Bandwidth Management in Interconnection Networks for Multiprocessor Architectures

Nayef Baker

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Department of Computing
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, U.K.

June 2003

© Nayef Baker 2003
Acknowledgements

I would like to thank my supervisor Dr Roger M.A. Peel for his continuous advice and support. Special thanks go to Professor Chris Jesshope who initially inspired my research, and to Professor Bernard Weiss for his support.

I am also very grateful to many friends and colleagues who helped me in many different ways.

Nayef
Contents

1 Introduction ........................................ 1

2 Review of Routing Algorithms ............... 3
  2.1 Introduction .................................... 3
  2.2 Routing algorithms ......................... 3
    2.2.1 Routing mechanisms ................. 3
    2.2.2 Routing Decisions .................... 4
  2.3 Routing Algorithms - Summary .......... 5
    2.3.1 Shortest Path Routing ............... 5
    2.3.2 Multi-path Routing .................... 6
    2.3.3 Centralised Routing ..................... 7
    2.3.4 Isolated Routing ....................... 7
    2.3.5 Hot-Potato algorithm ................. 8
    2.3.6 Combined Hot-Potato/Static algorithm .... 8
    2.3.7 Backward learning algorithm ........... 8
    2.3.8 Delta algorithm ....................... 8
    2.3.9 Flood Routing ......................... 8
    2.3.10 Flow-Based Routing ................. 9
    2.3.11 Hierarchical Routing ................. 9
    2.3.12 Broadcast Routing .................... 9
    2.3.13 Distributed Routing ................. 10
    2.3.14 Interval routing ....................... 10
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>

4 A Path-finding Algorithm

4.1 Introduction ............................... 40
4.2 Assumptions and definitions .......... 42
  4.2.1 Definition of bandwidth .......... 42
  4.2.2 Definition of path ................. 43
4.3 Modeling nodes and messages ......... 44
4.4 Algorithm for path-finding .......... 47
  4.4.1 Requirements specifications ...... 47
  4.4.2 Description of the algorithm ...... 48
  4.4.3 Stages of the algorithm .......... 49
  4.4.4 Routing Tables ...................... 55
4.5 The Algorithm in CSP ................. 56
4.6 Characteristics of the algorithm .... 60
  4.6.1 Correctness ......................... 60
  4.6.2 Termination ......................... 60
  4.6.3 Local termination ................. 61
  4.6.4 Global termination ............... 61
  4.6.5 Deadlock ............................ 62
  4.6.6 Performance ......................... 64
  4.6.7 Competition on bandwidth ....... 65
  4.6.8 Development issues ............... 65
4.7 Summary .................................. 66

5 Design of the Router ................. 67

5.1 Introduction ............................... 67
  5.1.1 Design considerations ............... 67
5.2 Router Structure ......................... 68
  5.2.1 Routing Layers ....................... 68
  5.2.2 State-Store Servers ............... 71
  5.2.3 Interface Block ....................... 72
5.3 Processing Element ...................... 73
5.4 Correctness of the design - the large picture .... 75
## Contents

5.4.1 Deadlock ................................................................. 75  
5.4.2 Fairness ................................................................. 76  
5.4.3 Update messages ..................................................... 77  
5.5 Summary ................................................................. 78  

6 Simulations and Results .................................................. 79  
6.1 Introduction ............................................................. 79  
6.2 The scope of simulations ............................................ 79  
  6.2.1 Latency measurements ........................................ 79  
  6.2.2 Targeted simulations .......................................... 80  
  6.2.3 Traffic models and patterns ................................ 81  
6.3 The Simulator ............................................................ 82  
  6.3.1 Limited-rate source ............................................ 83  
  6.3.2 The timing of events ........................................... 86  
  6.3.3 Message logging facility ..................................... 87  
6.4 Results of simulations ............................................... 88  
  6.4.1 Measurement method ......................................... 88  
6.5 Latency results .......................................................... 88  
  6.5.1 Path Request latency .......................................... 89  
  6.5.2 Path Acknowledgment latency .............................. 90  
  6.5.3 Path Set-Up latency ........................................... 91  
  6.5.4 Path Closure latency ......................................... 92  
  6.5.5 Data Forwarding latency ..................................... 92  
  6.5.6 Overall control latency ...................................... 94  
6.6 Performance results .................................................. 94  
6.7 Competitive comparisons .......................................... 99  
6.8 Further development issues ....................................... 102  
6.9 Summary ............................................................... 104
7 Conclusions 105
  7.1 The Path-finding Algorithm .......................................................... 105
  7.2 The implementation ................................................................. 106
  7.3 Performance ............................................................................ 106
  7.4 The simulations ....................................................................... 106
  7.5 Other issues ........................................................................... 107
  7.6 Final summary ....................................................................... 107
List of Figures

3.1 The ATM connection identifiers ................................................................. 21
3.2 ATM switching: virtual paths and virtual channels [1] .............................. 22
3.3 ATM header fields ...................................................................................... 23
3.4 The ATM layers ........................................................................................... 30
3.5 $I$ and $L$ parameters in GCRA ................................................................. 37

4.1 A model for end-to-end path set-up messaging ........................................... 41
4.2 A path and its parameters - an example ...................................................... 45
4.3 A generalised model for 2-D mesh of 5x5 nodes ........................................ 46
4.4 Generalised model for nodes in $k$-D mesh ................................................ 47
4.5 State diagram for paths ................................................................................ 50
4.6 A typical wave exploration pattern in a 2-D sub-mesh .............................. 52
4.7 Typical structure of a routing table associated with an output link ........... 56
4.8 States of node $N$ ......................................................................................... 58

5.1 A functional architecture of the Router ...................................................... 69
5.2 Routing Layers ............................................................................................ 70
5.3 A Routing-Table Server ............................................................................... 71
5.4 Interface Block ............................................................................................ 72
List of Figures

5.5 Functional diagram of the processing element ..................................................... 74
5.6 End-to-end path set-up messaging ........................................................................ 75
5.7 Time-stamping of messages at the routing layer .................................. 76
5.8 Update messages following backward messages .................................................. 77

6.1 Data packets source with bandwidth limit ........................................................... 84
6.2 A model for the generator of data packets ........................................................... 85
6.3 Fairness between inputs of a node ........................................................................ 86
6.4 Latencies of path request message in unloaded 8x8 mesh .............................. 89
6.5 Latencies of path acknowledgment message in unloaded 8x8 mesh .......... 91
6.6 Latencies of path set-up in 8x8 unloaded mesh ...................................................... 92
6.7 Latencies of path closure message in 8x8 unloaded mesh ............................... 93
6.8 Latencies of data packets in an 8x8 mesh with single path ............................... 93
6.9 Comparisons of latencies in 8x8 mesh with single path ...................................... 95
6.10 PFA latency in 8x8 mesh with isolated traffic ..................................................... 96
6.11 Latencies of in PFA with random uniform traffic .............................................. 97
6.12 Sub-mesh with correlated traffic ........................................................................ 98
6.13 Node propagation delays when using PFA with correlated traffic ................. 99
6.14 SIC-based model used in simulations ................................................................. 100
6.15 Comparisons of PFA and SIC-based routers .................................................... 101
List of Tables

3.1 Functional differences between generations of packet switching ........... 19
3.2 PTI status [2] ......................................................................................... 23
3.3 Support operations for AAL classes [1] .............................................. 29
3.4 Bandwidth management procedures [3] ............................................. 33
Chapter 1

Introduction

This work describes a new routing algorithm for the guided routing of packets in multiprocessor networks of \( k\text{-ary } n\text{-cube} \) topology. The algorithm is fully distributed and allows for an increase of throughput by minimising traffic density at (and around) busy nodes or busy areas. These busy areas are also known in the literature as "hot spots". Congestion at hot spots eventually causes severe contention over resources which in turn dramatically reduces throughput.

The algorithm uses pre-assigned paths for a particular connection between a pair of nodes (point-to-point communication). The capacity of the links is shared among multiple connections, similar to virtual paths (VP) in Asynchronous Transfer Mode (ATM). The advance assignment of each link's capacity allows for the smooth flow of packets along each virtual path. The algorithm uses an exploration (or route discovery) phase to select a unique route that is "capable" of handling the traffic load. The exploration phase ultimately avoids selecting paths that lead to hot spots. Each reservation is cleared when the connection is no longer needed.

The operations of selection and cancellation are carried out asynchronously, i.e. there is no global timing. Nodes are also autonomous, that is each node's response to an event (or a message) is solely dependent on its state; simply there is no global knowledge of traffic conditions nor central decision-making mechanism on selection of paths.

The network is modelled and simulated using many parallel Occam processes. The networked model is designed as a grid of processes each representing a node, with smaller processes therein representing functions within the node. A common clock process is used for time
measurements during the simulation runs without affecting the asynchronous behavior of the models. The simulation clock is distributed in a way that avoids any synchronisation of internode messaging between nodes.

This thesis is organised into six chapters. The next chapter is a summary of the most common and interesting routing algorithms and techniques that are used for packet routing. The third chapter is a review of Asynchronous Transfer Mode (ATM). It also include the concepts of virtual paths and bandwidth sharing. These concepts are re-used in the new algorithm.

Chapter four includes a description of the algorithm in depth. Routing, deadlock, and live-lock issues are also presented. In chapter five, a modular approach is outlined to specify nodes as functional modules. These functional modules are designed and fully specified to achieve the correct operation of the algorithm. These modules are then simulated using Occam processes and the results are shown in chapter six. Chapter seven includes a summary of simulation results, and comparisons with a close common competitor - minimal adaptive routing. Conclusions are finally presented in chapter eight.
Chapter 2

Review of Routing Algorithms

2.1 Introduction

The main function of the interconnection network in a multiprocessor architecture is to route packets between nodes. In this chapter, a number of common routing algorithms in computer networks are reviewed, with emphasis on routing for multiprocessor architectures.

2.2 Routing algorithms

2.2.1 Routing mechanisms

The temporal pattern of traffic between nodes in packet-switched networks can be either connection-oriented or connection-less.

1. In connection-oriented networks, a fixed connection path must be established before the actual data transfer occurs. The connection is maintained during the whole data transfer, where packets follow the path, arriving at their destination in the same order in which they were sent. Upon completion of the information transfer, the connection is closed. Allocated resources are then released and made available for further transfers. As the route is solely used by one connection, each packet header does not necessarily contain addressing information.
2.2. Routing algorithms

2. Connection-less routing allows a mixture of packets from different routes to be sent down a link. Resources (links, buffers, etc.) are shared among the streams of traffic. There is a need for addressing information in each packet. That information links a source to a destination, and thus is used by nodes to route each packet to its destination.

