, especially to systems where data transmission was infrequent. The over-sampling technique chosen for the NTR-FTM08 does not require back-to-back transmission for maintenance of synchronisation, but occasional transmission is desired for connection confirmation. To supply a reasonable reaction time to disconnection, the activity time-out must be selected to suit signalling rates, transmission distances, and packet sizes. To ensure link activity, the mechanism transmits flow control tokens when no data is available, which are sent at periods less than the activity time-out. Using the flow control tokens to produce link activity is valid in this specification, as it acts as a reaffirmation of the permission based flow control mechanism. A similar technique is used in the Myrinet system as highlighted in section 3.3.3. Although this

solution transmits with less frequency when compared to the IEEE Std. 1355-1995, transmitting at this reduced rate may still be to often for some systems. This fact encouraged the development of a configurable feature on each link that allowed each link to fall asleep after a predefined time of message inactivity on the link. This feature was called link dormancy. In practice, the link moves from the 'awake' state directly to the 'asleep' state. In addition, allowing the link to return to the 'asleep' state also required methods, with which to kick the sleeping link back awake (kick-start command), and remain asleep while not required. This aspect of the operation will be discussed as part of the description of the link initialisation procedure.

#### 4.3.2.1 Link Initialisation Procedure

Link initialisation defines a safe procedure that can supply confidence that both sides of the connection are ready to transfer data. As Figure 4-1 shows, from the power on condition, the link state machine enters the 'reset' state. In this state, the receiver is enabled to listen to activity on the link, but data connections are prohibited. The link will remain in this state until a CONREQ token is either detected by the receiver, or sent by the transmitter. Control logic will order the transmission of a CONREQ token after a period of time in the 'reset' state. The disconnected or dormant link may remain in the 'asleep' state indefinitely, during which time it will be unavailable. Link dormancy is a configurable setting that allows the link to return to the 'asleep' state. The receipt of any other token in this state will force the link back into the 'reset' state. If the link is configured as non-dormant, during the 'asleep' state, XOFF tokens will be occasionally transmitted. Therefore, if two devices are connected after both have entered the 'asleep' state, the transmission of the XOFF token will be received by the other side and initialisation will be triggered. In contrast, if both ends of the link are configured as dormant, the transmitters will remain silent, thus allowing both devices to remain in the 'asleep' state. Only a kick start command from the higher layers is able to reinitialise the link from this state, returning the link into the 'reset' state. The reasons for the generation of the kick-start command is related to the network layer, which will be addressed in the next section.

Once in the 'asleep' mode, a CONREQ token will only be sent after the receipt of a CONREQ token. Once the CONREQ token has been flagged as sent in the 'asleep' state, the link moves to the 'waking' state. The 'waking' state is the only state that may move to any of the three other states. A legal completion of the handshake will move the link in to the 'awake' state, which is the receipt of either of the flow control tokens or a CONREQ

token. The receipt of any other token results in the link moving to the reset state. The other possibility is no further activity on the link. Without activity within a specified time-out the link returns to the 'asleep' state. As the operation of the initialisation procedure is dependent on time, the mechanism will only function correctly within operational limits of signalling rate and transmission distance. This is acceptable, as the flow control mechanism also imposes these limits. Therefore, for a workable solution, both mechanisms must meet acceptable limits of signalling rates and transmission distances.

### 4.3.3 Stage One Network Layer Protocol Enhancements

The network layer modifications to the NTR-FTM08 implement the fault isolation features. The main aim of isolation is to maintain the appearance of network operation to the rest of the network. The enhancements achieve this using two techniques, namely: active packet recovery and link invalidation. Active packet recovery removes packets from the network that have been corrupted by failure. Link invalidation prevents the use of disabled links and uses redirection through group adaptive routing or removes undeliverable packets once all possible outputs have been confirmed as unavailable.

#### 4.3.3.1 Active Packet Recovery

The NTR-FTM08 has been specified to implement a buffered form of wormhole routing, which results in active packets holding resources across several switches. The detection mechanism operates on critical failure of the interconnection, thus failures can be considered as localised to the physical medium. As the system does not implement virtual channels, the worse case scenario of failure would be two packets active across a single link. Figure 4-2 depicts this worse case scenario in an example network with a failure that would have been flagged from one of the four methods described previously. Two packets are active across multiple router-switches, which has resulted in one interconnection with bi-directional data flow. A fault on this link would cause the two packets to be divided, which, without the isolation mechanism, would result in the related resources being held indefinitely. The shaded links on Figure 4-2 show the allocated resources for each packet.

The data link layer specified the definition of the BEOP token, which is used for message truncation, similar to the EOP_2 of the IEEE Std. 1355-1995. For a packet to free the resources it has been allocated, a termination token must be allowed to travel to its destination. In the example of Figure 4-2, each message has been divided into two by the failure; therefore the isolation procedure for each message must operate in two parts. The

first stage of the mechanism must free resources held by the head of the message, that is the path between the fault and the message destination. This is achieved by terminating any active message at the receiver of the failure point by the BEOP token. Referring back to the example of Figure 4-2, points *x* and *z* are effectively the receivers of that link on router-switches 3 and 2, and must carry out this function. To reiterate, this will allow the rest of the network to operate as normal as the truncated packet continues to its destination, and the arrival of the BEOP token at the destination flags a fault in transmission.

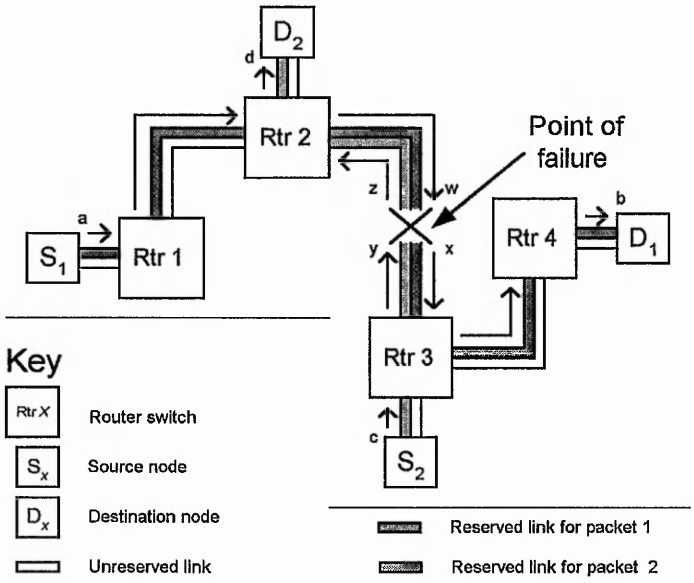



**Figure 4-2 : Theoretical network with a single point of failure, showing allocated resources**

The second part of the mechanism must remove the tail of the packet from between the source and the point of failure. Effectively, point *w* and point *y* are the transmitters of the failed link, and thus are now the new destinations of the tails of these corrupted packets. By spilling packets one and two at points *w* and *y* respectively, the tail of the packets behave normally with respect to the rest of the network.

#### 4.3.3.2 Link Invalidation

Once active packets have been dealt with by the isolation procedure, further steps must be taken to attempt to deliver subsequent packets that require the failed link. Alternatively, a link may have never successfully entered the 'awake' state, or a link was configured as dormant and it has returned to the 'asleep' state. The description of the detection mechanism showed how the link should attempt to re-establish a connection if an error is raised. In the interim, or failing a re-establishment of the connection, a procedure

is required to govern other pending connection-requests. There are three situations that must be handled:

1. connection-requests are pending and the output is not grouped and the output is not 'awake';
2. connection-requests are pending and the output is grouped but none of the outputs are 'awake';
3. connection-requests are pending and the output is grouped with alternative routes available.

The first and second situations must follow the same steps, that is attempt to re-establish a connection on the target link(s), and wait for a period of time. If, after the period of time, the connection is still unavailable, the packet that made the request should be removed. This should be performed for each subsequent packet, to ensure that each is give the same chance to be delivered. This is called the kick-start procedure, and is evident in the link state machine shown in Figure 4-1, as the kick-start command. Irrespective of the configured dormancy setting of the link, the kick-start procedure is used on each unavailable link that is required. This standardises the procedure for the availability of the link, which simplifies implementation.

Removal of the undeliverable packet is achieved by using the functionality of the isolation procedure. Following the time-out of the kick-start procedure, the allocation is enabled that will connect the undeliverable packet to the 'asleep' output. This will spill the packet as if it was active on the occurrence of the failure. Following the removal of the packet, the circuits are reset, which ensures that the router-switch returns to normal operation.

The third situation must utilise the other available routes, thus allowing the use of redundancy through group adaptive routing, for operation in the presence of faults. If some members of the group are shown to be inactive, they should still follow the kick-start procedure described above. If a suitable link is 'awake', but busy, the recovery of the other link would improve throughput. Packets that can be routed via alternative paths will never be removed from the network

## 4.4 Stage Two Development Features

Deadlock in wormhole routed networks is a major concern. As the review of section 2.2.3.7 showed, there have been many solutions, the most prevalent for irregular network being avoidance based source routing. Myrinet implements two stages of deadlock protection, namely avoidance and recovery mechanisms. This is sensible as bit errors in the data stream may negate the precautions of many avoidance systems. The aim of the stage two development was to investigate whether a more responsive deadlock detection and recovery mechanism than the basic time-out system was feasible.

The stage two enhancements evolved from work by Lopez and Martinez [57, 67]. Their heuristic approach allowed the detection of potential deadlock conditions by using information local to routing nodes. Their implementation, whilst still operating a time-out system, reduced messages marked for recovery, thus lowering false detection rates. This was achieved by following the premise that a tree structure of blocked messages forms while waiting for resources occupied by a single message that is advancing. The message that is advancing is known as the root and freeing the resources held by the root will allow the remaining messages to be delivered.

To maintain implementation flexibility and to minimise false deadlock detection, the detection mechanism, devised for the stage two modifications, propagates additional status information up and down the branches of the tree structure of reserved resources. In this way, possible deadlock conditions can be validated, which minimises false detection on non-deadlock cycles and heavy traffic patterns with minimal time-out requirements and without any limitations on packet size.

### 4.4.1 Stage Two Physical Layer Protocol Enhancements

Similar to stage one of the enhancements, there are no modifications to the physical layer.

### 4.4.2 Stage Two Data Link Layer Protocol Enhancements

The mechanism suggested here uses additional control tokens in the deadlock detection process to allow a bi-directional transfer of control information. By defining new control tokens, the passage of information relevant to the deadlock condition can be passed transparently to the data path. For this mechanism, three extra control tokens are defined as shown in Table 4-4. The data link layer implementation of these tokens relates to only transmission and receipt, which is controlled from the network layer implementation.

**Table 4-4 : Definition of the extra control tokens for stage two of the fault tolerance mechanism for the NTR-FTM08**

| Token Identifier | Abbreviation | Coding [Type, LSB ... MSB] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Deadlock probe | DLPRB | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Deadlock path clear | DLCLR | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Data movement | DLMOV | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

### 4.4.3 Stage Two Network Layer Protocol Enhancements

The network layer of the second stage implementations governs the transmission and effects of receipt of the three extra tokens that were defined in the data link layer. The operation of the second stage enhancements is constructed of two parts: deadlock detection, and deadlock recovery.

#### 4.4.3.1 Deadlock Detection

The deadlock detection mechanism uses the flow control information possessed by each link and the additional control tokens. The mechanism operates by sending the deadlock probe token up the path of stalled reserved resources to query the status of the path ahead. Thus if a cycle (deadlock or otherwise) is present, the original node will receive a DLPRB. However, the path clear token will be returned down the path of reserved resources, following a deadlock probe, if a valid exit path exists. The data movement token is used to enhance the information supplied by the data flow-control mechanism, as the hysteresis of the permission based system effects the information that it provides.

To control the distributed detection mechanism, each receiver at each port in the network possesses five flags, namely:

- *targetStalled* – the state of the flow control of the target output;

  The *targetStalled* flag is asserted when the required resources of the packet queued at the receiver have been stalled by the flow control mechanism.

- *iAmRoot* – the status of the packet at the receiver, whether it is the source of a deadlock;

  The *iAmRoot* flag is asserted when a packet is queued for an output and the *targetStalled* flag changes from false to true. Additionally, the *iAmRoot* can

be asserted if the *targetStalled* flag is true and a DLMOV token is received at the output.

The *iAmRoot* signal is de-asserted if the *targetStalled* becomes false or the target output receives a DLCLR token.

- *sentProbe* – whether the receiver requested the transmission of a DLPRB token at the output;

  The *sentProbe* flag is asserted when the receiver raises a request for its target output to generate a DLPRB token. It is de-asserted when a DLCLR token is received at the respective target output or if *targetStalled* becomes false.

- *gotProbe* – whether the receiver has received a DLPRB token;

  The *gotProbe* flag is asserted when the receiver link has detected the arrival of a DLPRB. It is de-asserted if the *targetStalled* is false.

- *cycleTimeout* – whether a timer has elapsed, which is started when a root node has generated a DLPRB token.

  The *cycleTimeout* is asserted after a timer has elapsed. The timer is started when *iAmRoot* is true and *sentProbe* is true.

The assertion of the *cycleTimeout* and the *gotProbe* flags triggers the operation of the recovery mechanism. Additional to the removal of the packet from the cycle, the execution of the recovery mechanism also resets all five flags to false.

The generation of the three control tokens is governed by these status signals. The rules for generation of each token are as follows:

- DLPRB token generation:

  1. if *iAmRoot* is true and *sentProbe* is false;
  2. if *gotProbe* is true and *sentProbe* is false and *iAmRoot* is false.

- DLCLR token generation:

  1. if *gotProbe* is true and the target output is not stalled;
  2. if *gotProbe* and the target output receives a DLCLR token.

- DLMOV token generation:

    1. if data is taken from the receiver FIFO, but not enough to trigger the flow control.

Using these five receiver flags and the rules for control token generation, a description of the mechanism operation, as a simple deadlock cycle is formed, can now be provided. To aid this further the diagrams from Figure 4-3 to Figure 4-7 will be used for reference.

Referring to Figure 4-3, the diagram depicts three packets in an example of a simple network. This state shows a number of connections that possess no obvious problems. Packet 1 entered the network first, and has connected across router-switches 1, 2 and 4. Thus, the five receiver flags for each link that are reserved by this packet would be all false.

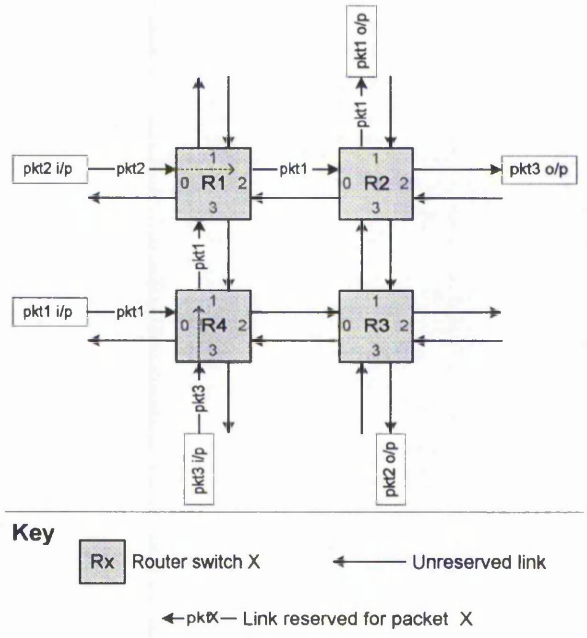



**Figure 4-3 : A depiction of an example network prior to the formation of a deadlock cycle**

Packets 2 and 3 entered subsequently and stalled at router-switch 1 and 4 respectively waiting for the resources held by packet 1. Although the packets are stalled while queuing for their outputs, the packet holding their target outputs is not stalled. Therefore, the receiver flags for packet 2 and 3 at link 0 of router-switch 1 and link 3 of router-switch 4 respectively also would be all false.

The fourth packet to enter the network does so by link 2 of router-switch 3 as shown in Figure 4-4. Uncontended, packet 4 connects to output 0 and is queued at input 2 of router-switch 4. The receiver flags of packet 4 at router-switch 3 are all false, as the packet has established a connection. The receiver flags of input 2 of router-switch 4 are configured as the flags associated with packet 1 and 2. Packet 4 is stalled waiting for resources held by packet 1, and packet one is not stalled, therefore all the flags also are set to false.

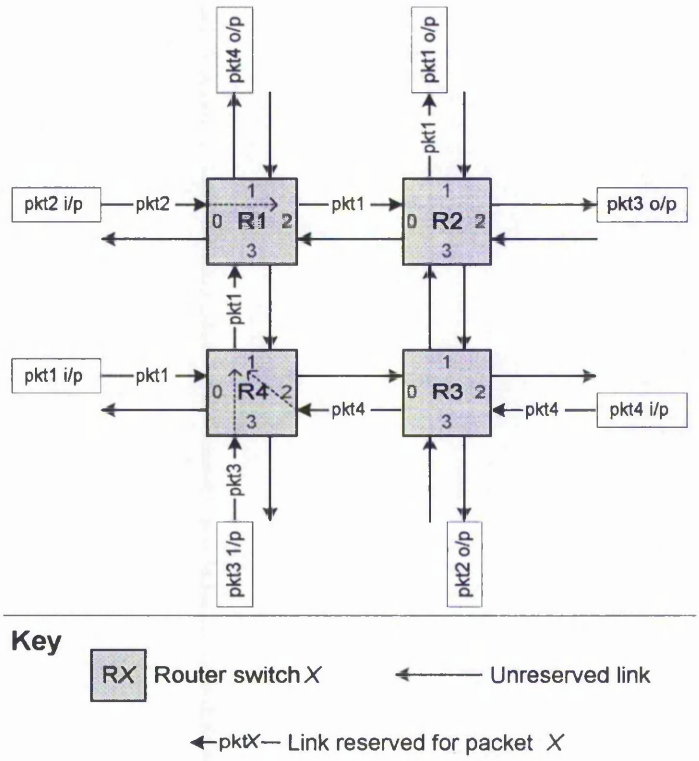



**Figure 4-4 : Introduction of a fourth packet into and example network prior to the formation of a deadlock cycle**

Figure 4-5 depicts two main changes in the status of the network. The first is the entry of the fifth packet on input 1 of router-switch 2, which connects across router-switch 2 to output 3 and is queued at router-switch 3 waiting for output 0. The flags of input 1 of router-switch 2 are set all false as the packet has formed a connection. However, the *targetStalled* flag of input 1 of router-switch 3 is true, as packet 4, which hold the resources of output 0 in router-switch 3, stalled whilst queuing for output 1 of router-switch 4. The remaining flags remain false, as the target output was stalled as the packet was queued. In this state the receiver flags are set thus - {targetStalled, !iAmRoot, !sentProbe, !gotProbe, !cycleTimeout}. Although this inhibits packet 5 generating a

DLPRB token, it is permitted to forward one if the status does not change and DLPRB token is received at input 1.

The second modification to the network state, is the release of some network resources. Packet 1 has release all resources associated with router-switch 4, which has allowed packet three to connect across router-switch 4 to output 1. This action has not changed the state of any flags as packet 2 is still waiting for output 2 of router-switch 1, and packet 4 is still waiting for output 1 of router-switch 1, both of which are still not stalled.

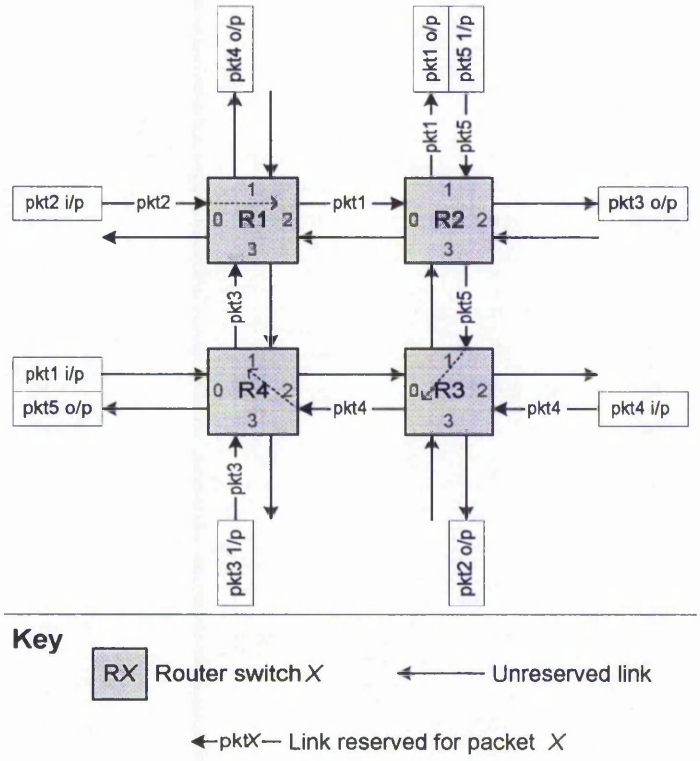



**Figure 4-5 : The entry of the third packet in the formation of a deadlock cycle**

The state of the network is progressed further in Figure 4-6, as packet 3 is queued for output 2 of router-switch 1. Additionally, packet 2 connects across router-switch 1 and arrives at router-switch 2. As packet 3 stalls at router-switch 1 and head of packet 2 is transmitted to router-switch 2, the mechanism remains dormant in router-switch 1. However, as the link flow control mechanism between router-switch 4 and router-switch 1 comes into effect the status of packet 4 changes. As the state of the target output changes to stalled, the deadlock mechanism triggers, which flags packet 4 as a possible root of a deadlock cycle. This forces the generation of a DLPRB token which is sent from router-

switch 4 to router-switch 1. As highlighted above, packet 2 has not been stalled, therefore, as yet, no deadlock cycle exists. This allows the input 3 of router-switch 1 to send a DLCLR token back to router-switch 4, the receipt of which returns packet 4 back into the !root status. Therefore, the receiver flags at router-switch 4, input 2 and router-switch 3, input 1 are the same, that is {targetStalled, !iAmRoot, !sentProbe, !gotProbe, !cycleTimeout}. The receiver flags associated with packet 2 and packet 3 are all set to false, as the packet is connected over the respective router-switch or the target output is waiting for a non-stalled output.

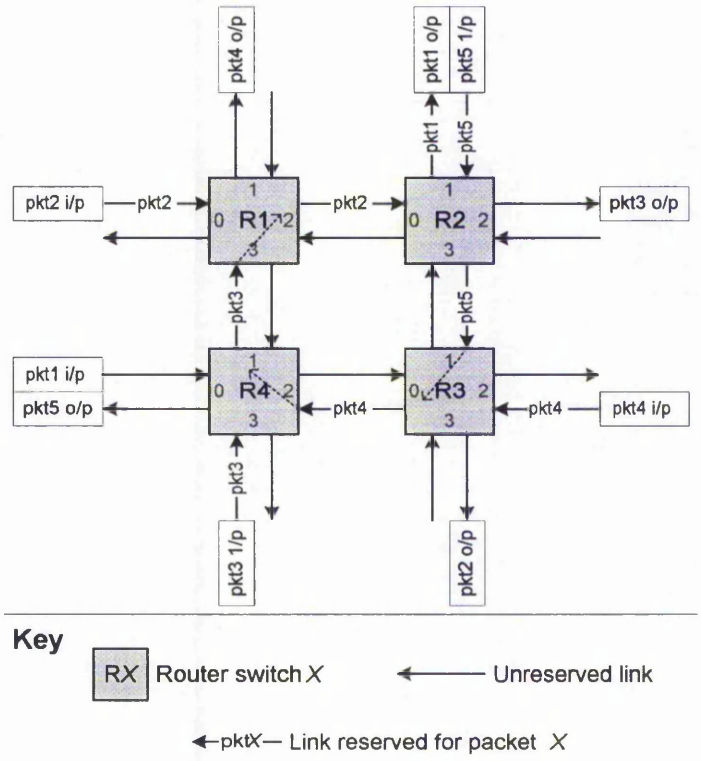



**Figure 4-6 : Depiction of the final stages of the formation of a deadlock cycle**

Figure 4-7 shows packet 1 completely removed from the network, and the completion of the deadlock cycle by packet 2. As packet 2 queues for output 3 on router-switch 2, the packet stalls. As the target output of the packet 2 was already stalled as it was queued the receiver flags are set thus - {targetStalled, !iAmRoot, !sentProbe, !gotProbe, !cycleTimeout}. Recall that the receiver flags for packet three at router-switch 1, input 3 were all set to false. As packet 2 stalls on router-switch 1, output 2 these flags change. As the state of the target output of packet 2 changed to stalled, this makes packet 3 a possible root of a deadlock cycle. Therefore *iAmRoot* and *targetStalled* are set to true.

This generates a DLPRB token for output 2 of router-switch 1, which also sets *sentProbe* to true. Thus the receiver flags for packet 3 at input 3 of router-switch 1 are set {targetStalled, iAmRoot, sentProbe, !gotProbe, !cycleTimeout}.

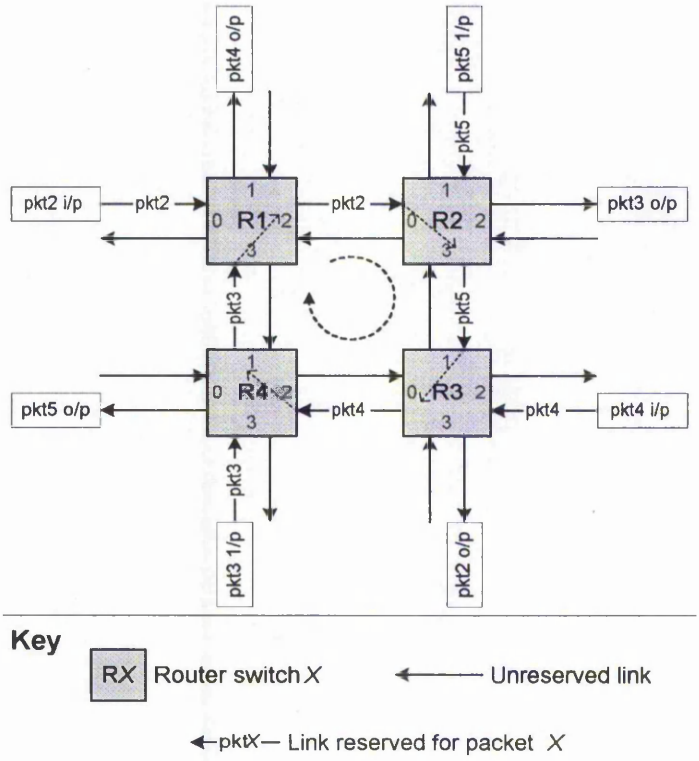



**Figure 4-7 : The complete deadlock cycle**

The arrival of the DLPRB token at input 0 of router-switch 2 is forwarded on to output 3, as 3 is the stalled target output of packet2 and it is not flagged as root or hasn't sent a probe. This sets the *sentProbe* and *gotProbe* flag to true. A similar action propagates the DLPRB token from router-switch 3 to switch4, and from router-switch 4 to router-switch 1. The receipt of the DLPRB token at input 3 of router-switch 1 sets the *gotProbe* to true. This receipt of the deadlock probe has highlighted a possible cycle. This is where the cycle timer is used. It is possible that traffic loading under normal operating condition may have created many possible root nodes, each of which would have generated DLPRB tokens. To ensure that there is enough time to allow any DLCLR tokens to return to the root node, the timer is used. As the deadlock control tokens are transparent to the data stream, this timer can be significantly smaller to those of a pure timer based system. The time-out is based on the diameter of the network and the bit-rate of the physical medium. If a DLCLR token does not arrive back at the node before the time-out, the

recovery mechanism will be triggered. In this example, a DLCLR token will not return as no escape path exists.

#### 4.4.3.2 Deadlock Recovery

Once a deadlock condition has been detected, a recovery mechanism must be initiated to resolve the cyclic dependency problem. Resolution can only be achieved by removing a message from the cycle, therefore relinquishing the resources held by it.

This forces extra functionality at the source node to detect packet loss so that retransmission can take place. Although this is not the most favourable solution, it is the simplest to implement on the router-switch architecture used in this work. For that reason, at this stage of development, message deletion was chosen for the recovery method.

To cleanly delete a message from the network, three steps must be taken. First, all reserved output port resources must be relinquished, second, any pending connection-requests should be cancelled and finally, the message should be removed from the receiving FIFO.

# 5 Detailed Router-Switch Design

This chapter details the implementation of the NTR-FTM08 in three stages. The first stage describes the core routing features of the design. This is followed by the details of the stage one developments that incorporated the fault tolerant mechanisms. Finally, the implementation of the deadlock detection and recovery mechanism is presented, which were part of the stage two developments.

## 5.1 Basic Skeletal Switch

This section presents all basic routing functions, but omits any of the implemented fault tolerant features, which are left for the subsequent sections.

### 5.1.1 Top-Level Router-Switch

The top-level module, illustrated in Figure 5-1, is constructed from three main components, namely the link unit, controller and exchange. This module connects these blocks together and distributes the clock and reset signals.

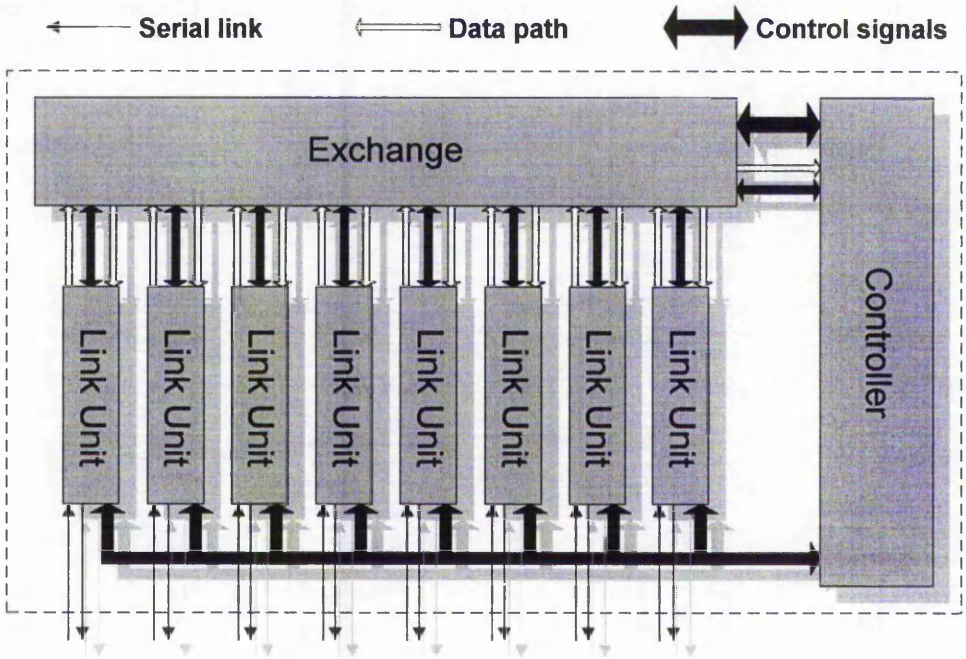



**Figure 5-1 : Block diagram of the top level components of the NTR-FTM08**

### 5.1.2 Link Unit

The link unit is a modular block that contains four sub-blocks, which provide the interface to a single link of the asynchronous transmission medium, in addition to unicast

decoding and connection-requests, which are directed to the controller. The module, as shown in Figure 5-2, is constructed from four blocks, namely the buffered link, receiver controller, interval decoder and transmitter buffer.

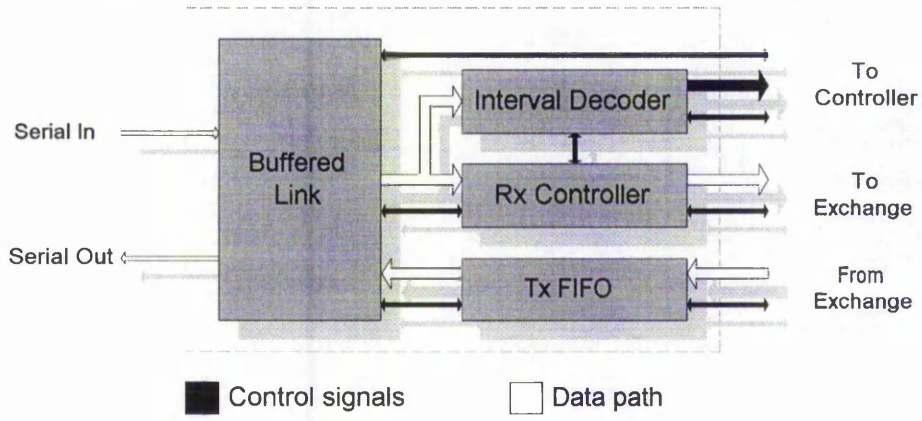



**Figure 5-2 : Block diagram of the link unit of the NTR-FTM08**

**Buffered Link Interface**

The buffered link interface possesses the functionality of an enhanced buffered UART. The module contains four sub-blocks, receiver, transmitter, receiver FIFO and synchroniser, in addition to control logic which implements the data link layer fault tolerance features as described in section 4.3.2. The receiver module synchronises the asynchronous serial bit-stream and converts it to a parallel byte-stream. The transmitter converts a parallel byte-stream into the serial bit stream of the asynchronous protocol. The receiver and transmitter modules work co-operatively to form a fully flow controlled link. The receiver FIFO provides buffering for the incoming data stream, the size of which has been set to thirty-one bytes with an 'almost-full' value of twenty and an 'almost-empty' value of ten. The synchroniser module interfaces the necessary control signals between the sampling and core clocks at the input of the receiver FIFO and the output of the transmitter FIFO. By synchronising at these points, the latency of the link can mask the synchronisation delays. Synchronisation is required as the bulk of the buffered link operates using the sampling clock, whereas the remaining functionality of the router-switch operates using the core clock.

**Receiver Controller**

The receiver controller operates based on the state machine shown in Figure 5-3, which ultimately controls the network layer status of the receiving link. After a device reset and before a packet arrives, the state machine is in the *idle* state. The placement of

data into the receiver FIFO moves the controller through the *header decode* and *decode select* states. These two states provide the clock cycles required by the interval decoder block to decode the lead packet header. The result provided by the interval decoder block controls which state is entered next. If an illegal header is decoded, the receiver will enter the *flush* state. If a valid interval header or physical header is found, the *queue request* state is entered. Finally, if a logical addressing header is detected, the receiver enters the *logical decode request* state.

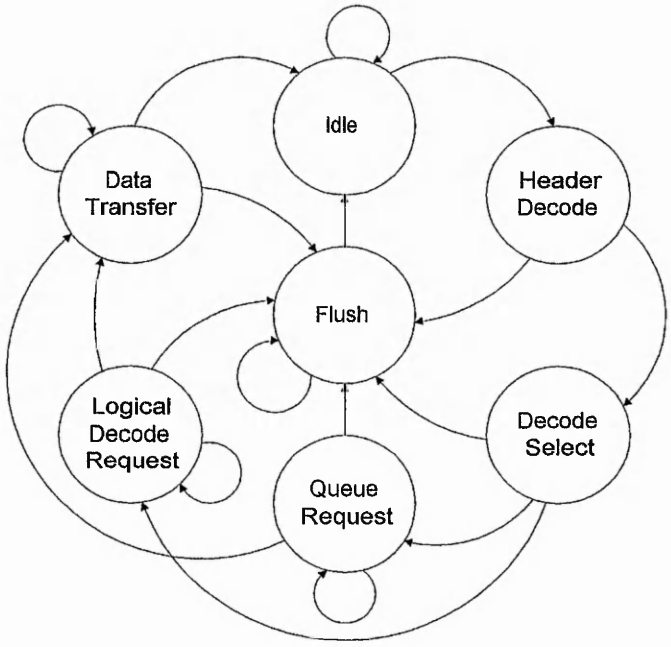



**Figure 5-3 : Receiver controller finite state machine**

Whilst in the *queue request* state, the decoded unicast connection-request is asserted which is produced from the decoding cycles. The controller will wait for a connection-request acknowledgement, which moves the receiver into the *data transfer* state. The *logical decode request* state produces a decode request, which is directed, together with the address header byte, to the logical address decoder within the router-switch controller block. The decode acknowledgement from the logical address decoder also validates the header. If an illegal header is flagged with the acknowledgement, the receiver enters the flush state. Otherwise, the acknowledgement signifies that the header has been decoded and the connection-request has been queued and the receiver moves into the *data transfer* state. Once in the *data transfer* state, the receiver unmasks the receiver FIFO control signals, which are connected to the exchange block. The receiver continues

to monitor the head of the receiver FIFO whilst in the *data transfer* state and flags a disconnection-request to the router-switch controller when the packet delimiter token is detected. This detection also moves the receiver back to the *idle* state, where the whole cycle may repeat.