The temporal pattern of packet forwarding at a node follows one of three possible modes [4]:

1. **Store-and-forward** routing. Each packet is completely received at a node, stored in a local buffer, and is then re-transmitted to the next node [5].

2. **Wormhole** routing, the packet header is advanced directly from incoming to outgoing links before the rest of the packet is received. Only a small part of the header is buffered at each node [6], [7]. Wormhole routing is described later in section 2.3.15 below.

3. **Virtual Cut-Through** routing, This is similar to wormhole routing, but it buffers the whole of a packet when it is blocked at a node [8].

2.2.2 Routing Decisions

The journey time a packet takes to reach its destination mainly depends on the route (i.e. how many hops), and the total delays. An ideal algorithm must take into account all factors that affect packet delivery. Eventually, the more complicated the algorithm is, the longer it takes to make routing decisions. A trade off between complexity and decision time is often required.

The way routing decisions are made is typically fixed, and the choice of a routing algorithm is made at the configuration stage. For example, the MPI developed at Surrey selects either routing x-then-y or y-then-x at start-up [9] and [10]. In theory, temporal changes of network utilisation give a good argument for dynamic changes in routing decisions, changes in topology, or both. Networks that allow changes in topology - the so called re-configurable networks - are emerging. One example is the re-configurable optical network shown in [11].

Routing algorithms can be classified according to how routing decisions are made, for example time taken, locations where decisions are taken, etc. Two categories are found: non-adaptive and adaptive.
2.3. Routing Algorithms - Summary

1. **Non-adaptive algorithms** (also called deterministic or static routing)

   In this category, routing decisions are not affected by measurements or estimates of the current traffic or topology. Instead, the route is computed in advance (off-line). The route taken by a message is determined by its destination, and not by other traffic in the network. It is either downloaded to nodes when the network is re-booted, or inherited by the design. One example is restrictive routing that uses fixed dimension ordering. One dimension is always traversed first, then another, and so on. Oblivious routing gives the choices of routing along either dimension first then along the other, depending on blocking situations.

2. **Adaptive Algorithms** (also called non-deterministic)

   These attempt to change routing decisions according to the change in topology and current traffic patterns. These can be further divided into three sub-classes [12]:

   - **Global Algorithms** (or centralised routing) use information collected from the entire network in an attempt to make optimal routing decisions.
   - **Local Algorithms** (isolated routing) run separately on each Interface Message Processor (or IMP), and only use local information available at each IMP to make routing decisions.
   - **Global/Local Algorithms** are mixtures of the above two sub-classes.

2.3 **Routing Algorithms - Summary**

The following sections describe the most common routing algorithms [12]:

2.3.1 **Shortest Path Routing**

Shortest path routing is the simplest routing algorithm that finds the path from source to destination. This is achieved by calculating delay costs along all possible paths and choosing the one with the smallest delay as the shortest one. It is assumed that the traffic between a pair of nodes should follow the shortest path. As some links may become busier than others, the
path with the least delay may differ from the shortest physical path. The path length can be the number of hops (label = 1), geographic distance (label = distance) or the mean queuing and transmission delay (determined by static metrics or by test runs).

The discovery of the 'shortest' path is achieved by building a graph of the net (or a part of the net). Thus, the problem becomes finding the shortest path on a graph.

Among various algorithms to find the shortest path, one repeatedly (sequentially) searches (at each IMP) for the minimum label value among neighboring nodes. Once a smaller label value is found, it is marked as a tentative node. On completion, the node with smallest label value is made permanent. A new search can begin from this permanent node on a similar scenario. The previous permanent node may be excluded, as it has been checked earlier. The search process continues until the path search is completed. Labeling can also be calculated in more complex ways. It can be a function of the distance, bandwidth, average traffic, communication costs, mean queue length, measured delay, etc. [12]. However, complex functions will impose an overhead on routing decisions, and therefore greater delays in each path set-up.

2.3.2 Multi-path Routing

Shortest path algorithms are essentially sequential. In multi-path routing, it is possible to send traffic over different paths. Instead of directing the traffic between a pair of nodes along one particular path (e.g. the shortest one), it is possible to split the traffic over many other equally "good" paths. This will reduce the load on each of the communication lines along the shortest path. In data-gram nets, the choice of link along which to route a packet is made at each intermediate node for each packet. The choice of link is independent of the previous choice for other packets heading to same destination. In virtual circuit nets, whenever a virtual circuit is set up, a route is chosen, but different virtual circuits with the same destination are routed independently. Each IMP maintains a table with one entry for each possible IMP destination. The tables are worked out by the operator and loaded into IMPs and not changed thereafter. Each entry contains all the outgoing lines in preference order together with their weights. Before forwarding a packet, an IMP generates a random number, and chooses among alternatives using the weight as probabilities. Multi-path allows more than one class of traffic to proceed concurrently. Reliability is also increased since the net can withstand the loss of some links due
to disjoint routes in routing tables. Multi-path can use a shortest path calculation to find first, second, etc. path preferences for a pair of nodes. It can be implemented by removing links used in the shortest path from the graph, and then calculating the shortest path again. This algorithm assures that IMP or line failures on the first path will not also cause the second path to fail.

2.3.3 Centralised Routing

Each IMP periodically sends status information to a particular node, called the Routing Control Centre (RCC). The RCC is usually located at the centre of the network. The status information held by an IMP includes a list of its neighbours that are alive, and current queue lengths. In this way, a global knowledge of the entire network is continuously made available to the RCC. The RCC can therefore compute all optimal routes between each IMP pair in the network. The routing algorithm implemented at the RCC can continually build new routing tables for all IMPs, which can be regularly distributed. The RCC relieves the IMPs of the burden of routing computations. In contrast, this algorithm has drawbacks:

1. The RCC has to perform computations fairly quickly,
2. The RCC may need a backup machine to allow for sudden RCC or link failures.
3. Delays in distributing the tables may also cause inconsistencies.
4. Links leading into the RCC may be heavily loaded compared to other links. This is due to a higher proportion of status and table information flowing along these links.

2.3.4 Isolated Routing

In contrast to centralised routing, IMPs do not exchange routing information with other IMPs when they use isolated routing. Instead, each IMP tries to adapt to changes to topology and traffic. For that it is called isolated adaptive routing. The following algorithms fall in this category:
2.3.5 **Hot-Potato algorithm**

Each IMP counts the number of packets queued up for transmission on each output. It puts the new packet on the shortest queue regardless of where that output leads to.

2.3.6 **Combined Hot-Potato/Static algorithm**

This is a combination of multi-path and static algorithms. The algorithm takes into account both the static weights of the links and the queue lengths. An example is to use the best static choice unless its queue exceeds a certain threshold.

2.3.7 **Backward learning algorithm**

This algorithm requires that the identity of the source IMP be included into the packet together with a count of the number of hops that the packet had traveled. An IMP will record the hop count of each incoming packet, thus the smallest (among packets coming from the same source) is the best. Then it marks that line as the choice for traffic to it. Repetition of this learning will result in every IMP discovering the shortest path to every other IMP. This mechanism allows IMPs continuously to choose better paths. Should any line go down or become overloaded, there is no way of recording that fact. The solution is periodically to clear IMP records, and then to start learning all over again.

2.3.8 **Delta algorithm**

Each IMP assigns a cost value to each link. The cost is computed as a result of some function of delay, queue length, bandwidth, etc. The cost of each link is sent to the RCC. According to the cost of each line, The RCC sends each IMP a list of all initial links for good paths for each of its possible destinations.

2.3.9 **Flood Routing**

Flooding relies on the forwarding of packets with minimal processing. Flood routing guarantees the fast arrival of messages with minimum en-route computations at the expense of excess-
2.3. Routing Algorithms - Summary

sive bandwidth usage (e.g. by copying messages in several directions). A controlled flooding, however, limits the extent to which a message is flooded [13]. There are a few variants:

- **Selective Flooding**: nodes send out packets only on these links that are going approximately in the right direction.

- **Random Walk**: nodes select a link at random and forward the packet on it.

- **Optimal Routing**: A set of optimal routes from all sources to a given destination is calculated. The set forms a tree (sink tree) rooted at the destination. The traffic from one node to another will follow a certain path along the corresponding sink tree.

### 2.3.10 Flow-Based Routing

This algorithm assumes that the data flow between each pair of nodes is relatively stable and predictable. The capacity of a link and the average flow are also assumed known. Using queuing theory, it is possible to compute the mean packet delay on a given link. It is therefore possible to calculate a flow-weighted average to obtain the mean packet delay for the whole network. The routing problem is therefore reduced to finding the routing algorithm that produces the minimum average delay for the network. This requires prior knowledge of the network topology, traffic matrices, and the capacity of each link.

### 2.3.11 Hierarchical Routing

In networks with a large number of IMPs, the network is divided into regions. Each IMP stores the routing information of other IMPs in its region. Inter-region traffic is routed to one of a few pre-assigned IMPs. Several levels of hierarchy may be used. This is similar to the telephone network.

### 2.3.12 Broadcast Routing

Some applications require to send a message from a node simultaneously to all other nodes in a sub-net. One possible way to do this is to send a distinct packet to each destination. Another
way is by flooding. A more efficient way is multi-destination routing. This can be achieved by inserting a list of the destinations into the packet. Alternatively, the packet can be simply copied to the next node. One more approach to broadcasting is to explicitly use a sink tree (or any other convenient spanning tree) for the source node.

2.3.13 Distributed Routing

Distributed routing is a category of algorithms rather than a single algorithm. One such algorithm requires that each IMP maintain a table that contains routes to each other IMP. The routing tables consist of entries. Each entry contains two parts: the preferred outgoing link to use for that destination, and some estimate of the cost to that destination. The cost can be calculated as a number of hops, estimated time delay, estimated total number of packets queued along the path, excess bandwidth, etc.

A variant called Street-sign routing implements table searches at each IMP to look up the next outgoing link that each message should use [6]. Interval routing is presented separately below.

2.3.14 Interval routing

Interval routing is a distributed routing scheme that distinctly labels nodes and links. Although this routing method does not implement searches for paths, it does use distributed tables, stored at nodes, successively to select the link that a message should take to reach its destination. To route a message \( m \) from node \( i \) to node \( j \), the \texttt{SEND} procedure is recursively executed at each node until it reaches its destination, if ever [14], [15]:

\begin{verbatim}
procedure SEND (i, j, m)
begin
    if \( i = j \) then process \( m \) else
    begin
        find label \( \alpha_k \) in the labeling at node \( i \) such that \( j < \alpha_k + 1 \)
        \( \hat{i} := \) the neighbour of \( i \) reached over link \( \alpha_k \)

        \( \hat{i} := \) the neighbour of \( \hat{i} \) reached over link \( \alpha_k \)

        \( \hat{i} := \) the neighbour of \( \hat{i} \) reached over link \( \alpha_k \)

        \texttt{SEND} (\( \hat{i} \), j, m)
    end
end
\end{verbatim}
\[ \text{SEND}(i, j, m) \]

end.

Labeling of nodes is implemented using the depth-first search approach. Once labeling of nodes and links is completed, path selection for each source-destination pair is fixed. A multi-dimensional interval routing scheme (k-IRS) is presented in [16].

### 2.3.15 Worm-hole routing

Worm-hole routing is a pipelined circuit-switching mechanism that is used in several architectures such as the Syrmul 2010, NCube, and Iwarp [17]. It uses on-line header processing. Once the header of a packet has arrived at an intermediate node, it is forwarded on to the next node immediately. The remainder of the message trails along behind its header. At particular nodes, such as the switch from one dimension to another, the header is removed, and the following flow-digits (so called flits) become the new header. Subsequent headers are repeatedly deleted until the message arrives at its destination. A small amount of storage for a few flits is provided at switches to allow for header checks and deletion [9], [10], and [6].

Once the first flit of a message is injected, the whole message must follow. No other messages are allowed on a link until the last flit of a message passed by. If the message is blocked, it freezes in the network. Every other message attempting to use a blocked link waits indefinitely until the link is free again. The provision of extra buffering allows for the temporary removal of a blocked message from the network, allowing other messages to proceed. This extra buffering is regarded as a virtual channel [18], to be discussed in Chapter 5.

When the message rate increases on a wormhole network, more messages become blocked. To allow a message to avoid blocked regions, it may routed along a more complicated path. In this case, the header must be large enough to contain all the details of that path before the message is sent. As the header size increases, compared with the information size, bandwidth efficiency drops.

Fully adaptive virtual cut-through (VCT), as proposed in [19], outperforms both deterministic and adaptive worm-hole. This means the throughput curve becomes saturated at lower traffic...
levels in wormhole compared to virtual cut-through. The reservation of some channels for
deadlock freedom makes free channels not fully available in wormhole [18]. The algorithm
developed later in this thesis is a distributed algorithm which adopts ATM-like scheme for
virtual pathing and bandwidth allocation. It uses a small-size fixed header for data packets.
A variant of wormhole routing is called universal routing is used in SGS-Thomson STC104
switch [20]. It randomly selects an intermediate node that is used as a “temporary” destination,
which then becomes a source that forwards packets to their final destination.

2.3.16 Time-Optimal Routing

In this algorithm, packets are deterministically routing to some intermediate nodes then deliv­
ered to their destinations. These intermediate nodes are randomly selected as described below.
The routing latencies are reduced by selecting intermediate nodes that act as a interim destination. The algorithm runs into three phases [21]:

- **Phase I**: divides rows into \(1/e\) strips of \(e\) rows, each \(e \geq \log(n)\). For a packet at \((i,j)\)
destined at \((r,s)\), it picks a processor \((k,j)\) at random in the same column and strip as
\((i,j)\). It then sends the packet to \((k,j)\) along the column;

- **Phase II**: sends the packet to \((k,s)\) along the row, then

- **Phase III**: sends the packet to \((r,s)\) along the column.

2.4 Previous research on path-finding

Path-finding algorithms select paths according to the current state of nodes and traffic; therefore
these algorithms are similar to shortest-path routing.

Distributed path-finding algorithms provide loop-free paths in various topologies by blocking
potential loops. Due to processing overheads, they are most suited for computer networks and
large Internets. In these algorithms, a router sends path information to its neighbours in update
messages of variable size that can contain the complete path.
2.4. Previous research on path-finding

Versions of the Distributed Bellman-Ford algorithm (DBF) generally implement iteration of distance calculations to find the shortest paths [22], [23]. The distance from any node $i$ to a given destination node is denoted by $x_i$ in a network of $n$ nodes. The $k$-th iteration of the DBF algorithm has the form:

$$
\begin{align*}
    x_i^k &:= \min_{j \in A(i)} (a_{ij} + x_j^{k-1}) \\
    x_i^1 &:= 0
\end{align*}
$$

$A(i)$ is the set of all nodes $j$ for which there is an outgoing arc $(i,j)$ from node $i$. The algorithm terminates after $k$ iterations if $x_i^k = x_i^{k-1}$ for all $i$. It converges to the solution for an arbitrary initial vector $x$ with $x_i = 0$. The iteration for each node $i$ can be carried out simultaneously with the iteration for every other node. The number of iterations strongly depends on the initial conditions.

The main drawback of the DBF is the looping (or counting) phenomenon that occurs when a node repeatedly attempts to exchange information about topology changes due to frequent link failures.

Due to the complexity and irregularity of the Internet topology, complex algorithms were developed. Internet routing based on the Routing Information Protocol (RIP) uses the Distributed Bellman-Ford. Large complex tables are continuously updated at nodes to record changes in topology [24]. A change in a link status triggers operations on tables, which causes an overhead.

Ercal and Lee [25] presented various algorithms for finding the Absolute Shortest Path (ASP), shortest duplex path (SDP), and the single shortest path (SSP) in a standard 2-D re-configurable mesh (RMESH). They assume an RMESH consisting of $n$ nodes that are circuit switched (bus connected) and arranged as columns and rows. In each node, any combination of the input ports can be connected to any combination of the output ports. This allows multiple buses to pass through a particular node.

1. In the ASP algorithm, nodes "listen" to signals on the bus by enabling all input and output ports. Blocked nodes disable all of their input and output ports. The source $r_s$ broadcasts a signal ($s$) on the bus. If destination node $r_d$ receives $s$, then there is a path between $r_s$ and $r_d$. 
2. To find all nodes that are ASP-reachable, a node $n_s$ broadcasts its coordinates $(x_s, y_s)$ to all the nodes and they record it. Each non-blocked node $n_i(x_i, y_i)$ enables some of its input and output ports according to the distance between $n_i$ and $n_s$. $n_s$ again broadcasts another signal $(x)$, and all nodes read the bus. Every node that receives $x$ marks itself as a reachable node from $n_s$.

3. To select one of the ASPs, the previous algorithm runs twice to find all reachable points to $n_s$ and $n_d$. Then each node $n_i$ communicates with its neighbours to determine whether they are in the set of ASP or not. Each node in that set enables its inputs and one of its outputs according to coordinates of $n_s$ and $n_d$. Finally, $n_s$ sends a signal $s$, and all the nodes that receive $s$ form a unique ASP. The ASP algorithm runs in $O(1)$ time. Another algorithm for finding the shortest duplex path (SDP) may be found in [25].

4. A fourth algorithm for finding a single shortest path between $n_s$ and $n_d$ iteratively prunes the unnecessary branches of the reachability tree (the reachability network for $n_s$ or the H-tree). The SSP algorithm runs in $O(N)$ in the worst case.

2.4.1 Other algorithms

- Murthy and Aceves [26] presented a series of path-finding algorithms (PFAs) based on Distributed Bellman-Ford. These algorithms are fully distributed and assume that no specific topology information is known. Under these algorithms (in [26]), each node stores path information to every destination. Whenever a node detects a change of topology (such as a link failure or a change of cost), that node updates its tables, and sends update messages to its neighbours, and so on.

These algorithms are: an ideal link state algorithm (ILS), a loop-free routing algorithm using diffused computations (so called DUAL), and a loop-free path-finding algorithm (LPA) ([27], [28], [29], and [30]). As these algorithms assume an unknown topology, more calculations for path finding are needed. Clearly there are nodes that do not communicate at all. In this case, the results of such calculations may never be needed nor used.

- In [13], a controlled flooding algorithm is presented. In this algorithm, each node is
assigned a cost, and every message carries a wealth. Once a message arrives at a node, it will be duplicated and forwarded along all outgoing links (except the link that it came from) whose cost is lower than the message's wealth. The cost of the link traversed is subtracted from the message's wealth.

2.4.2 Summary

A review of relevant routing algorithms has been presented. Fully distributed algorithms are in particular well suited for routing in multiprocessor architectures. Adding some form of "intelligence" to routers (at nodes) would increase the overall throughput, and allow asynchronous messaging between nodes. The algorithm designed for this thesis utilises a fully distributed approach which will be presented later in a separate chapter.
Chapter 3

Review of ATM

3.1 Introduction

In this chapter, a summary of transfer modes is briefly presented. One of these is Fast Packet Switching or Asynchronous Transfer Mode (ATM) which uses bandwidth allocation and virtual paths. These concepts will be reused in Chapter 4 when designing a practical solution for data transfer between the processing elements (PEs) in a multiprocessor parallel architecture.

3.2 Transfer Modes

A transfer mode was described by the CCITT (now the ITU) as "a technique which is used in a telecommunication network and the aspects of transmission, multiplexing and switching". Published research divides transfer modes into five categories: Circuit Switching, Multi-Rate Circuit Switching, Fast Circuit Switching, Packet Switching, and Fast Packet Switching (or ATM). The following sections briefly look at these modes. The rest of this chapter examines ATM in greater detail.

3.2.1 Circuit Switching (CS)

This is an approach in which a circuit is established for the complete duration of the connection. This mode has been used, and still is, in telephone networks. A common implementation of CS
3.2. Transfer Modes

uses time division multiplexing (TDM) for transporting information from one node to another sharing physical link.

In TDM, several connections are time-multiplexed over one link. Each connection uses a particular time slot in a frame for the complete duration of the session. Circuit switching can internally be performed by space-switching or time-switching, or a combination of both. The first is carried out by using different links for each connection. The later uses shared physical links at different time-slots to serve different connections.

The switching of a circuit of an incoming link to an outgoing link is controlled by a translation table. Circuit switching is simple, but is very inflexible as it requires constant synchronisation between end points while switching to time-slots, hence the bit rate is fixed.

3.2.2 Multi-Rate Circuit Switching (MRCS)

MRCS overcomes the inflexibility of a single bit rate in circuit switching. This is achieved in an identical switching network, but with the ability to allocate more than one basic channel to a connection. Therefore, a single connection can be made up from several fixed-rate basic channels. This option is retained for video-phony in narrow-band integrated service digital networks (NISDN).

MRCS requires synchronization of the individual channels belonging to the same connection. This synchronization makes the switching system more complex compared to pure circuit switching. Another disadvantage of MRCS is inflexibility in choosing the basic rate. A low basic rate requires a large number of channels for broad-band connections; therefore, more complex management of these channels is required. A high basic rate will waste bandwidth. The basic time-frame can be divided into time slots of different lengths, to provide a multiple basic rate solution.