The *flush* state operates similarly to the *data transfer* state, where the controller monitors the head of the receiver FIFO for the packet delimiter token and returns to the *idle* state when it is detected. In addition to this, the controller also generates receiver FIFO read commands to remove the invalid packet from the receiver FIFO. Therefore, any packets that possess an illegal header are removed from the system, thus freeing the resources. Illegal headers are detected by the interval decoder and logical address decoder unit (discussed in section 5.1.3).

**Interval Decoder**

The interval decoder reads the head of the receiver FIFO and provides decoding information. The module decodes physical and interval addresses, and flags the presence of logical addressed headers to the receiver controller. As the routing headers are spilt into four blocks of sixty-four, the operation of the interval decoder is defined by the upper two bits of a routing header. "00" defines an interval address. "01" defines a logical address. "10" defines a physical address. "11" defines a connection to the configuration port.

An interval addressed packet is decoded using information passed from the configuration port from the router-switch controller module to all link units. The interval information is presented as a six-bit value and a corresponding five-bit destination port. The header is passed in parallel to ten, six-bit comparators; the result of which is used to select the correct interval. The corresponding interval five-bit destination port information is a four-bit binary-coded decimal value of the destination port and an interval validity bit. If a physical address is detected the three lower bits are used as a binary-code decimal value to define the required destination, and the header is always valid. Similarly, if the header defines a connection to the configuration port, the header is marked as valid and the configuration port is selected as the destination port. Any logical addressed header is flagged to the receiver controller, which then generates a logical decode request.

Finally, an overriding feature of the interval decoder is the detection of a control token at the head of any packet, which is indicated by a '0' at bit zero of the nine-bit token.

Such a condition is illegal and indicates a bad packet format, which overrides all other operations to flag an illegal header.

**Transmitter Buffer**

The transmitter buffer implements a two-token FIFO buffer to conceal both the latency of the exchange and the change of connections, enabling back-to-back data transfers out of the link transmitter block. The module also performs header stripping, and generates the transmitter disconnection-request. Both optional and mandatory header stripping is performed by the module, which also allows multicast headers to be stripped on selected outputs. The FIFO buffer block has been implemented in logic, as there are insufficient embedded memory blocks in the target device.

### 5.1.3 Controller

The controller contains the system blocks that regulate the connections of the router-switch. As shown in Figure 5-4, the module is comprised of the following blocks: connection-request queue, link allocater, logical address decoder, logical connections status register and configuration port.

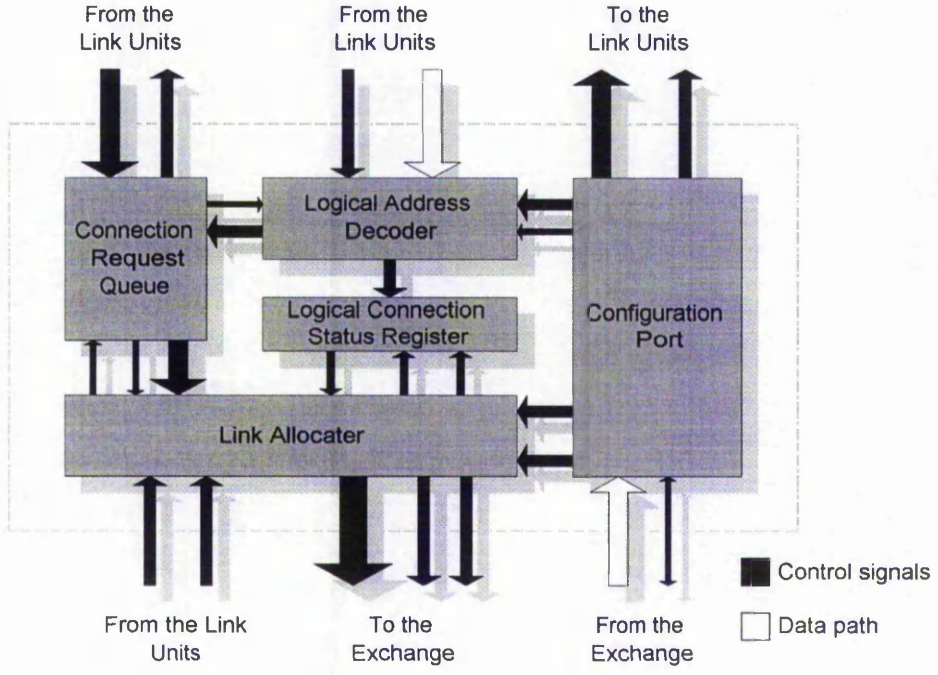



**Figure 5-4 : Block diagram of the controller of the NTR-FTM08**

**Connection-request Queue**

As packets arrive at the receiver ports, the link unit block generates a connection-request for interval and physical addressed packets. Logical decode requests are directed to the logical decoder, which also creates connection-requests. The connection-request queue utilises a unified request selector, which places the requests on to one or more of the nine separate, FIFO queues. Each FIFO queue, which are associated with an output, contains three entries and stores the input code of the pending request. Logical requests are given a higher priority for queuing than interval and physical addressed packets, as logical packets may form multicast connections. By providing a higher priority, the request is presented for allocation sooner, which reduces the delay on multicast connections. As only one logical request is queued at a time, allocation deadlock of multicast connections cannot occur.

Once a connection queue is full of requests, further requests for that output are ignored until a space on the queue is freed. A cyclic priority is used to prevent connection starvation due to the limited queuing entries. The term cyclic priority refers to the priority selection technique if two or more requests are made concurrently. The technique uses the last selection to set the point of priority. For example, if the last selection was number 4 out a choice of 1 to 8, then for the next selection the priority would be {5, 6, 7, 8, 1, 2, 3, 4} (in decreasing order of priority). Although the cyclic priority does not guarantee connection starvation, its use makes such occurrences improbable.

The implementation of the connection FIFO queue has been realised in logic, as there is an insufficient number of embedded memory blocks within the target device.

**Link Allocater**

The link allocater sequentially forms the connections between input and output ports across the exchange. As part of this task, the module maintains the status of the output and input links and the exchange configuration, which the exchange uses to control data transfers across the connections.

The operation of the allocation block works on a priority scheme, such that requests on the lower order queues are serviced before one on the higher ordered queues. This form of priority is permissible as allocation starvation is made improbable by the connection-request queuing stage. The configuration port supplies output grouping information, which is used in the allocation process, together with pending requests from the connection-

request queues. Allocation operates using a 'last minute' grouping technique, which ensures that any pending request for a grouped output is connected to an unconnected group output irrespective on which output queue the request is held. Once an output is reserved for a connection, the status of the output is flagged as in use. This status flag can only be reset with a disconnection-request from the respective output port. If the connection is flagged as a logically addressed packet, the allocation unit must wait for an activation signal from the logical connection status register before the connection can be enabled. If it is not a logically addressed packet, once the output has been flagged, the connection can be activated. The activation of the connection permits the exchange to transfer data between the input and output ports. Each connection active flag is reset by a disconnection-request from the input link unit.

**Logical Address Decoder**

The logical address decoder module receives address decode requests from the link unit modules, which it prioritises and sequentially services. The logical address decoding follow a four state process, namely 'selection', 'decode', 'queue request' and 'acknowledgement'. The first state selects the decode request to service, the result of which is used in the 'decode' state. Logical address decoding is achieved by using the header associated with the request, to address a sixty-four entry, ten-bit wide RAM. The ten-bits of the RAM entry consist of a header validity bit, and a nine-bit, bit vector that relates to the destination outputs required for the connection. Following the address look-up, the resultant bit-vector is masked to remove the source link from the vector and any grouped outputs associated with the source link. This has been implemented so that a multicast group could be defined, and any member could use the header without causing packet livelock or avalanche packet replication. The servicing state machine then moves either to the 'acknowledgement' state or 'queue request' state. The 'acknowledgement' state is entered if an invalid header has been decoded. A header is deemed invalid for one of two reasons: firstly, if the address validity bit is set to 'false', or secondly if the masked bit-vector contained zero valid destinations. The service acknowledgement is asserted with an invalid address flag, which instructs the receiver controller of the link unit to spill the invalid packet. If the 'queue request' state is entered, the masked bit vector is used to raise a connection-request, and this state is maintained until the connection-request queue acknowledges the request. The connection-request is also passed to the logical connection status register, to aid connection allocation of multicast packets.

The logical address decoder also includes a memory interface that allows control signals from the configuration port to configure the look-up table.

**Logical Connection Status Register**

As logical addressed headers can be used to implement multicast connections, a method was required to ensure that all required outputs are reserved before the connection was enabled for data transfer. To allow multiple logical requests to be serviced concurrently, the logical connections status register has been implemented, such that normal connection-request queuing could be utilised. Each input port has a status register associated with it, which maintains the state of any logical connection it may make. This information is supplied as the logical address decoder raises a connection queue request. As the link allocater reserves outputs for connections, the information is also sent to the logical connection status register. Once all the outputs of a request have been made, a signal is sent to the link allocater to activate the connection.

**Configuration Port**

As stated previously, the configuration port for the NTR-FTM08 is accessible via the data path. To achieve this, the configuration port acts as a valid output port, which provides access via any input port, through the exchange. Thus, configuration information takes the same form as normal system packets; that is, a routing header, payload and termination token. With future design implementations in mind, a scaleable programming method has been implemented that is based on the byte-stream supplied by the exchange. The specification of the NTR-FTM08 contains the registers for addressing, stripping and grouping configuration, namely:

- eleven, five-bit registers for interval port information;
- ten, six-bit registers for interval limit information;
- eight, three-bit registers for grouping information;
- one, eight-bit register for stripping information.

Additionally, the configuration port possesses an interface to configuration the look-up table used for logical address decoding.

By using the byte stream from the exchange as a stream of atomic instructions, as listed in Table 5-1, all these registers can be configured in a standardised method. Using

these instructions, each configuration register within the configuration port is accessible by a data register, and for multiple registers, an address register. The data register is ten bits long, which is equal to the longest configuration feature, namely the logical look-up table. The address register is six bits long to support the greatest address range, namely the logical address register. The data register is used to store the configuration data before it is transferred to the correct configuration register. The address register is used if an offset is required into a group of configuration registers; namely, the interval registers, the logical address look-up table and the grouping registers.

**Table 5-1 : Atomic instructions for the configuration port of the NTR-FTM08**

| Token Identifier | Abbr. | Coding [MSB ... LSB, Type] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Load address register | CLAR | 0 | 0 | 0 | 0 | $A_0$ | $A_1$ | $A_2$ | $A_3$ | 1 |
| Load data register | CLDR | 0 | 0 | 0 | 1 | $D_0$ | $D_1$ | $D_2$ | $D_3$ | 1 |
| Write interval port register | CWIP | 0 | 1 | 0 | 0 | X | X | X | X | 1 |
| Write interval limit register | CWIR | 0 | 1 | 0 | 1 | X | X | X | X | 1 |
| Write logical lookup table | CWLR | 0 | 1 | 1 | 1 | X | X | X | X | 1 |
| Write stripping register | CWSR | 0 | 1 | 1 | 0 | X | X | X | X | 1 |
| Write grouping register | CWGR | 1 | 0 | 0 | 0 | X | X | X | X | 1 |

Configuration data and the offset address are loaded into the associated register using the CLDR and CLAR commands respectively. The configuration data is then latched to the correct configuration register by the associated instructions. Each load instruction carries four bits of information, which means that multiple CLAR and CLDR instructions are required to carry all the information required to configure many of the registers. The nibble carried with the load instruction is pushed into the least significant bits of the relevant register, thus longer words of information must be sent with the most significant bits first. Table 5-2 shows an example packet that would configure entry '53' (110101 binary) of the logical addressing lookup table, with a broadcast header. Note that the routing header and end of packet token are included, to show the packet construction.

The configuration unit inhibits all address decoding, queue requests and connection allocations while the configuration unit is connected, but established connections are unaffected. Additionally, the control port initially delays the processing of configuration data to ensure that all servicing has ceased.

**Table 5-2 : Example packet for configuration of the NTR-FTM08**

| Instruction | Coding [MSB … LSB, Type] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Connect to control port | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Load address register with | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| '00 0000'(binary) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| Load data register with | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| '11 1111 1111'(binary) | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Write to logical lookup table | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| Disconnect from control port | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

### 5.1.4 Exchange

The exchange forms the connection between the receiver and transmitter(s). In this role, the module actively transfers data between the receiver and transmitter FIFOs. Figure 5-5 shows the basic structure of the exchange, which delivers the data to the correct output with the use of the exchange unit sub-modules and control signals derived from status information. The exchange forms a non-blocking crossbar; that is, up to eight connections can transfer data concurrently. The nine-bit transfers operate over four clock cycles, which results in an effective maximum throughput of eighteen bits per clock cycle.

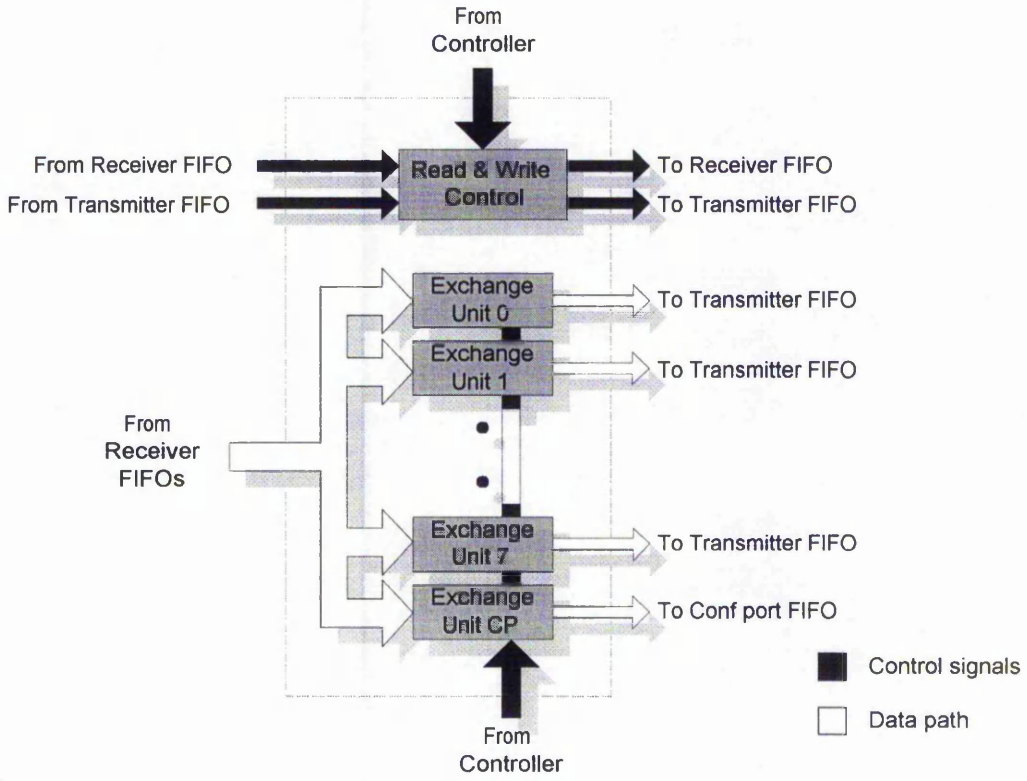



**Figure 5-5 : Block diagram of the exchange of the NTR-FTM08**

As multicast connections can be formed, the movement of data across the exchange is synchronised between the input and all the outputs. This is achieved by the control logic for the read signal for each receiver to be dependent on all transmitter FIFO full flags and output status flags that are associated with the connection. Connection information is provided by the link allocater from within the router-switch controller. A transfer is made if the connection is enabled and the receiver FIFO is not empty and all of the target transmitter FIFOs are not full.

**Exchange Unit**

The exchange unit module operates as a three-bit multiplexor, which provides data to a nine-bit shift register. The exchange transfers a nine-bit token over four clock cycles. The first cycle is used to check the associated receiver and transmitter FIFO buffers were ready, and the remaining cycles are used to transfer the data. In this way, on the third transfer cycle the data is ready to be latched into the target transmitter FIFO.

## 5.2 Fault Tolerance - Stage One

The stage one fault tolerance modifications concentrated on features for fault detection and localisation. This limited alterations to two areas, namely the link unit and the controller.

### 5.2.1 Stage One Enhancements to the Link Unit

The enhancements to the link unit are centred on error detection and isolation based on the truncation and removal of corrupted packets. The enhancements are focused on the lower levels of data transfer, that is the physical link operation and packet format.

**Alterations to the Buffered link**

As described earlier in section 4.3, fault detection is based on link status and expected operational procedures. The enhancements of the basic functionality include features for the transmission and recognition of the additional control tokens specified in section 4.3.2. This allows the implementation of the initialisation and link status state machine, which operates with the control signalling of the link described in 4.3.2.

The operation of the error detection and the link initialisation procedure depends on two periodic control signals, namely heartbeat and check-pulse. These signals are used for all states of the link, as they supply triggers for initialisation and disconnection detection

procedures. The signals are created by a centralised counter to minimise logic requirements. The current implementation generates a check-pulse once every 511 sample clock cycles, which is approximately equivalent to the transmission of 31 tokens, where the heartbeat is generated once every 255 sample clock cycles, which is approximately equivalent to the transmission of 15 tokens.

For the disconnection detection mechanism, the heartbeat signal is used to create link activity and the check-pulse signal is used to verify link activity, while the link is 'awake'. Link activity is achieved by the periodic transmission of idle tokens. Idle tokens were defined in the specification as flow control token relative to the flow state; that is GO tokens when the link is free, and STOP tokens when the link is stalled. Verification of link activity is achieved by logging the arrival of any token, which is recorded by the 'link activity' flag. If the 'link activity' flag is not asserted on the occurrence of the check-pulse signal, a disconnection error is flagged. The check-pulse also resets the 'link activity' flag, to restart the verification procedure. The synchronisation error is flagged by the sampling circuits, which check the token framing to ensure a valid stop token is received after the assumed start token. The receiver circuits also provide a flag, which indicates the receipt of a 'connection-request' control token. An overflow error is detected by a small control circuit, which monitors the state of the receiver FIFO and the write signal.

The triggering of any error detection mechanisms moves the link state machine to the 'reset' state. This results in the truncation of any incoming packet that is active and invalidates the receipt of any further tokens until the link has been successfully initialised to maintain packet integrity. For truncation, a 'receiver packet status' flag monitors the receiver data stream to flag receiver packet activity. On the event of a reset, if the 'receiver packet status' flag is true, a BEOP token is written to the receiver FIFO buffer, effectively truncating the corrupted packet.

The final modification to the buffered link is the support of link dormancy. This configuration setting allows a link, which has been idle for a predetermined time, to return to the 'asleep' state. To achieve this functionality, an inactivity counter and a 'transmitter packet activity' flag have been included. The 'transmitter packet activity' flag monitors the transmitter data stream, and indicates a packet is active across the transmitter link. The 'packet inactivity' counter is incremented at the occurrence of the check-pulse signal if the both the transmitter and receiver packet activity flag are false. The 'packet inactivity' counter is reset if either activity flag are true. If the inactivity counter reaches a predefined

value, the transmission of idle tokens is inhibited, which results in a disconnection error. However, the activity and link dormancy status information are used to ensure that the link safely returns to the 'asleep' state as opposed to raising an error flag. Whilst in the 'asleep' state, the dormancy feature inhibits further transmission of idle tokens. The packet inactivity counter is set to a maximum of four in the prototype NTR-FTM08, although it is suggested that this value would be much larger in a final implementation to suit application needs.

If the dormant link is required following a return to the 'asleep' state, a kick-start request can be used to restart the link. The kick-start request forces the link from the 'asleep' state to the 'reset' state, which initiates the initialisation procedure. The kick-start request is generated by the controller if the link status makes the link unavailable and the output is required. In addition to the function of waking a dormant link, the kick-start also doubles as a request for the confirmation of the link status. The possible responses to this request are either the link moving to the 'awake' state, and therefore becoming available, or the assertion of the still-dead signal. The still-dead signal is asserted if the link has not returned to the 'awake' state after an appropriate time has elapsed since the kick-start request. In the current implementation, this period equates to three occurrences of the check-pulse signal, which equates in time to between 93 and 124 tokens. This period was arbitrarily chosen, but could be easily altered to a more appropriate value for application requirements.

**Alterations to the Transmitter FIFO**

The transmitter FIFO buffer needed packet removal functionality for the isolation procedure of the stage one mechanism, which involves the removal of the tail of severed packets. The mechanism operates on the status of the 'link active' flag in conjunction with an 'active packet' status flag. The 'active packet' status flag monitors the transmitter data stream for active packets across the link. If the 'active packet' status flag is true and the associated output is inactive, the remaining part of the packet is spilt. The packet removal is controlled by a 'spill' flag, which is reset on the arrival of the packet termination token. This ensures that the whole packet is removed from the data path, and each packet is spilt according to the link status as it moves through the transmitter FIFO. The remaining operation of the transmitter is unaffected, and a disconnection-request is raised as the packet termination token is latched in the transmitter FIFO.

### 5.2.2 Stage One Enhancements to the Controller

The enhancements to the controller are centred on error isolation, which ensured that packets were given a suitable chance to be delivered, but any undeliverable packets were removed to provide maximum network availability.

**Alterations to the Link Allocater**

Fault isolation procedures within the controller involve the connection of undeliverable packets to the target outputs that are unavailable, which will result in the removal of the packet from the system via the transmitter buffer. However, the main aim of the link allocater is to ensure that each packet should be given adequate chance to establish a connection to its required destination before isolation procedures are initiated. Thus, for each connection-request that requires an unavailable output, the output must be queried to confirm its status before any further action is taken. As described in the previous section, the buffered link is involved in this procedure with the use of the kick-start, still-dead and link availability signals. The link allocater generates kick-start requests based on grouping configuration, output availability states and pending connection-requests. After a kick-start request is generated, the link allocater ignores the associated connection-requests until either a link becomes available or the still-dead confirmation signal returns. The still-dead signal allows a connection-request for the associated output to be serviced to remove a single packet. Thus, subsequent connection-requests for the same output also must validate the destination output by the use of the same procedure.

For fault tolerance through redundancy, group adaptive routing may be used to supply more routes in the network. For this reason, the link allocater considers the status of all the outputs that may be grouped. Similarly, with grouped outputs, the kick-start is generated if a connection-request is pending for an output, which is in a group with unavailable links. This ensures that all routing possibilities are verified for use when required to maximise concurrency. Packets are only connected for removal if all outputs within the group have been confirmed as unavailable, then the packet marked for removal is spilt through the output for which it is queued. This simplifies the validation and removal selection mechanism, and allows subsequent packets the same chance for a connection.

**Alterations to the Configuration port**

There are no enhancements to the configuration port which relate to the fault detection or isolation procedure, but the additional link dormancy feature requires configuration registers. The dormancy configuration register is eight bits in length, each bit is associated with an individual link. To configure this register, an additional configuration register command was defined. Table 5-3 shows the format of this additional write command.

**Table 5-3 : Additional atomic instruction for the configuration port of the NTR-FTM08**

| Token Identifier | Abbr. | Coding [Type, LSB … MSB] |
|---|---|---|
| Write dormancy register | CWDR | 1 0 0 0 0 1 0 0 1 |

## 5.3 Fault Tolerance – Stage Two

The primary aim of the second stage of research was the investigation of a deadlock handling procedure. The implementation only deals with basic unicast messaging.

### 5.3.1 Stage Two Enhancements to the Link Unit

**Alterations to the Buffered Link**

The bulk of deadlock detection algorithm operates within the higher levels of router-switch operation. This minimises the involvement of the link to injection and extraction of extra control tokens; namely the 'deadlock probe' (DLPRB), 'deadlock path clear' (DLCLR), data movement (DLMOV) tokens, which were defined in section 4.4.2. Transmission of DLPRB and DLCLR only occurs after a request from the controller. The DLMOV token is an autonomous operation that is triggered if data is removed from the associated receiver FIFO, which has been stalled. The transmission of the DLMOV is restricted, such that it is sent only after a certain time has elapsed after data is removed. This ensures that it is sent only if the movement out of the receiver FIFO is not sufficient to trigger the flow control mechanism. The time equates to the time it would take to transmit the number of tokens equal to the difference of the full capacity of the FIFO and activity of the 'almost-empty' flag. This equates to the approximate transmission time of twenty-four tokens in the simulation implementation as, the total capacity was thirty-one tokens, and 'almost-empty' was asserted when the FIFO contained less than eight tokens.

**Alterations to the Receiver Controller**

The remaining enhancements within the link unit module are involved in the recovery mechanism of the deadlock procedure. Once a packet has been flagged as the root of a deadlock by the procedure, that packet is marked for removal from the network. This decision for removal is made in the controller, but a command signal has been integrated into the receiver controller that removes a packet from the receiver FIFO and resets the receiver. The implementation is a reuse of the packet flushing functionality as defined in the skeletal router-switch is section 5.1.2.

### 5.3.2 Stage Two Enhancements to the Controller

**Deadlock Detection Unit**

The deadlock detection unit (DDU) is an additional modular block that had been included to supply the functionality for the deadlock detection procedure. The DDU is constructed using modular sub-block, each of which is associated with an input link. The sub-blocks contain the five status flags and associated control logic as described in section 4.4.3.1 (namely, targetStalled, iAmRoot, sentProbe, gotProbe and cycleTimeout). The receipt of deadlock control tokens is reported to the DDU and it generates requests for the transmission of deadlock control tokens based on requests from the sub-blocks. The queuing and connection status of each input is maintained by monitoring the connection queue requests and input status flags. Although, this includes replication of status information, it allows an isolated operation that required minimal modification to the remaining parts of the switch. Implementation details operate as governed by the detection mechanism as described in section 4.4.3.1.

Once a packet has been flagged as the root of a deadlock cycle, the removal mechanism is initiated. As discussed earlier, the DDU sends a request to the associated link unit, at which the packet arrived, to flush the packet at the receiver FIFO. In addition to this, the connection-request must also be removed from the system to maintain connection integrity. The DDU is responsible for generating the control signals for this removal, which involves halting the link allocation operation, and supplying the relevant information to the connection-request queue to target the correct request for removal.

The deadlock time-out value used is an equivalent time of approximately 23 tokens. This figure should be proportional to the diameter of the network, which would then allow a DLCLR token to propagate over the maximum number of hops. By using a value close

to twenty-three tokens, it is estimated that networks with a diameter of eight router-switches could be supported (due to transmission time and worse case delays). However, further testing in larger networks is required to validate this premise.

**Alterations to the Connection-request Queue**

The connection-request queue is based on a FIFO storage structure as described in section 5.1.3, so the removal of entries that may not be at the head of queues dramatically alters the operation of the module. As described earlier, the implementation of the connection queue was realised in logic rather than with an embedded memory block. The read and write control logic for each entry is controlled by register status flags. Thus, additional logic allows entry comparison, which generates localised read and write signals, which are used to remove the request from the queue.

**Alterations to the Link Allocater**

Features are included in the link allocater to stall its operation. This is to ensure that the removal of the connection-request does not interfere with the integrity of connection servicing. Early simulations of the implementation showed that certain network states could generate multiple root nodes in deadlock cycles, which were problematic in connection integrity without with this precaution. The implementation of the stalling functionality includes stalling the earlier stages of allocation. Thus, connections that are in the final stages of completion are allowed to finish, after which the DDU initiates the removal of the packet from the network.

# 6 Design Synthesis & Verification

This chapter details the simulation and hardware results from the implementation of the specification described in chapters 4 and 5. All data presented here have been normalised to show the true operation of the device relative to the calculated ideal values. The normalisation enables a clearer analysis of the device operation and allows comparisons with systems that operate at different transfer rates. The raw figures and calculations for each data set within this chapter are included in appendices A, B, C and F for completeness.

## 6.1 Preliminary Simulations

### 6.1.1 Credit-based and Permission-based Flow Control Comparison Analysis

The first simulations were based on a four-port model of an earlier device that was derived from the NTR-M04. Minor modifications were made to the device, which included the option to use the original credit-based flow control mechanism or a permission-based STOP/GO mechanism, both of which operated with a twenty-byte receiver FIFO. The credit-based mechanism set an 'almost-full' threshold of twelve bytes, whereas the permission-based mechanism used an 'almost-full' threshold of sixteen bytes and an 'almost-empty' threshold of four bytes. A single device was simulated with a range of workloads based on a random (uniform) destination and injection rate. Random test vectors were used to provide a view of the behaviour of the device under a varying load. Although the random nature of the packets may not have matched the traffic pattern of any one application, it did subject the network to load conditions that involve contention between packets within the network. Each set of test vectors was generated statically before each simulation, which allowed the same sets to be used for both flow control mechanisms. Each simulation injected 8000, 32-byte packets into the network, which equated to 2000 packets from each source. The 'start of transmission' time and 'end of receipt' time was recorded and stored for each packet by a VHDL test bench. This information was then processed and analysed following the completion of the simulation.

Figure 6-1 shows the effective data throughput of the four-port router-switch against an increasing offered data load, which reveals the that the device reached saturation at an accepted load of between 60% and 70% of the calculated ideal throughput. The difference in throughput between the two flow control mechanisms under saturated conditions is shown to reach approximately 6%. This is primarily due to the higher levels

of guaranteed control signalling of the credit-based mechanism. As the applied workload increases, so does the chance of bi-directional data. Thus, the guaranteed loss of bandwidth of the credit-based mechanism affects the performance of the device as a whole.

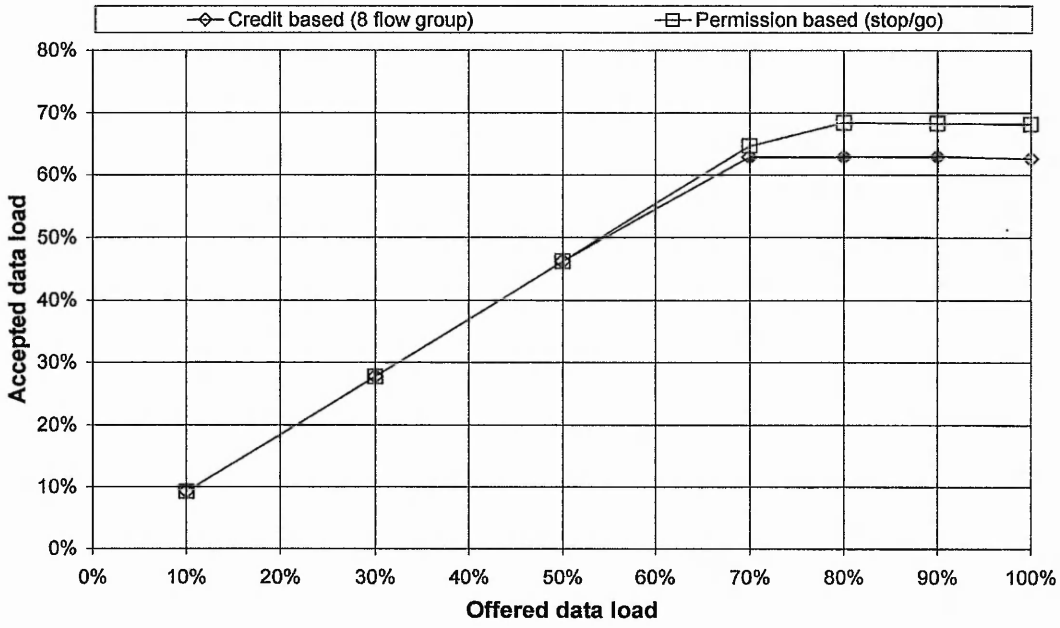



**Figure 6-1 : Offered load verses accepted load for a behavioural model of a four port router-switch for permission and credit based flow control**

From the results of Figure 6-1, the limiting effects of the credit-based mechanism do not significantly affect the throughput when the network operates below the level of saturation. However, there are sub-saturation differences between the mechanisms; packet latency is increased with the use of the credit-based mechanism. Figure 6-2 displays the average packet latency against the accepted data load, showing obvious differences in network behaviour when the device was subjected to sub-saturation conditions. Ideally, the average packet latency should remain constant as the applied workload increases. However, the contention in packet connection-requests as the workload rises, makes the ideal case impossible. Figure 6-2 reveals that the flow control mechanisms do affect the operational characteristic of the device even below the saturated workload conditions. The tabular form of these results is included in appendix A.

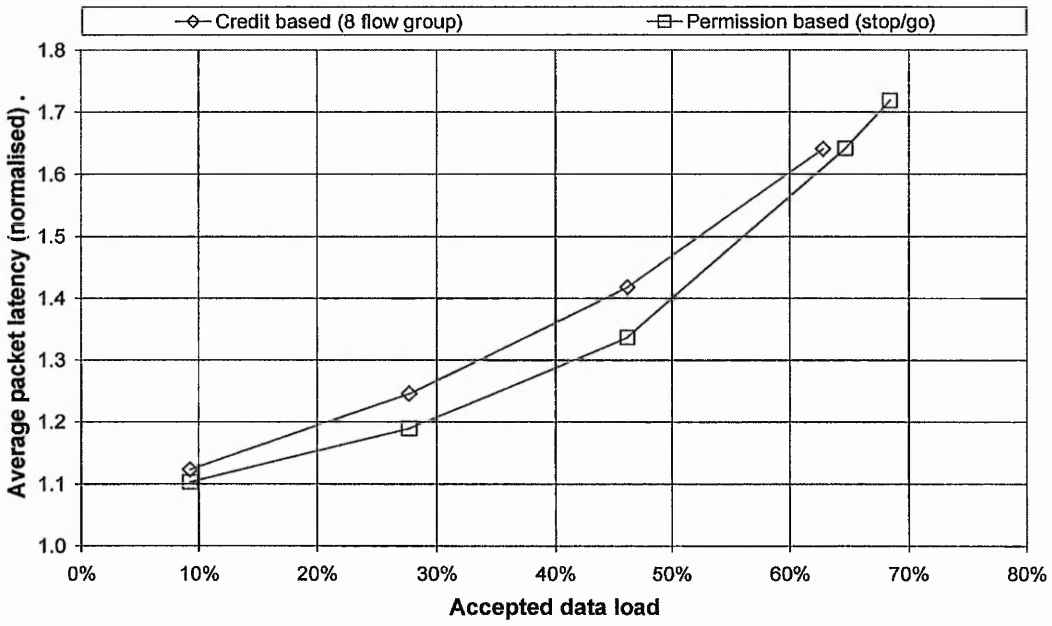



**Figure 6-2 : Accepted data load verses average packet latency for a behavioural model of a four port router-switch for permission and credit based flow control**

Following the development of the first behavioural model of the skeletal NTR-FTM08 router-switch, further simulations were carried out that concentrated on the permission-based flow control mechanism. This basic analysis centred on the effects of the 'almost-full' and 'almost-empty' thresholds within the mechanism. As discussed in section 2.2.2.2, the minimum value of the 'almost-full' threshold is defined by the network operational parameters. Alteration of the 'almost-full' figure from the recommended value would either result in flow control failure or wasted buffering resources. Similarly, the 'almost-empty' threshold also requires a minimum value to prevent data starvation. However, by increasing the value of the 'almost-empty' flag, the differential value between the two thresholds decreases and alters the behaviour of the device. This results in increased levels of control signalling in most situations, but as the flow control mechanism is more responsive to the network state, it should also provide better utilisation of buffering.

### 6.1.2 Further Permission-based Flow Control Analysis

Two regular, multiple router-switch networks were used for the threshold analysis simulations, as opposed to a single device of the earlier simulations. The primary reason for using multiple router-switch networks was to place greater strain on the flow control mechanism. The more complex routes within the networks increased the levels of bi-

directional data transfers and packet contention. Regular topologies were chosen as deadlock free configurations could be quickly generated, and results could be contrasted with previous research to validate the trend of the network response. The two topologies included a four-switch, 2-D mesh and an eight-switch, 2D torus. Both test networks included sixteen nodes that transmitted and received packets to and from the network. The diagrams and configuration details for these networks are provided in appendix B.

The tests vectors were generated statically, containing 16000 packets, which equated to 1000 packets from each node. Each set of test vectors conformed to a workload rating, which related to the rate of packet injection. The packet destinations and injection intervals were based on a random (uniform) destination and injection rate. The VHDL test bench monitored and recorded the transmission and receipt of all packets to allow post-simulation analysis. Each router-switch contained a larger amount of buffering than used previously, to allow larger threshold differentials to be investigated. Each receiver FIFO contained forty-eight bytes with the 'almost-full' threshold remaining constant at forty bytes. Three differentials were simulated; namely eight, sixteen and thirty-two. Figure 6-3 and Figure 6-4 present the results of the analysis. Full results in tabular form can be found in appendix B.

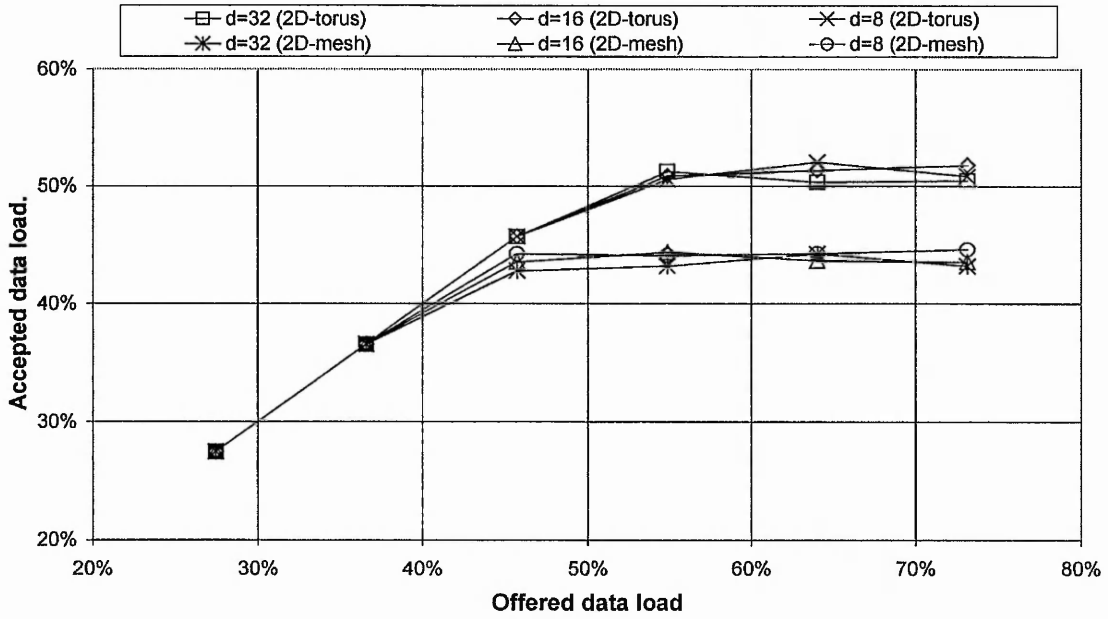



**Figure 6-3 : Offered data load verses accepted data load over a range of workloads for a torus and mesh network with variation on the flow control threshold value differential**

Figure 6-3 reveals that the value of the differential had little effect to the overall throughput figures, as devices saturated around the same level of accepted throughput, with a maximum variance of approximately one percent between the different thresholds. The small size and irregular nature of the variances indicate that the differences seen in the results are most probably due to an insufficient quantity of test vectors. To validate this assumption and to see the true effect of the variations of differentials, larger test vector sets would be required. Nevertheless, the trend of these results does show that the effects of the varying the threshold values are insignificant when compared to the network topological effects.

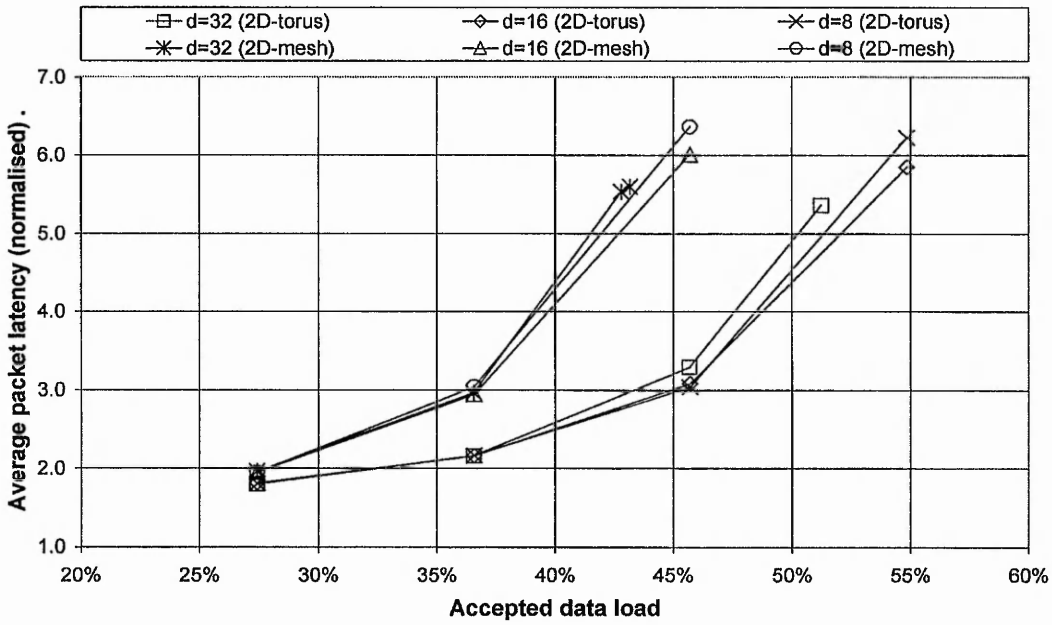



**Figure 6-4 : Accepted data load verses average packet latency over a range of workloads for a torus and mesh network with variation on the flow control threshold value differential**

In contrast to Figure 6-3, the analysis presented in Figure 6-4 does display a trend in the results, which show that the differential value d=16 provided the best characteristic as the network approached saturation. Although d=32 resulted in similar and occasionally smaller average packet latencies to that of d=16, the comparable saturation level was smaller. This is the result of packet injection throttling, which reduces the level of saturation, and therefore improves packet latencies at the cost of network throughput. Although the range of these simulations was small, the results of Figure 6-4 imply that there is an optimum settings for the 'almost-empty' threshold, which can not be assumed to

be the same as the operational limits defined in section 2.2.2.2. Future research based on more complex networks, where variations occur in packet size, topology, traffic patterns and node behaviour, may provided the required insight in this subject.

## 6.2 Basic Design Plus Stage One Enhancements

This section details the simulations, synthesis, and hardware tests that were carried out during the development cycle of the stage one NTR-FTM08. Performance test results are presented from both simulation and hardware implementations.

### 6.2.1 Design Simulation

Simulation of the router-switch design was carried out for the verification of device operation and preliminary performance analysis. In this way, the device operation could be analysed at each stage of connection servicing.

#### 6.2.1.1 Verification

Functional simulations were carried out to prove the design before synthesis was performed. A brief description is now provided for each test bench that was produced.

**Sampling tests**

The NTR-FTM08 operates using an over-sampling circuit to recover the data from the incoming asynchronous data-stream. These tests validated the interface circuits, verifying the operation of both the transmitter and receiver circuits.

**Device testing**

Following basic module verification, a number of simulations were produced to verify the device operation as a whole. The test benches were written in VHDL, and were based on a standard single controlling process model. To minimise the length of the simulation, a number of architectures were implemented that each dealt with a different aspect of the design and could be executed independently.

- Testbench 1 : Unconfigured router-switch tests

  This architecture operated using physical addressed packets and no configuration was sent to the device. The test concentrated on areas such as connection, disconnection, valid routing, valid delivery, flow control operation (back pressure across router-switch), concurrent contentious/non-

contentious connections, and multiple contentious connection-request integrity.

- Testbench 2 : Grouping tests

  This test used physical addressed packets, and various forms of output grouping, which required configuration of the router-switch. Thus, the test verified all major routing functions, as described above in testbench 1, but with grouping applied.

- Testbench 3 : Interval tests

  This was a basic repetition of testbench 1, but with the physical addressed packets being replaced by interval addressing. The tests also proved the operation of the configuration procedure of the interval addressing registers.

- Testbench 4 : Logical tests

  In addition to the equivalent unicast tests of testbench 1, this test included multicast connection, grouping and multiple packets with mixed addressing, invalid packet recovery, and configuration of the logical address look-up table.

- Testbench 5 : Fault detection tests

  Testbench 5 verified that the detection and recovery procedures of the fault tolerance specification operated as expected. This included receiver overflow, synchronisation errors, link resets, and disconnection errors at the link level, and the operation of the allocation for disabled links for grouped and non-grouped outputs.

- Testbench 6 : link dormancy tests

  As link dormancy was a configurable option, this testbench was implemented to proved its operation, which involved a range of conditions, such as grouping, multicast and unavailable links.

#### 6.2.1.2 Device Performance

The following sub-section presents the simulation based performance analysis of the NTR-FTM08. The analysis was derived from both unloaded and synthetically loaded network simulations.

**Unloaded Network**

Although point-to-point bandwidth measurements of an unloaded network show little of the router-switch behaviour, these figures have been produced to allow a basic comparison with the devices produced by the earlier research. In these simulations a number of packets were monitored as they were sent across the router-switch. The average packet latency was recorded for a range of packet sizes, which were used to calculate the average data bandwidth figures. The results are presented as a percentage of raw bit rate to allow direct comparisons with earlier devices. Figure 6-5 presents five plots that depict the data bandwidth utilisation against packet size. The shape of the plot is governed by the packet overheads and switching latencies, which remain constant as the packet size increases, thus lessening their effect. The maximum level of data bandwidth is restricted ultimately by the data-link protocol overhead, which governs the token format and flow control mechanism. Figure 6-5 presents the physical addressed result for the NTR-FTM08 only, to aid clarity, as the difference in performance between the addressing modes is small, and physical addressing mode was the closest to the switching operation of the earlier devices.

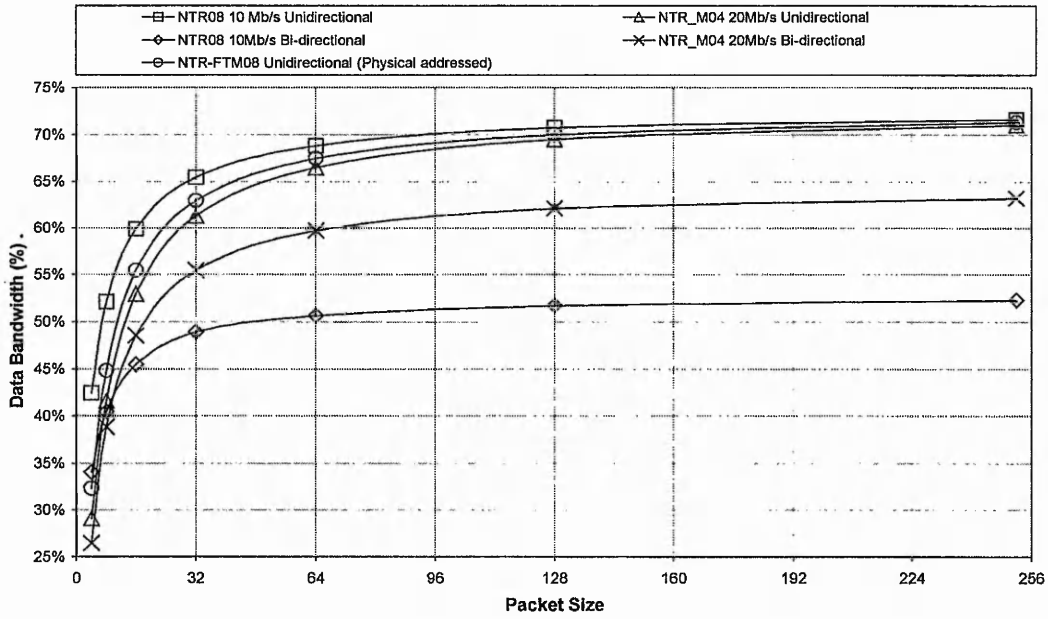



**Figure 6-5 : Simulation results for raw bandwidth comparison with the earlier devices of an unloaded network**

Figure 6-5 shows that the NTR-FTM08 operates marginally more efficiently than the NTR-M04 but less efficiently than the NTR08 at 10 Mb/s. This was due to the

differences in the switching latencies of the individual router-switches when compared to their link data rate. The lower core clock rate of the NTR-FTM08 penalised the switching efficiency of the device, but the improved pipelined architecture of the controller still provided a slight improvement on the earlier PLD-based NTR-M04. The alteration of the flow control mechanism for the NTR-M04, which increased the flow group to eight, provided a distinct improvement on the NTR08. Figure 6-5 does not present the results of a bi-directional transfer for the NTR-FTM08, as contention-free bi-directional transfers do not require any control signalling, and are therefore equal to unidirectional transfers. All the results for these tests are shown in tabular form in appendix C.

Although the unicast connection latencies for the three addressing modes of the NTR-FTM08 are very similar, the multicast latencies increase linearly with the number of outputs due to the sequential operation of connection allocation. Figure 6-6 shows the small linear decrease in effective bandwidth as the number of output increases. The link bandwidth is degraded by 2 % with a packet payload of four compared to a unicast logical connection and that of a multicast connection with seven outputs. This figure decreases to 0.2 % with a packet payload of two hundred and fifty-two.

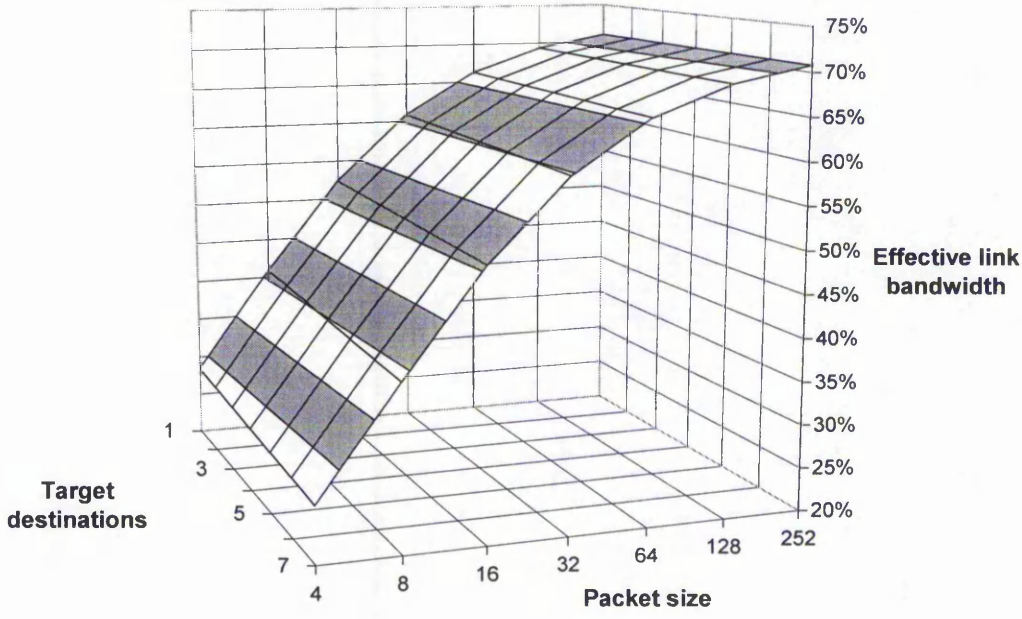



**Figure 6-6 : Unloaded network, raw bandwidth comparison of effective link bandwidth for multicast connections**

Only a small number of applications would use only a single router-switch. The effects of switching latencies and small packet sizes become even more prominent in

multiple router-switch networks. As the distance between source and destination increases, so does the overall packet latency, due to the switching latency incurred at each router-switch. This effect is shown clearly in the small packet sizes of Figure 6-7, Figure 6-8, and Figure 6-9.

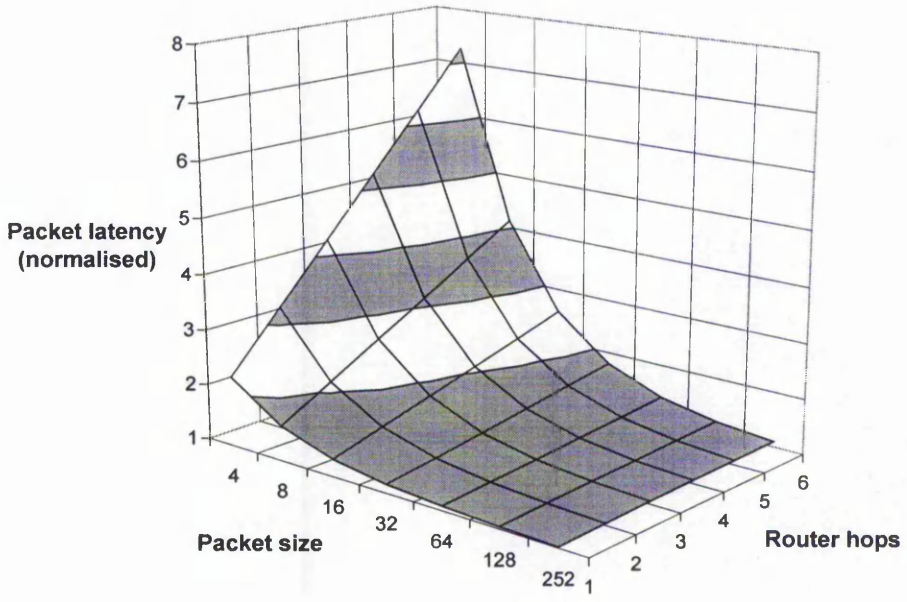



**Figure 6-7 : Packet latency verses router-switch hops and packet size for physical addressed packets**

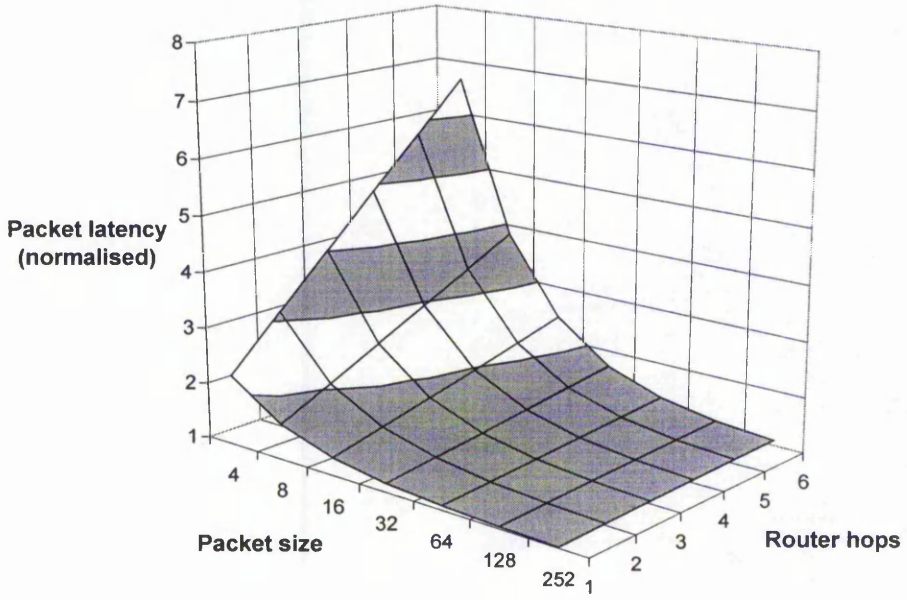



**Figure 6-8 : Packet latency verses router-switch hops and packet size for interval addressed packets**

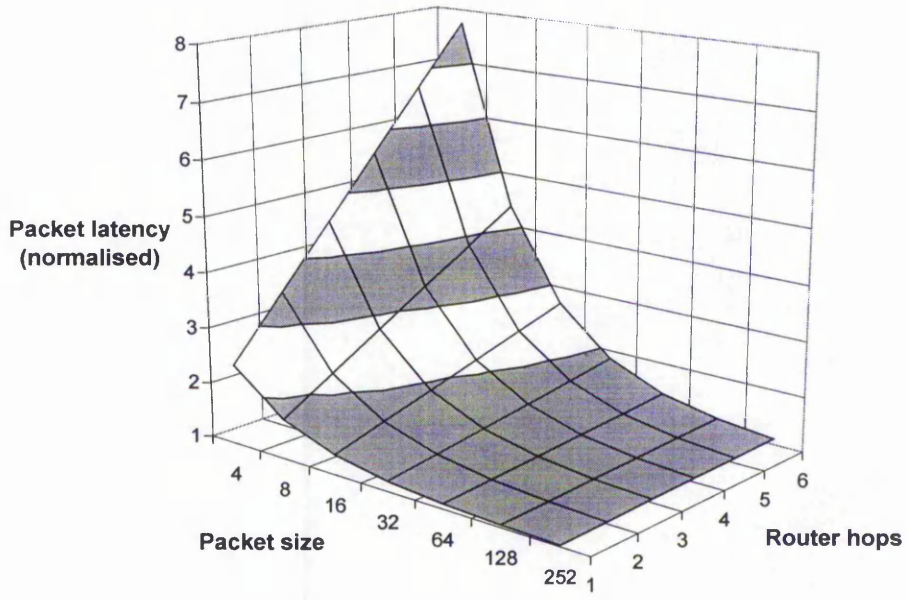



**Figure 6-9 : Packet latency verses router-switch hops and packet size for logical addressed packets**

The main aim of using configurable headers is to reduce the routing overheads for networks with multiple router-switches. However, these results show the logical addressed packets incur a marginal increase in latency over the physically addressed packets, which in turn incur a marginal increase in latency over the interval addressed packet. This is a feature of wormhole routing, which minimises the transmission latencies by effectively pipelining the switching of the packet across the network. This has the effect of obfuscating the routing overhead in physical addressed packets, as routing bytes are removed as the packet traverses the network. However, these results are misleading as they display little of the effect of the packet on the rest of the network. As wormhole routing allows a packet to connect across a router-switch before the entire packet has arrived, the tail of the packet may span one or more devices, which may result in other packets stalling until the resources are freed. Figure 6-10 shows the normalised time for which resources are held by a connection relative to its data packet size. As the figure shows, there is little or no difference between the addressing modes if a single header is considered. However, a single header for logical and interval addressed packets may be valid for many router-switches, where a single header for physical addressing would be only valid for a single router-switch. Figure 6-11 shows the linear increase of the time that resources are held as the number of router-switches in the transmission path increases.

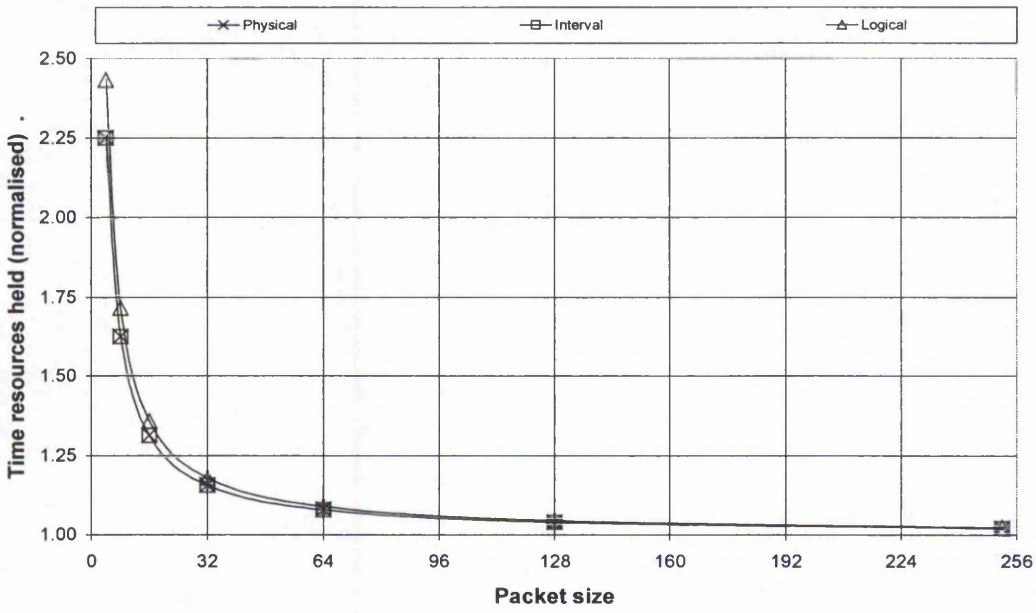



**Figure 6-10 : Time resources are held for a packet relative to the packet size for a single routing header for interval, logical and physical addressed packet in an unloaded network**

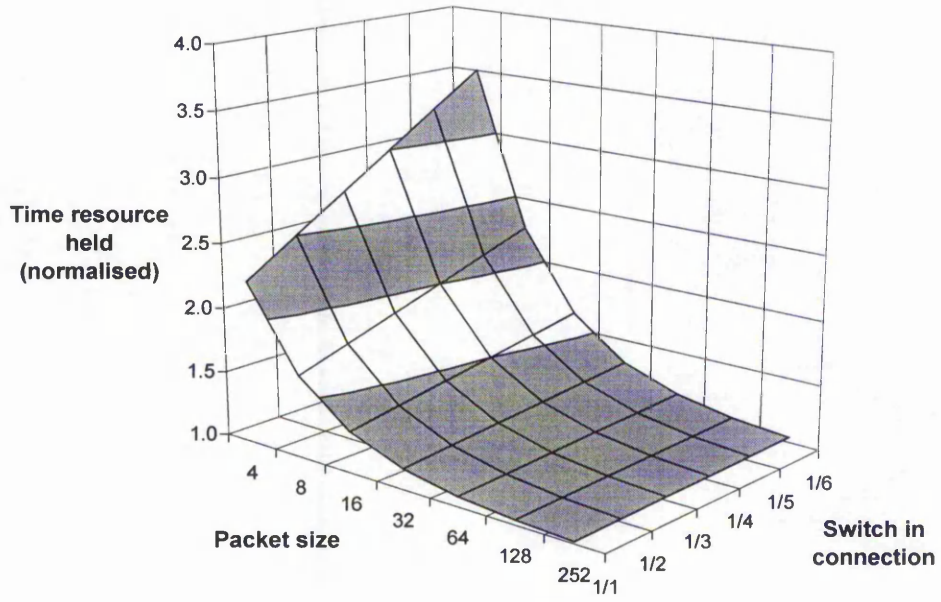



**Figure 6-11 : Time resources that are held for a physically addressed packet relative to the packet size for connections over 1 to 6 router-switches in an unloaded network**

**Network Under a Synthetic Load**

A number of performance simulations were carried out to investigate the behaviour of the device under a synthetic load. This allowed a further analysis of the differences between the three addressing methods under load conditions, which take into account the contention between packets within the network. A number of test vectors were generated, each of which injected packets into the network at varying load. Seven workloads were generated for each addressing type each with four different packet sizes. The workloads varied at intervals of 10% from 30% to 100%, with packet sizes of sixteen, thirty-two, sixty-four and one hundred and twenty-eight. Each packet was injected at an interval that was relative to the target workload and packet size. The interval was varied using a Poisson distribution, which ensured that 70% of the packets were within 30% of the target injection interval. Packet destinations were uniformly selected, but with a governing factor that a two consecutive packets only had a 30% chance of being sent to the same destination. The source code for the program that generated the test vectors can be found in appendix D. The test bench recorded the transmission and receipt times of the packets, which were then used for post-simulation analysis. The source code for the program that performed the post-simulation analysis can also be found in appendix D. Header stripping is mandatory for physical addressing, but not so for interval and logical. For these simulations, header stripping was omitted for the two configurable addressing modes.

Figure 6-12 and Figure 6-13 present the results of the analysis from the load simulations for all addressing methods for the tests with sixteen bytes per packet and one-hundred and twenty-eight bytes per packet. The full results for these simulations can be found in appendix C. Figure 6-12 shows the accepted load against the offered load, which clearly depicts the differences in the addressing algorithms with small packet sizes, due to the differences in switching latencies whereas differences are less significant with large packet sizes. Figure 6-13 also displays these effects in the differences of the switching latencies. The distances between the plots of the smaller packet sizes display the significance of the switching latency in comparison to the transmission time. The effect of the switching latency lessens as the packet size increases as displayed in the plots for the larger packet sizes. The switching latencies across the device creates a two percent difference in bandwidth between the logical and interval addressing methods, and the byte stripping creates a two percent difference between the physical and interval addressed packets. If saturation points of the NTR-FTM08 and the STOP/GO modified version of the NTR-M04 from the results in section 6.1 are compared, the NTR-FTM08 appears to

8 % lower than the NTR-M04. Although some of the difference in bandwidth could be due to a disparity in the test vector traffic patterns, it is conjectured that it is primarily due to the bottleneck of the servicing mechanism. The NTR-M04 possessed four links, which could concurrently generate connection-requests, whereas the NTR-FTM08 had to deal with eight. As the servicing mechanism in both devices was effectively sequential, concurrent connection-request had to time-share the functionality. The only solution to this problem is to implement more concurrency within the servicing algorithms as the router-switch scales.

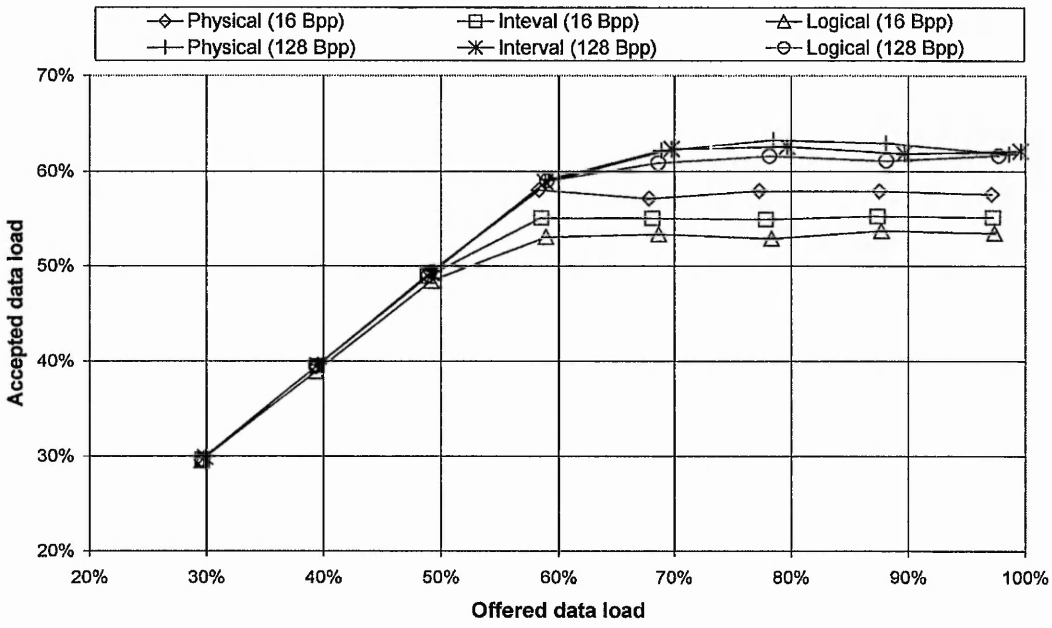



**Figure 6-12 : Offered data load verses accepted data load over a range of workloads for a single router-switch for 16 and 128 byte packets**

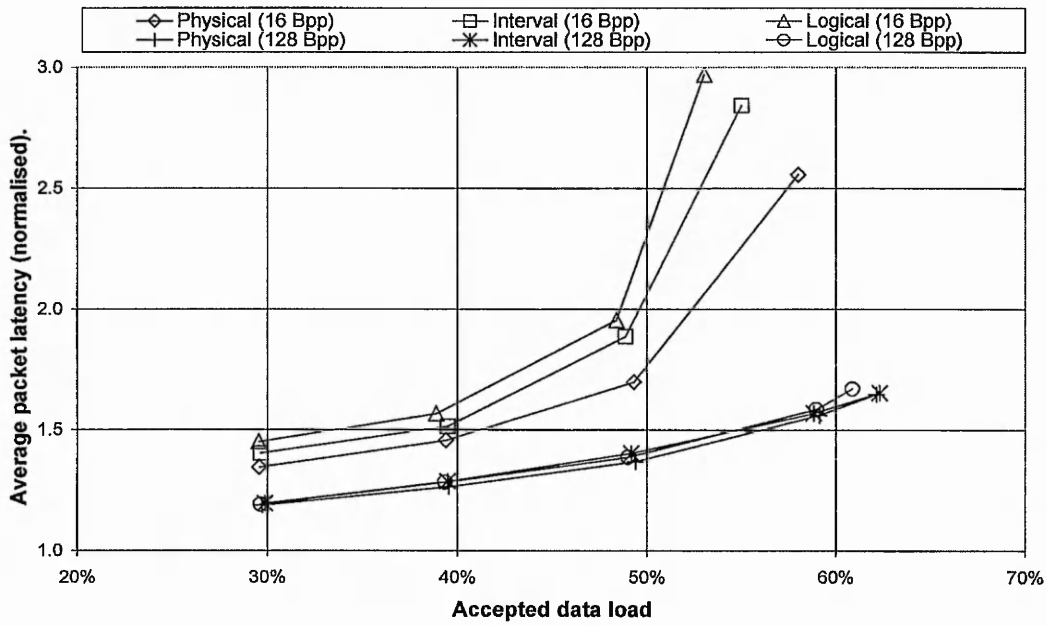



**Figure 6-13 : Accepted data load verses average packet time over a range of workloads for a single router-switch for 16 and 128 byte packets**

### 6.2.2 Synthesis

The design of the router-switch was targeted to an Altera EPF10K130EQC240-2 device. This device was selected due to its amount of logic resources and embedded memory, and based on the research groups' familiarity with the FLEX 10K product family [99]. The FLEX 10K family is based on a SRAM technology, which requires the device to be programmed following power up. This suits a prototyping design flow as the reprogrammability allows hardware verification to supplant timing-based simulation work, which greatly reduces the design cycle time. Figure 6-14 is a photograph of the device and driver board that was developed to allow for hardware performance and operational verification.