3.2.3 Fast Circuit Switching (FCS)

FCS is suitable for sources of a fluctuating and bursty nature. Resources are allocated to services only when information is being sent, and are then released again when no information is being sent.
3.2. Transfer Modes

At call set-up, users request a connection with a bandwidth equal to some integer multiple of the basic rate. However, the system does not allocate the resources. Instead information on the required bandwidth and the selected connection are stored in the switch. The system also allocates a header (or a tag) to the signaling channel, identifying that connection.

When the source actually starts sending information, a request by the sender is made to allocate the necessary resources immediately. However, it may happen that the system is unable to satisfy the instantaneous requests because not enough resources are available. Therefore, any remaining resources will be not fully utilised.

3.2.4 Packet Switching

Under this mode, user information is encapsulated into packets that contain additional information (a header) which is used inside the network for routing, error correction, flow control, etc. A connection is often composed of a series of links. Complex protocols are necessary to perform error and flow control on every link of the connection. This link-by-link error control is often required due to the low quality of the links. Three generations of packet switching networks have been developed since the 1960s: X.25, Frame Switching, and Frame Relaying.

Packets of variable length require rather complex buffer management within the network. When the operation speed is not too high, software buffer control is feasible. Buffer management causes extra delays, hence this mode lacks time transparency.

The next generations of packet switching for NISDN were Frame-relaying and Frame switching. These have less functionality than X.25 and have better quality links. Table 3.1 shows a comparative summary.

3.2.5 Fast Packet Switching (or ATM)

Fast Packet Switching is also called Asynchronous Transfer Mode (ATM). It uses a small packet length (53 bytes) with minimal functionality in the network [2]. In ATM, a virtual connection has to be set-up between communicating nodes before any transmission can be initiated. Once the connection is set-up, user information is segmented into packets (or cells) of equal length. Packets can be inserted into the network access multiplexer at an arbitrary
3.3 Asynchronous Transfer Mode (ATM)

Information is routed to its destination using the information stored in the header. Hence, a header identifies a unique virtual connection. The header also allows an easy multiplexing of different virtual connections over a single link. The very limited functionality of ATM headers guarantees fast processing in the network. These headers contain routing information, i.e. identifiers of a virtual channel, a virtual path, etc. The detailed description of the header is shown in Section 3.3.2 on page 21.

The information field length in ATM cells is kept short. This offers the following two advantages: (i) reduction of internal buffers in the switching nodes, and (ii) limitation of queuing delays in buffers. It also guarantees a small delay and low delay jitters as required by real-time services.
3.3. Asynchronous Transfer Mode (ATM)

It is possible to route the information belonging to a virtual channel along different routes. Multiple parallel paths can be used to achieve an aggregate data rate up to the giga-bit range [33]. Considering that packets may encounter variable delays over the network links, they may arrive in a totally different order from that in which they were transmitted. The disadvantage of this method is that extra time is required to correctly reconstruct the information at the destination.

Limited functionality of the switching system allows the system to operate at a higher rate, compared to usual packet switching. The sender clock and the receiver clock are not synchronised. The difference between both clocks is resolved by inserting empty packets in the information stream. These packets do not contain useful information and are dropped at the receiver.

Errors such as bit errors, packet loss and packet insertion errors can happen in ATM networks. Errors caused by noise, such as transmission errors (single bit errors) and burst errors (multiple-bit errors), can occur in any transfer mode. Such errors are partially corrected, for example, using CRC.

3.3.1 ATM switching

An ATM connection is identified through two labels called the virtual path identifier (VPI) and the virtual channel identifier (VCI). The VPI can be viewed as a bundle of virtual channels. Each bundle must have the same end points. Hence, VPI is used to identify a group of virtual channel connections (see Figure 3.1).

Different virtual paths are multiplexed onto a physical circuit. Switching in the ATM network is performed by the ATM switch examining both the VCI and VPI fields in the cell or only the VPI field (see Figure 3.2). This choice is dependent on how the switch is designed and if VCIs are terminated within the network.

A virtual channel (VC) link is terminated when the VCI is assigned, translated or removed. Likewise, a virtual path (VP) link is terminated when the VPI is assigned, translated, or removed [1]. The VCI/VPI pair can be used in operations like point-to-point or point-to-multipoint communications, pre-established virtual connections or set-up on demand channels.
3.3. Asynchronous Transfer Mode (ATM)

3.3.2 ATM header

The ATM header is 5 bytes in length and consists of the following identifiers (see Figure 3.3):

- **GFC**: Generic flow control, 4-bits, user-network interface (UNI) only;
- **VPI**: Virtual path identifier, 8-bits in UNI, 12-bits in network-network interface (NNI);
- **VCI**: Virtual channel identifier, 12-bits;
- **PTI**: Payload type identifier, 3-bits;
- **CLP**: Cell loss priority, 1-bit; and
- **HEC**: Header error control, 8-bits.

The function of each field is summarized as follows [2]:

- **Generic flow control** (GFC) provides flow control at user-network interface for the traffic that originates at the user equipment and is directed to the network. It does not control the traffic in the other direction. It is only used outside the network for the implementation of different access levels and priorities, thus it has no use within the network. However, it can be used in the network to enhance path-identification capabilities. In this case, it can be a part of Virtual path identifier at network-network interfaces.
3.3. Asynchronous Transfer Mode (ATM)

Virtual path identifier (VPI) and virtual channel identifier (VCI) together provide the necessary information for packet routing. Virtual path is a collection of virtual channels between two nodes. At call set-up, a route is defined, and hence associated with a virtual path in the physical network. Each virtual path has its own bandwidth, limiting the number of virtual channels that can be multiplexed on a virtual path. Virtual path identifiers are used to distinguish between different connections. Virtual channel identifiers are used to route packets between two nodes that originate, remove, or terminate the virtual paths.

A payload type identifier (PTI) is used to define the payload type and is shown in Table 3.2.
3.3. Asynchronous Transfer Mode (ATM)

- **Cell loss priority** (CLP) this single bit field is used for cell-loss priority. If the single bit cell-loss priority field set in a cell, then this cell may be discarded by the network due to congestion. Cells with the CLP bit not set have higher priority and should not be discarded if at all possible.

- The **Header error control** (HEC) is used for discarding cells with corrupted headers and cell delineation. When it is used for header error correction, it provides single-bit error correction and low-probability corrupted cell delivery capabilities. It can also be used to identify the cell delineation.

![Figure 3.3: ATM header fields](image)

<table>
<thead>
<tr>
<th>PTI code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 000</td>
<td>User data cell, congestion not experienced, SDU type=0</td>
</tr>
<tr>
<td>1 001</td>
<td>User data cell, congestion not experienced, SDU type=1</td>
</tr>
<tr>
<td>2 010</td>
<td>User data cell, congestion experienced, SDU type=0</td>
</tr>
<tr>
<td>3 011</td>
<td>User data cell, congestion experienced, SDU type=1</td>
</tr>
<tr>
<td>4 100</td>
<td>Segment OAM flow-related cell</td>
</tr>
<tr>
<td>5 101</td>
<td>Segment OAM flow-related cell</td>
</tr>
<tr>
<td>6 110</td>
<td>Resource management cell</td>
</tr>
<tr>
<td>7 111</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 3.2: PTI status [2]
In ATM, there is no need for destination addressing or for sequence number. Instead, every virtual connection is identified by a number (identifier), which has local significance in the virtual connection. Identification of the virtual connection is performed by two sub-fields of the header: the Virtual Channel Identifier (VCI) and the Virtual Path Identifier (VPI).

The error control function can also be removed on high quality links. With optical links in mind, broad-band networks can allow up to ten thousand simultaneous channels on the same link. This requires up to 16 bit Virtual Channel Identifier. In ATM, the Virtual Channel Identifier is characterised at call set-up. When the connection is released, the Virtual Channel Identifier values will be released too, and can be reused by other connections.

Resources are allocated semi-permanently to allow for the simple and efficient management of resources on virtual paths. The Virtual Path Identifier can allow the management of these paths on a bundle of logical connections. The VPI header can also support the differentiation of logical connections by different priorities. Priority can divide the networks into different logical networks. However, it can also ensure that only low priority connections will lose information in the case of overloading.

The Payload Type Identification (PTI) field can allow the network to transport two types of information: data and maintenance. Special cells can be inserted, per virtual connection, and routed as normal cells, but which contain dedicated maintenance information. These special cells can be inserted and extracted in specific places in the network. Multiple access is allowed in some point-to-multi-point connections, e.g. multiple users on the same physical link. To achieve this, additional information is added to the header to indicate multiple recipients.

### 3.3.3 ATM performance

ATM performance depends on these factors:

1. **Time transparency**: Delay characteristics in ATM networks are very different from those of classical packet switching networks. The overall ATM network delay is the sum of:

   - *Transmission Delay (TD)*: which depends on the physical link bandwidth and the distance between both end-points, (typically 4-5 μs/km).
3.4 Services & Performance Requirements

- **Packetization Delay (PD):** i.e. the time of conversion of information into packets.

- **Switching Delay:** is composed of two parts.
  - **Fixed Switching Delay (FD):** caused by internal packet transfer through hardware.
  - **Queuing Delay (QD):** statistically caused by switching and multiplexing ATM packets. This delay varies with the load on the network and the behaviour of the queues.

- **De-Packetization Delay (DD):** caused by the reconstruction of the original bit stream.

2. Semantic Transparency: Errors in ATM networks are mainly caused by transmission and switching/multiplexing systems. The overall BER can be determined by three main factors [34]:

- Loss and incorrect arrival of bits of the information fields due to transmission errors,
- Loss of packets in the switching/multiplexing systems due to queue overflow, and
- Loss and incorrect arrival of packets caused by mis-routing due to misinterpretation of the header in the switching system.

3. Information field length: The choice of information field length is an important issue in ATM networks. The information field length can be either fixed or variable. Factors affecting the choice of information field length are: transmission bandwidth efficiency, switching performance, queuing memory size and management.

### 3.4 Services & Performance Requirements

A service can be described as a single connection with some bit rate. The bit rate of services varies from low bit rate (e.g. telemetry), to medium (e.g. voice), to high (e.g. High Definition TV or HDTV). Connection times also vary from a few minutes up to several hours. Therefore, different requirements exist for each service.
A single network that can cope with various types of current services (and is future proof) is required, a so called standard broad-band network. The increase of the number of users (i.e. customers) requires high speed switches, while trying to maintain quality of service (QoS). A closer look at a comparative survey of switches for commercial local area networks (LAN), and wide area networks can be seen in [35]. Technical challenges can be summarised as follows:

- **Compression** to reduce information volume (mostly graphics and video) while in transit,
- **Cost** stands for itself,
- **Management** at various levels: operations administration and management monitoring (OAM) for monitoring virtual circuits, and network management. OAM consists of three functions:
  - Fault and performance management (operations),
  - Addressing, data collection, and usage monitoring (administration), and
  - Analysis, diagnosis, and repair of network faults (maintenance).
- **Protocols and protocol processing** impose an extra processing overhead due to their complexity, and logical redundancy (considering low error rates in fiber-optic links).
- **Class of service** associated with a variety of users needs to be translated into fair levels,
- **Security** using encryption and fire-walls to protect commercial or confidential information from damage, misuse or unauthorised access, etc., and
- **Fault tolerance** to cope with hardware faults, software bugs, or "unusual" traffic patterns (possibly due to incorrect design decisions).

QoS is assessed according to errors and delays in the network [1]. A short list is:

- **Bit Error Rate** (BER) is defined as the number of erroneously received bits divided by the total number of bits transmitted over a representative period of time. Bit errors can occur as isolated (singular) errors or in groups (burst errors). The first are mainly caused by noise or system imperfections (e.g. due to imperfect clocks). Burst errors can be caused by packet errors or impulsive noise.
3.5. Traffic Management in ATM

The main objective of ATM traffic management is to ensure that each of the ATM bearer service categories is offered with an adequate quality of service (QoS).

3.5.1 Service Categories

Types of services that can be offered by an ATM network can be categorized into four groups [36]:

1. Variable bit rate - (no reserved bandwidth service)
   Bit rate is variable which allows a reduction of cost on the expense of quality of service. This type is suitable for applications that accept and adapt to network performance degradation (due to congestion) for economical reasons.

2. Constant bit rate - (reserved bandwidth service)
   This service offers no cell loss and a very low cell delay variation. Network resources are reserved for these connections to ensure that the specified Quality of Service is maintained. This service is intended for the carriage of CBR traffic such as voice, video channels and critical data transfers.
3. Variable bit rate - (reserved connection bandwidth service)

Cell loss rate and delay in this category are higher than those in the constant bit rate reserved-bandwidth-service to allow a lower Quality of Service. Higher Layer Plane Management Functions are used for statistical multiplexing of traffic. More efficient utilization of the network resources is achieved by specifying the service bit rate as a function of the peak rate of the connection and traffic parameters.

4. reserved burst bandwidth service category

This service can be offered at many peak bit rates but with a single burst blocking QoS. The user can choose one of a set of peak bit rates at subscription time. This choice is based on both application requirements and cost. Before sending a burst into the network, the source must send a request for burst transmission into the network and wait for confirmation. If no confirmation is received by the source, then it will hold its burst transmission and re-send the request again. It is allowed to send only after confirmation of acceptance is received. This service is suitable for applications that require spontaneous transfers of large bursts of data such as images or data files.

3.5.2 ATM Reference Model

Four classes of applications that can be supported by ATM are defined by the CCITT as the following [37]:

- Class 1: A continuous (constant)-bit-rate application such as pulse code modulation (PCM) telephony,
- Class 2: A variable-bit-rate non-data application such as compressed video,
- Class 3: A connection-oriented data application, and
- Class 4: A connection-less data application.

The reference model consists of three sections (see Figure 3.4) [1]:

- *The Physical Layer* which transports cells between source and destination.
3.5. Traffic Management in ATM

- **The Transfer Mode**, which are ATM protocol functions

- **The ATM Adaptation Layer (AAL)** is service-specific and consists of two parts:
  - **Constant bit rate** (CBR), and
  - **Variable bit rate** (VBR) can be further divided into two sub-layers: **Convergence** and **Segmentation & re-assembly** (SAR).

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timing</strong></td>
<td>Synchronous</td>
<td>Synchronous</td>
<td>Asynchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td><strong>Bit transfer</strong></td>
<td>Constant</td>
<td>Variable</td>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Connection mode</strong></td>
<td>CO</td>
<td>CO</td>
<td>CO</td>
<td>CL</td>
</tr>
<tr>
<td><strong>AAL type</strong></td>
<td>1</td>
<td>2</td>
<td>3/4 and 5</td>
<td>3/4 and 5</td>
</tr>
</tbody>
</table>

*CO*: Connection-oriented  
*CL*: Connection-less

Table 3.3: Support operations for AAL classes [1]

Higher Layer Functions are application-specific and can be classified into three main categories: signaling, connection-less, and connection-oriented services. The following sections summarise these layers.

1. **Physical layer** The function of the physical layer is to transport ATM cells between two ATM entities. It also guarantees (within a certain probability) the integrity of the cell header, and minimises user cells transmission overheads and generate a continuous bit stream across the physical medium. Therefore, physical layer functions are divided into two layers [2]:

   - **Physical media** (PM) sub-layer: which provides bit-transmission capabilities, and insertion and extraction of symbol timing information. In optical links, it also provides a transformation of signals from electrical to optical form and vice versa.
   - **Transformation convergence** (TC) sub-layer: performs HEC generation and verification, frame and cell delineation, and line coding. It receives cells from the ATM layer and pack them into the appropriate PM format. It also inserts idle cells.
3.5. Traffic Management in ATM

Connection-oriented data services
- VBR

Connectionless data services
- VBR

Connection-oriented voice/video services
- CBR

ATM adaptation layer (AAL)
- SAR

Segmentation and reassembly sublayer (SAR)
- SSCS

Service-specific convergence sublayer (SSCS)
- CPCSCPCS

Convergence sublayer (CS)

Physical Layer

Figure 3.4: The ATM layers

AAL: ATM adaptation layer
CBR: Constant bit rate
CPCS: Common part convergence sublayer
CS: Convergence sublayer
SAR: Segmentation and reassembly sublayer
SSCS: Service-specific convergence sublayer
VBR: Variable bit rate

These cells are identified by a specific header value and are not passed to the ATM layer.

Cell delineation determines cell boundaries in the stream received from the PM layer. According to CCITT Recommendation I.432, the receiver can be in any of the following three states (see Fig 12): hunt, pre-synch, synch [2].

In the Hunt state, the receiver monitors the incoming bit stream to detect a 5-byte word with correct CRC. Once the CRC is detected, it is assumed that this is a header. The receiver moves to pre-synch state.

In the Pre-synch state, the receiver searches for consecutive matches. If found, it moves from pre-synch to synch.

The Synch state is the normal receiving state. However, a consecutive number of mismatches (say) will cause the receiver to go back to the hunt state.
Four types of physical layer interfaces are used: The SONET STS-3, DS3, 100-Mbps multi-mode fiber, and 155-Mbps multi-mode [2].

- **SONET STS-3** physical interface operates at 155.520 Mbps (although the effective transport rate is 149.632 Mbps). Two main sub-layers exist: Transformation Convergence (TC), and Operations Administration and Management (OAM). The functions of the TC sub-layer are:
  - Header error control generation,
  - Cell framing indication,
  - Cell delineation,
  - Path signal identification,
  - Frequency justification / pointer processing,
  - Multiplexing, and
  - Transmission frame generation/recovery.

The functions of Operations Administration and Management (OAM) are:

  - Performance monitoring, which includes the monitoring of: cell header, line error, path error, and section error.
  - Fault management to provide detection, isolation, and correction of failure functions in the network. These are provided by the alarm indication signal (AIS), the far-end remote failure (FERF), and the remote alarm indication (RAI), to indicate the loss of cell delineation, or the loss of a frame, signal, or pointer.
  - Facility testing which permits verification of the connections between two path ends.

- The **DS3** physical interface operates at 44.736 Mbps. Cells are transported using the physical layer convergence protocol (PLCP). PLCP uses 12 ATM cells, each preceded by 4 bytes of overhead. To adjust the length of the frame, nibble stuffing is used after the 12th ATM cell. These 4-bytes are: 2-bytes frame alignment, 1-byte path overhead indicator, 1-byte path overhead. This gives a total of 40.704 Mbps effective bandwidth.
3.5. Traffic Management in ATM

- The **100-Mbps multi-mode** physical interface is intended to be used in private networks. An interface unit is used for connection with an ATM switch.

- The **155-Mbps multi-mode** physical interface uses 27-cell frames, which include 26 cells of payload, a 5-byte delimiter, and 48 bytes reserved for OAM functions. The payload rate is 149.76 Mbps.

2. The **ATM Adaptation layer (AAL)** is the protocol layer that converts higher-level *protocol data units (PDU)* into 48-byte ATM cells. In order for ATM to support many kinds of services with different traffic characteristics and system requirements, it is necessary to adapt the different classes of applications to the ATM layer. This function is performed by the AAL, which is service-dependent. Four types of AAL were originally recommended by CCITT. Two of these (3 and 4) have been merged into one, AAL 3/4. AAL5 was added later.

  - **AAL1** supports connection-oriented services that require constant bit rates and have specific timing and delay requirements. Example are constant bit rate services like the DS1 or DS3 transports.
  
  - **AAL2** supports connection-oriented services that do not require constant bit rates. In other words, variable bit rate applications like some video schemes.
  
  - **AAL3/4** is intended for both connection-less and connection oriented variable bit rate services. Originally two distinct adaptation layers AAL3 and 4, they have been merged into a single AAL whose name is AAL3/4 for historical reasons.
  
  - **AAL5** supports connection-oriented variable bit rate data services. It is a very lean AAL compared with AAL3/4 at the expense of error recovery and built in re-transmission. This trade-off provides a smaller bandwidth overhead, simpler processing requirements, and reduced implementation complexity.

AALs are composed of a convergence sub-layer (CS) and a segmentation and re-assembly (SAR) sub-layer. The CS is further composed of a common part (CPCS) and a service specific part (SSCS). SAR segments higher layer *protocol data units* into 48-byte chunks that are fed into the ATM layer to generate 53-byte cells. The ATM Forum is working on an AAL6 for supporting MPEG2 video streams.
3.6 ATM Bandwidth Management

Managing the available bandwidth to avoid congestion and provide guaranteed levels of Grade of Service (GoS) poses new challenges that are very different from the ones present in traditional packet-or circuit-switched networks. Bandwidth management strategies are also affected by the nature of the traffic.

The network must provide some ability to allocate and manage its finite resources (link bandwidth, buffer space, switch capacity etc.). That will allow guaranteed levels of service to all types of traffic.

3.6.1 Bandwidth management procedures

Bandwidth management procedures operate at two different scales: connection-level controls, and packet-level controls [3], [38]. The following Table 3.4 summarises these controls.

<table>
<thead>
<tr>
<th>Connection level controls</th>
<th>Packet level controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Bandwidth allocation</td>
<td>Access control</td>
</tr>
<tr>
<td>2. Path selection &amp; admission</td>
<td>Traffic monitoring &amp; adaptation control</td>
</tr>
<tr>
<td>3. Call set-up</td>
<td>Buffer management &amp; scheduling</td>
</tr>
</tbody>
</table>

Table 3.4: Bandwidth management procedures [3]

- **Connection-level controls** are applied at connection set-up time and are based on the connection characterization and the network state at that time. They include path selection and admission control functions that decide whether or not to permit a new connection access to the network, and determine which path the connection will be routed over. They also carry out bandwidth allocation and connection set-up functions to update (and distribute) the network state information, and to establish the connection.