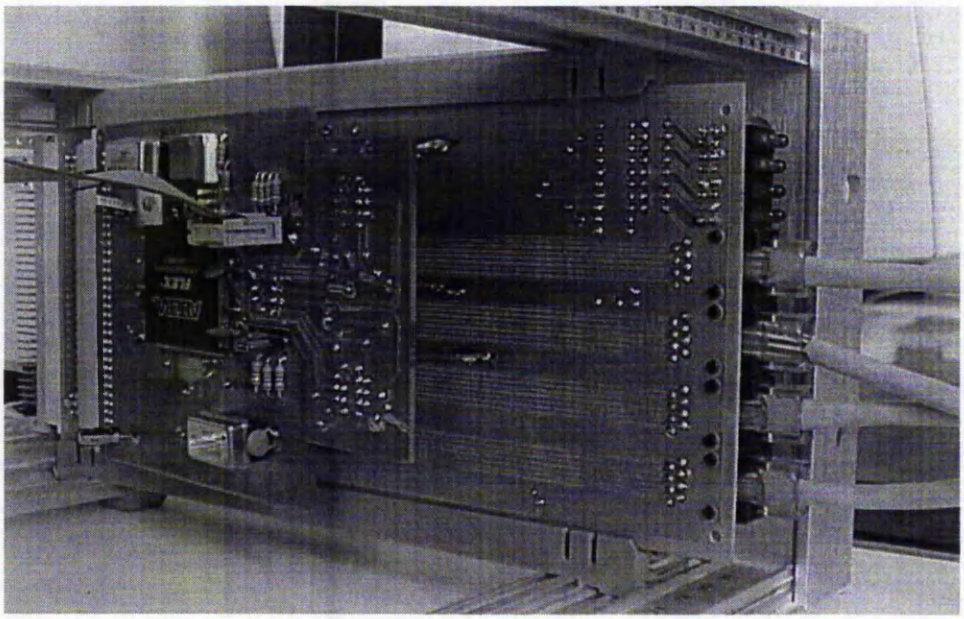


**Figure 6-14 : A photograph of the NTR-FTM08 prototype device and driver board**

One of the device design aims was improve data bandwidth figures. The NTR-FTM08 was designed to operate with two clock domains, which allowed the link interface to be optimised to operate at the high rate, whilst the timing restraints on the core of the device could be less stringent. As Table 6-1 shows, the final synthesis provided a device that could operate at up to 51 MHz and 33 MHz for the sampling and core clocks respectively. These figures produced a link bit rate of 34 Mb/s. As the table shows, each part of the design can operate at higher operating frequencies than the overall device; for example, all modules for the sampling clock will operate at a frequency of at least 60 MHz. However, as part of the synthesis stage of the development flow, the design as a whole is routed and placed, which results in each part of the design competing for resources. If the design requirements are too great, the limited signal routing resources on the programmable architecture can result in a reduction in operating performance. For this reason, most designs are recommended to use less than 80% of the device. The NTR-FTM08 design has not been fully optimised, and with the device utilisation at 84%, improvements could be made on size and speed of the device.

**Table 6-1 : Synthesis results for the NTR-FTM08 router-switch**

| Module Name | Module Size (in logic cells) | Sampling Clock (MHz) | Core Clock (MHz) |
|---|---|---|---|
| **Link Unit** | **429** | **61.72** | **45.24** |
| Receiver Controller | 51 | N/A | 84.03 |
| Interval Decoder | 130 | N/A | 63.69 |
| Transmitter Buffer | 43 | N/A | 75.75 |
| Buffered Link | 221 | 60.97 | 84.74 |
| Receiver Block | 43 | 94.33 | N/A |
| Transmitter Block | 45 | 60.97 | N/A |
| Synchroniser Block | 42 | 200 | 200 |
| **Controller** | **1729** | **N/A** | **36.90** |
| Link Allocater | 585 | N/A | 42.73 |
| Configuration Port | 349 | N/A | 46.66 |
| Logical Address Decoder | 167 | N/A | 44.44 |
| Logical Connection Register | 215 | N/A | 117.11 |
| Connection-request Queue | 435 | N/A | 43.29 |
| Queue FIFO | 19 | N/A | 142.85 |
| **Exchange** | **506** | **N/A** | **57.47** |
| Exchange Unit | 21 | N/A | 163.00 |
| **Router-switch** | **5649** | **51.54** | **33.44** |

### 6.2.3 Design Hardware Tests

#### 6.2.3.1 Verification

A number of hardware tests were devised to test the operation of the router-switch for extended periods. Figure 6-15 shows a block diagram that depicts the hardware that was used for all of the verification tests. Each test repeated a sequence of operations, which were designed to test a single primary function of the switch continually in a repeating loop until an error occurred. Errors included the arrival of incorrect data at the receiving node, link stability failure, and test inactivity. All the tests have been successfully executed over extended periods with the sampling and core clocks operating at 48 MHz and 30 MHz respectively. These clocking frequencies provided a 32 Mb/s raw bit rate. The areas that each hardware test verified now will be summarised. However, further description of each test is provided in appendix E.

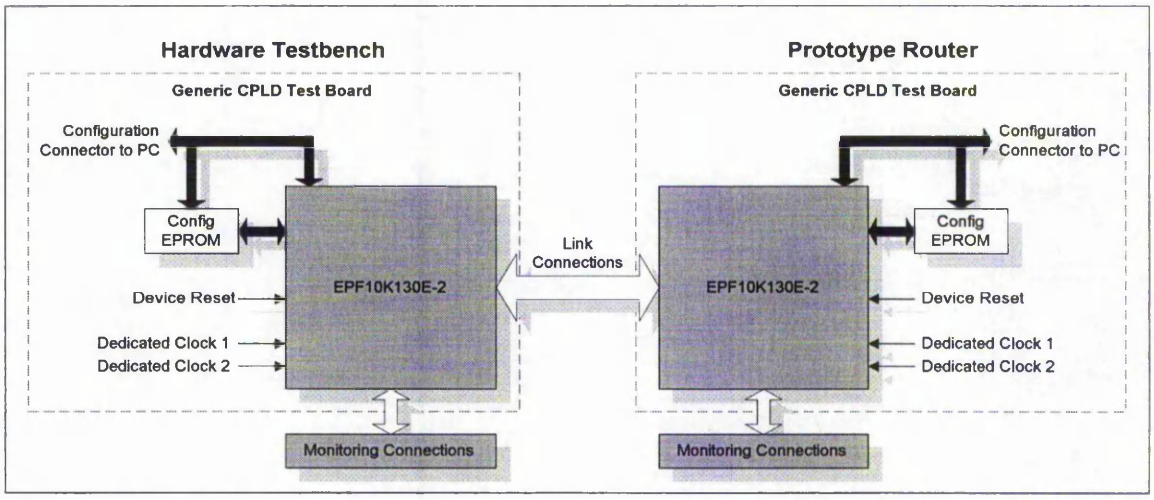



**Figure 6-15 : Block diagram of the hardware used for verification tests**

**Sampling test**

The sampling test used eight buffered link modules to verify the sampling, link status and flow-control mechanisms.

**HW test 1 - Physical addressed test**

The physical addressed test was performed on an unconfigured NTR-FTM08. Thus, all interval or logical addressed headers were illegal, no links were grouped, and no links were dormant. The test verified the operation of sampling, flow control, service requesting, handling of contentious queuing requests, connection allocation, disconnection, and illegal packet spillage.

**HW test 2 - Grouped physical addressed test**

This test was the first to include configuration, which required a configuration packet to be sent to the router-switch before the start of the test. Thus, in addition to the features tested by HW test 1, this test verified the operation of the link allocation with grouped outputs under contentious conditions and the basic operation of the configuration port. The functionality verified within the configuration port were connection, disconnection, operation of the data and address register and the write command for the grouping registers.

**HW test 3 - Logical addressed test (unicast)**

HW test 3 verified the use of each logical address header, which proved the logical decoding, queuing and allocation mechanisms under contentious conditions. Further functionality of the configuration port was verified, including the configuration of the logical address look-up table.

**HW test 4 - Interval addressed test**

This test repeated the verification of the standard operation of packet transfer as described with the HW test 1, but included the verification of interval decoding, and the relevant features of the configuration port. Interval addressed packets employ the same queuing and allocation stages as physical addressed packets, although header decoding differs.

**HW test 5 - Broadcast test**

The broadcast test verified the same features as the HW test 3, but with multiple destinations. This ensured that all required destinations were allocated and the packet was not sent back to the originating link.

**HW test 6 - Disconnection time-out test**

HW test 6 was the first of three tests that verified an aspect of the fault tolerance mechanism. This test verified that the disconnection detection mechanism and recovery procedure operated successfully and switching functions were not compromised.

**HW test 7 - Overflow reset test**

This test was only performed on the buffered link module, and validated the overflow detection mechanism and correct operation of the subsequent recovery procedure. Overflow was achieved by modifying the receiver FIFO 'almost-full' flags such that the link delay forced a failure in the flow control mechanism.

**HW test 8 - Link Dormancy test**

Dormancy allowed the configured links to return to the 'asleep' state following a period of inactivity. This test verified that this procedure operated successfully, and that links could be 'woken' when required. This test also verified the operation of parts of the configuration port, link allocation and buffered link.

#### 6.2.3.2 Device Performance

To provide a comparison with the earlier simulation based results, unloaded bandwidth measurements were carried out on the physical device. The router-switch operated with a 48 MHz sampling clock, and a 30 MHz core clock. The test measurements were taken at the interface of the NTR-FTM08 using a logic analyser, where the figures were taken from the average of ten packets for each packet size. Figure 6-16 and Figure 6-17 show traces captured from the logic analyser of the eight-byte packets for all three addressing modes. Figure 6-16 shows the test with unidirectional data flow and Figure 6-17 shows the test with bi-directional data flow.

The waveforms in Figure 6-16 and Figure 6-17 are labelled by link number and mode, that is either receiver or transmitter. For example, the receiver and transmitter of link 0 are labelled R0 and T0 respectively. Links that are used to transfer a test packet are further labelled with three letters. The first letter denotes the type of addressing; P – physical, I – interval, and L – logical. The second two letters denote the direction; that is RX – arriving and TX – departing. For example, the links for the physical addressed packet are marked as PRX and PTX for the arriving and departing link respectively.

All measurements were recorded, and the effective percentage of the raw bit rate was calculated. Figure 6-18 shows both the simulated and measured results from the device. The hardware results show an improvement on the simulation results, but the overall trend is identical. The differences arose due to an improved switching latency because of the clock frequencies used in the hardware test. As the design was targeted to operate with a core frequency of half the sampling clock, the simulations were implemented as such. As stated above, the sampling and core clocks operated at 48 MHz and 30 MHz respectively. These result in the core operating $5/8^{ths}$ of the sampling rate, thus providing the better results shown. A full tabular form of the results can be found in appendix C.

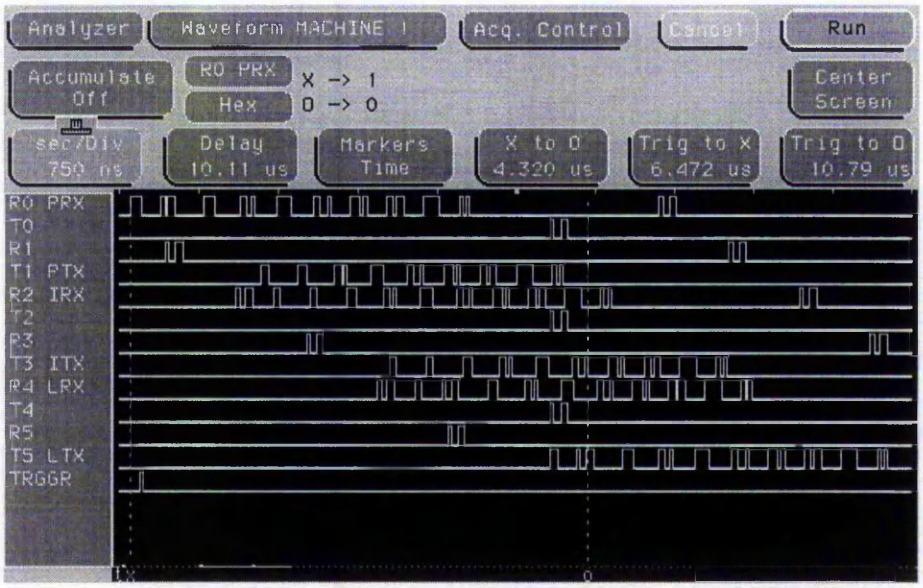



**Figure 6-16 : A captured trace from the unloaded router-switch tests for all three addressing modes for unidirectional data flow**

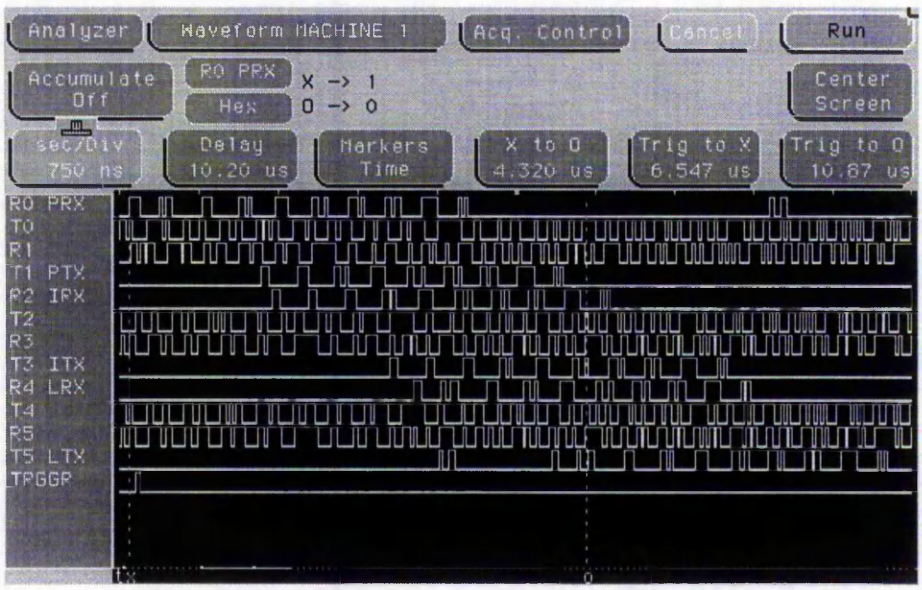



**Figure 6-17 : A captured trace from the unloaded router-switch tests for all three addressing modes for bi-directional data flow**

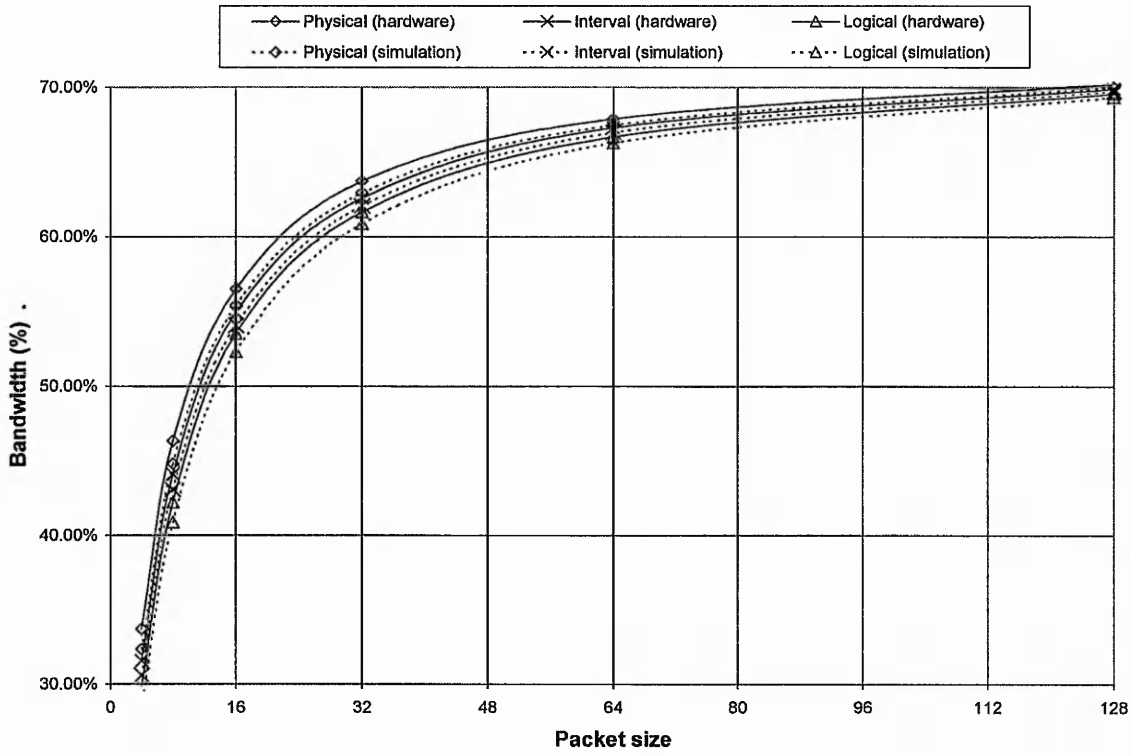



**Figure 6-18 : Hardware results for raw bandwidth comparison with the earlier devices of an unloaded network**

## 6.3 Stage Two Enhancements

### 6.3.1 Stage Two Simulation

The deadlock detection mechanism, which was devised as part of the second stage of enhancements to the skeletal NTR-FTM08, was implemented in behavioural level VHDL to enable concept validation. A simple four router-switch network was implemented in a mesh format with sixteen communicating nodes, as depicted in Figure 6-19. A deadlock prone router-switch configuration was used, which guaranteed the occurrence of a single cycle deadlock in a certain position of the network. The test-bench was designed to monitor for the deadlock cycle, so network status and the operation of the mechanism could be easily documented and verified. As documented by Pinkston [66], the probability of deadlock increases with the saturation level of the network. To create 'worse case' conditions, the network configuration created network hot spots on the path of the deadlock cycle. Additionally, to increase the amount of stalling, each communicating entity stalled the egress link shortly after receiving the start of each packet, which was equivalent to approximately five tokens.

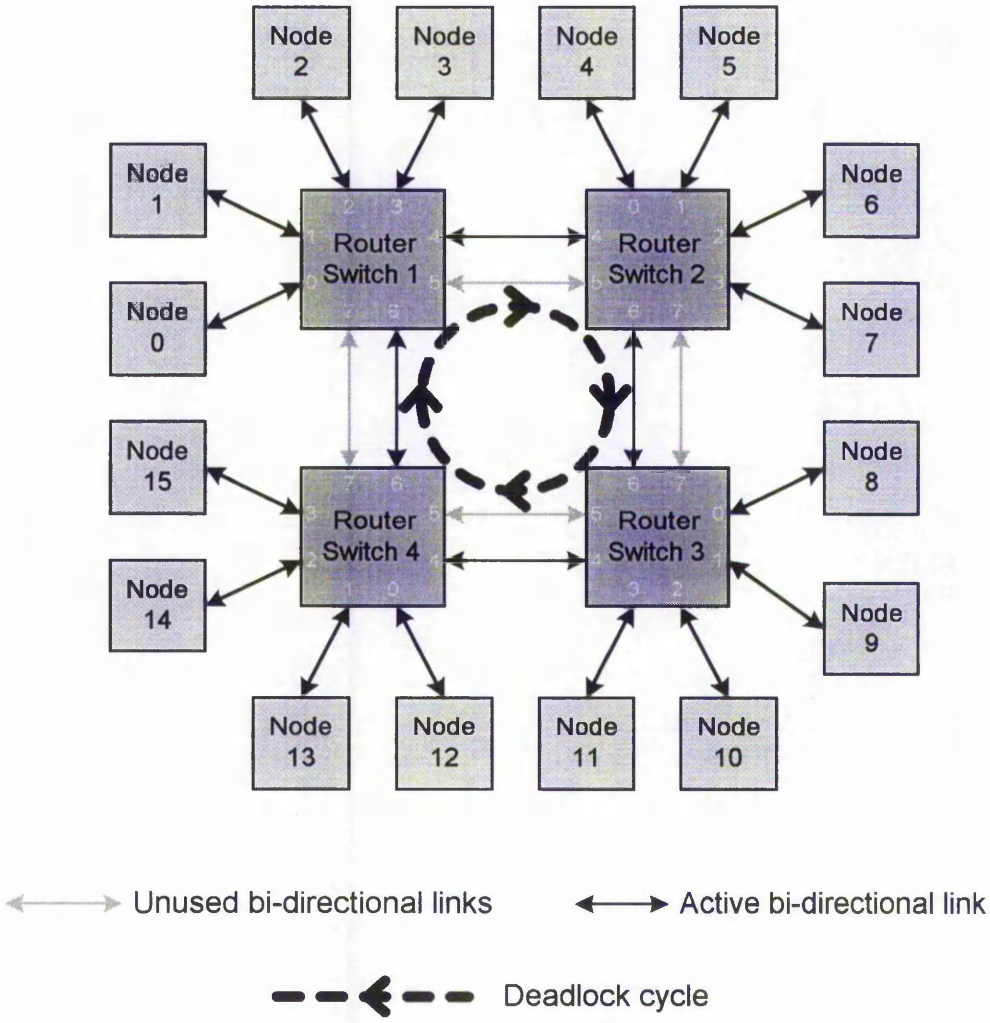



**Figure 6-19 : Network structure used for deadlock testing**

The network was injected with packets as defined by statically generated test vectors, which varied with workload and packet size. The test vectors were generated using the same software that was used for the performance analysis (as described in section 6.2.1.2); the listing of which can be found in appendix D. Each set of test vectors contained 8000 packets of either 16, 32, 64 or 128 bytes. Interval addressing was used to configure the network to be prone to the deadlock cycle shown Figure 6-19. The configuration information for the network can be found in appendix F. The NTR-FTM08 used slightly different receiver FIFO settings than the performance tests, which set the STOP and GO values at 24 and 8 respectively.

The test bench generated monitoring information, on which post-simulation analysis was performed. Software was written that parsed the monitoring information, and

collated all the details relevant to the deadlock conditions. The source code for the software is listed in appendix F. The collated results provided information based on number of deadlock cycles, number of non-deadlock cycles, and number of removed packets due to the recovery mechanism. The complete results from this analysis in tabular form can be found in appendix F. As the meaning of the results cannot be represented easily in graphical form, the following salient points can be produced from their analysis:

- all deadlocks were successfully detected;
- no false detections were recorded, even in the presence of non-deadlock cycles;
- the packet size or workload did not affect the rate of false detections, but had an affect on number of packets marked for removal.

As the time-out was used to provide a period of grace for the return of a 'path clear' control token, the relatively short period, supplied a quick detection and recovery mechanism irrespective of workload and packet sizes. The results show that an average of 17.6% of the packets was removed from the network. While this figure may appear high, the workload configuration and network topology was designed to produce a highly saturated and congested network. This increased the chance of deadlock and created worse case conditions. Proof of extreme saturation was evident in the number of deadlock cycles in each test. Following Pinkston [66], the results from this test analysis demonstrated saturation since they showed no significant increase in the number of deadlocks with the increasing injected workload. However, the results showed that a lower number of deadlock cycles formed with the smaller packet sizes. This finding also complies with Pinkston's analysis [66], where the amount of buffering and the ratio between buffering and packet sizes affect deadlock probability. The number of packets that were removed with each deadlock cycle decreased as packet sizes increased, which related to the way deadlocks were formed in these tests. The large packets would cause links to be stalled for longer, which therefore gave the mechanism more chance to validate waiting packets, thus a higher probability that a single packet would be highlighted as root.

# 7 Discussion, Conclusions and Further Work

## 7.1 Discussion

Much of recent research regarding switched networks has been based around fault tolerance and quality of service. The work on fault tolerance for parallel processing systems has concentrated on network adaptivity to bypass permanently disabled links in regular topologies. Most of such research has ignored the recovery aspect of the network and focused on ensuring the network can operate in a degraded state. Little or nothing is said about the unreachable devices in the network. In contrast, fault tolerance in distributed processing or LAN replacement systems, which are predominantly irregular topology systems, have focused on detection and recovery in addition to the maintenance of network operability. The research detailed in this thesis has concentrated on fault handling procedures, while supplying some fairness in packet servicing, but without additional features for providing bandwidth guarantees.

The remaining part of this section discusses points regarding the fault tolerance features investigated by this work, followed by some more general points regarding switched communications in small area networks.

### 7.1.1 Fault tolerant features

This research is aimed at supplying a network into which the user can inject a packet with the confidence that it will not critically hinder the delivery of any other packet. To achieve this aim two areas were investigated, firstly critical fault detection and recovery, and secondly deadlock handling procedures.

#### 7.1.1.1 Link Fault Detection and Recovery

**Distributed Architecture**

The NTR-FTM08 has been designed to operate as a building block for an irregular-topology communication network of a distributed or embedded processing system. A primary aim of the design was to supply maximum network availability (not to supply an error free medium). This was achieved by implementation of distributed critical link failure detection and recovery procedures that localised the corruption caused by a fault. This method has been carried out in many other systems over the last decade, such as Autonet [40], STC104 [78], Myrinet [88] and EtheReal [100]. A distributed implementation is a primary feature possessed by all of these fault tolerant networks. A

distributed model allows the fault tolerant mechanism to scale linearly with the network, although the recovery latencies of some systems increase with system size. In contrast, the NTR08, which relied on additional resources for monitoring and intervention, required a centralised fault handling mechanism, which became more complex and increased resource requirements above a linear rate as the network scaled.

The majority of the distributed fault handling systems based their recovery on network reconfiguration, which reduced the responsiveness of the network to failure and did not guarantee an improvement in the network connection graph. However, automatic reconfiguration provided a flexible method for a dynamic network, where devices could be added or removed from the system with no adverse effect. In contrast, the technique used by the STC104 and the NTR-FTM08 negated the need for reconfiguration by allowing redundancy to be built in at design time, although the features were a configurable option in the STC104. Both of these devices detected and isolated failures through features that identified incorrect link operation, and removed or truncated corrupt packets. Unrecoverable failures caused links to become disabled and only could be reused if the link was successfully reinitialised. Subsequent packets were automatically removed if the packet was undeliverable due to the failure.

Although the basic operation of the recovery mechanism of the NTR-FTM08 and STC104 resulted in a much-improved response to failure, without some form of path redundancy, areas of the network could become disjoint. The implementation of group adaptive routing provided the required redundancy, which improved network throughput in error-free conditions and allowed a rapid response to network failure. This quick response to failure minimised traffic corruption while the network still offered a degraded but operational level of service without the need to wait for reconfiguration. As such, this technique better suited static applications, such as embedded or distributed systems.

**Modified Protocol**

The similarities of the fault handling procedures in the NTR-FTM08 and the STC104 were separated by differing protocol choices. The NTR-FTM08 improved some aspects of link operation. The first improvement was the additional packet delimiter token. The inclusion of three packet delimiter tokens supported an identifiable message format that simplified requirements of the interface for message packetisation and corrupt packet detection. Additionally, the data recovery technique of over-sampling allowed the link to

be asynchronous, which permitted the link dormancy feature of the NTR-FTM08. Link dormancy was intended for use in distributed control systems where communication would be infrequent. Thus, to minimise power requirements and to minimise signalling activity, allowing an idle link to return to the dormant state seemed the most suitable solution. This feature also required that each connection-request was given a suitable chance to form a connection if the desired link was initially unavailable. While this could reduce the network throughput, it was greatly advantageous for networks with low traffic volumes. Finally, the use of the STOP/GO flow control mechanism improved the bi-directional data transfers in comparison to the credit-based mechanism of the previous devices. In addition, the re-use of the flow control tokens as idle token allows reaffirmation of the flow control status. This, in conjunction with the receiver overflow recovery, provided guaranteed recovery of flow control failure.

**Reliance on Time-outs**

All of the fault tolerant features that have been implemented in the NTR-FTM08 have some dependence on time. This can introduce difficulties in the mechanism, as time-out periods must be long enough to tolerate network latencies, but not too long to make the mechanisms unresponsive. There were two intervals chosen from which all other features were derived, namely 'heartbeat' and 'check-pulse'. The heartbeat signal occurred once every 256 sample clock cycles, which equated to an approximate transmission time of 15.5 tokens. The check-pulse signal occurred at half the rate of the heartbeat signal, at 512 sample clock cycles. The detection of disconnection was based on this longer interval, which equated to an approximate transmission time of 31 tokens. As the performance results showed, average packet sizes of 32 tokens provided a reasonable amount of efficiency, thus the time-out was set approximately to this figure. However, there is a distinct danger that an invalid link could be used by multiple small packets before the disconnection is detected. Fortunately, the concept of bad packet removal or truncation requires further support in the upper layers of the protocol, which covers the shortcomings of the disconnection detection resolution. The design centralised the timer from which all parts of the detection and recovery mechanism operated. While this minimised logic requirements, the lockstep operation of each link was detrimental to the device as a whole, as it caused large amounts of switching at periodic intervals. Thus, as the operating frequency increased, electrical noise would also increase, which could degrade reliability.

**Redundancy**

The concept of fault tolerance can be vague. The technique here was designed to use redundancy to improve the network availability of resultant systems. Thus, if one or more point-to-point connections failed, the extra lines in the circuit would supply alternate paths. Although this method was dependent on the topology used and the network configuration, some statements can be made about the abilities of the device.

The NTR-FTM08 was an eight-port router-switch and, in theory, each port could be grouped with one another. This is impractical as this would mean that they would all be connected to the same location. Similarly, two groups of four would also be impractical as it could be replaced by four wires. Thus, to make the design features useful, the device must operate with three or more groups. With the eight-port device the maximum number of groups are four; that is, four groups of two. If all configurations are considered that ensure that each direction possesses some redundancy; that is, where each port is part of a group, there are only three possibilities; they are:

{2,2,2,2}

{3,3,2}

{4,2,2}

Obviously, other configurations exist where some links are not grouped but these are discounted, as they will not supply resilience against faults for those routes. While route redundancy may not always be required, this shows that to construct networks of reasonable sizes, the eight port router-switch with this form of grouping seems inadequate.

Two solutions are evident; the first is to increase the number of ports per router-switch. This would increase the number of possible configurations and therefore improve the scope of topologies. The second solution is to modify how redundancy is implemented into the network. A limiting factor of the current implementation is the basic form of group adaptive routing, which minimises resource requirements by offering alternative paths by associated outputs and not by header information.

With the improvement of technology, more flexible route decoding methods have evolved, which have provided more flexibility on the way routing headers operate. Many networks possess a large number of valid routes between two points. Thus, if technique

could be devised that operated individually on each header by considering the arrival port and the target address, it is possible that more route flexibility could be found. This has two obvious problems, the first being implementation costs. The more complex the solution the more expensive it will be to implement. Deadlock is the second consideration, as any solution should ensure that the increase in network availability does not detrimentally affect the network procedures for avoiding, preventing or recovering from deadlock.

#### 7.1.1.2 Deadlock handling procedures

The review of deadlock handling procedures showed the majority of switched point-to-point systems rely on avoidance schemes. Deadlock avoidance schemes restrict the routes that packets may take, this ensures that deadlock-cycles cannot form. However, the routing restrictions limit network utilisation, which provoked research in less restrictive algorithms or removal of all restrictions while relying on deadlock detection and recovery mechanisms. The investigative work on deadlock detection and recovery included in this research was based on providing a resilient network, not to improve adaptivity. It is obvious that to reduce the requirements of user intervention for a distributed or embedded system, some form of deadlock detection and recovery would be valuable, even for systems that implement an avoidance mechanism. The major drawback with basic time-out solutions, is the low level of responsiveness inherent in the implementation that is necessary to minimise false detection. As discussed earlier, some work has attempted to alleviate this problem by increasing the complexity of the detection mechanism [57]. While these results showed improvements on basic time-outs, the operations of the time-out still the determining factor on the length of time for which a link was blocked. Thus, packet size and routing algorithms still had to be considered for the selection of a suitable time-out period. In contrast, the work described here targeted a system that was designed to operate with unrestricted packet sizes, thus a mechanism that determined the time-out period from the network diameter was more suitable. To achieve this, control signalling was used to verify network conditions. The fact that the control signalling carried no detailed information meant that no buffering was required to forward the query. However, the localised mechanism could not identify a deadlock probe token that it had generated, from that of any other part of the mechanism. Therefore, the mechanism did not remove the chance of multiple packet removal for a single deadlock cycle. As the control signalling was transparent to the flow control mechanism, the control signalling could

travel on stalled links and possessed a higher priority than data traffic, providing an optimum propagation time for detection resolution. The simulation results of section 6.3 showed that the relationship between packet size and network receiver buffering affected the number of packets removed per cycle, but did not affect false deadlock detection. Hence, as the size of a network increases and the size of possible deadlock cycles increases, the incremental amount of buffering per link would also increase the number of removed packets. Although increased buffering should decrease the probability of deadlock, it means more packets may be present in the network, which can result in cascaded deadlock cycles. That is, after a deadlock cycle is cleared, another may form immediately due to the network saturation. However, in the preliminary simulations, even with the packet sizes at half the size of the receiver FIFO buffer, the average loss per cycle was less than 1.3 packets. If such figures could be maintained in larger networks, this packet loss ratio would be a good compromise to obtain a quick resolution to a critical network state.

Other areas of the simulation results were also very promising, as all deadlocks were detected with zero false detection. This included the correct validation of non-deadlock cycles. It is difficult to compare this mechanism with the systems from which it evolved, as previously published results were based on more complex regular networks that operated with various traffic patterns. An aim of the previous work was to show how infrequently deadlocks occurred within an unrestricted network and how a deadlock recovery could improve on avoidance techniques. In contrast, the tests performed within this research, as detailed in section 6.3, were designed to create an abnormally high number of deadlocks to prove the mechanism under extreme conditions. Unlike the work described in this thesis, much of the previously published results do not provide a percentage of false detection.

Similar to the earlier systems, the response to deadlock cycles within the NTR-FTM08 solution was a short time-out. However, this system used the time-out to wait for validation of an exit path. This was in contrast to the earlier systems, which waited for the exit to clear. Due to the control signalling validation, a basic time-out system would require time-outs of much greater values to attain the zero false detection rate achieved by this mechanism.

The NTR-FTM08 mechanism needs no prior knowledge about network layout and the mechanism should be effective irrespective of the traffic pattern, as the control

signalling follows the path of stalled resources. However, if the mechanism was implemented in an irregular topology, the network may or may not possess possible connections that could form cyclic paths. This oblivious mechanism treats each link with a standard procedure, which could result in loss of bandwidth due to unnecessary control signalling. Even in networks with possible deadlock cycles, the 'path clear' and 'data movement' tokens are sent on the return path of the stalled resources. These return paths may not be stalled, hence, the control tokens may utilise available data bandwidth. Logically, as the network approaches saturation, more deadlock queries would be made. Thus, the operation of the mechanism could produce a lower saturation point due to the percentage of data bandwidth used for control signalling. However, the strict creation rules of the tokens maintained a low volume of control signalling, which minimised these effects. A more complete analysis would be required to verify how greatly the utilisation of the validation signalling degrades the operation of the network.

Although the deadlock detection mechanism operated well under simulation conditions, meeting the requirements, it is unfortunate that the control mechanism would require a significant amount of resource if synthesised for implementation. As previously stated, for an effective fault tolerant switch, more than eight ports would be preferred, and this would worsen resource requirements as the deadlock detection mechanism would scale greater than linearly. Extra ports would also increase the requirements for the support of group adaptive routing and multicast connections. This is because the mechanism would be required to monitor all possible permutations of connections and routes to correctly query and validate possible deadlock paths. These factors, in addition to the fact that this mechanism was supposed to be a secondary fail-safe system, make it impracticable for the networks targeted for this work. A back-up system should not utilise such significant resources, however, it could be suitable for use in the parallel 4+1 networks that use fully adaptive routing algorithms and a detection mechanism as the primary defence against deadlock. As the primary mechanism, the resource requirement would be acceptable and the lower number of ports would simplify the implementation, although virtual channels would have to be supported.