Each connection has the following metrics: peak rate, mean rate, and average duration of a burst period. These metrics are initially used at set-up as parameters for a given connection.
3.7. Traffic and Congestion Control in ATM

- *Packet level controls* operate after the successful set-up of a connection. They ensure that data flow is at a steady rate and that the traffic injected into the network behaves as assumed. They are applied at the access points to the networks as well as within the network. At access points, they consist of a rate control mechanism and a traffic estimation module.

3.6.2 Bandwidth Allocation

As each connection requires an allocation of sufficient bandwidth, a mechanism for bandwidth allocation and removal is needed. One approach is to allocate a certain bandwidth for a connection for the duration of the connection life cycle. This does not allow for the efficient usage of networks efficiently, due to variation of the source burst rate. Alternatively, and more efficiently, the bandwidth allocated to a connection should be continuously adapted.

*Fast bandwidth reservation protocol* (FRP) proposed by [39] uses in-band signaling to negotiate changes to a connection's information transfer rate. This is achieved by sending a special request cell to network elements along the connection path. Network elements along the path will therefore attempt to reserve network capacity at the connection's peak rate. A successful allocation at all nodes will activate an acknowledgment which will be sent, informing the source that it can start its transmission.

3.7 Traffic and Congestion Control in ATM

ATM layer traffic control aims at providing three objectives [34]: *flexibility* in supporting various classes of services, *simplicity* to minimize network complexity, and *robustness* to achieve high resource efficiency under any traffic circumstances while maintaining simple control functions.

3.7.1 Basic ATM traffic control

There are two ATM traffic control functions. The first is used before the connection is utilised, and the second is used during the lifetime of a connection. These functions are [34]:
1. **Connection admission control (CAC)** describes actions of the network at call set-up to accept or reject an ATM connection. Acceptance occurs only if sufficient resources are available to carry the new connection at the requested QoS without affecting the QoS of existing connections. Hence the following information is negotiated and agreed between the user and the network to enable the CAC unit to make reliable connection acceptance/denial decisions:

- Specific limits on the traffic volume the network is expected to carry;
- A requested QoS class expressed in terms of cell transfer delay, cell jitter, and cell loss ratio; and
- A tolerance to accommodate cell delay variation introduced by Terminal Equipment or Customer Premises Equipment, which may alter the negotiated limits of the expected traffic volume.

This information may be renegotiated during the lifetime of the connection at the request of the user. The network itself may limit the frequency of these re-negotiations.

2. **Usage/network parameter control (UPC/NPC)** are performed at the user-network interface (UNI) and at the node-network interface (NNI) respectively. These represent the set of actions taken by the network to monitor and control traffic on an ATM connection in terms of cell traffic volume and cell routing validity, hence called policing. Ideally, a UPC/NPC algorithm should feature:

- the capability of detecting any illegal traffic situation;
- a rapid response time to parameter violations; and
- simplicity of implementation.

Controlling of traffic flow within the network typically relies on end-to-end exchanges of control massages in order to regulate traffic flow [3]. This also is called *explicit congestion notification* (ECN) [36], [40], [39]. The source node can use these control messages, possibly with added congestion information by intermediate nodes, to regulate its traffic. As the propagation delay across the network dominates the switching and queuing delays in high speed networks, the feedback from the network is usually outdated. In this case, any action the source takes
is too late to resolve the congestion. This argues for mechanisms that do not rely heavily on network feedback.

### 3.7.2 Generic cell rate algorithm (GCRA)

The ATM Forum and the ITU-T have defined algorithms for policing traffic at the sender. These use traffic parameters to detect excessive traffic. Congestion is reduced by regulating the traffic at the source, or so-called open-loop control. There are two equivalent versions of GCRA: the virtual scheduling (VS) and the leaky-bucket schemes (LB). Both VS and LB determine whether a cell is conforming or non-conforming to source traffic descriptor. Source traffic descriptors include the peak cell rate, sustainable cell rate, and the burst tolerance. The definitions of these parameters can be found in ATM literature such as [34] and [1].

*Virtual scheduling* (also called *cell delay variation tolerance*) compares the actual arrival time of a cell with the predicted arrival time (allowing some tolerance value) to decide if the cell arrived too early or not. If early arrival is found then the cell is non-conforming.

*The leaky bucket* version uses two parameters, the increment $I$ and the limit $L$ [1], [34]. The parameter $I$ affects the cell rate, and $L$ affects the cell bursts. An analogy of this algorithm is a bucket (hence the *leaky bucket algorithm*) with a finite capacity, containing liquid that leaks out at a continuous rate. The leaky-bucket allows controlling the peak load and smoothing out the burstiness of the input rates [41], [3]. Its contents can be filled (incremented) by $I$ if $L$ is not exceeded. Otherwise, the incoming cell is defined as non-conforming (see Figure 3.5).

### 3.7.3 Available bit rate (ABR)

ABR provides a mechanism for controlling traffic flow from LAN-based workstations and the routers that service these workstations. Several solutions are proposed. Among these are two based on *explicit congestion notification* (ECN): the *backward ECN* (or BECN), and *forward ECN* (or FECN). These two send the notification signal to downstream and upstream devices respectively.

Congestion information is continuously generated at each network element along the connection and is sent to the end-points. This information is carried as a single bit indicator in the cell.
header. This bit is set once a node on the connection detects a congestion by monitoring its buffer. Once the risk of congestion is over, this bit is reset appropriately. The source node uses these control messages to regulate its traffic [39], [40].

ECN facilitates the reduction of congestion, thus the cell loss ratio (CLR) is also reduced. As a result, the re-transmission of higher layer data units is greatly reduced, thus higher throughput of the network during congestion periods is achieved [39].

Congestion often occurs when links come under increased demand. It causes dramatic degradation of the overall network throughput. In worst cases, it may bring the network into a complete deadlock.

Two methods for congestion control are used: avoiding congestion in advance (to prevent its occurrence), or coping with congestion after it has occurred. Five algorithms for congestion control may be found in [12]. The first three follow the avoidance method, while the rest follow the second method.

1. Pre-allocation of resources: At the set-up of a virtual connection, a table of entries is created at each node the set-up request visits. These table entries reserve some resources (e.g. buffer size, or bandwidth). When the request has arrived at the destination, the route is defined, and the appropriate resources are allocated to that connection. If the resources are adequate, the problem of congestion is solved altogether [12]. The allocation of adequate bandwidth can ensure that each node can cope with the amount of incoming
traffic. This principle is used in our routing algorithm.

2. *Isarithmic congestion control* keeps the volume of injected traffic into the network below a certain limit (by limiting the number of packets). It uses permits that are circulated in the network. Before a node sends a packet, it must capture a permit and destroy it. Once the packet reached its destination and is consumed, the destination again generates the permit. This method however does not guarantee the prevention of congestion [12].

3. The *flow control* method is designed to restrict the mean rate of a sender to some limit. Flow control does not completely solve the congestion problem, as the peak rate can be much higher than the mean rate. Also, it is possible that more than one sender simultaneously transmits at their peak rates.

4. *Packet discarding:* Nodes discard extra packets at will once the buffers are full or the number of buffered packets has reached a threshold. The source (or previous) node can repeatedly re-transmit discarded packets until they are received. Another method is to keep timing out and re-transmitting until the packets are received. A combined method is to limit the number of re-transmissions and then to time-out.

5. *Choke Packets:* Each node monitors the percentage utilisation of each of its output links. If the utilisation of a link rises above a limit, the node sends a choke packet to the source, requesting the reduction of transmission rate by an amount. The source then reduces the rate for an interval. If no more chokes are received from the same destination, then the source may increase the rate again to its original level.

6. *Deadlock* (or lockup) occurs when nodes wait for each other to start transmission in cyclic form. Deadlocked nodes indefinitely wait, which wastes network resources, and hence congestion is more likely to occur. Avoiding deadlock relies on the prevention of dependency cycles.

7. *ECN* is used in ATM to control congestion, and typically relies on an end-to-end exchange of control massages (see Section 3.7.3 on page 36).

In a *Multi-stage* network, congestion information is dynamically sent backwards as feedback information to previous nodes by diverting part of the traffic to another route along the 'tree' [42].
3.8 Summary

This chapter presented features of Asynchronous Transfer Mode. Among these, the concepts of bandwidth allocation and virtual paths in ATM can be re-used for traffic management in multiprocessor interconnection networks. However, some modifications on ATM technology would be required to avoid a complex solution. Data transfer between processing elements would appear similar to a single type service, which should simplify the bandwidth allocation mechanism.
Chapter 4

A Path-finding Algorithm

4.1 Introduction

In this chapter, a description of a fully distributed algorithm for efficiently utilising resources in a multiprocessor interconnection network is presented. An efficient utilisation of resources basically relies on sharing resources in an orderly manner, to increase the overall throughput. The objective is to boost the overall routing performance of the network. In particular, the effects of heavily-congested areas (or hot-spots) in a $k$-$D$ topology are reduced compared to routing without bandwidth management. The hot-spot avoidance relies on minimal routing, thus it is a "partial" avoidance. A similar approach uses a "hot-spot avoidance strategy" (HSA), shown in [43], that is based on hot-potato routing. It uses semi-isolated routing (i.e. utilizes knowledge of the state of neighbouring nodes). HSA allows for full avoidance of congestion due to its use of non-minimal routing, thus packets are routed around hot-spots.

The minimal routing algorithm developed in this chapter combines both path-finding and bandwidth allocation to distribute excessive traffic away from congested areas, and to limit the traffic at hot spots to acceptable levels. The amount of traffic (number of packets per unit time) is spread more evenly (or near evenly) over links in congested areas. The solution uses minimal routing, that is with every packet’s move, the packet becomes closer to its destination. Minimal routing imposes limits on the number of possible routes that may be used. At nodes that are close to end-points, there will fewer choices of links, thus the scheme becomes less effective.
4.1. Introduction

The devised strategy manages point-to-point communications in the form of virtual paths (VCs). The decisions taken while creating these VCs are based on the “average” utilisation of links. The processing overhead of managing the bandwidth on links is kept to a minimum.

To allow routing decisions to be based on average traffic conditions, routing information is stored locally at nodes. Models for node structure, node functions, and traffic types are presented in Chapter 5. The algorithm presented here, together with those models is a complete scheme for implementing virtual paths in a mesh topology.