### 7.1.2 Basic Routing Features

#### 7.1.2.1 Flow Control Mechanisms

The earlier research, from which this work developed, was designed to remedy communication bottlenecks of the first generation transputers. The OS-link used a credit based flow control mechanism with a single token flow group as a compromise between buffering requirements and bandwidth, as embedded memory in the ASIC technology of the time was very expensive. The modifications of the transputer communication model, which allowed the inclusion of a router-switch, altered the types of traffic that utilised the OS-links. Although the volume of traffic and chance of bi-directional transfer increased, and therefore reduced data bandwidth due to the high level of flow control signalling, the increased connectivity of the network lowered software overheads and improved system performance in most cases. The second-generation transputers recognised the communication problems and altered the communication model to operate with virtual connections. As this modified connection model increased communication traffic on the links and increased the bit rate, the single token flow group was increased to an eight token flow group. While this improved the overall transfer performance of the link, the larger flow group demanded larger amounts of buffering.

As the work of this research group moved away from the transputer family, restrictions on the communication model were lifted. Although the communication model was very attractive for embedded and distributed applications, the flow control mechanism was one area that possessed some redundant features. The simulation results documented in section 6.1 showed the performance was increased by switching from a credit-based flow control mechanism to a permission-based STOP/GO system. Additionally, as no fault detection mechanism was based on the information provided by the credit-based flow control systems, there was no basis for maintaining the system. The decision to change to the new flow control mechanism was supported further by the improved fault tolerant capabilities. For example, the worse case failure scenario for a STOP/GO system would be buffer overrun, which can be detected easily, and immediately recovered. As the flow control information simply implies flow status, the mechanism can be used as reaffirmation of the flow control state. In contrast, the worse case failure scenario for a credit-based system would be the loss of credit, which could result in an indefinite stall. Solutions for this problem can be produced, but at the cost of increased complexity and significant robustness to remove all chances of false operation.

#### 7.1.2.2 Target Technology

As the research moved away from the transputer families, a change of target technology to PLDs was made. Although memory is still a relatively expensive commodity in semi-custom ASIC technology, its integration has become prolific as application devices become more complex, especially in communication applications. Many PLD families include large amounts of embedded memory to target communication applications. The increasing amounts of memory within these devices has had a significant positive effect on the use of PLDs in many systems, especially in prototype systems. In this research, PLDs have made buffering requirements a minor concern. This is evident if the NTR-M04 and NTR-FTM08 devices are compared. The skeletal router-switch of the NTR-FTM08 utilised approximately the same logic resources as the NTR-M04, which is due to the replacement of buffering logic by embedded memory. The amount of embedded memory within PLDs continues to rise with the latest families, this will certainly introduce new possibilities in future switched network research.

#### 7.1.2.3 Connection servicing

Connection servicing is one of the most critical operations of a router-switch, as it is the major influence on the latencies incurred by the packet. The comparable differences between the three NTR devices have shown how switching latencies can affect the performance of a system, and how a sequential mechanism can form a bottleneck, which can degrade the device throughput. Servicing includes destination decoding and allocation, both of which can vary greatly between implementations.

The NTR-FTM08 servicing differed from the earlier devices, as three addressing modes were supported. While the support of all three modes would not be useful in a practical system, the implementation of it in this research allowed a brief analysis of absolute and configurable schemes. The results show that configurable systems can be used to improve system efficiency with small packet sizes or in larger systems. However, absolute systems, in conjunction with wormhole routing, can be equally effective in small networks or with large packet sizes. From this finding, the choice of utilising a configurable addressing mode would seem to be the better selection, as the scope of application networks is much greater. However, the inclusion of an absolute addressing mode would be advisable for implementation of a reliable default routing system. Two obvious reasons for this, highlighted by this research, are the configuration and mapping of the network. The NTR-M04 supported just logical addressing, but the reset state emulated

physical addressing to supply a functional network after reset. However, following configuration this default mode would be overwritten, thus any problems could make the network unusable. In such cases, using the data network for configuration becomes a problem and to recover from any configuration problems a system wide reset would be required. Network mapping was used in Myrinet, which implemented relative addressing. This scheme allowed a mapping packet to be sent into the network that could help construct the connection model dynamically, irrespective of the connection topology.

As the NTR-FTM08 implemented two configurable modes, the features of these modes could be analysed further. Interval address decoding was a parallel implementation, which unified the servicing of physical and interval addressed packets for connection-request queuing. In contrast, the logical address decoding was implemented as a centralised structure, which minimised resources and helped prevent allocation deadlock with multicast connections (but with increased servicing latencies). Working with both the interval and logical addressing modes provided an insight to their most appropriate use. It was found that interval addressing should be used in devices with large address ranges, or if total concurrent servicing is to be employed. The parallel implementation of the interval address decoding provides a compact solution, which an equivalent logical addressed system could not supply. The embedded and distributed applications, for which this research was targeted, would consider a 16-bit address as large. In contrast, logical addressing should be used for support of multicast packets, or for systems with small address ranges. The definition of small address ranges depends on the target technology, but for the PLDs used here, an 8-bit header would be a suitable limit. Obviously, any configurable headers could extend their scope by cascading headers to multiply possible destinations.

The second stage in connection servicing is allocation, which includes selecting a connection-request and forming the connection. The order in which connections are made depends on the servicing method used. All the router-switches produced in the Nottingham Trent University have employed some form of FIFO queuing. The use of request queuing in the NTR08 allowed the separation of address decoding and connection allocation. This introduced concurrency in the pipeline, in addition to providing fairness in allocation and reduction of switching latencies in times of resource contention. Although the NTR-M04 also used a request queue primarily to maintain some allocation fairness, its secondary function was to supply a mutually exclusive connection between its decoding

and servicing stages. The unified decoding and allocation structure was a compromise made due to the limited resources imposed by buffering requirements. The NTR-FTM08 maintained a FIFO queue for connection-request, and returned to a separate output queue structure similar to the NTR08. This structure permitted concurrent servicing of multiple multicast connection-requests, which was prohibited by the unified queue of the NTR-M04 due to allocation deadlock concerns. However, concurrency was limited, as request queuing and allocation was performed sequentially, due to resource restrictions. As the allocation cycle was sequential, and the probability of concurrent connection-requests was predicted to be low, the sequential form of queuing was used to reduce resource requirements. Request selection operated on a cyclic priority structure to ensure allocation starvation would not occur. The allocation stage was made sequential to allow for the support of 'last minute' group adaptive routing, which attempted to supply improved functionality to the mechanism. In contrast, the NTR08 implemented group adaptive routing at the connection-request queuing stage, which allowed parallel allocation. However, this could result in detrimental output allocation as the decision was made at one instance of time, and the conditions on which that choice was made, could alter with the dynamics of the network. Implementation of a parallel form of last minute group adaptive routing would have required large amounts of resources, and as the predicted probability of concurrent allocation was deemed low, the choice was acceptable. The differences in the effect on the network behaviour between the earlier implementations and the last minute mechanism have not been investigated in this work, and should be targeted for future research.

The primary reason for use of a connection-request queue was to provide allocation fairness when two or more sources request a single resource. The aim of allocation fairness is to prevent allocation starvation and to minimise average packet times. This was produced by both the unified and parallel implementations of the request queue. However, all three router-switch implementations have shown weaknesses, which implies there is much research required to develop the most suitable solution. This future research could include the investigation of other selection servicing mechanisms. Only network analysis, which is based on typical application architecture and traffic patterns, can identify the most suitable service algorithms.

#### 7.1.2.4 Network Configuration

A final concern with the operation of the network is the validation of configuration data. The operation of the fault tolerance within the network does not concentrate on identifying bit errors, it only deals with critical failure. Block checking techniques can be used in the upper layers of the protocol to locate data errors in the point-to-point data stream between communicating entities. However, the current NTR-FTM08 specification does not include any such support for the verification of configuration data. A solution to this could be the use of parity, which could be included in each token instruction. However, to maintain support for the current configuration-port instructions and to support future additions, the inclusion of parity would require the reduction from four to three data bits per load data or address instruction. Alternatively, configuration packets could be limited in size, which could then allow block-checking features to be added to the configuration port to validate the data. While neither of these solutions is infallible, their inclusion would increase configuration confidence, although they do not contain a recovery mechanism. As the configuration network is integrated into the data network, it would be impossible to say where the configuration commands have come from, which makes it impossible to send an error message. The simplest solution is to clear all configuration registers to the power-on reset default. A more proactive option would be the inclusion of an additional control token that is propagated across the network to all nodes, which highlights a configuration failure. This would simplify the problem, and allow the reconfiguration procedure to be triggered. In addition, such a token could halt all data transmissions until the situation had been rectified, which might degrade the network operation but would help with communication integrity.

## 7.2 Conclusions

This thesis has described the development of a novel VLSI hardware packet routing switch. It has detailed the implementation of distributed fault tolerance features that were implemented in two stages. The first stage was concerned with critical interruption of the link stability, validated through simulation and hardware verification: these features detected and localised the effects of network failure, while supplying reduced link activity. The second stage dealt with deadlock detection and recovery, including investigation of a detection mechanism, which minimised time and false detections irrespective of network traffic.

Previous research in embedded parallel or distributed systems has seen the production of many dynamic packet router-switches, which could be used to construct efficient, low-latency networks for use in distributed architectures. Tolerance to network failure in these earlier systems was limited as the resultant network systems were targeted towards board-level or rack-level systems, which possessed very high reliability figures. This resulted in systems relying on system-wide reset recovery techniques or complicated mechanisms of network monitoring and recovery techniques that became more complex as the system scaled. The work described in this thesis has remodelled the system methodology with respect to network faults by turning to a distributed detection and recovery approach. By supplying a holistic automated fault tolerant mechanism, systems can be constructed where network availability gracefully degrades in the presence of faults and that scales without added complexity. This research has maintained the aspects of the connection-switching model of earlier research, whilst contributing to improvements in fault tolerance and core switching, which is concentrated in the following key elements:

**Fault tolerance**

- Integrated a critical link failure detection mechanism for disconnection, synchronisation, and buffer overrun.
- Developed a recovery mechanism which attempts to re-establish valid connection where possible.
- Allowed isolation of critical faults through packet truncation or spillage whilst allowing the remainder of the network to maintain normal operation.
- Investigated deadlock detection and recovery, including implementation of a simulation based mechanism that experimentally supplied 0% false detections whilst minimising the recovery latencies.

**Core Router-switch Design**

- Implemented a dormant mode and the features to re-establish connections, which allowed the links to return to an inactive state, to stop link activity on unused links.
- Implemented three addressing modes.
- Implemented a 'last minute' group adaptive routing algorithm.

- Implemented concurrent servicing of multicast connections.
- Maintained switching overheads whilst using multiple clock domains to improve link speed.
- Changed the flow control mechanism to improve on bi-directional data transfers.

The implementation of the design specification has been demonstrated to operate successfully in simulation and extended hardware tests. The resilience of the fault detection, recovery and isolation mechanism has been informally verified in extended hardware tests, which show no user intervention is required in network recovery. Increased network availability has been supported through 'last minute' group adaptive routing. This technique allows the optimum use of valid links, and maintains a last option of packet deletion to ensure undeliverable packets do not further reduce network availability.

Analysis of the simulation based deadlock detection mechanism has shown it to be an effective solution that minimises recovery latencies whilst prohibiting false detection. Through the discussion, the deadlock detection mechanism that was developed was presented as being too complex for a back-up support for avoidance network-configuration. However, the low latency and 0% false detections make it suitable for further investigation as a primary mechanism within parallel processing networks that use smaller router-switches.

The core operation of the router-switch has been improved compared to earlier research devices, through a number of alterations to the basic operation. Link dormancy was included to reduce link activity on devices with low traffic requirements, yet maintaining coherence with the fault tolerant methodology. The technique allows each connection to be given sufficient chance to be made, in networks with disabled links, whilst taking advantage of the 'last minute' group adaptive routing algorithm where network configuration permits.

Three addressing modes were implemented which included absolute and configurable systems. The discussion presented the appropriateness of introducing multiple addressing modes for a flexible network. Additionally, the research helped validate the use of configurable addressing modes, and target their appropriate use.

The basic operation of earlier forms of group adaptive routing was analysed, which led to the development of last minute group adaptive routing. This form of the mechanism operates more dynamically with the state of the network, improving on some limitations of earlier implementations. It is also required to support the fault tolerant aspects of connection allocation, which ensures that network availability is optimally utilised, where the configuration permitted.

The centralised mechanism for logical address decoding and subsequent operation of connection-request queuing did not differentiate between unicast and multicast connections. The use of a parallel request queue and sequential decoding mechanism allowed concurrent servicing of all logical addressed connections, which contrasted with the sequential multicasting servicing algorithm of the NTR-M04.

The servicing model effectively formed a three-stage pipeline, which reduced the number of core cycles for packet switching compared to the NTR-M04 implementation. Much of the improvement in clocking rates was based on the up-to-date PLD technology, however, the pipelined decode-queue-allocate structure of the NTR-FTM08 did supply an improvement even with the dual clock domains. This was shown in the comparison of the normalised unloaded network bandwidth figures. The primary aim was to minimise the number of core cycles but to maintain a reasonable clock rate. A second clock domain for the sampling circuits allowed optimisation to increase the link speed, without increased design effort in the core. The overall effect maintained similar switching latencies when the normalised figures were compared to the NTR-M04, where the core was clocked at half the rate of the sampling circuits.

Analysis of basic network models was used to select an optimum flow control mechanism, which improved the throughput of the device by 6% in a synthetically loaded environment of a single router-switch. The STOP/GO flow control mechanism, which replaced the credit-based mechanism, was also shown to improve average packet latencies in relation to the network load. The alteration to the flow control mechanism also aided in the operation of the fault detection mechanism, as the control tokens were re-used as idle tokens, which also reaffirmed flow status.

## 7.3 Further Work

There are two main areas of research that need to be further addressed to continue to progress the switched network architecture for use in embedded and distributed systems.

### 7.3.1 System Level Work

Throughout this research project, much has changed in technology and application requirements. In order to aid further research in this area, up-to-data application studies are necessary that will identify the operational requirements of current target systems. However, the immediate work that needs to be carried out includes the integration of this and other collaborative research into an operational system. This will allow more subtle research into the effectiveness that each area of the system contributes. The increased interest of digital communications in consumer markets has led to increased levels of multimedia traffic with embedded and distributed systems. The effects of multimedia traffic and their requirements need to be investigated to help evolve network strategies to maintain an effective solution.

### 7.3.2 Further Routing Techniques

**Further Flow Control Investigation**

The results of the preliminary investigation into the permission-based STOP/GO flow control implied that optimum settings existed for the threshold values of the mechanism. To provide a clearer insight into the effects of the flow control mechanism, further research is required that investigates more complex networks in detail. Areas of investigation should include networks with variations in buffering capacity, threshold values, packet size, topology, traffic patterns, and node behaviour.

**Link Dormancy**

The current link dormancy feature of the NTR-FTM08 operates as designed when both ends of the link are set to the dormant mode. However, if the configuration settings are contrary, it results in intermittent link activity as one side attempts to return to the sleep state, while the other attempts to maintain a connection. Future work could include modification of the link dormancy feature to allow a non-dormant link to be connected to a dormant link, which would allow the dominant configuration to supersede initial configuration settings.

**Configuration Considerations**

As discussed in section 7.1.1.1, the current implementation does not possess any protection against data errors in a configuration data packet. The discussion suggested including some form of error detection within the data stream, such as parity or block

checking, and a predefined recovery procedure on the event of error detection. As the configuration network is integrated into the data network to improve connection availability to the configuration port of each switch, it would be impossible to say where the configuration commands have come from, which complicates the recovery mechanism. Additional research is required to identify a suitable solution to this problem, and integrate it in to an operational system for validation.

**Network Interface Considerations**

This work has not undertaken any steps to integrate the required functionality to any network interface device (a hardware device that links a switched network to a processing unit that does not support the communications protocol [6]). This will be required to support the fault tolerant methodology presented in this thesis. The network interface device would require all the features that the individual link units possess. Additionally, features related to delivery verification would be required. The NTR-FTM08 can cause packets to be removed from the network or truncated, which must be detected to trigger the appropriate recovery mechanism.

Another concern is the order in which packets are delivered. With the inclusion of group adaptive routing and large amounts of buffering within networks, it is possible that consecutive packets of a single message could arrive at the destination out of order. A simple solution would be to acknowledge packets as they arrive, which would permit the transmission of subsequent packets (as implemented in the second generation of transputers). This solution could be implemented in hardware or software, but this would require analysis to provide an effective solution.

**Group Adaptive Routing**

As described in section 7.1.1.1, the current implementation of group adaptive routing provides little advantage within small router-switches. An evolution of this technique would be required that extends the routing flexibility without massive costs in implementation. Such mechanisms would provide improved adaptivity and availability, which would aid both performance and the fault tolerant aspects of resulting networks. Additionally the advantages of 'last minute' grouping compared to the earlier queuing stage mechanisms should be investigated, to discover whether the increased resource overhead is worth the improvements gained.

**Multicast Implementation**

Hardware support of multicast connections complicated the NTR-M04 and NTR-FTM08 designs. In fact, the complications were not concentrated solely in the controller as the crossbar suffers also. Following allocation of all the associated outputs, the data transfer across the crossbar had to be in lockstep to ensure the whole packet was delivered to each destination output. This synchronous nature increased hardware resource requirements dramatically. Such synchronisation meant that a blockage at a single output would inhibit the progression of the whole packet, and therefore degrade the operation of the device. Although some interface-based multicast implementations have been shown to improve the latencies for the delivery of multicast packets, the synchronous based implementations of the NTR-M04 and NTR-FTM08 could cause greater latencies in other traffic patterns. As the emphasis within embedded and distributed systems is on low latency, efficient communication, the implementation of synchronous hardware support for multicast may be an unwise selection. A study of the use of multicast communications in the target applications should be undertaken, providing an analytical comparison between an interface-based solution and the switched based solution. Supplying other routing options within the switch, such as priority, may help interface-based solutions that could result in overall improvements in communication efficiency.

**Connection Servicing**

The NTR08, NTR-M04, and NTR-FTM08 all implemented FIFO buffering based connection-request queuing to supply a form of first-come-first-served service ordering, as discussed previously in section 7.1.2. This servicing methodology can greatly affect the behaviour of the router-switch, which is evident in the response to traffic patterns and workloads. Based on the application review, research needs to be carried out that investigates the effectiveness of the first-come-first-served servicing mechanism and compares it with other techniques within example application networks with realistic target traffic patterns and workloads. Additionally, the comparison of the results from sections 6.1 and 6.2.1.2 stated that sequential servicing algorithms may form a bottleneck that reduces device throughput under loaded conditions. This implies that as the number of links on the router-switch scales, more concurrency should be implemented within the connection servicing mechanism to ease the bottleneck. This should be investigated further to ascertain how the bottleneck affects the device throughput as the device scales.

**System Throughput**

The results showed that with the selected PLD technology the maximum operating frequency of the link interface was around 60 MHz, which equated to 45 Mb/s rate link rate. Complete optimisation of the design and removal of superfluous features could allow the design to reach this operating rate. Improvement on this figure could be simply achieved by changing technology, but the limiting factor always would be the physical encoding technique. Differential signal drivers were used in the physical implementation of this work for basic line drivers, and have been shown to be capable of long transmission distances at high transmission rates in other collaborative work [6]. Some PLD vendors are now offering low voltage differential signal driver circuits as optional integrated IO circuits for their devices, which provide recoverable data streams that operate up to bit rates of 624 Mb/s in a synchronous mode [26]. The feasibility of this or similar technology for use in embedded and distributed system needs to be investigated, concentrating on cost effectiveness and suitability to target applications.

In addition to the link bit rate, switching latencies also must be reduced. The results showed that the current implementation lost up to 19 % of the effective data bandwidth with a payload size of four bytes. It is possible that many embedded and distributed systems may use small packets, which would result is severe loss of network bandwidth. Therefore, to support small packet sizes, switching latencies should be no greater than the time for the transmission of a single token.

**Low Power Implementations**

A major concern in most consumer products is the use of power. The current PLD implementation technology may provide excellent prototyping attributes, but at the cost of power consumption. This situation could be eased by selecting another technology after prototyping, such as a compatible fuse-based architecture. Alternatively, the design could be remodelled to reduce power by either the use of asynchronous design techniques or systems that implement stoppable clocks.

## Publications

R.Hotchkiss, B.C.O'Neill, S.Clark

"*Fault Tolerance for an Embedded Wormhole Switched Network*"

PARLEC 2000 Conference, Trois-Rivieres, Canada, August 2000, ISBN 0-7695-0759-X

R.Hotchkiss, B.C.O'Neill, S.Clark

"*A Fault Tolerant Router for Parallel Networks*"

PREP 2000 Conference, Nottingham, UK, April 2000, ISBN 0-86341-3218

R.Hotchkiss, B.C.O'Neill, K.L.Wong, G.C.Coulson, S.Clark & P.D.Thomas

"*The Building Blocks for a Parallel Network Incorporating the StrongARM Microprocessor*"

PDPTA Conference, Las Vegas, USA, July 1998, ISBN 1–892512–08–4

## References


[1] J.W.ELLIS; "*A hardware routing device for transputer arrays*"; Ph.D. Thesis, The Nottingham Trent University, UK; Oct 1995

[2] G.C.COULSON; "*An ASIC implementation of a multicast message routing switch for interprocessor communications*"; Ph.D. Thesis, The Nottingham Trent University, UK; Sept 1998

[3] CRAY RESEARCH INC.; "*The Cray T3D System Architecture Overview*"; Cray Research Inc.; Technical document HR-04033; 1993

[4] D.RIDGE, D.BECKER, P.MERKEY, T.STERLING; "*Beowulf: harnessing the power of parallelism in a pile-of-PCs*"; 1997 IEEE Aerospace Conference. Proceedings (Cat. No.97CH36020). IEEE. Part vol.2, vol.2. New York, USA, 1997, pp 79-91

[5] G.R.HENDRY; "*Standard ETHERNET as an Embedded Communication Network*"; MSc Thesis; Dept. of Electrical and Computer Engineering, Carnegie Mellon University; April 1999

[6] KL.WONG; "*A message controller for distributed processing systems*"; Ph.D. Thesis, The Nottingham Trent University, UK; June 2000

[7] INTEL CORPORATION; "*Intel Architecture Software Developer's Manual. Volume 1: Basic Architecture*", Order Number 243190; Intel Corporation, 1997, pp 2.2-2.4

[8] M.C. BECKER, M.S.ALLEN, C.R.MOORE, J.S.MUHICH, D.P.TUTTLE; "*The Power PC 601 Microprocessor*", IEEE Micro, vol. 13 no. 5, Oct 1993, pp 54-68

[9] M.FLYNN; "*Some computer organizations and their effectiveness*"; IEEE Trans. Computers,Vol. 21; 1972, pp. 948-960

[10] AMD INC.; "*AMD Athlon Processor. Technical Brief*", Publication Number 22054, Revision D, AMD Inc., Dec 1999

[11] TRANSMETA CORP.; "*Crusoe Processor Model TM3120 Data Sheet*", Transmeta Corp, Data Sheet TM3120, Jan 2000

[12] ANALOG DEVICES INC; "*TigerSHARC DSP Microcomputer. Preliminary Technical Data ADSP-TS001*", Analog Devices Inc., Preliminary Data Sheet ADSP-TS001, Rev PrC, Dec 1999

[13] X.ZHANG; "*System Effects of Interprocessor Communication Latency in Multicomputers*"; IEEE Micro, April 1991; pp 12-15, 52-55

[14] A.C.DÖRING, G.LUSTIG, W.OBELÖER; "*The Impact of Routing Decision Time on Network Latency*"; Proceedings of the 4th PASA Workshop on Parallel Systems and Algorithms; 1997; pp 67-83

[15] M.PIRVU, L.BHUYAN, N.NI; "*The Impact of Link Arbitration on Switch Performance*"; Proceedings of HPCA-5, 5th Int. Conference on High Performance Computer Architecture; IEEE Computer Soc.; Orlando, USA; January 1999; pp 228-235

[16] C.A.R. HOARE: "*Communicating Sequential Processes*", Hemel Hempstead : Prentice-Hall International, 1985

[17] R.A.QUINNELL; "*USB: a neat package with a few loose ends*"; EDN, Vol. 41, No. 22; October 1996; pp 38-46, 48, 50, 52

[18] I.J.WICKELGREN; "*The facts about FireWire serial communication bus*"; IEEE Spectrum, Vol. 34, No. 4; April 1997; pp 19-25

[19] C.EDWARDS; "*Big three backing for switch fabric*"; Electronics Times, No. 984; 6th Mar 2000; pp 6

[20] J.CHILD; "*Get ready for channel-based links in I/O subsystems*"; Electronic Design, Vol. 47, No. 9; Penton Publishing, USA ;May 1999; pp 65-66, 68, 70, 72

[21] T.FOREMSKI; "*InfiniBand is new name for future server design standard*"; Electronics Weekly Archive, Online at http://www.electronicsweekly.co.uk; 28th Oct 1999

[22] R.BALL; '*IP and trendy*'; Electronics Weekly, No. 1915; 23rd June 1999; pp 18

[23] I.A.GLOVER, P.M.GRANT; "*Digital Communications*"; Prentice Hall, Europe 1998; ISBN 0 13 565391 6; 1998; pp 670-673

[24] L.R.DENNISON, W.S.LEE, W.J.DALLY; "*High-performance bidirectional signalling in VLSI systems*"; 1993 Symposium on Research on integrated systems; Seattle, USA; 1993

[25] T.M.PINKSTON, Y.CHOI, M.RAKSAPATCHARAWONG; "*Architecture and Optoelectronic Implementation of the WARRP Router*"; Proceedings, 5th High performance Interconnects, (Hot interconnects V); Palo Alto, CA, USA.; 1997; pp ??

[26] ALTERA CORPORATION; "*APEX 20K. Programmable Logic Device Family*"; Data Sheet A-DS-APEX20K-02.06; Ver. 2.06; March 2000

[27] B.VOM BERG, P.GROPPE; "*The CAN bus, intelligent, decentralized, data communications. Part 2*"; Elektor Electronics, No. 280; September 1999; pp 39

[28] G.HELD; "*Data Communications Networking Devices*"; Wiley 1999, Fourth Edition; ISBN 0-471-97515; pp 303-327

[29] E.TRAN; '*Multi-bit Error Vulnerabilities in the Controller Area Network Protocol*'; M.Sc. Thesis, Carnegie Mellon University, Pittsburgh, USA; May 1999

[30] VITA STANDARDS ORGANISATION; '*Myrinet-on-VME. Protocol Specification Draft Standard*'; VITA 26-199x Draft 1.1; VITA Standards Organisation, August 1998; pp 27

[31] M.D.MAY, P.W.THOMPSON, P.H.WELCH; '*Networks, routers & transputers: Function, performance and application*'; IOS Press; ISBN 90 5199 129 0; 1993; pp 40

[32] I.A.GLOVER, P.M.GRANT; "*Digital Communications*"; Prentice Hall, Europe 1998; ISBN 0 13 565391 6; 1998; pp 45-48

[33] W.J.DALLY, L.R.DENNISON, D.HARRIS, K.KAN, T.XANTHOPOULOS; "*The Reliable Router: A Reliable and High Performance Communication Substrate for Parallel Computers*"; 1st Int. Parallel Computer Routing & Communication Workshop; Seattle, USA, May 1994; pp 241-255

[34] W.J.DALLY, H.AOKI; "*Deadlock-free adaptive routing in multicomputer networks using virtual channels*", IEEE Trans. Parallel and Distributed Systems; Vol. 4, No. 4; April 1993

[35] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 51-211

[36] B.VOM BERG, P.GROPPE; "*The CAN bus, intelligent, decentralized, data communications. Part 1 & 2*"; Elektor Electronics, No. 280; September 1999; pp 24-29, 36-41

[37] M.GERLA, P.PALNATI, S.WALTON; '*Multicasting in Myrinet – A high speed, wormhole-routing network*'; IEEE GLOBECOM 1996. Communications: The key to global prosperity; IEEE. Part Vol. 2, ISBN 0 7803 3336 5; New York, USA; pp 1064-1068

[38] R.SIVARAM, R.KESAVAN, D.K.PANDA, C.B.STUNKEL; "*Where to provide support for efficient multicasting in irregular networks: Network interface or switch?*"; IEEE Computer Soc. 27th Int. Conference on Parallel Processing; Ohio, USA; Aug 1998; pp 452-459

[39] P.K.MCKINLEY, H.XU, A.H.ESFAHANIAN, L.M.NI; "*Unicast-Based Multicast Communication in Wormhole-Routed Networks*"; IEEE Trans. Parallel and Distributed Systems, Vol. 5, No. 12, Dec 1994; pp 1252-1265

[40] M.D.SCHROEDER, A.D.BIRRELL, M.BURROWS, H.MURRAY, R.M.NEEDHAM, T.L.RODEHEFFER, E.H.SATTERTHWAITE, C.P.THACKER; "*Autonet: a high-speed self-configuring local area network using point-to-point links*"; Digital Equipment Corporation, SRC Research Report 59, April 1990

[41] C.B.STUNKEL, R.SIVARAM, D.K.PANDA; '*Implementing multidestination worms in switch-based parallel systems: architectural alternatives and their impact*'; Computer Architecture News, Vol. 25, No. 2; ACM, USA; May 1997; pp 50-61

[42] L.M.NI, P.K.MCKINLEY; "*A Survey of Wormhole Routing Techniques in Direct Networks*"; IEEE Computer, Vol. 26, No 2; February 1993; pp 62-76

[43] J.DUATO, S.YALAMACHILI, M.C.CAMINERO, D.LOVE, F.J.QUILES: "*MMR: A high-performance multimedia router – architecture and design trade-offs*", Proceedings of the 5th Symposium on High-performance computer architecture, 1999, Los Alamitos, CA, USA, pp 300-309

[44] S.W.DANIEL, K.G.SHIN, S.K.YUN; "*A router architecture for flexible routing and switching in multihop point-to-point networks*"; IEEE Trans. Parallel and Distributed Systems, Vol. 10, No. 1, Jan 1999, pp 62,75

[45] W.J.DALLY; "*Virtual-channel Flow Control*", IEEE Trans. Parallel and Distributed Systems, Vol. 3, No. 2; March 1992, pp 194-204

[46] C.C.SU, K.G.SHIN; "*Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel*"; IEEE Transactions on Computers, vol.45, no.6, June 1996, pp.666-683

[47] M.D.MAY, P.W.THOMPSON, P.H.WELCH; '*Networks, routers & transputers: Function, performance and application*'; IOS Press; ISBN 90 5199 129 0; 1993; pp 21

[48] N.R.MCKENZIE, K.BOLDING, C.EBELING, L.SNYDER; "*Cranium: An Interface for Message Passing on Adaptive Packet Routing Networks*"; Proceedings of the 1994 Parallel Computer Routing and Communication Workshop; Seattle, USA; May 1994; pp ??

[49] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 17-18

[50] H.J.LEE, B.Y.SONG; "*Performance of multiple links over single link in STC104 networks*"; Proceedings. 1997 Int. Conference on Parallel and Distributed Systems, (CAT. No. 97TB100215); Los Alamitos, CA, USA; 1997; pp 196-202

[51] W.J.DALLY, C.L.SEITZ; "*Deadlock-free message routing in multiprocessor interconnection networks.*"; IEEE Transactions on Computers, Vol. C-36, No. 5; May 1987; pp 547-533

[52] J.DUATO; "*A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks*"; IEEE Transactions on Parallel & Distributed Systems, Vol. 4, No. 12; December 1993; pp 1320-1331

[53] K.BOLDING; "*Chaotic Routing – Design and Implementation of an Adaptive Multicomputer Network Router*"; Ph.D. Thesis, University of Washington, USA; 1993

[54] C.J.GLASS, L.M.NI; "*Fault tolerant wormhole routing in meshes without virtual channels*"; IEEE Trans. Parallel and Distributed Systems, Vol. 7, No. 6; June 1996; pp 620-636

[55] T.D.NGUYEN, L.SNYDER; "*Performance Analysis of a Minimal Adaptive Router*"; Proceedings of the 1994 Parallel computer routing and communication workshop; Seattle, USA; May 1996; pp 31-44

[56] K.V.ANJAN; T.M.PINKSTON; J.DUATO; "*Generalized Therory for Deadlock-Free Adaptive Wormhole Routing and its Application to Disha Concurrent*"; IPPS 96;

[57] LOPEZ P., MARTINEZ J.M., DUATO J.: "*A Very Efficient Distributed Deadlock Detection Mechanism for Wormhole Networks*"; Proceedings of the 4th International Symposium on High-Performance Computer Architecture. IEEE Computer Soc. (1998) 57-66

[58] MYRICOM, INC.; '*Myrinet Link Specification*'; Archived specification of specification available from Myricom Inc. USA. Also available http://www.myri.com/scs/documentation/link/index.html (as of July 2000)

[59] SGS-THOMPSON MICROELECTRONICS; "*C104 Asynchronous Packet Switch Engineering Data*"; Document No. 42 1470 06; April 1995; pp 8-11

[60] G.HELD; "*Data Communications Networking Devices*"; Wiley 1999, Fourth Edition; ISBN 0-471-97515; pp 54-58

[61] S.WARNAKULASURIYA, T.M.PINKSTON; "*Characterization of deadlocks in interconnection networks*"; Proceedings, 11th Int. Parallel processing symposium; IEEE Computer Soc. Press 1997; Los Alamitos, CA, USA; April 1997; pp 80-86

[62] A.FOLKESTAD, C.ROCHE; "*Deadlock probability in unrestricted wormhole routing networks*"; IEEE Int. Conference on Communications, Vol.3; Montreal, Canada; 1997; pp 1401-1405

[63] P.T.GAUGHAN, S.YALAMANCHILI; "*Pipelined circuit switching: A fault-tolerant variant of wormhole routing*"; Proceedings, IEEE symposium on Parallel and distributed processing; ISBN 0 8186 3200 3; 1992; pp 148-155

[64] B.V.DAO, J.DUATO, S.YALAMANCHILI; "*Configurable flow control mechanisms for fault-tolerant routing*"; Proceedings, 22nd Annual Int. Symposium on Computer Architecture; 1995; pp 220-229

[65] J.FLICH, M.P.MALUMBRES, P.LOPEZ, J.DUATO; "*Performance evaluation of a new routing strategy for irregular networks with source routing*"; Proceedings, Int. conference on supercomputing; ACM Press, ISBN 1 58113 270 0; May 2000; pp 34-43

[66] T.M.PINKSTON, S.WARNAKULASURIYA; "*On Deadlocks in Interconnection Networks*" Computer Architecture News. Vol. 25, No. 2; 1997; pp 38-49

[67] MARTINEZ J.M., LOPEZ P., DUATO J., PINKSTON T.M.; "*Software-Based Deadlock Recovery Techniques for True Fully Adaptive Routing in Wormhole Networks*"; proceedings, 1997 Int. Conference on Parallel Processing; 1997; pp 182-189

[68] K.V.ANJAN, T.M.PINKSTON; "*An efficient, fully adaptive deadlock recovery scheme: DISHA*"; Proceedings, 22nd Int. symposium on Computer architecture; ISBN 0 89791 698 0; New York, USA; June 1995; pp 201-210

[69] VITA STANDARDS ORGANISATION; '*Myrinet-on-VME. Protocol Specification Draft Standard*'; VITA 26-199x Draft 1.1; VITA Standards Organisation, August 1998; pp 21

[70] T.M.PINKSTON; "*Flexible and Efficient Routing Based on Progressive Deadlock Recovery*"; IEEE Transactions on Computers; Vol. 48, No. 7; 1999; pp 649-669

[71] INMOS LTD.: "*The Transputer Databook*" (INMOS document number: 72 TRN 203 02, 1992) 3rd edn; pp 208-209

[72] INMOS LTD.: "*The Transputer Databook*" (INMOS document number: 72 TRN 203 02, 1992) 3rd edn; pp. 397-407

[73] J.Y. LEE, S.J. JEONG, W.M. JANG: "*An Implementation of a Dynamic Reconfiguration in POPA (POhang PArallel) Computer*", Applications of Transputers 3, (IOS Press, Ohmsha, 1991) Vol. 1, Proceedings of the 3rd International Conference on Applications of Transputers; August 1991; Glasgow, UK, pp. 38-47

[74] L.W. WIGGERS, J.C. VERMEULEN: "*The application of transputers in High-Energy Physics*", Applications of Transputers 2, (IOS Press, Ohmsha, 1990), Proceedings of the 2nd International Conference on Application of Transputers; July 1990; Southampton, UK; pp. 34-39

[75] P.A.SHALLOW: "*Processor Independent and Extendable Routing System using a Cyclic Routing Algorithm*", OCCAM and The Transputer – Current Developments, (IOS Press, 1991), proceedings of the 14th World OCCAM and Transputer User Group Technical Meeting, Sept. 1991, Loughborough, UK, pp 225-233

[76] G.C. COULSON, B.C. O"NEILL, J.W. ELLIS, S. CLARK; "*Optimisation of a Processor Farm using Hardware Routing*"; Transputer Applications and Systems ed. B M Cook, M R Jane, P Nixon & P M Welch; IOS Press, Vol 46; ISSN: 1383-7575; 1995; pp 70-77.