Packets often require multiple hops to reach their destinations. In practice, packets can compete over resources and congestion may arise, reducing performance. Therefore, it is vital to utilise network resources (buffers, bandwidth, etc.) efficiently.

The algorithm implements a single fixed (“static”) policy for routing and switching packets. As shown below, the policy is composed of a series of steps that are carried out collectively by a subset of nodes in the $k$-ary $n$-cube topology. It also implements a method for setting-up virtual paths between pairs of nodes, i.e. a point-to-point communication pattern. A single path $P$ between an arbitrarily-chosen pair of nodes in a “direct” $k$-$D$ mesh $M$ of radix $w$ (i.e. $w^k$ nodes) is used for communicating packets. A direct network is a network that allows input/output of packets at every node (i.e. a processor is attached to each node).

From an abstract perspective, messages for path operations (e.g. path set-up) can be viewed as pe-network-pe messaging (see Figure 4.1).

![Figure 4.1: A model for end-to-end path set-up messaging](image-url)
If more than one path exists between a given pair of nodes, only the “first available” path is selected. Traffic conditions at set-up favour selection of some paths over others. As these conditions change over time, a path that is selected at a given time may not be the best thereafter.

4.2 Assumptions and definitions

The following is a short list of basic assumptions made throughout this chapter. These assumptions, however, are similar to those found in many routing algorithms:

1. Nodes can control their output links, and have no control on their inputs except for blocking incoming messages.
2. Nodes are identical, and links have equal bandwidth (or capacities),
3. The bandwidth of the links from any node to its local processor is unlimited. This bandwidth is large enough to accommodate all packets incoming from and injected by the local processor at full rate (full capacity).
4. Nodes are permitted to temporarily block an incoming message, but indefinite blocking is not allowed.
5. Each message is eventually consumed at its destination node. Refusal or redirection of messages at their destinations is not allowed.

Failures of links within the network, and error recovery, are beyond the scope of this work.

4.2.1 Definition of bandwidth

The data rate of a link is defined in [44] as “the total amount of data transferred divided by the total time taken”. The bandwidth (or capacity) of a link is defined as “the maximum amount of data that can be transferred on the link divided by the total time taken”.

If the rate is expressed in bits-per-second (bps), then the bandwidth is the baud rate. The actual rate for data transfer is however smaller than the bandwidth. This is because a header
is transferred with each packet. In the current context of virtual paths, the amount of allocated bandwidth on a virtual path can be numerically represented. As an example, the full capacity is assumed to be 100 (i.e. 100%). Allocating a bandwidth of 25 to a path designates that the injection rate must not exceed 25% of the full link’s capacity.

4.2.2 Definition of path

A path, \( P \), in a directed graph is “a sequence of nodes \( (n_0, \ldots, n_k) \) with \( k \geq 2 \) and a corresponding sequence of \( k - 1 \) arcs such that the \( i \)th arc in the sequence is either a forward arc \((n_i, n_{i+1})\) or a backward arc \((n_{i+1}, n_i)\)” [44].

More specifically, to distinguish arcs of the two directions between a given pair of nodes (and their perspective bandwidths), \( P \) may be defined as: “an ordered set of uni-directional links that can transfer packets from a source node \((n_s)\) to a destination node \((n_d)\) at a specified bandwidth \((b)\) or rate. Also, \( P \) can visit any node only once”. Thus, the path cannot contain cycles.

The link can also be defined as “a unidirectional connection between two neighbouring nodes”. The available bandwidth of a path is eventually limited by the smallest available bandwidth on its individual links.

Several paths may share some links if their bandwidth is sufficient. Similarly to virtual paths (VP) in ATM [34], the path is a virtual connection that is:

- **Uni-directional**, that is from the source to the destination;
- **Limited in bandwidth**, and that bandwidth is quasi-guaranteed at set-up; and
- **Intermediate nodes are transparent to traffic** (except for header processing).

This definition ignores the details of the links comprising the path. In a \( k \)-D mesh, assume that \( n_s = (s_1, \ldots, s_k) \) and \( n_d = (d_1, \ldots, d_k) \). The source is defined by its coordinates within the mesh. The distance (or shift) from the source to the destination is defined by the number of hops between along each dimension, i.e. the distance along the dimension \( i \) is denoted as \( d_i \). Thus, using relative addressing, \( n_d \) can be specified with respect to \( n_s \), i.e. as its distance from
4.3 Modeling nodes and messages

The Manhattan distance $D$ is the number of hops from $n_s$ to $n_d$ [25]. If the distance along dimension $i$ is $\delta_i = (d_i - s_i)$, then

$$\Delta = (\delta_1, \ldots, \delta_k) = ((d_1 - s_1), \ldots, (d_k - s_k))$$

For a 2-D mesh, $n_s = (s_x, s_y)$ and $n_d = (d_x, d_y)$, so the distance $\Delta = (\delta_x, \delta_y)$. Assume $L_{s,d}$ is the ordered set of links $l_{i,j}$ that comprises path $P$. The path is:

$$P_{s,d,b} = \{n_s, \Delta, b, L_{s,d}\}$$

The three parameters $n_s$, $\Delta$, and $b$ define a class (or a set) of paths $P_{s,d,b}$ from $n_s$ to $n_d$, or:

$$P_{s,d,b} = \bigcup P_{s,d,b}$$

The region $R_{s,d}$ is the sub-mesh whose $n_s$ and $n_d$ are at its corners:

$$R_{s,d} = \bigcup_{i=s_1}^{s_1+\delta_1} \bigcup_{j=s_y}^{s_y+\delta_y} n_{i,j}$$

where $i, \ldots, r \in \{0, 1, \ldots, w - 1\}$.

In 2-D mesh:

$$R_{s,d} = \bigcup_{i=s_1}^{s_1+\delta_1} \bigcup_{j=s_y}^{s_y+\delta_y} n_{i,j} \text{ where } 0 \leq i \leq w - 1 \text{ and } 0 \leq j \leq w - 1$$

The source [45] states that the number of possible paths in $R_{s,d}$ is:

$$\frac{(|\delta_1| + \ldots + |\delta_k|)!}{(|\delta_1|)! \ldots (|\delta_k|)!} \text{ and } \frac{(|\delta_x| + |\delta_y|)!}{(|\delta_x|)! (|\delta_y|)!}$$

Figure 4.2 shows an example of a path (the shaded line) and its conventions in a 2-D mesh. If only a single path is allowed between any pair of nodes, then the path would be simply called $P_{s,d}$ instead of $P_{s,d,b}$ assuming that the bandwidth is value is valid.

4.3 Modeling nodes and messages

A standard $k$-Dimensional (or $k$-D) mesh $M$, consisting of $w^k$ nodes, is labeled as $n_{i_1,i_2,\ldots,i_k}$.

Using CSP, $M$ can be specified as a collection of $w \times w$ processes $N$ (see Figure 4.3). Each

The symbol $b$ is used to indicate the requested bandwidth $b_r$ that has actually been allocated. In a fully dynamic schema, intermediate nodes may offer (or negotiate) a lower value.
4.3. Modeling nodes and messages

node runs asynchronously from the other nodes; there is no global clock (i.e. synchronisation
clock). Nodes can only communicate via an exchange of messages over the links. Faults in
nodes and links are beyond the scope of this work.

The node’s model \( N \) is composed of (Figure 4.3): a processing element \( pe \), a routing machine
\( rm \), and a set of unidirectional links (a bidirectional link is a pair of two unidirectional links).
There are no wrap-round links on the mesh.

A general model for \( N \) in a \( k\)-\( D \) model as shown in Figure 4.4. Each node \( n_{i_1,i_2,...,i_k} \) is an
instance of \( N \) (i.e. \( i_1 : i_2 : \ldots : i_k : N \)) that interacts with its environment (neighbours)
through communication channels.
4.3. Modeling nodes and messages

The two independent parts, pe and rm of each node forever and asynchronously run in parallel, or \( N = pe \parallel rm \) (in CSP notation). Both rm and pe can only synchronise on common events, that is messages over the links \( in_0 \) and \( out_0 \) (also called \( out_p \) and \( in_p \) respectively). \( rm \) does routing messages as well as resolving competition for resources (e.g. links) between packets. In this model, a packet incoming from any input \( in_i \) can be routed virtually to any output \( out_j \).

The structure of \( rm \) and \( pe \) will be described in the following sections.

The node's model \( N \) has \( 2(k+1) \) links which are arranged as follows:

- \( in_1, \ldots, in_k \) and \( out_1, \ldots, out_k \) are the input and the output links to nodes in the positive
4.4 Algorithm for path-finding

4.4.1 Requirements specifications

The algorithm requirements are similar to those found in the literature (such as [18], and [46]), and are listed below:

R.1 Set-up a single virtual path $P_{k,d}$. Once set-up is completed, there remain no “branches” (active or idle) for the same path.

---

2This link indexing is chosen to allow the selection of the link in the opposite direction by changing just one bit in the binary representation (i.e. $k_{th}$-bit or the MSB) of a link label.
R.2 The path has no cycles.

R.3 The path is chosen to be short, but not necessarily the shortest.

R.4 The allocated bandwidth must not exceed the maximum bandwidth of its links.

R.5 The bandwidth $b$ of the path is fixed during the life of the path.

R.6 On demand, any link must provide the required bandwidth $b_r$ if its available bandwidth $b_a$ is sufficient, that is $b_a \geq b_r$.

R.7 The allocated bandwidth $b$ is reserved on all links comprising the path, and other paths cannot use $b$, even if the path is "idle".

R.8 The path must be closed if it is no longer needed.

R.9 The reserved bandwidth is released upon closure; available bandwidth is re-adjusted and immediately made available to other path requests.

R.10 If any request fails to set-up a path, all the bandwidth that it allocated on links must be released.

### 4.4.2 Description of the algorithm

The algorithm utilises topology information (mesh orientation) to find a virtual path between any pair of nodes with a quasi-guaranteed bandwidth. It also allows the sharing of physical links among virtual paths via the selection of routes that avoid congested areas or hot-spots. As the traffic pattern changes over time, a bounded search for a possible path has to be implemented. The search would only involve the nodes in the sub-mesh with the pair of nodes at its corners (i.e. $n_i \in R_{s,d}$). Other nodes $n_i \notin R_{s,d}$ need not "know" about the path $P_{s,d}$ at all.

Each node keeps track of available resources by saving state information locally. Each node is a state machine that maintains a small amount of storage in the form of a routing table (RT) which is continuously updated as paths are created and destroyed over time. Once a path is discarded upon path closure, its related information is also discarded (i.e. no history is maintained). The detailed structure of the routing table is shown in Section 4.4.4 below.