[77] IC-ROUTING Ltd.; "*16-port Dynamic Routing Switch for Transputer Link*"; Company data sheet; Ver. 1.3; 1996

[78] SGS-THOMPSON MICROELECTRONICS; "*C104 Asynchronous Packet Switch Engineering Data*"; ; 42 1470 06; April 1995

[79] W.J.DALLY, L.R.DENNISON, D.HARRIS, K.KAN, T.WANTHOPOLOUS; "*Architecture and Implementation of the Reliable Router*"; Proceedings of Hot Interconnects II, Stanford, USA, Aug 1994; pp 122-133

[80] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 7-8

[81] INSTITUTE ELECTRICAL AND ELECTRONIC ENGINEERS, INC.; "*IEEE Standard for Heterogeneous Interconnects (HIC)(Lost-Cost, Low-Latency Scaleable Serial Interconnect for Parallel System Construction)*"; IEEE Std 1355-1995, SH94378; IEEE, NY, USA; June 1996; pp 25-32

[82] K.BOLDING, W.YOST; *"Design of a Router for Fault-tolerant Networks*"; Proceedings of the 1994 Computer Routing and Communication Workshop; May 1994; pp 226-240

[83] L.R.DENNISON, W.J.DALLY, D.XANTHOPOULOS; "*Low-latency plesiochronous data retiming*"; 1995 Conference on Advanced Research in VLSI; Chapel Hill NC, March 1995; (Available ftp://ftp.ai.mit.edu/pub/cva/plesio.ps.Z – June 2000)

[84] M.D.MAY, P.W.THOMPSON, P.H.WELCH; '*Networks, routers & transputers: Function, performance and application*'; IOS Press; ISBN 90 5199 129 0; 1993

[85] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 260

[86] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 45-48

[87] SPACEWIRE WORKING GROUP, S.M.PARKES; '*ECSS-E-50-12 Space Engineering. SpaceWire: Links, nodes, routers and networks. Draft*'; Document No. UoD-DICE-TN-9201, Issue D; ESA Contract 12693/97/NL/FM; June 2000; pp 96-97; (http://www.estec.esa.nl/tech/spacewire/ - July 2000)

[88] N.J.BODEN, D.COHEN, R.E.FELDERMAN, A.E.KULAWIK, C.L.SEITZ, J.N.SEIZOVIC, W.SU; "*Myrinet – A Gigabit-per-Second Local Area Network*"; IEEE Micros, Vol. 15, No. 1; February 1995; pp 29-36

[89] K.VERSTOEP, K.LANGEDOEN, H.BAL; '*Efficient reliable multicast on Myrinet*'; Proceedings, 1996 Int. Conference on Parallel processing, Vol. 3 software; IEEE Computer Soc. Press; ISBN 0 8186 7623 X; Los Alamitos, USA; pp 156-165

[90] R.SIVARAM, D.K.PANDA, C.B.STUNKEL; '*Efficient broadcast and multicast on multistage interconnection networks using multiport encoding*'; IEEE Transactions on Parallel and Distributed Systems; Vol. 9, No. 10; Oct 1998; pp 1004-1028

[91] J.REXFORD, J.HALL, K.G.SHIN; '*A router architecture for real-time communication in multicomputer networks*'; IEEE Transactions on Computers, Vol. 47, No. 10; Oct 1998; pp 1088-1101

[92] M.D.MAY, P.W.THOMPSON, P.H.WELCH; '*Networks, routers & transputers: Function, performance and application*'; IOS Press; ISBN 90 5199 129 0; 1993; pp 21

[93] K.G.SHIN, S.HAN; "*FastLow-Cost Recovery for Reliable Real-Time Multimedia Communication*"; IEEE Network, Vol. 12, No. 6; November-December 1998; pp 56-63

[94] VITA STANDARDS ORGANISATION; '*Myrinet-on-VME. Protocol Specification Draft Standard*'; VITA 26-199x Draft 1.1; VITA Standards Organisation, August 1998; pp 45 (or online – http://www.myri.com/myri-types.html – July 2000)

[95] A.M.JONES, N.J.DAVIES, M.A.FIRTH, C.J.WRIGHT; "*PACT The Network Designer"s Handbook*"; IOS Press, Ohmsha, Concurrent systems engineering series, Vol. 51; ISSN 1383-7575; 1997; pp 18-19

[96] J.FLICH, M.P.MALUMBRES, P.LOPEZ, J.DUATO; "*Improving Routing Performance in Myrinet Networks*"; Proceedings. 14th Int. Parallel and Distributed Processing Symposium; IEEE Computer Soc.; ISBN 0-7695-0574-0; May 2000; pp 27-32

[97] SGS-THOMPSON MICROELECTRONICS; "*C104 Asynchronous Packet Switch Engineering Data*"; ; 42 1470 06; April 1995; pp 22-38

[98] F.PETRINI, M.VANNESCHI; "*Performance Analysis of Minimal Adaptive Wormhole Routing with Time-Dependent Deadlock Recovery*"; Proceedings, 11th Int. Parallel processing symposium; Genva, Switzerland; April 1997; pp 587-595

[99] ALTERA CORPORATION; "*FLEX 10KE Embedded Programmable Logic Family*"; Data sheet A-DS-F10KE-01.01; Version 1.01; November 1998

[100] S.VARADARAJAN, T.CHIEUH; "*Automatic Fault Detection and Recovery in Real Time Switched Ethernet*"; Proceedings of INFOCOM'99, Conference on Computer Communications, IEEE Computer Soc.; New York, USA; March 1999; pp 21-25

# Appendix A : Preliminary Flow Control Comparison Results

This appendix contains the results of preliminary simulations of the flow control comparisons on a synthetically loaded router switch.

The estimated workloads were based on the average rate of packets that were injected into the network. The test vectors were statically defined before the tests. Average packet time was the observed time taken from post simulation analysis of the test results. Normalised average packet time was calculated thus:

$$\text{normalised packet time} = \frac{\text{average packet time}}{\text{calculated minimum packet time}}$$

where the calculated minimum packet time was derived from the operational parameters of the system thus :

$$\text{bits per packet} = (\text{packet overhead} + \text{tokens per packet}) \times \text{bits per token}$$

$$\text{calculated mimimum packet time} = \text{bits per packet} \times (\text{sample clock period} \times 1.5)$$

The throughput figures where produced by the division of the number of transmitted bits by the duration of the test. The percentage throughput was equal to the percentage of the observed throughput of the calculated maximum system throughput, which was calculated thus:

$$\text{link data rate} = \frac{1}{\text{Sample Clock Period} \times 1.5}$$

$$\text{calculated system maximum throughput} = \text{link data rate} \times \text{number of links}$$

The data throughput and percentage data throughput were calculated similarly to the basic throughput figures, but only data bits were included in the calculation. The calculated maximum system data throughput was calculated thus:

$$\text{calculated maximum system data throughput} = \text{link data rate} \times \frac{8}{11} \times \text{number of links}$$

**Results for the STOP/GO permission based flow control.**

| Estimated Workload (%) | Average Packet Time (seconds) | Normalised Average Packet time |
|---|---|---|
| 100.00% | 27.194E-6 | 1.717 |
| 90.00% | 27.160E-6 | 1.715 |
| 80.00% | 27.216E-6 | 1.718 |
| 70.00% | 25.998E-6 | 1.641 |
| 50.00% | 21.174E-6 | 1.337 |
| 30.00% | 18.833E-6 | 1.189 |
| 10.00% | 17.468E-6 | 1.103 |

| Estimated Workload (%) | Test Duration (seconds) | Throughput (bits per second) | Percentage Throughput |
|---|---|---|---|
| 100.00% | 108.894E-3 | 64.65E+6 | 72.73% |
| 90.00% | 108.690E-3 | 64.77E+6 | 72.87% |
| 80.00% | 108.519E-3 | 64.87E+6 | 72.98% |
| 70.00% | 114.784E-3 | 61.33E+6 | 69.00% |
| 50.00% | 160.782E-3 | 43.79E+6 | 49.26% |
| 30.00% | 268.008E-3 | 26.27E+6 | 29.55% |
| 10.00% | 804.138E-3 | 8.75E+6 | 9.85% |

| Estimated Workload (%) | Data Throughput (bit per second) | Percentage Data Throughput |
|---|---|---|
| 100.00% | 44.08E+6 | 68.19% |
| 90.00% | 44.16E+6 | 68.31% |
| 80.00% | 44.23E+6 | 68.42% |
| 70.00% | 41.82E+6 | 64.69% |
| 50.00% | 29.85E+6 | 46.18% |
| 30.00% | 17.91E+6 | 27.70% |
| 10.00% | 5.97E+6 | 9.23% |

| | |
|---|---|
| Number of messages | 20000 |
| Packet overhead | 2 |
| Tokens per message | 30 |
| Bit per token | 11 |
| Number of bits per test | 7040000 |
| Data bits per test | 4800000 |

| | |
|---|---|
| Sample Clock Period | 30.00E-9 |
| Single packet tx time (s) | 14.850E-6 |
| Number of links | 4 |
| System Throughput | 88.89E+6 |
| System Data Throughput | 64.65E+6 |
| Calculated Min Packet Time | 15.840E-6 |

**Results for the credit based flow control (flow group = 8).**

| Estimated Workload (%) | Average Packet Time (seconds) | Normalised Average Packet time |
|---|---|---|
| 100.00% | 26.017E-6 | 1.642 |
| 90.00% | 26.163E-6 | 1.652 |
| 80.00% | 26.063E-6 | 1.645 |
| 70.00% | 25.984E-6 | 1.640 |
| 50.00% | 22.459E-6 | 1.418 |
| 30.00% | 19.732E-6 | 1.246 |
| 10.00% | 17.784E-6 | 1.123 |

| Estimated Workload (%) | Test Duration (seconds) | Throughput (bits per second) | Percentage Throughput |
|---|---|---|---|
| 100.00% | 118.590E-3 | 59.36E+6 | 66.78% |
| 90.00% | 118.027E-3 | 59.65E+6 | 67.10% |
| 80.00% | 118.046E-3 | 59.64E+6 | 67.09% |
| 70.00% | 118.230E-3 | 59.55E+6 | 66.99% |
| 50.00% | 160.782E-3 | 43.79E+6 | 49.26% |
| 30.00% | 268.008E-3 | 26.27E+6 | 29.55% |
| 10.00% | 804.138E-3 | 8.75E+6 | 9.85% |

| Estimated Workload (%) | Data Throughput (bits per second) | Percentage Data Throughput |
|---|---|---|
| 100.00% | 40.48E+6 | 62.61% |
| 90.00% | 40.67E+6 | 62.91% |
| 80.00% | 40.66E+6 | 62.90% |
| 70.00% | 40.60E+6 | 62.80% |
| 50.00% | 29.85E+6 | 46.18% |
| 30.00% | 17.91E+6 | 27.70% |
| 10.00% | 5.97E+6 | 9.23% |

| | |
|---|---|
| Number of messages | 20000 |
| Packet overhead | 2 |
| Tokens per message | 30 |
| Bit per token | 11 |
| Number of bits per test | 7040000 |
| Data bits per test | 4800000 |

| | |
|---|---|
| Sample Clock Period | 30.00E-9 |
| Single packet tx time (s) | 14.850E-6 |
| Number of links | 4 |
| System Throughput | 88.89E+6 |
| System Data Throughput | 64.65E+6 |
| Calculated Min Packet Time | 15.840E-6 |

# Appendix B : Basic STOP/GO A nalysis Details

This appendix contains the configuration settings, network diagrams and simulation results for the STOP/GO flow control analysis. Two small regular networks were used within these simulations, as opposed to a single router switch, to provide increased levels of bi-directional data. Both test networks were set-up with a deadlock avoidance configuration.

To help clarify how the results were calculated from the test results, please refer to Appendix A. Although the operational parameters of the test networks are different between the two tests, the same terms of reference and calculations were used.

## Mesh Network

### Test Configuration Set-up of the Mesh Network

The mesh network test used the connection topology as shown in Figure B-1. The device configuration for this mesh network follows the diagram.

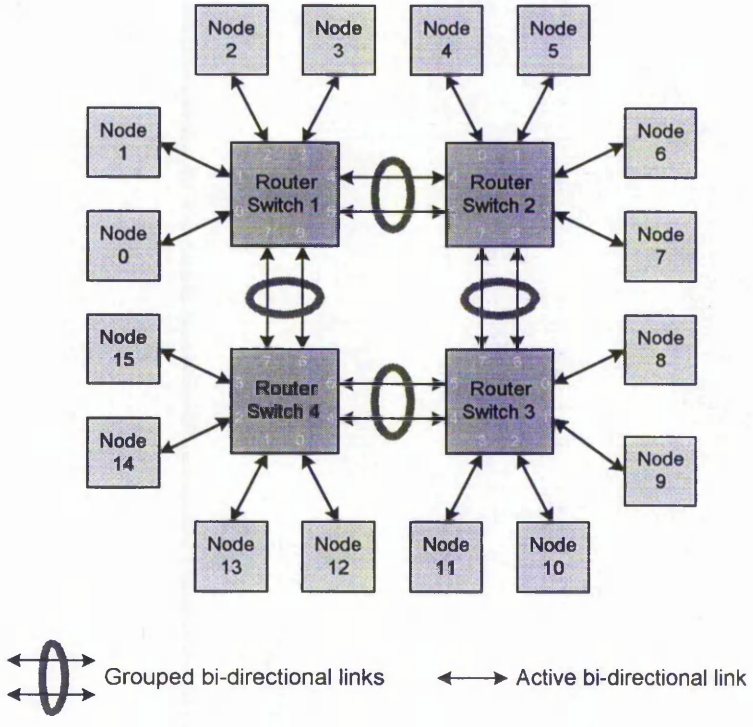



**Figure B-1 : Four router switch mesh network used within the STOP/GO flow control analysis simulations**

| Router switch 1 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 11 | 4 |
| 5 | 15 | 6 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 2 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 4 |
| 1 | 4 | 0 |
| 2 | 5 | 1 |
| 3 | 6 | 2 |
| 4 | 7 | 3 |
| 5 | 11 | 6 |
| 6 | 15 | 4 |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 3 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 4 |
| 1 | 7 | 6 |
| 2 | 8 | 0 |
| 3 | 9 | 1 |
| 4 | 10 | 2 |
| 5 | 11 | 3 |
| 6 | 15 | 4 |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 4 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 6 |
| 1 | 11 | 4 |
| 2 | 12 | 0 |
| 3 | 13 | 1 |
| 4 | 14 | 2 |
| 5 | 15 | 3 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

Each router switch within the mesh network was configured with same group adaptive routing settings; namely, group 0 = 4 & 5, group 1 = 6 & 7

The operational parameters used for these simulations and the analysis were as follows:

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 3 |
| Tokens per packet | 32 |
| Bit per token | 11 |
| Number of bits per test | 3080000 |
| Data bits per test | 2048000 |

| | |
|---|---|
| Sample Clock Period | 16.00E-9 |
| Single packet tx time (s) | 8.448E-6 |
| Number of links | 16 |
| System Throughput | 666.67E+6 |
| System Data Throughput | 484.85E+6 |
| Calculated Min Packet Time | 9.240E-6 |

**Test Results for the Mesh Network**

| Average packet time (seconds) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 62.871E-6 | 59.148E-6 | 63.992E-6 |
| 70 | 62.627E-6 | 58.412E-6 | 63.934E-6 |
| 60 | 61.187E-6 | 57.929E-6 | 63.556E-6 |
| 50 | 58.818E-6 | 55.488E-6 | 62.532E-6 |
| 40 | 28.132E-6 | 27.240E-6 | 27.763E-6 |
| 30 | 17.985E-6 | 18.003E-6 | 17.990E-6 |

| Normalised average packet time | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 6.81 | 6.40 | 6.40 |
| 70 | 6.78 | 6.32 | 6.32 |
| 60 | 6.62 | 6.27 | 6.27 |
| 50 | 6.37 | 6.01 | 6.01 |
| 40 | 3.05 | 2.95 | 2.95 |
| 30 | 1.95 | 1.95 | 1.95 |

| Test duration (seconds) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 9.476E-3 | 9.702E-3 | 9.639E-3 |
| 70 | 9.549E-3 | 9.676E-3 | 9.629E-3 |
| 60 | 9.578E-3 | 9.524E-3 | 9.640E-3 |
| 50 | 9.551E-3 | 9.704E-3 | 9.712E-3 |
| 40 | 11.557E-3 | 11.557E-3 | 11.557E-3 |
| 30 | 15.407E-3 | 15.407E-3 | 15.407E-3 |

| Observed Throughput (bits per second) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 325.04E+6 | 317.46E+6 | 319.53E+6 |
| 70 | 322.56E+6 | 318.33E+6 | 319.86E+6 |
| 60 | 321.56E+6 | 323.40E+6 | 319.50E+6 |
| 50 | 322.47E+6 | 317.41E+6 | 317.14E+6 |
| 40 | 266.52E+6 | 266.52E+6 | 266.51E+6 |
| 30 | 199.91E+6 | 199.91E+6 | 199.91E+6 |

| Percentage Throughput | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 48.76% | 47.62% | 47.93% |
| 70 | 48.38% | 47.75% | 47.98% |
| 60 | 48.23% | 48.51% | 47.93% |
| 50 | 48.37% | 47.61% | 47.57% |
| 40 | 39.98% | 39.98% | 39.98% |
| 30 | 29.99% | 29.99% | 29.99% |

| Data Throughput (bits per second) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 216.13E+6 | 211.09E+6 | 212.46E+6 |
| 70 | 214.48E+6 | 211.67E+6 | 212.68E+6 |
| 60 | 213.82E+6 | 215.04E+6 | 212.45E+6 |
| 50 | 214.42E+6 | 211.05E+6 | 210.88E+6 |
| 40 | 177.22E+6 | 177.22E+6 | 177.21E+6 |
| 30 | 132.93E+6 | 132.93E+6 | 132.93E+6 |

| Percentage Data Throughput | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 44.58% | 43.54% | 43.82% |
| 70 | 44.24% | 43.66% | 43.87% |
| 60 | 44.10% | 44.35% | 43.82% |
| 50 | 44.22% | 43.53% | 43.49% |
| 40 | 36.55% | 36.55% | 36.55% |
| 30 | 27.42% | 27.42% | 27.42% |

## Torus Network

### Test Configuration Set-up of the Torus Network

The torus network test used the connection topology as shown in Figure B-2. The device configuration for this torus network follows the diagram.

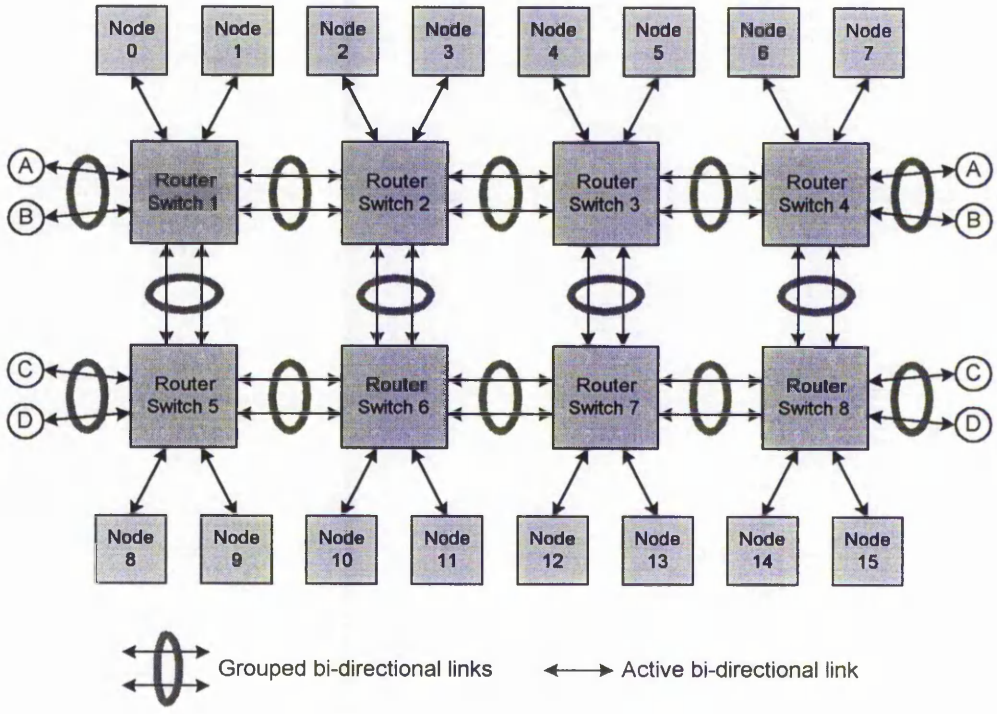



**Figure B-2 : Eight router-switch torus network used within the STOP/GO analysis simulations**

| Router switch 1 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 5 | 2 |
| 3 | 7 | 6 |
| 4 | 15 | 4 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 2 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 1 | 6 |
| 1 | 2 | 0 |
| 2 | 3 | 1 |
| 3 | 5 | 2 |
| 4 | 7 | 6 |
| 5 | 15 | 4 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 3 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 6 |
| 1 | 4 | 0 |
| 2 | 5 | 1 |
| 3 | 7 | 2 |
| 4 | 15 | 4 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 4 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 2 |
| 1 | 5 | 6 |
| 2 | 6 | 0 |
| 3 | 7 | 1 |
| 4 | 15 | 4 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 5 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 7 | 4 |
| 1 | 8 | 0 |
| 2 | 9 | 1 |
| 3 | 13 | 2 |
| 4 | 15 | 6 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 6 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 7 | 4 |
| 1 | 9 | 6 |
| 2 | 10 | 0 |
| 3 | 11 | 1 |
| 4 | 13 | 2 |
| 5 | 15 | 6 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 7 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 7 | 4 |
| 1 | 11 | 6 |
| 2 | 12 | 0 |
| 3 | 13 | 1 |
| 4 | 15 | 2 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 8 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 7 | 4 |
| 1 | 11 | 2 |
| 2 | 13 | 6 |
| 3 | 14 | 0 |
| 4 | 15 | 1 |
| 5 | 63 | invalid |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

Each router switch within the torus network was configured with same group adaptive routing settings; namely, group 0 = 2 & 3, group 1 = 4 & 5, group 2 = 6 & 7

The operational parameters used for these simulations and the analysis were as follows:

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 3 |
| Tokens per packet | 32 |
| Bit per token | 11 |
| Number of bits per test | 3080000 |
| Data bits per test | 2048000 |

| | |
|---|---|
| Sample Clock Period | 16.00E-9 |
| Single packet tx time (s) | 8.448E-6 |
| Number of links | 16 |
| System Throughput | 666.67E+6 |
| System Data Throughput | 484.85E+6 |
| Calculated Min Packet Time | 9.240E-6 |

**Test Results for the Torus Network**

| Average packet time (seconds) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 61.495E-6 | 57.833E-6 | 51.573E-6 |
| 70 | 60.423E-6 | 56.932E-6 | 51.088E-6 |
| 60 | 57.502E-6 | 54.026E-6 | 49.505E-6 |
| 50 | 28.082E-6 | 28.483E-6 | 30.435E-6 |
| 40 | 19.956E-6 | 19.920E-6 | 19.934E-6 |
| 30 | 16.656E-6 | 16.739E-6 | 16.632E-6 |

| Normalised average packet time | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 6.66 | 6.26 | 5.58 |
| 70 | 6.54 | 6.16 | 5.53 |
| 60 | 6.22 | 5.85 | 5.36 |
| 50 | 3.04 | 3.08 | 3.29 |
| 40 | 2.16 | 2.16 | 2.16 |
| 30 | 1.80 | 1.81 | 1.80 |

| Test duration (seconds) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 8.311E-3 | 8.162E-3 | 8.375E-3 |
| 70 | 8.121E-3 | 8.234E-3 | 8.394E-3 |
| 60 | 8.354E-3 | 8.308E-3 | 8.246E-3 |
| 50 | 9.246E-3 | 9.246E-3 | 9.246E-3 |
| 40 | 11.557E-3 | 11.557E-3 | 11.557E-3 |
| 30 | 15.407E-3 | 15.407E-3 | 15.407E-3 |

| Observed Throughput (bits per second) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 370.57E+6 | 377.38E+6 | 367.74E+6 |
| 70 | 379.25E+6 | 374.04E+6 | 366.93E+6 |
| 60 | 368.70E+6 | 370.74E+6 | 373.53E+6 |
| 50 | 333.12E+6 | 333.13E+6 | 333.12E+6 |
| 40 | 266.51E+6 | 266.52E+6 | 266.52E+6 |
| 30 | 199.91E+6 | 199.91E+6 | 199.91E+6 |

| Percentage Throughput | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 55.59% | 56.61% | 55.16% |
| 70 | 56.89% | 56.11% | 55.04% |
| 60 | 55.30% | 55.61% | 56.03% |
| 50 | 49.97% | 49.97% | 49.97% |
| 40 | 39.98% | 39.98% | 39.98% |
| 30 | 29.99% | 29.99% | 29.99% |

| Data Throughput (bits per second) | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 246.41E+6 | 250.93E+6 | 244.53E+6 |
| 70 | 252.18E+6 | 248.71E+6 | 243.99E+6 |
| 60 | 245.16E+6 | 246.52E+6 | 248.37E+6 |
| 50 | 221.50E+6 | 221.51E+6 | 221.50E+6 |
| 40 | 177.21E+6 | 177.22E+6 | 177.22E+6 |
| 30 | 132.93E+6 | 132.93E+6 | 132.93E+6 |

| Percentage Data Throughput | | | |
|---|---|---|---|
| Estimated Workload (%) | Threshold Differential | | |
| | 8 | 16 | 32 |
| 80 | 50.82% | 51.75% | 50.43% |
| 70 | 52.01% | 51.30% | 50.32% |
| 60 | 50.56% | 50.84% | 51.23% |
| 50 | 45.69% | 45.69% | 45.69% |
| 40 | 36.55% | 36.55% | 36.55% |
| 30 | 27.42% | 27.42% | 27.42% |

# Appendix C : NTR-FTM08 Performance Results

This appendix contains the results for the unloaded device simulations and hardware tests, and synthetically loaded simulations.

## Simulation based unloaded network

### Unidirectional

The following operational parameters were used in the unidirectional simulations on the unloaded router switch.

| | |
|---|---|
| Sampling Clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |

| | |
|---|---|
| Bit rate | 41.67E+6 |
| Packet overhead | 2 |
| Bit per token | 11 |
| Data bits per token | 8 |

The following tables contain the observed results for physical, interval, and logical addressed packets. The interval and logical addressed packets did not strip the header.

| Payload Packet Size (bytes) | Calculated Minimum Packet Time (seconds) | Measured Packet Time (seconds) | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 1.584E-6 | 2.376E-6 | 2.51E-6 | 2.70E-6 |
| 8 | 2.640E-6 | 3.432E-6 | 3.57E-6 | 3.76E-6 |
| 16 | 4.752E-6 | 5.544E-6 | 5.68E-6 | 5.87E-6 |
| 32 | 8.976E-6 | 9.768E-6 | 9.90E-6 | 10.10E-6 |
| 64 | 17.424E-6 | 18.216E-6 | 18.35E-6 | 18.54E-6 |
| 128 | 34.320E-6 | 35.112E-6 | 35.25E-6 | 35.44E-6 |
| 252 | 67.056E-6 | 67.848E-6 | 67.98E-6 | 68.18E-6 |

| Payload Packet Size (bytes) | Maximum Effective Data Bandwidth (bits per second) | Effective Data Bandwidth (bits per second) | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 20.20E+6 | 13.47E+6 | 12.74E+6 | 11.83E+6 |
| 8 | 24.24E+6 | 18.65E+6 | 17.94E+6 | 17.02E+6 |
| 16 | 26.94E+6 | 23.09E+6 | 22.54E+6 | 21.80E+6 |
| 32 | 28.52E+6 | 26.21E+6 | 25.85E+6 | 25.36E+6 |
| 64 | 29.38E+6 | 28.11E+6 | 27.90E+6 | 27.61E+6 |
| 128 | 29.84E+6 | 29.16E+6 | 29.05E+6 | 28.89E+6 |
| 252 | 30.06E+6 | 29.71E+6 | 29.65E+6 | 29.57E+6 |

| Payload Packet Size (bytes) | % Maximum Effective Data Bandwidth | Percentage Effective Data Bandwidth | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 48.48% | 32.32% | 30.57% | 28.40% |
| 8 | 58.18% | 44.76% | 43.05% | 40.85% |
| 16 | 64.65% | 55.41% | 54.08% | 52.32% |
| 32 | 68.45% | 62.90% | 62.04% | 60.86% |
| 64 | 70.52% | 67.46% | 66.96% | 66.26% |
| 128 | 71.61% | 69.99% | 69.72% | 69.34% |
| 252 | 72.15% | 71.31% | 71.17% | 70.97% |

The following tables contain the observed results of an unloaded network with logical addressed packets for multicast connections, for all ranges of target addresses.

| Packet size = 4 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 2.70E-6 | 11.83E+6 | 28.40% |
| 2 | 2.74E-6 | 11.70E+6 | 28.07% |
| 3 | 2.77E-6 | 11.56E+6 | 27.75% |
| 4 | 2.80E-6 | 11.43E+6 | 27.43% |
| 5 | 2.83E-6 | 11.30E+6 | 27.12% |
| 6 | 2.86E-6 | 11.17E+6 | 26.82% |
| 7 | 2.90E-6 | 11.04E+6 | 26.50% |

| Packet size = 8 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet Duration (bits per second) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 3.76E-6 | 17.02E+6 | 40.85% |
| 2 | 3.79E-6 | 16.88E+6 | 40.51% |
| 3 | 3.82E-6 | 16.74E+6 | 40.17% |
| 4 | 3.86E-6 | 16.60E+6 | 39.83% |
| 5 | 3.89E-6 | 16.46E+6 | 39.51% |
| 6 | 3.92E-6 | 16.33E+6 | 39.18% |
| 7 | 3.95E-6 | 16.19E+6 | 38.87% |

| Packet size = 16 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 5.87E-6 | 21.80E+6 | 52.32% |
| 2 | 5.90E-6 | 21.68E+6 | 52.03% |
| 3 | 5.94E-6 | 21.56E+6 | 51.75% |
| 4 | 5.97E-6 | 21.45E+6 | 51.47% |
| 5 | 6.00E-6 | 21.33E+6 | 51.20% |
| 6 | 6.03E-6 | 21.22E+6 | 50.93% |
| 7 | 6.06E-6 | 21.11E+6 | 50.66% |

| Packet size = 32 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (seconds) | % Effective data bandwidth |
| 1 | 10.10E-6 | 25.36E+6 | 60.86% |
| 2 | 10.13E-6 | 25.28E+6 | 60.66% |
| 3 | 10.16E-6 | 25.20E+6 | 60.47% |
| 4 | 10.19E-6 | 25.12E+6 | 60.28% |
| 5 | 10.22E-6 | 25.04E+6 | 60.09% |
| 6 | 10.26E-6 | 24.96E+6 | 59.91% |
| 7 | 10.29E-6 | 24.88E+6 | 59.71% |

| Packet size = 64 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 18.54E-6 | 27.61E+6 | 66.26% |
| 2 | 18.58E-6 | 27.56E+6 | 66.15% |
| 3 | 18.61E-6 | 27.52E+6 | 66.04% |
| 4 | 18.64E-6 | 27.47E+6 | 65.92% |
| 5 | 18.67E-6 | 27.42E+6 | 65.81% |
| 6 | 18.70E-6 | 27.37E+6 | 65.70% |
| 7 | 18.74E-6 | 27.33E+6 | 65.58% |

| Packet size = 128 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 35.44E-6 | 28.89E+6 | 69.34% |
| 2 | 35.47E-6 | 28.87E+6 | 69.28% |
| 3 | 35.50E-6 | 28.84E+6 | 69.22% |
| 4 | 35.54E-6 | 28.82E+6 | 69.16% |
| 5 | 35.57E-6 | 28.79E+6 | 69.10% |
| 6 | 35.60E-6 | 28.76E+6 | 69.03% |
| 7 | 35.63E-6 | 28.74E+6 | 68.97% |

| Packet size = 252 (bytes) | | | |
|---|---|---|---|
| # Target outputs | Packet duration (seconds) | Data bandwidth (bits per second) | % Effective data bandwidth |
| 1 | 68.18E-6 | 29.57E+6 | 70.97% |
| 2 | 68.21E-6 | 29.56E+6 | 70.93% |
| 3 | 68.24E-6 | 29.54E+6 | 70.90% |
| 4 | 68.27E-6 | 29.53E+6 | 70.87% |
| 5 | 68.30E-6 | 29.52E+6 | 70.84% |
| 6 | 68.34E-6 | 29.50E+6 | 70.80% |
| 7 | 68.37E-6 | 29.49E+6 | 70.77% |

## Hardware based unloaded network

The following operational parameters were used in the both hardware tests on the unloaded router switch.

| | |
|---|---|
| Sampling Clock period | 20.08E-9 |
| Core clock period | 33.33E-9 |

| | |
|---|---|
| Bit rate | 32.00E+6 |
| Packet overhead | 2 |
| Bit per token | 11 |
| Data bits per token | 8 |

### Unidirectional

The following tables contain the observed results for physical, interval, and logical addressed packets. The interval and logical addressed packets did not strip the header.

| Packet Size | Calculated minimum packet Time (seconds) | Measured Packet Time (seconds) | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 2.063E-6 | 2.970E-6 | 3.170E-6 | 3.350E-6 |
| 8 | 3.438E-6 | 4.320E-6 | 4.540E-6 | 4.740E-6 |
| 16 | 6.188E-6 | 7.080E-6 | 7.280E-6 | 7.470E-6 |
| 32 | 11.688E-6 | 12.560E-6 | 12.790E-6 | 12.980E-6 |
| 64 | 22.688E-6 | 23.570E-6 | 23.780E-6 | 24.000E-6 |
| 128 | 44.688E-6 | 45.570E-6 | 45.790E-6 | 46.000E-6 |

| Packet Size | Maximum Bandwidth (bits per second) | Effective Link Bandwidth (bits per second) | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 15.515E+6 | 10.774E+6 | 10.095E+6 | 9.552E+6 |
| 8 | 18.618E+6 | 14.815E+6 | 14.097E+6 | 13.502E+6 |
| 16 | 20.687E+6 | 18.079E+6 | 17.582E+6 | 17.135E+6 |
| 32 | 21.904E+6 | 20.382E+6 | 20.016E+6 | 19.723E+6 |
| 64 | 22.567E+6 | 21.723E+6 | 21.531E+6 | 21.333E+6 |
| 128 | 22.915E+6 | 22.471E+6 | 22.363E+6 | 22.261E+6 |

| Packet Size | Maximum Bandwidth | Percentage Effective Link Bandwidth | | |
|---|---|---|---|---|
| | | Physical (unidirectional) | Interval (unidirectional) | Logical (unidirectional) |
| 4 | 48.48% | 33.67% | 31.55% | 29.85% |
| 8 | 58.18% | 46.30% | 44.05% | 42.19% |
| 16 | 64.65% | 56.50% | 54.95% | 53.55% |
| 32 | 68.45% | 63.69% | 62.55% | 61.63% |
| 64 | 70.52% | 67.88% | 67.28% | 66.67% |
| 128 | 71.61% | 70.22% | 69.88% | 69.57% |

**Bi-directional**

The following tables contain the observed results for physical, interval, and logical addressed packets under bi-directional transfer conditions. The interval and logical addressed packets did not strip the header.

| Packet Size | Calculated minimum packet time (seconds) | Measured Packet Time (seconds) | | |
|---|---|---|---|---|
| | | Physical (bi-directional) | Interval (bi-directional) | Logical (bi-directional) |
| 4 | 2.063E-6 | 2.970E-6 | 3.160E-6 | 3.370E-6 |
| 8 | 3.438E-6 | 4.320E-6 | 4.540E-6 | 4.740E-6 |
| 16 | 6.188E-6 | 7.080E-6 | 7.290E-6 | 7.480E-6 |
| 32 | 11.688E-6 | 12.560E-6 | 12.780E-6 | 12.980E-6 |
| 64 | 22.688E-6 | 23.570E-6 | 23.800E-6 | 24.000E-6 |
| 128 | 44.688E-6 | 45.570E-6 | 45.800E-6 | 45.990E-6 |

| Packet Size | Maximum Bandwidth (bits per second) | Effective Link Bandwidth (bits per second) | | |
|---|---|---|---|---|
| | | Physical (bi-directional) | Interval (bi-directional) | Logical (bi-directional) |
| 4 | 15.515E+6 | 10.774E+6 | 10.127E+6 | 9.496E+6 |
| 8 | 18.618E+6 | 14.815E+6 | 14.097E+6 | 13.502E+6 |
| 16 | 20.687E+6 | 18.079E+6 | 17.558E+6 | 17.112E+6 |
| 32 | 21.904E+6 | 20.382E+6 | 20.031E+6 | 19.723E+6 |
| 64 | 22.567E+6 | 21.723E+6 | 21.513E+6 | 21.333E+6 |
| 128 | 22.915E+6 | 22.471E+6 | 22.358E+6 | 22.266E+6 |

| Packet Size | Maximum Bandwidth | Percentage Effective Link Bandwidth | | |
|---|---|---|---|---|
| | | Physical (bi-directional) | Interval (bi-directional) | Logical (bi-directional) |
| 4 | 48.48% | 33.67% | 31.65% | 29.67% |
| 8 | 58.18% | 46.30% | 44.05% | 42.19% |
| 16 | 64.65% | 56.50% | 54.87% | 53.48% |
| 32 | 68.45% | 63.69% | 62.60% | 61.63% |
| 64 | 70.52% | 67.88% | 67.23% | 66.67% |
| 128 | 71.61% | 70.22% | 69.87% | 69.58% |

## Simulation based synthetically loaded network

The following tables contain the observed results for physical, interval, and logical addressed packets for a synthetically loaded router switch. The interval and logical addressed packets did not strip the header.

**Physical - 16 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 4.35E-3 | 364.14E+6 | 235.41E+6 | 97.11% |
| 90 | 4.82E-3 | 328.35E+6 | 212.27E+6 | 87.56% |
| 80 | 5.47E-3 | 289.82E+6 | 187.36E+6 | 77.29% |
| 70 | 6.23E-3 | 254.36E+6 | 164.43E+6 | 67.83% |
| 60 | 7.24E-3 | 218.68E+6 | 141.37E+6 | 58.31% |
| 50 | 8.59E-3 | 184.32E+6 | 119.16E+6 | 49.15% |
| 40 | 10.73E-3 | 147.66E+6 | 95.46E+6 | 39.38% |
| 30 | 14.28E-3 | 110.89E+6 | 71.69E+6 | 29.57% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 12.737E-6 | 7.335E-3 | 215.94E+6 | 139.60E+6 |
| 90 | 12.756E-6 | 7.295E-3 | 217.14E+6 | 140.38E+6 |
| 80 | 12.638E-6 | 7.292E-3 | 217.23E+6 | 140.43E+6 |
| 70 | 12.767E-6 | 7.398E-3 | 214.11E+6 | 138.42E+6 |
| 60 | 12.145E-6 | 7.279E-3 | 217.61E+6 | 140.68E+6 |
| 50 | 8.057E-6 | 8.566E-3 | 184.93E+6 | 119.55E+6 |
| 40 | 6.919E-6 | 10.726E-3 | 147.68E+6 | 95.47E+6 |
| 30 | 6.387E-6 | 14.282E-3 | 110.91E+6 | 71.70E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 3.015 | 64.78% | 57.58% |
| 90 | 3.020 | 65.14% | 57.90% |
| 80 | 2.992 | 65.17% | 57.93% |
| 70 | 3.023 | 64.23% | 57.10% |
| 60 | 2.875 | 65.28% | 58.03% |
| 50 | 1.907 | 55.48% | 49.31% |
| 40 | 1.638 | 44.31% | 39.38% |
| 30 | 1.512 | 33.27% | 29.58% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 4.224E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 16 |
| Bit per token | 11 |
| Number of bits per test | 1584000 |
| Data bits per test | 1024000 |

**Physical - 32 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 8.65E-3 | 345.95E+6 | 236.80E+6 | 97.68% |
| 90 | 9.61E-3 | 311.36E+6 | 213.12E+6 | 87.91% |
| 80 | 10.71E-3 | 279.43E+6 | 191.27E+6 | 78.90% |
| 70 | 12.22E-3 | 244.78E+6 | 167.55E+6 | 69.12% |
| 60 | 14.24E-3 | 210.05E+6 | 143.78E+6 | 59.31% |
| 50 | 17.07E-3 | 175.31E+6 | 120.00E+6 | 49.50% |
| 40 | 21.31E-3 | 140.40E+6 | 96.10E+6 | 39.64% |
| 30 | 28.38E-3 | 105.41E+6 | 72.15E+6 | 29.76% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 16.938E-6 | 14.100E-3 | 212.20E+6 | 145.25E+6 |
| 90 | 16.743E-6 | 13.857E-3 | 215.92E+6 | 147.80E+6 |
| 80 | 16.905E-6 | 14.256E-3 | 209.88E+6 | 143.66E+6 |
| 70 | 16.922E-6 | 14.024E-3 | 213.34E+6 | 146.03E+6 |
| 60 | 16.404E-6 | 14.271E-3 | 209.66E+6 | 143.51E+6 |
| 50 | 13.600E-6 | 17.073E-3 | 175.25E+6 | 119.96E+6 |
| 40 | 12.106E-6 | 21.317E-3 | 140.36E+6 | 96.08E+6 |
| 30 | 11.302E-6 | 28.390E-3 | 105.39E+6 | 72.14E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 2.005 | 63.66% | 59.91% |
| 90 | 1.982 | 64.78% | 60.97% |
| 80 | 2.001 | 62.96% | 59.26% |
| 70 | 2.003 | 64.00% | 60.24% |
| 60 | 1.942 | 62.90% | 59.20% |
| 50 | 1.610 | 52.57% | 49.48% |
| 40 | 1.433 | 42.11% | 39.63% |
| 30 | 1.338 | 31.62% | 29.76% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 8.448E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 32 |
| Bit per token | 11 |
| Number of bits per test | 2992000 |
| Data bits per test | 2048000 |

**Physical - 64 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 17.20E-3 | 337.62E+6 | 238.10E+6 | 98.22% |
| 90 | 19.08E-3 | 304.34E+6 | 214.63E+6 | 88.54% |
| 80 | 21.38E-3 | 271.69E+6 | 191.60E+6 | 79.04% |
| 70 | 24.40E-3 | 238.03E+6 | 167.87E+6 | 69.25% |
| 60 | 28.66E-3 | 202.67E+6 | 142.93E+6 | 58.96% |
| 50 | 34.30E-3 | 169.33E+6 | 119.42E+6 | 49.26% |
| 40 | 42.61E-3 | 136.32E+6 | 96.14E+6 | 39.66% |
| 30 | 56.71E-3 | 102.41E+6 | 72.22E+6 | 29.79% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 29.951E-6 | 27.215E-3 | 213.41E+6 | 150.50E+6 |
| 90 | 30.099E-6 | 27.368E-3 | 212.22E+6 | 149.67E+6 |
| 80 | 29.966E-6 | 27.504E-3 | 211.17E+6 | 148.92E+6 |
| 70 | 29.745E-6 | 27.391E-3 | 212.04E+6 | 149.54E+6 |
| 60 | 28.739E-6 | 28.659E-3 | 202.66E+6 | 142.92E+6 |
| 50 | 24.951E-6 | 34.301E-3 | 169.32E+6 | 119.41E+6 |
| 40 | 22.802E-6 | 42.607E-3 | 136.31E+6 | 96.13E+6 |
| 30 | 21.248E-6 | 56.713E-3 | 102.41E+6 | 72.22E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.773 | 64.02% | 62.08% |
| 90 | 1.781 | 63.67% | 61.74% |
| 80 | 1.774 | 63.35% | 61.43% |
| 70 | 1.760 | 63.61% | 61.68% |
| 60 | 1.701 | 60.80% | 58.96% |
| 50 | 1.477 | 50.80% | 49.26% |
| 40 | 1.350 | 40.89% | 39.66% |
| 30 | 1.258 | 30.72% | 29.79% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 16.896E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 5808000 |
| Data bits per test | 4096000 |

**Physical - 128 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 34.27E-3 | 333.84E+6 | 239.06E+6 | 98.61% |
| 90 | 38.36E-3 | 298.24E+6 | 213.56E+6 | 88.10% |
| 80 | 43.05E-3 | 265.71E+6 | 190.27E+6 | 78.49% |
| 70 | 49.08E-3 | 233.08E+6 | 166.90E+6 | 68.85% |
| 60 | 57.13E-3 | 200.24E+6 | 143.39E+6 | 59.15% |
| 50 | 68.40E-3 | 167.25E+6 | 119.77E+6 | 49.40% |
| 40 | 85.49E-3 | 133.82E+6 | 95.82E+6 | 39.53% |
| 30 | 113.66E-3 | 100.65E+6 | 72.07E+6 | 29.73% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 56.743E-6 | 54.668E-3 | 209.26E+6 | 149.85E+6 |
| 90 | 56.137E-6 | 53.706E-3 | 213.01E+6 | 152.53E+6 |
| 80 | 55.907E-6 | 53.432E-3 | 214.10E+6 | 153.32E+6 |
| 70 | 56.492E-6 | 54.362E-3 | 210.44E+6 | 150.69E+6 |
| 60 | 53.530E-6 | 57.133E-3 | 200.23E+6 | 143.38E+6 |
| 50 | 46.934E-6 | 68.402E-3 | 167.25E+6 | 119.76E+6 |
| 40 | 43.348E-6 | 85.493E-3 | 133.81E+6 | 95.82E+6 |
| 30 | 40.753E-6 | 113.664E-3 | 100.65E+6 | 72.07E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.679 | 62.78% | 61.81% |
| 90 | 1.661 | 63.90% | 62.92% |
| 80 | 1.654 | 64.23% | 63.24% |
| 70 | 1.672 | 63.13% | 62.16% |
| 60 | 1.584 | 60.07% | 59.15% |
| 50 | 1.389 | 50.17% | 49.40% |
| 40 | 1.283 | 40.14% | 39.53% |
| 30 | 1.206 | 30.19% | 29.73% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 33.792E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 11440000 |
| Data bits per test | 8192000 |

**Interval - 16 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 4.34E-3 | 364.64E+6 | 235.73E+6 | 97.24% |
| 90 | 4.83E-3 | 327.84E+6 | 211.93E+6 | 87.42% |
| 80 | 5.42E-3 | 292.01E+6 | 188.78E+6 | 77.87% |
| 70 | 6.20E-3 | 255.31E+6 | 165.05E+6 | 68.08% |
| 60 | 7.22E-3 | 219.38E+6 | 141.82E+6 | 58.50% |
| 50 | 8.64E-3 | 183.27E+6 | 118.48E+6 | 48.87% |
| 40 | 10.70E-3 | 148.08E+6 | 95.73E+6 | 39.49% |
| 30 | 14.25E-3 | 111.13E+6 | 71.84E+6 | 29.64% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 13.596E-6 | 7.668E-3 | 206.58E+6 | 133.55E+6 |
| 90 | 13.414E-6 | 7.649E-3 | 207.08E+6 | 133.87E+6 |
| 80 | 13.567E-6 | 7.697E-3 | 205.78E+6 | 133.03E+6 |
| 70 | 13.565E-6 | 7.679E-3 | 206.27E+6 | 133.35E+6 |
| 60 | 13.499E-6 | 7.677E-3 | 206.34E+6 | 133.39E+6 |
| 50 | 8.967E-6 | 8.644E-3 | 183.24E+6 | 118.46E+6 |
| 40 | 7.188E-6 | 10.698E-3 | 148.06E+6 | 95.72E+6 |
| 30 | 6.663E-6 | 14.255E-3 | 111.12E+6 | 71.84E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 3.219 | 61.98% | 55.09% |
| 90 | 3.176 | 62.12% | 55.22% |
| 80 | 3.212 | 61.73% | 54.88% |
| 70 | 3.211 | 61.88% | 55.00% |
| 60 | 3.196 | 61.90% | 55.02% |
| 50 | 2.123 | 54.97% | 48.86% |
| 40 | 1.702 | 44.42% | 39.48% |
| 30 | 1.577 | 33.34% | 29.63% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 4.224E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 16 |
| Bit per token | 11 |
| Number of bits per test | 1584000 |
| Data bits per test | 1024000 |

**Interval - 32 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 8.73E-3 | 342.85E+6 | 234.68E+6 | 96.81% |
| 90 | 9.67E-3 | 309.42E+6 | 211.79E+6 | 87.36% |
| 80 | 10.78E-3 | 277.44E+6 | 189.90E+6 | 78.34% |
| 70 | 12.23E-3 | 244.58E+6 | 167.41E+6 | 69.06% |
| 60 | 14.22E-3 | 210.45E+6 | 144.05E+6 | 59.42% |
| 50 | 17.09E-3 | 175.07E+6 | 119.84E+6 | 49.43% |
| 40 | 21.33E-3 | 140.25E+6 | 96.00E+6 | 39.60% |
| 30 | 28.41E-3 | 105.33E+6 | 72.10E+6 | 29.74% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 17.301E-6 | 14.341E-3 | 208.64E+6 | 142.81E+6 |
| 90 | 17.257E-6 | 14.320E-3 | 208.94E+6 | 143.02E+6 |
| 80 | 17.304E-6 | 14.374E-3 | 208.16E+6 | 142.48E+6 |
| 70 | 17.210E-6 | 14.207E-3 | 210.60E+6 | 144.15E+6 |
| 60 | 17.255E-6 | 14.441E-3 | 207.19E+6 | 141.82E+6 |
| 50 | 14.435E-6 | 17.092E-3 | 175.05E+6 | 119.82E+6 |
| 40 | 12.571E-6 | 21.335E-3 | 140.24E+6 | 95.99E+6 |
| 30 | 11.633E-6 | 28.408E-3 | 105.32E+6 | 72.09E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 2.048 | 62.59% | 58.91% |
| 90 | 2.043 | 62.68% | 59.00% |
| 80 | 2.048 | 62.45% | 58.77% |
| 70 | 2.037 | 63.18% | 59.46% |
| 60 | 2.042 | 62.16% | 58.50% |
| 50 | 1.709 | 52.52% | 49.43% |
| 40 | 1.488 | 42.07% | 39.60% |
| 30 | 1.377 | 31.60% | 29.74% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 8.448E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 32 |
| Bit per token | 11 |
| Number of bits per test | 2992000 |
| Data bits per test | 2048000 |

**Interval - 64 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 17.09E-3 | 339.89E+6 | 239.70E+6 | 98.88% |
| 90 | 19.17E-3 | 302.93E+6 | 213.64E+6 | 88.13% |
| 80 | 21.52E-3 | 269.84E+6 | 190.30E+6 | 78.50% |
| 70 | 24.55E-3 | 236.61E+6 | 166.87E+6 | 68.83% |
| 60 | 28.73E-3 | 202.14E+6 | 142.56E+6 | 58.80% |
| 50 | 34.37E-3 | 168.96E+6 | 119.16E+6 | 49.15% |
| 40 | 42.84E-3 | 135.58E+6 | 95.61E+6 | 39.44% |
| 30 | 56.53E-3 | 102.74E+6 | 72.46E+6 | 29.89% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 30.614E-6 | 27.772E-3 | 209.13E+6 | 147.49E+6 |
| 90 | 30.191E-6 | 27.475E-3 | 211.39E+6 | 149.08E+6 |
| 80 | 30.504E-6 | 27.774E-3 | 209.12E+6 | 147.48E+6 |
| 70 | 30.397E-6 | 27.585E-3 | 210.55E+6 | 148.48E+6 |
| 60 | 29.462E-6 | 28.734E-3 | 202.13E+6 | 142.55E+6 |
| 50 | 25.152E-6 | 34.376E-3 | 168.95E+6 | 119.15E+6 |
| 40 | 22.840E-6 | 42.840E-3 | 135.57E+6 | 95.61E+6 |
| 30 | 21.456E-6 | 56.532E-3 | 102.74E+6 | 72.46E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.812 | 62.74% | 60.84% |
| 90 | 1.787 | 63.42% | 61.50% |
| 80 | 1.805 | 62.74% | 60.83% |
| 70 | 1.799 | 63.16% | 61.25% |
| 60 | 1.744 | 60.64% | 58.80% |
| 50 | 1.489 | 50.69% | 49.15% |
| 40 | 1.352 | 40.67% | 39.44% |
| 30 | 1.270 | 30.82% | 29.89% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 16.896E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 5808000 |
| Data bits per test | 4096000 |

**Interval - 128 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 33.92E-3 | 337.23E+6 | 241.49E+6 | 99.61% |
| 90 | 37.68E-3 | 303.61E+6 | 217.41E+6 | 89.68% |
| 80 | 42.40E-3 | 269.79E+6 | 193.19E+6 | 79.69% |
| 70 | 48.44E-3 | 236.17E+6 | 169.12E+6 | 69.76% |
| 60 | 57.44E-3 | 199.16E+6 | 142.62E+6 | 58.83% |
| 50 | 68.71E-3 | 166.50E+6 | 119.23E+6 | 49.18% |
| 40 | 85.61E-3 | 133.63E+6 | 95.69E+6 | 39.47% |
| 30 | 113.15E-3 | 101.10E+6 | 72.40E+6 | 29.86% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 57.054E-6 | 54.439E-3 | 210.14E+6 | 150.48E+6 |
| 90 | 57.061E-6 | 54.667E-3 | 209.27E+6 | 149.85E+6 |
| 80 | 56.347E-6 | 54.003E-3 | 211.84E+6 | 151.69E+6 |
| 70 | 56.680E-6 | 54.215E-3 | 211.01E+6 | 151.10E+6 |
| 60 | 53.810E-6 | 57.442E-3 | 199.16E+6 | 142.61E+6 |
| 50 | 48.109E-6 | 68.710E-3 | 166.50E+6 | 119.23E+6 |
| 40 | 44.151E-6 | 85.613E-3 | 133.62E+6 | 95.69E+6 |
| 30 | 40.976E-6 | 113.152E-3 | 101.10E+6 | 72.40E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.688 | 63.04% | 62.07% |
| 90 | 1.689 | 62.78% | 61.81% |
| 80 | 1.667 | 63.55% | 62.57% |
| 70 | 1.677 | 63.30% | 62.33% |
| 60 | 1.592 | 59.75% | 58.83% |
| 50 | 1.424 | 49.95% | 49.18% |
| 40 | 1.307 | 40.09% | 39.47% |
| 30 | 1.213 | 30.33% | 29.86% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 33.792E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 11440000 |
| Data bits per test | 8192000 |

**Logical - 16 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 4.34E-3 | 364.99E+6 | 235.96E+6 | 97.33% |
| 90 | 4.81E-3 | 329.04E+6 | 212.71E+6 | 87.74% |
| 80 | 5.39E-3 | 293.77E+6 | 189.91E+6 | 78.34% |
| 70 | 6.15E-3 | 257.39E+6 | 166.39E+6 | 68.64% |
| 60 | 7.17E-3 | 220.92E+6 | 142.81E+6 | 58.91% |
| 50 | 8.59E-3 | 184.34E+6 | 119.17E+6 | 49.16% |
| 40 | 10.73E-3 | 147.62E+6 | 95.43E+6 | 39.37% |
| 30 | 14.29E-3 | 110.87E+6 | 71.67E+6 | 29.57% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 14.222E-6 | 7.906E-3 | 200.34E+6 | 129.51E+6 |
| 90 | 14.219E-6 | 7.870E-3 | 201.26E+6 | 130.11E+6 |
| 80 | 14.327E-6 | 7.994E-3 | 198.15E+6 | 128.10E+6 |
| 70 | 14.363E-6 | 7.923E-3 | 199.92E+6 | 129.24E+6 |
| 60 | 14.108E-6 | 7.965E-3 | 198.86E+6 | 128.56E+6 |
| 50 | 9.288E-6 | 8.732E-3 | 181.41E+6 | 117.27E+6 |
| 40 | 7.441E-6 | 10.865E-3 | 145.78E+6 | 94.24E+6 |
| 30 | 6.883E-6 | 14.297E-3 | 110.79E+6 | 71.62E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 3.367 | 60.10% | 53.42% |
| 90 | 3.366 | 60.38% | 53.67% |
| 80 | 3.392 | 59.45% | 52.84% |
| 70 | 3.400 | 59.98% | 53.31% |
| 60 | 3.340 | 59.66% | 53.03% |
| 50 | 2.199 | 54.42% | 48.38% |
| 40 | 1.762 | 43.73% | 38.88% |
| 30 | 1.630 | 33.24% | 29.55% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 4.224E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 16 |
| Bit per token | 11 |
| Number of bits per test | 1584000 |
| Data bits per test | 1024000 |

**Logical - 32 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 8.55E-3 | 349.96E+6 | 239.55E+6 | 98.81% |
| 90 | 9.60E-3 | 311.51E+6 | 213.22E+6 | 87.95% |
| 80 | 10.78E-3 | 277.45E+6 | 189.91E+6 | 78.34% |
| 70 | 12.23E-3 | 244.68E+6 | 167.48E+6 | 69.09% |
| 60 | 14.25E-3 | 209.98E+6 | 143.73E+6 | 59.29% |
| 50 | 17.08E-3 | 175.19E+6 | 119.92E+6 | 49.47% |
| 40 | 21.29E-3 | 140.54E+6 | 96.20E+6 | 39.68% |
| 30 | 28.36E-3 | 105.49E+6 | 72.21E+6 | 29.79% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 17.819E-6 | 14.535E-3 | 205.85E+6 | 140.90E+6 |
| 90 | 17.820E-6 | 14.630E-3 | 204.51E+6 | 139.98E+6 |
| 80 | 17.852E-6 | 14.600E-3 | 204.94E+6 | 140.28E+6 |
| 70 | 17.757E-6 | 14.565E-3 | 205.43E+6 | 140.61E+6 |
| 60 | 17.741E-6 | 14.560E-3 | 205.49E+6 | 140.66E+6 |
| 50 | 14.420E-6 | 17.080E-3 | 175.17E+6 | 119.91E+6 |
| 40 | 12.810E-6 | 21.291E-3 | 140.53E+6 | 96.19E+6 |
| 30 | 11.812E-6 | 28.364E-3 | 105.49E+6 | 72.20E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 2.109 | 61.75% | 58.12% |
| 90 | 2.109 | 61.35% | 57.74% |
| 80 | 2.113 | 61.48% | 57.86% |
| 70 | 2.102 | 61.63% | 58.00% |
| 60 | 2.100 | 61.65% | 58.02% |
| 50 | 1.707 | 52.55% | 49.46% |
| 40 | 1.516 | 42.16% | 39.68% |
| 30 | 1.398 | 31.65% | 29.78% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 8.448E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 32 |
| Bit per token | 11 |
| Number of bits per test | 2992000 |
| Data bits per test | 2048000 |

**Logical - 64 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 17.40E-3 | 333.79E+6 | 235.40E+6 | 97.10% |
| 90 | 19.29E-3 | 301.04E+6 | 212.30E+6 | 87.57% |
| 80 | 21.34E-3 | 272.11E+6 | 191.90E+6 | 79.16% |
| 70 | 24.37E-3 | 238.36E+6 | 168.10E+6 | 69.34% |
| 60 | 28.40E-3 | 204.53E+6 | 144.24E+6 | 59.50% |
| 50 | 33.90E-3 | 171.33E+6 | 120.83E+6 | 49.84% |
| 40 | 42.36E-3 | 137.10E+6 | 96.69E+6 | 39.88% |
| 30 | 56.47E-3 | 102.85E+6 | 72.54E+6 | 29.92% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 31.172E-6 | 28.558E-3 | 203.38E+6 | 143.43E+6 |
| 90 | 31.231E-6 | 28.422E-3 | 204.35E+6 | 144.11E+6 |
| 80 | 31.107E-6 | 28.718E-3 | 202.24E+6 | 142.63E+6 |
| 70 | 30.952E-6 | 28.123E-3 | 206.52E+6 | 145.65E+6 |
| 60 | 30.132E-6 | 28.399E-3 | 204.51E+6 | 144.23E+6 |
| 50 | 25.860E-6 | 33.901E-3 | 171.32E+6 | 120.82E+6 |
| 40 | 23.240E-6 | 42.365E-3 | 137.10E+6 | 96.68E+6 |
| 30 | 21.658E-6 | 56.470E-3 | 102.85E+6 | 72.53E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.845 | 61.01% | 59.16% |
| 90 | 1.848 | 61.30% | 59.45% |
| 80 | 1.841 | 60.67% | 58.83% |
| 70 | 1.832 | 61.96% | 60.08% |
| 60 | 1.783 | 61.35% | 59.50% |
| 50 | 1.531 | 51.40% | 49.84% |
| 40 | 1.375 | 41.13% | 39.88% |
| 30 | 1.282 | 30.86% | 29.92% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 16.896E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 5808000 |
| Data bits per test | 4096000 |

**Logical - 128 bytes per packet**

| Workload (%) | Required test duration (seconds) | Required raw throughput (bits per second) | Required data throughput (bits per second) | Required data throughput (%) |
|---|---|---|---|---|
| 100 | 34.58E-3 | 330.86E+6 | 236.92E+6 | 97.73% |
| 90 | 38.33E-3 | 298.44E+6 | 213.71E+6 | 88.15% |
| 80 | 43.21E-3 | 264.75E+6 | 189.59E+6 | 78.20% |
| 70 | 49.25E-3 | 232.30E+6 | 166.35E+6 | 68.62% |
| 60 | 57.30E-3 | 199.67E+6 | 142.98E+6 | 58.98% |
| 50 | 68.96E-3 | 165.88E+6 | 118.79E+6 | 49.00% |
| 40 | 85.87E-3 | 133.23E+6 | 95.40E+6 | 39.35% |
| 30 | 114.04E-3 | 100.32E+6 | 71.84E+6 | 29.63% |

| Workload (%) | Average packet time (seconds) | Observed test duration (seconds) | Observed throughput (bits per second) | Observed data throughput (bits per second) |
|---|---|---|---|---|
| 100 | 57.452E-6 | 54.814E-3 | 208.71E+6 | 149.45E+6 |
| 90 | 57.591E-6 | 55.329E-3 | 206.76E+6 | 148.06E+6 |
| 80 | 57.060E-6 | 54.871E-3 | 208.49E+6 | 149.30E+6 |
| 70 | 57.258E-6 | 55.502E-3 | 206.12E+6 | 147.60E+6 |
| 60 | 54.311E-6 | 57.298E-3 | 199.66E+6 | 142.97E+6 |
| 50 | 47.575E-6 | 68.966E-3 | 165.88E+6 | 118.78E+6 |
| 40 | 44.045E-6 | 85.869E-3 | 133.23E+6 | 95.40E+6 |
| 30 | 40.857E-6 | 114.040E-3 | 100.32E+6 | 71.83E+6 |

| Workload (%) | Normalised average packet time (seconds) | Observed throughput (%) | Observed data throughput (%) |
|---|---|---|---|
| 100 | 1.700 | 62.61% | 61.65% |
| 90 | 1.704 | 62.03% | 61.08% |
| 80 | 1.689 | 62.55% | 61.58% |
| 70 | 1.694 | 61.84% | 60.88% |
| 60 | 1.607 | 59.90% | 58.98% |
| 50 | 1.408 | 49.76% | 49.00% |
| 40 | 1.303 | 39.97% | 39.35% |
| 30 | 1.209 | 30.09% | 29.63% |

| | |
|---|---|
| Sample clock period | 16.00E-9 |
| Core clock period | 32.00E-9 |
| Number of links | 8 |
| System throughput | 333.33E+6 |
| System data throughput | 242.42E+6 |
| Minimum Packet Time | 33.792E-6 |

| | |
|---|---|
| Number of messages | 8000 |
| Packet overhead | 2 |
| Tokens per message | 128 |
| Bit per token | 11 |
| Number of bits per test | 11440000 |
| Data bits per test | 8192000 |

# Appendix D : Analysis Utilities

This appendix contains the source code for the test vector generation program and results analysis program, which were used in the synthetic load simulations.

### Program listing for poisson.cpp

```
/*
Filename : poisson.cpp
Author   : Robin Hotchkiss
Date     : Autumn 1999
Version  : 1.00
Description :
   Generates 6 files of test vectors, each at different injection rates.
   30,40,50,60,70,80 (approximate)

   70% of messages are sent within 30% of the injection rate.
*/
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <time.h>
#include <math.h>

#define NUM_LINKS 8
#define DATA_PER_MSG 16
#define ROUTING_OVERHEAD 2
#define TOKENS_PER_MSG DATA_PER_MSG+ROUTING_OVERHEAD

#define BITS_PER_TOKEN 11
#define CORECLK_MULTIPLIER 0.75
#define CYCLES_PER_MSG (BITS_PER_TOKEN*TOKENS_PER_MSG)*CORECLK_MULTIPLIER

#define MSGS_PER_LINK 1000

void main( int argc, char *argv[])
{
ofstream outFile_of;
time_t t;
char filename_c255[255];
int msgsPerLink_i = MSGS_PER_LINK;
int numLinks_i = NUM_LINKS;
int randomVal_i;
int requiredWorkload_i;
int sequenceID_i16[ 16];
int lastDest_i;
int destLink_i;
int destSrcData_i;
int msgID_i;
double variance_d;
double repeatProb_d;
double prob_d;
double scalingFactor_d;
double percentVariance_d = 0.30;
double percentMsg_d = (1 - 0.70);
double intervalCentre_d;
unsigned long maxTxTime_ul = 0;
unsigned long minTxTime_ul = 32000;
long double transmissionTime_ld;
int addrMode_i = 0;


   //addrMode_i = 0; // physical
   //addrMode_i = 1; // interval
   addrMode_i = 2; // logical

   // create for 30, 40, 50, 60, and 70
   for( requiredWorkload_i = 30; requiredWorkload_i <= 100; requiredWorkload_i+=10){
      // zero max and min watches
      maxTxTime_ul = 0;
      minTxTime_ul = 32000;
```

```
        // calculate the average interval point
        intervalCentre_d = (1/( (double) requiredWorkload_i));
        intervalCentre_d *= CYCLES_PER_MSG;
        intervalCentre_d *= 100;

        // scaling factor give 70% of messages within +/- 30% variance of required msg time
        scalingFactor_d = (-1) * (log( percentMsg_d)/ (percentVariance_d*CYCLES_PER_MSG));

        // reset the seed for the random generator
        srand( (unsigned) time(&t));

        // generate output filename and open file
        sprintf( filename_c255, "p_vt%d.txt", requiredWorkload_i);
        outFile_of.open( filename_c255);
        if( outFile_of.fail()){
              cout << "Couldn't generate output file\n";
              exit(0);
        }

        // include vector file information
        outFile_of << "-- test vectors at " << requiredWorkload_i << "% injection rate.\n";
        outFile_of << "-- poisson load of " << msgsPerLink_i*numLinks_i << " messages.\n";
        outFile_of << "-- " << numLinks_i << " Links. Bytes/msg = " << TOKENS_PER_MSG <<
".\n";
        outFile_of << "-- Scaling factor of " << (1 - percentMsg_d) * 100 << "% of msgs\n";
        outFile_of << "-- within +/-" << percentVariance_d*100 << "% variance of required msg
time.\n";
        if( addrMode_i == 0){
           outFile_of << "-- Addressing mode = physical\n";
        }
        else if( addrMode_i == 2){
           outFile_of << "-- Addressing mode = logical\n";
        }
        else{
           outFile_of << "-- Addressing mode = unicast\n";
        }

        // for each source, create a set of messages
        for( int src_i = 0; src_i < numLinks_i; src_i++){
           // reset the sequence ID
           for( int index = 0; index < 16; index++){
              sequenceID_i16[ index] = 0;
           }
           // reset the start transmission time
           transmissionTime_ld = 0.0;

           // preset last dest
           randomVal_i = rand();// generates a number between 0 & 32767
           prob_d = (((double) randomVal_i) / 32768.0)*numLinks_i;
           lastDest_i = (int)prob_d;

           // create the n messages
           for( int msgCnt_i = 0; msgCnt_i < msgsPerLink_i; msgCnt_i++){

              // generate the message interval time
              randomVal_i = rand();
              prob_d = (((double) randomVal_i+1) / 32768.0)*2;
              if( prob_d <= 1){
                 variance_d = (-1) * ( log( prob_d) / scalingFactor_d);
              }
              else if( prob_d <= 2){
                 variance_d = ( log( (prob_d-1)) / scalingFactor_d);
              }
              else{
                 cout << "\nError : Invalid probability!\n";
                 exit(0);
              }
              // update the message transmission time
              transmissionTime_ld += (intervalCentre_d + variance_d);

              // generate the destination - 30% chance of repeated destination
              randomVal_i = rand();     // generates a number between 0 & 32767
              repeatProb_d = (((double) randomVal_i) / 32768.0);
              do{
                 randomVal_i = rand();  // generates a number between 0 & 32767
                 prob_d = (((double) randomVal_i) / 32768.0)*numLinks_i;
```

```
            destLink_i = (int)prob_d;
         }while( (destLink_i == lastDest_i) && (repeatProb_d > 0.3) || (destLink_i ==
src_i));

         // format the first packet byte
         destSrcData_i = src_i << 4;
         destSrcData_i = destSrcData_i + destLink_i;

         // format the second packet byte
         msgID_i = src_i << 4;
         msgID_i = msgID_i + sequenceID_i16[ destLink_i];

         // modify the msgID for the next
         if( sequenceID_i16[ destLink_i] >= 15){
            sequenceID_i16[ destLink_i] = 0;
         }
         else{
            sequenceID_i16[ destLink_i]++;
         }

         // update last destination
         lastDest_i = destLink_i;

         // write the message out to file
         outFile_of << ": src =" << src_i;                // message source
         if( addrMode_i == 0){ // physical
            outFile_of << ": DHDR=" << destLink_i;          // routing header
            outFile_of << ": XHDR=" << destSrcData_i;       // src-dest ID
            outFile_of << ": size=" << (DATA_PER_MSG-1);        // #bytes in msg
         }
         else if( addrMode_i == 2){ // logical
            outFile_of << ": MHDR=" << destLink_i;          // routing header
            outFile_of << ": size=" << DATA_PER_MSG;        // #bytes in msg
         }
         else{ // unicast
            outFile_of << ": UHDR=" << destLink_i;          // routing header
            outFile_of << ": size=" << DATA_PER_MSG;        // #bytes in msg
         }
         outFile_of << ": valu=" << msgID_i;             // byte value
         outFile_of << ": EoPKT";                       // end delimeter
         outFile_of << ": time=" << ((unsigned long) transmissionTime_ld) << "\n";
// time of tx


         // find earliest transmission
         if( (unsigned long) transmissionTime_ld < minTxTime_ul){
            minTxTime_ul = (unsigned long) transmissionTime_ld;
         }
      }
      // find latest transmission
      if( (unsigned long) transmissionTime_ld > maxTxTime_ul){
         maxTxTime_ul = (unsigned long) transmissionTime_ld;
      }
   }
   outFile_of << "endData\n";
   outFile_of << "-- First tx time = " << minTxTime_ul << " cycles.\n";
   outFile_of << "-- Last tx time = " << maxTxTime_ul << " cycles.\n\n";

   cout << "Closing " << filename_c255 << "\n";
   outFile_of.close();
}
cout << "\nProcess complete.\n";
}
```

## Program listing for msglat.cpp

```
/*
Filename : msglat.cpp
Author   : Robin Hotchkiss
Date     : Autumn 1999
Version  : 1.02
Description :
   Reads in the test results from the VHDL testbench
   and calculates the average message time.
   It also presents the shortest and longest
   message time, as well as the duration of the
   test.

Modifications :
   8/10/98 : included the time based matching

   16/12/99 modified to work with new results file from tstbench
       tx file format = destID seqID txCount pktCount
       rx file format = srcID seqID rxCount pktCount
*/
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define STARTFILE  "txLink"

#define ENDFILE    "rxLink"

#define MAX_LINKS    16
#define SRCLINKS   16
#define C_PERIOD   24


int main(int argc, char *argv[])
{
ifstream rxFileHandle_if;
ifstream txFileHandle_if;
ofstream resFileHandle_of;
char buffer_s255[255];
int fileMode_i;
int numLinks_i;
int clkPeriod_i;
int paramIndex_i;
int destMsgID_i;
int destMsgSeq_i;
int txMsgCnt_i;
int srcMsgID_i;
int srcMsgSeq_i;
int rxMsgCnt_i;
int rxPktCnt_i;
int txPktCnt_i;
int byteStrip_i;
int strSize_i;
long maxTxCnt_l;
long minTxCnt_l;
unsigned long firstTx_ul;
unsigned long lastRx_ul;
long transmissionCnt_l;
unsigned long msgCnt_ul;
unsigned long totalTxCnt_ul;
char txFilenameStem_s256[256];
char rxFilenameStem_s256[256];
char preEmpt_s8[8];


   if( argc > 1){
      numLinks_i = (-1);
      sprintf( txFilenameStem_s256, "%s", "NULL");
      sprintf( rxFilenameStem_s256, "%s", "NULL");
      sprintf( preEmpt_s8, "%s", "NULL");
      clkPeriod_i = (-1);
      byteStrip_i = (-1);
```

```
// parse the input parameters
for( paramIndex_i = 1; paramIndex_i < argc; paramIndex_i++){
   // odd and even. all parameter have pre-empts
   if( (paramIndex_i % 2) > 0 ){ // || paramIndex_i == 3 || paramIndex_i == 5){
      // pre-empt
      if( strcmp( argv[ paramIndex_i], "-num_link") == 0){
         if( numLinks_i == (-1)){
            sprintf( preEmpt_s8, "%s", "numlink");
         }
         else{
            cout << "Invalid arguments\n\n";
            paramIndex_i = argc +2;
         }
      }
      else if( strcmp( argv[ paramIndex_i], "-tx") == 0){
         if( strcmp( txFilenameStem_s256, "NULL") == 0){
            sprintf( preEmpt_s8, "%s", "tx");
         }
         else{
            cout << "Invalid arguments\n\n";
            paramIndex_i = argc +2;
         }
      }
      else if( strcmp( argv[ paramIndex_i], "-rx") == 0){
         if( strcmp( rxFilenameStem_s256, "NULL") == 0){
            sprintf( preEmpt_s8, "%s", "rx");
         }
         else{
            cout << "Invalid arguments\n\n";
            paramIndex_i = argc +2;
         }
      }
      else if( strcmp( argv[ paramIndex_i], "-period") == 0){
         if( clkPeriod_i == (-1)){
            sprintf( preEmpt_s8, "%s", "period");
         }
         else{
            cout << "Invalid arguments\n\n";
            paramIndex_i = argc +2;
         }
      }
      else if( strcmp( argv[ paramIndex_i], "-#byte_strip") == 0){
         if( byteStrip_i == (-1)){
             sprintf( preEmpt_s8, "%s", "s_bytes");
         }
         else{
            cout << "Invalid arguments\n\n";
            paramIndex_i = argc +2;
         }
      }
      else{
         cout << "Invalid argument. " << paramIndex_i << "\n";
         paramIndex_i = argc +2;
      }
   }
   else{
   // actual parameter
      if( strcmp( preEmpt_s8, "numlink") == 0){
         numLinks_i = atoi( argv[ paramIndex_i]);
      }
      else if( strcmp( preEmpt_s8, "tx") == 0){
         sprintf( txFilenameStem_s256, "%s", argv[ paramIndex_i]);
      }
      else if( strcmp( preEmpt_s8, "rx") == 0){
         sprintf( rxFilenameStem_s256, "%s", argv[ paramIndex_i]);
      }
      else if( strcmp( preEmpt_s8, "period") == 0){
         clkPeriod_i = atoi( argv[ paramIndex_i]);
      }
      else if( strcmp( preEmpt_s8, "s_bytes") == 0){
         byteStrip_i = atoi( argv[ paramIndex_i]);
      }
      else{
         cout << "Invalid argument. " << paramIndex_i << "\n";
         paramIndex_i = argc +2;
      }
      sprintf( preEmpt_s8, "%s", "NULL");
```

```
      }
   }
   if( (paramIndex_i == argc + 3)){
      cout << "msglat [-num_link <# num links>] [-tx <file name>]\n";
      cout << "       [-rx <file name>] [-period <clk period>]\n";
      cout << "       [-#byte_strip <bytes removed by delivery>]\n";
      cout << "Takes in the test vectors and end msg times and generates\n";
      cout << "a single file with the message times.\n\n";

      cout << "Default settings : number of links = " << SRCLINKS << "\n";
      cout << "                   transmitted file = " << STARTFILE << "?.txt\n";
      cout << "                   received file = " << ENDFILE << "?.txt\n";
      cout << "                   clock period = " << C_PERIOD << " ns\n";
      cout << "                   bytes stripped = 1\n";
      exit(0);
   }
   else{
      // check the parameters and fill in the blanks
      if( numLinks_i == (-1)){
         numLinks_i = SRCLINKS;
      }
      if( clkPeriod_i == (-1)){
         clkPeriod_i = C_PERIOD;
      }
      if( byteStrip_i == (-1)){
          byteStrip_i = 1;
      }
      if( strcmp( txFilenameStem_s256, "NULL") == 0){
         sprintf( txFilenameStem_s256, "%s", STARTFILE);
      }
      if( strcmp( rxFilenameStem_s256, "NULL") == 0){
         sprintf( rxFilenameStem_s256, "%s", ENDFILE);
      }
      cout << "Using parameters :\n";
      cout << "                  Number of links = " << numLinks_i << "\n";
      cout << "                  Transmitted file = " << txFilenameStem_s256 << "?.txt\n";
      cout << "                  Received file = " << rxFilenameStem_s256 << "?.txt\n";
      cout << "                  Clock period = " << clkPeriod_i << "ns\n";
      cout << "                  bytes stripped = " << byteStrip_i << "\n\n";
   }
}
else{
   numLinks_i = SRCLINKS;
   sprintf( txFilenameStem_s256, "%s", STARTFILE);
   sprintf( rxFilenameStem_s256, "%s", ENDFILE);
   clkPeriod_i = C_PERIOD;
   byteStrip_i = 1;
   cout << "Using default settings :\n";
   cout << "                    number of links = " << numLinks_i << "\n";
   cout << "                    transmitted file = " << txFilenameStem_s256 << "?.txt\n";
   cout << "                    received file = " << rxFilenameStem_s256 << "?.txt\n";
   cout << "                    clock period = "<< clkPeriod_i << "ns\n";
   cout << "                    bytes stripped = " << byteStrip_i << "\n\n";
}

// parameters have been processed, now work with the data

maxTxCnt_l = 0;
minTxCnt_l = 3000000;
msgCnt_ul = 0;
lastRx_ul = 0;
firstTx_ul = 3000000;
totalTxCnt_ul = 0;

resFileHandle_of.open( "result.txt");
if( resFileHandle_of.fail()){
   cout << "\n\nError : Couldn't open the results file";
   exit (0);
}

fileMode_i = ios::nocreate|ios::out;
cout << "\nStarting processing\n";

for(int srcLink = 0; srcLink < numLinks_i; srcLink++){
   // we have the src and dest ID. open each pair in order and search for matches
   // out of sequence should flag errors
   // cout << "Starting " << txFilenameStem_s256 << srcLink << ".txt : ";
```

```
      for(int destLink = 0; destLink < numLinks_i; destLink++){
         // open the source and dest files
         sprintf( buffer_s255, "%s%d.txt", txFilenameStem_s256, srcLink);
         txFileHandle_if.open( buffer_s255, fileMode_i, 0);
         if( txFileHandle_if.fail()){
            cout << "\nFile IO error, couldn't open " << buffer_s255;
            cout << " at itteration " << destLink << "\n\n";
            exit (0);
         }
         sprintf( buffer_s255, "%s%d.txt", rxFilenameStem_s256, destLink);
         rxFileHandle_if.open( buffer_s255, fileMode_i, 0);
         if( rxFileHandle_if.fail()){
            cout << "\nFile IO error, couldn't open " << buffer_s255;
            cout << " at itteration " << srcLink << "\n\n";
            exit (0);
         }
         while( !rxFileHandle_if.eof() && !txFileHandle_if.eof()){
            while( !rxFileHandle_if.eof() && (srcMsgID_i != srcLink)){  // get next arrival
               rxFileHandle_if >> srcMsgID_i >> srcMsgSeq_i >> rxMsgCnt_i >> rxPktCnt_i;
               if( srcMsgID_i != srcLink)
                  srcMsgID_i = (-1);
            }

            while( !txFileHandle_if.eof() && (destMsgID_i != destLink)){
            // get transmission
               txFileHandle_if >> destMsgID_i >> destMsgSeq_i >> txMsgCnt_i >> txPktCnt_i;
               if( destMsgID_i != destLink)
                  destMsgID_i = (-1);
            }
            while( destMsgSeq_i != srcMsgSeq_i && destMsgID_i != (-1) && srcMsgID_i != (-
1)){
               if( destMsgSeq_i > srcMsgSeq_i || (destMsgSeq_i < 2 && srcMsgSeq_i > 13)){
                  // tx sequence is greater than source - out of order delivery
                  sprintf( buffer_s255, "Out of sequence delivery - Src = %d, dest = %d,
req seq = %d, found seq = %d\n\0", srcLink, destLink, destMsgSeq_i, srcMsgSeq_i);
                  for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                  resFileHandle_of.write( buffer_s255, strSize_i);
                  sprintf( buffer_s255, "Tx time = %d, rx time = %d\n\0", txMsgCnt_i,
rxMsgCnt_i);
                  for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                  resFileHandle_of.write( buffer_s255, strSize_i);
                  cout << "s";
                  do{   // get next arrival
                     rxFileHandle_if >> srcMsgID_i >> srcMsgSeq_i >> rxMsgCnt_i >>
rxPktCnt_i;
                     if( srcMsgID_i != srcLink )
                        srcMsgID_i = (-1);
                  }while( !rxFileHandle_if.eof() && (srcMsgID_i != srcLink));
               }

               else if(destMsgSeq_i < srcMsgSeq_i || (srcMsgSeq_i < 2 && destMsgSeq_i >
13)){
                  // tx sequence is less than source - possible message lost
                  sprintf( buffer_s255, "Possible packet loss - Src = %d, dest = %d, req
seq = %d, found seq = %d\n\0", srcLink, destLink, destMsgSeq_i, srcMsgSeq_i);
                  for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                  resFileHandle_of.write( buffer_s255, strSize_i);
                  sprintf( buffer_s255, "Tx time = %d, rx time = %d\n\0", txMsgCnt_i,
rxMsgCnt_i);
                  for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                  resFileHandle_of.write( buffer_s255, strSize_i);
                  cout << "l";
                  do{   // get next arrival
                     txFileHandle_if >> destMsgID_i >> destMsgSeq_i >> txMsgCnt_i >>
txPktCnt_i;
                     if( destMsgID_i != destLink)
                        destMsgID_i = (-1);
                  }while( !txFileHandle_if.eof() && destMsgID_i != destLink);
               }
            }
            if( destMsgID_i == (-1) && srcMsgID_i == (-1));
            else if( destMsgID_i == (-1)){
               // uneven message set
               sprintf( buffer_s255, "Left over message at receiver - Src = %d, dest = %d,
seq = %d, rx time = %d\n\0", srcLink, destLink, srcMsgSeq_i, rxMsgCnt_i);
               for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
               resFileHandle_of.write( buffer_s255, strSize_i);
```

```
                cout << "r";
            }
            else if( srcMsgID_i == (-1)){
                // uneven message set
                sprintf( buffer_s255, "Left over message at transmitter - Src = %d, dest =
%d, seq = %d, tx time = %d\n\0", srcLink, destLink, destMsgSeq_i, txMsgCnt_i);
                for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                resFileHandle_of.write( buffer_s255, strSize_i);
                cout << "t";
            }

            else if(txMsgCnt_i > rxMsgCnt_i){
                sprintf( buffer_s255, "Negative msg time - Src = %d, dest = %d, tx time =
%d, rx time = %d\n\0", srcLink, destLink, txMsgCnt_i, rxMsgCnt_i);
                for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                resFileHandle_of.write( buffer_s255, strSize_i);
                cout << "x";
            }
            else{
                if( (rxPktCnt_i + byteStrip_i) != txPktCnt_i){
                    sprintf( buffer_s255, "Received wrong number of bytes - Src = %d, dest =
%d, tx #bytes = %d, rx #bytes = %d\n\0", srcLink, destLink, txPktCnt_i, rxPktCnt_i);
                    for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
                    resFileHandle_of.write( buffer_s255, strSize_i);
                    cout << "#";
                }
                msgCnt_ul++;
                transmissionCnt_l = (long) rxMsgCnt_i - (long) txMsgCnt_i;
                totalTxCnt_ul = totalTxCnt_ul + (unsigned long) transmissionCnt_l;
                if( transmissionCnt_l > maxTxCnt_l )
                    maxTxCnt_l = transmissionCnt_l;

                if( transmissionCnt_l < minTxCnt_l)
                    minTxCnt_l = transmissionCnt_l;

                if( (unsigned long) txMsgCnt_i < firstTx_ul)
                    firstTx_ul = (unsigned long) txMsgCnt_i;

                if( (unsigned long) rxMsgCnt_i > lastRx_ul)
                    lastRx_ul = (unsigned long) rxMsgCnt_i;
            }
            destMsgID_i = (-1);
            srcMsgID_i = (-1);
        }
        // finished with this rx-tx pair, close files
        cout << ".";
        txFileHandle_if.close();
        rxFileHandle_if.close();
    }

    // cout << " Done\n";
    cout << "|";
  }
  cout << "\nFound " << msgCnt_ul << " valid messages\n";
  cout << "The first transmission began at " << firstTx_ul << " clock cycles (";
  cout << (firstTx_ul * (unsigned long) clkPeriod_i) << " ns)\n";
  cout << "The last receipt finshed at " << lastRx_ul << " clock cycles (";
  cout << (lastRx_ul * (unsigned long) clkPeriod_i) << " ns)\n";
  cout << "The tests took " << (lastRx_ul - firstTx_ul) << " clock cycles (";
  cout << ((lastRx_ul - firstTx_ul) * (unsigned long) clkPeriod_i) << " ns)\n";
  cout << "Longest transmission time was " << maxTxCnt_l << " cycles (";
  cout << (maxTxCnt_l * clkPeriod_i) << " ns)\n";
  cout << "Shortest transmission time was " << minTxCnt_l << " cycles (";
  cout << (minTxCnt_l * clkPeriod_i) << " ns)\n";
  cout << "Average transmission time was " << (totalTxCnt_ul/msgCnt_ul) << " cycles (";
  cout << (((long double)totalTxCnt_ul/ (long double)msgCnt_ul) * (long
double)clkPeriod_i) << " ns)\n";

  sprintf( buffer_s255, "\nTotal transmission time (cycles) = %d, message count = %d\n\0",
totalTxCnt_ul, msgCnt_ul);
  for( strSize_i = 0; buffer_s255[ strSize_i] != '\0'; strSize_i++);
  resFileHandle_of.write( buffer_s255, strSize_i);
  resFileHandle_of.write( "Processing complete\n", sizeof( "Processing complete\n"));
  resFileHandle_of.close();
  return 1;
}
```

# Appendix E : Hardware Test Descriptions

This appendix contains the descriptions of the hardware test benches that were used for extended verification of the NTR-FTM08 implemented in an Altera 10K130E. The hardware tests include : Sampling, physical, grouped physical, logical, interval, broadcast, disconnection, receiver overflow, and dormancy.

**Sampling test**

The sampling test used eight buffered link modules to verify the sampling, link status and flow-control mechanisms. In total, sixteen buffered links were connected together, eight in each PLD. Each link was offered data to transmit, while the receiver was monitored to ensure valid reception was achieved. The control circuits for the receiver was periodically stalled, which allowed the receiver FIFO to fill and trigger the flow control mechanism. An incremental byte sequence was used to test all possible data sequences.

Test failure was detected in one of three ways. The first method verified the stability of the link state; that is, if the link was reset following the test start. The second monitored the value of the received data, and ensured that if followed the predefined sequence. The final method verified data movement, by monitoring the read and write control signals to the link.

Each test unit possessed eight LEDs that were used in this test to indicate the test status. These were :

- permanently off – test not started, corresponding link did not initialise;
- regular flashes – test activity failure, no data movement;
- two, intermittent flashes - receiver error, incorrect data value has been received;
- three, intermittent flashes - link failure, the link detected and error and reset itself.
- Permanently on – test progressing normally.

**Physical addressed test**

The physical addressing test is performed on an unconfigured switch; that is, no grouping and all packets with interval or logical addressed headers are spilt at the receiver. This test possesses eight individual test links, which send a constant stream of packets to

the switch. It verifies the operation of sampling, flow control, service requesting, contentious queuing, connection allocation, disconnection, and illegal packet spillage.

Each test link transmits five, three token packets per header value, which results in all headers being transmitted. The five packets follow a set sequence of {physical, interval, physical, logical, physical}. This sequence allows each physical packet to be transmitted with all three end-of-packet markers; that is, end-of-packet, end-of-message and bad-end-of-packet. The second byte in the packet is a replication of the routing byte. Transmitting the packets with interval and logical headers proves the physical addressed packet has been disconnected successfully, and the illegal packet has been removed from the network.

The receiving channel of each test link monitors the incoming byte stream and validates all received data. Each link should receive only physically addressed packets, and therefore the receiver check the incoming data bytes to ensure correct delivery. All transmitters are enabled together, which creates contention as each link follows the same sequence of headers. Similar to the sampling test, data flow is verified to ensure, that none of the links are permanently stalled, and a reset link will also result in test failure.

Any failure inhibits further testing, and the test state is indicated by the operation of the eight LEDs, which are:

- permanently off – test has not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error, where the link has been unexpectedly reset;
- Permanently on – test progressing normally.

**Grouped physical addressed test**

This test is the first to include configuration, which requires a configuration packet to be sent to the switch before the start of the test. Thus, in addition to the features tested by the basic physical tests and verification of the operation of the link allocation with grouped outputs under contentious conditions, this test verifies the basic operation of the control port. That is, connection, disconnection, operation of the data and address register and the write command for the grouping registers.

The switch is configured to group all the links into four groups of two, namely:

- group 0 – links 0 & 1
- group 1 – links 2 & 3
- group 2 – links 4 & 5
- group 3 – links 6 & 7

Each test links sends eight packets of three tokens, which equates to two for each group. Thus, the test verifies that allocation operates for all headers for the group. The payload byte of each token contains the BCD value of the source link and destination link. This allows the receiving test link to verify each packet as it arrives, and confirm all packets have been delivered. Following the transmission of all eight packets, the transmitting control for each test link inhibits further transmission until all packets have been verified by the receiving links.

To indicate test status, the LEDs are used similar to the previously defined tests; thus:

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error;
- permanently on – test progressing normally.

**Logical addressed test (unicast)**

This test verifies the use of logical addressing for packet routing. Each link takes turns to be the receiver, and as such configures the logical address lookup table. The configuration validates the target destination for each offset of the output for the range of the sixty-four outputs. For example, link 0 configures headers 0, 8, 16, 24, 32, 40, 48, 56 to send the packet to output 0 and the remaining headers are invalid. Following configuration, the other 'non-receiving' links transmit sixty-four small packets, ensuring each header is sent. All the packets are sent concurrently which shows the operation of the device under contention. The receiver checks the incoming packet stream and checks for correct delivery, which ensures that all of the packets have been successfully received.

The receivers that are not configured to accept data will flag any data reception. Once all packets have been validated, the process is repeated with the next link.

This test checks for the standard operation of packet transfer in addition to the configuration of the LUT, logical decoding and queuing, logical connection allocation.

To indicate test status, the LEDs are used similar to the previously defined tests; thus:

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error;
- permanently on – test progressing normally.

**Interval addressed test**

The interval addressed test replicates the logical addressed test, but with the use of interval addressed headers. However, the headers are validated in offsets of sixteen, as opposed to the offset of eight as used in the logical addressed test. Thus link 1 configures headers 1, 17, 33, 49 to send the packet to output 0 and the remaining headers are invalid.

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error;
- permanently on – test progressing normally.

**Broadcast test**

The broadcast test ensured that multicast connections were processed successfully. The test allowed each test link to configure the router in turn, which validates eight headers to connect to all other links. The selected headers were relative to the offset of the originating link. For example, link 2 configures headers 2, 10, 18, 26, 34, 42, 50, 58 to send the packet to all outputs and the remaining headers are invalid. The receiving links

all verify the correct delivery of the packet before the test can progress. Similar to the other tests, the standard checks are made and are indicated by the LEDs as follows:

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error;
- permanently on – test progressing normally.

**Disconnection time-out test**

The test operated based on a finite state machine that allowed validation of the recovery procedure. The states were as follows:

1. Send the start of the packet;
2. validate the arrival of the start of the packet;
3. disconnect either the source or destination from the switch;
4. send the end of the packet;
5. check the packet was truncated with a BEOP token;
6. reconnect the severed links;
7. configure for the next set of packets

Each source sends a non-contentious packet to the switch concurrently, which ran in an incremental sequence that ensured that each link connected with all other links. Links were disconnected based on a sequence such that either the source or the destination connections were severed exclusively. Physical addressing was used for the test and the header was reused for the packet payload, which included the source and destination of the packet. This allowed receipt and packet format validation. Each transmitted packet was contained four tokens in total, where the header and first payload byte was transmitted in state 1. State 4 saw the transmission of the second payload token and the end of packet.

This test saw the allocation of concurrent non-contentious connection requests, which resulted in either receiver packet truncation or packet spillage at the transmitter. Maintaining the standard test indicators, the LEDs operated thus:

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error, a link was reset that was not an expected;
- permanently on – test progressing normally.

**Overflow reset test**

The overflow reset test used eight buffered link modules to verify the operation of the link recovery for flow-control failure. To ensure that the mechanism failed, the 'almost-full' flag was set at 30, with a receiver FIFO buffer of 31 entries. In total, sixteen buffered links were connected together, eight in each PLD. Eight links were set as receivers and eight were set as transmitters. The test was executed in the following sequence:

1. Allow the transmitters to start transmitting, but do not remove data from the receiver FIFO buffers;
2. verify the overflow reset at the receiving link, and reset at the transmitter;
3. check the contents of the receiver FIFO buffers, including a BEOP at the end.

The eight LEDs were used to indicate the test status, as follows:

- permanently off – test not started, corresponding link did not initialise;
- regular flashes – test activity failure, no data movement;
- two, intermittent flashes - receiver error, incorrect data value has been received;
- three, intermittent flashes – unexpected link failure, the link detected and error and reset itself outside of the period expected;
- Permanently on – test progressing normally.

**Link Dormancy test**

The link dormancy test verified that links would safely return to the link 'asleep' state and would successfully wake when required. The subsequent to the configuration of the link dormancy register, test executed in the following sequence:

1. Allow the links to return to the 'asleep' state;

2. enable the selected link to transmit a packet to the network;
3. wait for the packet to arrive on the destination link.

During this sequence the state of all links were monitored to ensure that the correct link was woken, and the packet arrived successfully. Physical addressing was used to address the packets. Only one packet was sent across the router at a time, and the test ensured that each link sent to every other link in ascending order. Maintaining the standard test indicators, the LEDs operated thus:

- permanently off – test not started, all links have not been initialised;
- regular flashes – no activity, the test is not progressing as expected;
- two intermittent flashes – registered an receiver error on that link;
- three intermittent flashes – link error, this link did not wake when expected or woke unexpectedly;
- permanently on – test progressing normally.

# Appendix F : Deadlock Detectio n Mechanism Details

This appendix contains the post-simulation analysis results of the deadlock detection tests, router switch configuration for the tests and the source code for the analysis program.

**16 bytes per packet**

| Workload (%) | Total packets removed | Cycles without removal | Cycles with removal | Average packets removed per cycle |
|---|---|---|---|---|
| 30 | 1376 | 270 | 1108 | 1.242 |
| 40 | 1342 | 310 | 1102 | 1.218 |
| 50 | 1283 | 265 | 1027 | 1.249 |
| 60 | 1325 | 305 | 1047 | 1.266 |

**32 bytes per packet**

| Workload (%) | Total packets removed | Cycles without removal | Cycles with removal | Average packets removed per cycle |
|---|---|---|---|---|
| 30 | 1480 | 202 | 1405 | 1.053 |
| 40 | 1520 | 191 | 1453 | 1.046 |
| 50 | 1355 | 213 | 1293 | 1.048 |
| 60 | 1361 | 223 | 1292 | 1.053 |

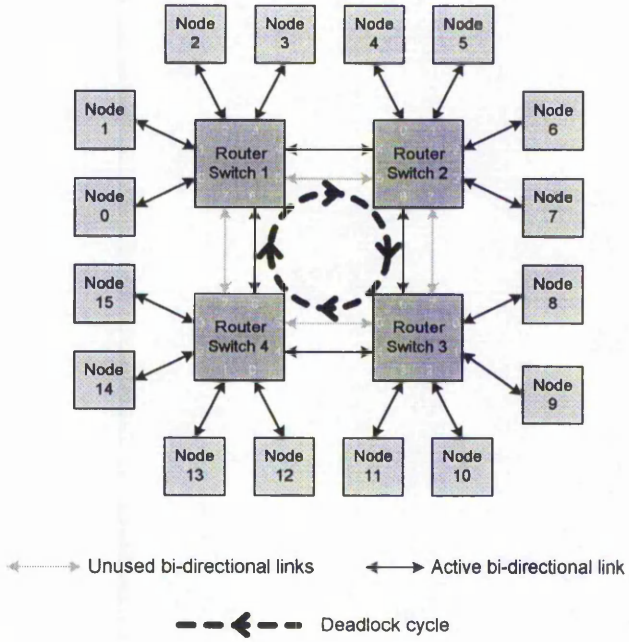
**64 bytes per packet**

| Workload (%) | Total packets removed | Cycles without removal | Cycles with removal | Average packets removed per cycle |
|---|---|---|---|---|
| 30 | 1450 | 153 | 1445 | 1.003 |
| 40 | 1483 | 159 | 1483 | 1.000 |
| 50 | 1449 | 168 | 1448 | 1.001 |
| 60 | 1452 | 159 | 1452 | 1.000 |

**128 bytes per packet**

| Workload (%) | Total packets removed | Cycles without removal | Cycles with removal | Average packets removed per cycle |
|---|---|---|---|---|
| 30 | 1491 | 160 | 1489 | 1.001 |
| 40 | 1535 | 179 | 1534 | 1.001 |
| 50 | 1483 | 175 | 1482 | 1.001 |
| 60 | 1469 | 161 | 1465 | 1.003 |

**Device configuration for the deadlock prone mesh network**




| Router switch 1 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 11 | 4 |
| 5 | 15 | 6 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 2 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 4 |
| 1 | 4 | 0 |
| 2 | 5 | 1 |
| 3 | 6 | 2 |
| 4 | 7 | 3 |
| 5 | 15 | 6 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 3 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 3 | 4 |
| 1 | 7 | 6 |
| 2 | 8 | 0 |
| 3 | 9 | 1 |
| 4 | 10 | 2 |
| 5 | 11 | 3 |
| 6 | 15 | 4 |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

| Router switch 4 | | |
|---|---|---|
| Interval | Interval Limit Register | Interval Port Register |
| 0 | 7 | 6 |
| 1 | 11 | 4 |
| 2 | 12 | 0 |
| 3 | 13 | 1 |
| 4 | 14 | 2 |
| 5 | 15 | 3 |
| 6 | 63 | invalid |
| 7 | 63 | invalid |
| 8 | 63 | invalid |
| 9 | 63 | invalid |
| 10 | MAX | invalid |

As the deadlock detection mechanism did not support group adaptive routing, the feature was not used in this test.

## Program listing for deadlock_chk.cpp

```
/*
Filename : deadlock_chk.cpp
Author   : Robin Hotchkiss
Date     : January 2000
Version  : 2.0
Description :
   Read the transcript file produced by the modelsim simulator
   and processes the assert statements
   to detect messages that are deleted due to the
   deadlock mechanism.
   User decisions are prompted when a message
   is cleared outside of a deadlock cycle

   Assert statements are provided when
   cycles form, deadlock cycles form,
   and the deadlock mechanism trigger

Modifications :

*/

#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main(int argc, char *argv[])
{
ifstream srcFileHandle_if;
int fileMode_i = 0;
char srcFilename_s255[255] = "transcript";
char lineBuffer_c255[255] = "empty";
unsigned long lineCount_ul = 0;
bool cyclePresent_b = false;
bool dlCyclePresent_b = false;
unsigned long dlCycleCnt_ul = 0;
unsigned long nonDlCycleCnt_ul = 0;
unsigned long badCycleCnt_ul = 0;
unsigned long dlMsgDelCnt_ul = 0;
unsigned long nonDlMsgDelCnt_ul = 0;
bool deletedInCycle_b = false;
bool deletedInLastCycle_b = false;

   fileMode_i = ios::nocreate|ios::out;
   srcFileHandle_if.open( srcFilename_s255, fileMode_i, 0);
   if( srcFileHandle_if.fail()){
      cout << "\n\nError : Couldn't open the source test vectors file";
      exit (0);
   }

   while( !srcFileHandle_if.eof()){
      lineCount_ul++;
      srcFileHandle_if.getline( lineBuffer_c255, 255);
      if( strstr( lineBuffer_c255, "Cycle has formed") != NULL){
         if( cyclePresent_b){
            cout << "Error in processing at line " << lineCount_ul;
            cout << " : cycle formed notification with cycle formed.\n";
         }
         cyclePresent_b = true;
         dlCyclePresent_b = false;
         deletedInCycle_b = false;
      }
      else if( strstr( lineBuffer_c255, "Cycle has cleared") != NULL){
         if( !cyclePresent_b){
            cout << "Error in processing at line " << lineCount_ul;
            cout << " : cycle clear notification without cycle formed.\n";
         }
         cyclePresent_b = false;
         if( deletedInCycle_b){
            if( dlCyclePresent_b){
               // count the cycles that caused deadlock recovery
               dlCycleCnt_ul++;
            }
            else{
```

```
                // the cycle shouldn't have caused recovery
                badCycleCnt_ul++;
            }
        }
        else{
            // count the cycles that didn't cause deadlock recovery
            nonDlCycleCnt_ul++;
        }
        deletedInLastCycle_b = deletedInCycle_b;
        deletedInCycle_b = false;
    }
    else if( strstr( lineBuffer_c255, "Deadlock flag") != NULL){
        if( !cyclePresent_b){
            cout << "DL-flag outside a cycle at line " << lineCount_ul;
            if( dlCyclePresent_b){
                cout << " (last cycle possessed a deadlock stall)" << endl;
            }
            else{
                cout << " (last cycle didn't possess a deadlock stall)" << endl;
            }
            cout << "Is this a false detection? (y/n) > ";
            cin >> lineBuffer_c255[0];
            if( lineBuffer_c255[0] == 'Y' || lineBuffer_c255[0] == 'y'){
                // out of cycle thang!
                nonDlMsgDelCnt_ul++;
            }
            else{
                // should be part of the last cycle
                dlMsgDelCnt_ul++;
                if( !deletedInLastCycle_b){
                    nonDlCycleCnt_ul--;
                    dlCycleCnt_ul++;
                    deletedInLastCycle_b = true;
                }
            }
        }
        else if( !dlCyclePresent_b){
            cout << "DL-flag without a deadlock stall at line " << lineCount_ul << endl;
            cout << "Is this a false detection? (y/n) > ";
            cin >> lineBuffer_c255[0];
            if( lineBuffer_c255[0] == 'Y' || lineBuffer_c255[0] == 'y'){
                // false detection, shouldn't have cleared the message
                nonDlMsgDelCnt_ul++;
            }
            else{
                // this cycle does have a deadlock stall
                dlMsgDelCnt_ul++;
            }
            deletedInCycle_b = true;
        }
        else{
            // valid deadlock, inside stall cycle
            dlMsgDelCnt_ul++;
            deletedInCycle_b = true;
        }
    }
    else if( strstr( lineBuffer_c255, "Dl cycle has formed") != NULL){
        // deadlock cycle detected, inside a stall cycle
        dlCyclePresent_b = true;
    }
}
if( cyclePresent_b){
    cout << "Error in processing at EOF : Cycle was not cleared.\n";
}

cout << "\n\n";
cout << dlCycleCnt_ul << " cycles with valid deadlocks.\n";
cout << badCycleCnt_ul << " cycles with illegal deadlock recovery.\n";
cout << nonDlCycleCnt_ul << " cycles without deadlocks.\n";
cout << dlMsgDelCnt_ul << " messages were removed due to valid deadlocks.\n";
cout << nonDlMsgDelCnt_ul << " messages were removed falsely.\n";

return 0;
}
```