Once a path is set up, "data" messages must follow that selected route similarly to street-sign routing [6]. Each data packet includes a path-specific header. The header is checked against
stored information at each intermediate node, then the message is directed to the appropriate output link. If that output link is not free, the message is blocked until the link becomes free again. The delay of message forwarding in this algorithm is similar to any "store-and-forward" algorithm, i.e. $O(d)$; where $d$ is the number of links comprising the path.

To close the path, the source sends a closure message down the path. In turn, the nodes serially remove the path information from the routing tables, and release the allocated bandwidth on their links. The released bandwidth is immediately made available to requests for other paths.

A similar "pipeline" table-lookup is used in the SPIDER router to reduce latency [47]. Upon receipt of a message at a node, the node uses the destination's identification (or ID) included in the message to look up routing instructions in the table. The table returns a direction or exit port which the next SPIDER chip uses for crossbar arbitration.

### 4.4.3 Stages of the algorithm

A communication over a virtual path from a source node $n_s$ to a destination node $n_d$ follows a sequence of three steps: (i) establish a connection, (ii) send the data over that connection, then (iii) close that connection. To do these, the algorithm runs the following phases to set up a path $P_{s,d}$:

1. **Exploration** of the links in the region $R_{s,d}$ to find a possible route;

2. **Building** a single short path $P_{s,d}$. Every other path (or sub-path) will have to be "rejected" (or "closed");

3. **Acknowledgment** that set-up of one path $P_{s,d}$ is completed;

4. **Utilisation** of the path $P_{s,d}$ by sending data over that path; and

5. **Closure** of $P_{s,d}$ once it is no longer needed.

The first two are performed in a single phase, the *construction* phase. The remaining are performed in *acknowledgment*, *utilisation*, and *closure* phases respectively. The source $n_s$ initiates this *construct-utilise-close* activity. The destination $n_d$ initiates the *acknowledge* function. The
4.4. Algorithm for path-finding

State diagram of the paths is shown in Figure 4.5. Utilisation (i.e. data traffic) is performed using one of the known routing methods, such as street-sign routing or as basic table-lookup.

A failing search does not complete the first phase; instead the state machine immediately proceeds to the closure phase. Extended blocking of the destination's acknowledgment may occur. Consequently, a successful search may fail if the source uses a time-out to check for the acknowledgment message (i.e. the dashed line in Figure 4.5). Some phases may overlap in time as shown below.

After the exploration has succeeded, each node \( n_t \in P_{s,t} \) should have stored the path parameters and a pair of link pointers to the predecessor (\( \text{pred} \)) and the successor (\( \text{succ} \)) nodes in its routing tables. Using these link pointers, individual nodes are chained to form a complete path.

Each node in the mesh contains a set of routing tables (RTs), one for each output link (see 4.4.4). Input links do not require routing tables because nodes can control packets only at the outputs. Each routing table is an array of entries associated with an input link. The routing table stores through-path information as entries. An \( RT_i \) of an input link \( l \) is:

\[
RT_i = \bigcup_{j=0}^{m-1} y_{i,j}
\]

where \( m \) is the maximum number of entries in \( RT_i \). Each entry \( y_{psd} \) in a routing table is a set of parameters of a path, and it consists of: \( y_{psd} = \{ n_s, \Delta, b, \text{pred}, \text{succ} \} \).

All routing table entries are initialised to the "empty" state at start-up. Since each routing table is uniquely attached with one output link, \( \text{succ} \) may not be stored. Instead, it can be obtained
4.4. **Algorithm for path-finding**

from the routing table's index. The structure of the routing tables are described in Section 4.4.4.

The stages of the algorithm are:

1. **Exploration**

   Exploration is performed via sending request messages \( m_{req} \) along some of the links. The choice whether to send (or not to send) the request message \( m_{req} \) along a direction \( i \) depends on:

   (a) whether the message has non-zero distance to travel in this dimension; and
   (b) the bandwidth on this link is sufficient for \( m_{req} \)'s request.

Directions that \( m_{req} \) is sent along are determined by the distance \( \Delta \) to the destination node \( n_d \), that is \( \Delta = (\delta_1, \delta_2, \ldots, \delta_k) \), \( \delta_i \in \mathbb{N}, i = 1, 2, \ldots, k \). Each \( \delta_i \) indicates the direction and the distance to node \( n_d \). The step is equal to +1 (i.e. a single hop in the positive direction) if the distance is positive on that direction. Likewise, if the distance is negative for that direction, then the step is -1. The step value is zero for a zero distance or insufficient bandwidth. The step vector \( S \) that is used to determine the steps along each direction is \( S = (s_1, s_2, \ldots, s_k) \). Each \( s_i \) step is:

\[
\forall i \in \{x, y\} \exists s_i \left\{ \begin{array}{ll}
-1 & : (\delta_i < 0) \land (b_a > b_r) \\
0 & : (\delta_i = 0) \lor (b_a < b_r) \\
+1 & : (\delta_i > 0) \land (b_a \geq b_r)
\end{array} \right.
\]

The \( b_a \) on any output link \( i \) is the difference between the maximum bandwidth (\( b_{max} \)) and the sum of allocated bandwidths \( b_{a,d} \) to paths on that link. Evaluating this term requires searching all the routing tables.

If \( S = 0 \), then \( m_{req} \) cannot be accepted and the search cannot proceed any further. Again, if \( S=0 \) and \( \Delta > 0 \), then the search has failed and the incomplete path must be closed (by echoing back a rejection message \( m_{req} \)). Upon accepting a request \( m_{req} \), each node \( n_t \in R_{a,d} \):

(a) stores path information as a new entry in the routing table, and
(b) sends a copy of \( m_{req} \) to the nodes in the correct direction based on the step value for that dimension (i.e. to the "prospective" nodes).

A failing search does not complete the first phase; instead the path immediately "collapses" and proceeds to the closure phase. Extended blocking of the destination’s acknowledgment may occur. Consequently, a successful search may fail if the source uses a time-out to check for the acknowledgment message (i.e. the dashed line in Figure 4.5).

The pattern of \( m_{req} \) migration is similar to agents in the WAVE language \([48]\). The agent \( m_{req} \) has mobile variables (that is \( n_s, \Delta, \) and \( b \)), and updates nodal variables (that is \( b_h, \) and \( RT_h \)). An example of the wave pattern of \( m_{req} \) in a 2-D mesh is shown in Figure 4.6. Once \( m_{ach} \) has reached \( n_s \), then the path set-up has been successfully completed.

A rule of "first-served rest-rejected" (FSRR) is used during exploration. The nodes executing the Explore function implement that rule in turn. That is, for a certain path, only the first request can be accepted. Any further requests are rejected.

This FSRR approach:
4.4. Algorithm for path-finding

(a) allows for the selection of the first available path (essentially fastest path selection). This results in less processing overhead of redundant requests;
(b) avoids prolonged exploration that leads to a slower path set-up. A full path finding search may never terminate due to extensive waiting that can live longer than the path-life.

The nodes $n_i \in R_{a,d}$ may receive up to $k$ copies of $m_{req}$ belonging to the same path request (see Figure 4.6). Using the FSRR rule, the first $m_{req}$ may be accepted. Any further copy $m_{req}$ arriving at any other input indicates merging of a new branch, and hence is rejected. A rejection message $m_{rej}$ is accordingly composed and "echoed" back to "kill" that new branch. At that occasion, the rejection of $m_{req}$ may miss out some outputs that have recently increased their $t_b$ (due to closure of some other paths).

2. Acknowledgment

The Acknowledgment message $m_{ack}$ is used to confirm path availability, and to remove sub-paths that may exist after a successful search reached it destination. Using $m_{ack}$, an acknowledgment is serially forwarded from one node to another up the path. Each node updates its routing tables to include an index of the path entry at the previous node, i.e. down the path, and is called "next". This is referred to as the "look-ahead" mechanism.

As $m_{ack}$ advances up the path, it may find a sub-path still active. In this case, a "closure" message is sent down that sub-path to close it. This closure message (as described below) releases the allocated bandwidth along the sub-path.

The values of $b$ is carried along within $m_{req}$. It is then used to adjust the available bandwidth on links where redundant sub-paths are to be closed. It is noticeable that $b$ can be looked-up from the routing table using the "next" index. However, it may be faster to copy the $b$ value in the message than using the "next" index for fetching $b$ from the routing table.

3. Rejection

Rejection is used to remove one or more links from a path or sub-path starting from its last node and then working backwards. Using a rejection message $m_{rej}$, rejection can be iteratively repeated from one node to another up the path or the sub-path. The $m_{rej}$
releases the allocated bandwidth on a link (i.e. the link is removed from the virtual path). Rejection of a request \( m_{\text{req}} \) for bandwidth reasons happens if either:

(a) A request \( m_{\text{req}} \) cannot be "served" due to insufficient bandwidth on one or more of its required links; or

(b) A sub-path is rejoining the path.

If \( m_{\text{req}} \) reaches a node that has sub-paths, \( m_{\text{req}} \) does not proceed further to allow the remaining sub-paths to either mature or die. It is noticeable that neither \( \Delta \) nor \( b \) are essential for \( m_{\text{req}} \). However, it is more efficient to have \( b \) "in hand" instead of fetching its value when updating \( b_0 \).

4. **Path Closure**

At some stage, the source processor (\( pe_s \)) decides that a path is no longer needed. In this case it does two things: (i) it stops sending data packets, and (ii) it initiates path closure. It sends a closure message \( (m_{\text{cls}}) \) down the path. Closure is similar to rejection, but it starts from the first node (the source) and then proceeds forwards down the path.

As \( m_{\text{cls}} \) advances toward \( n_d \), it causes iterative destruction of the path, one link at a time. Each node will release the reserved bandwidth, \( b \), on its link, and then forwards \( m_{\text{cls}} \) to the next node. Once \( m_{\text{cls}} \) has reached \( n_d \), then the path is destroyed and no longer exists. \( n_d \) does not need to send any acknowledgment message to confirm the path closure.

5. **Forwarding data packets**

The source processor (\( pe_s \)) starts sending data in the form of \( m_{\text{del}} \). Each data packet is bound to follow the path down to its destination where it will be "consumed". At this point there should be an existing single path (with no branches) \( P_{s,d} \). Using \( m_{\text{del}} \), data messages are serially forwarded from each node to the next node down the path.

The format of \( m_{\text{del}} \) uses a "look-ahead" forwarding mechanism that is similar to street-sign routing. It includes an output pointer \( \text{out} \) to indicate which link the packet should use at the current node, and an index \( \text{next} \) that points to the entry that should be used for table-lookup at the next node.
Inter-leaving of multiple paths operations allows for the management of virtual paths. Virtual paths are independent. Therefore, any path operation can proceed concurrently with other operations. At any node, processing of a packet belonging to a path is not interleaved with processing of packets of other paths. However, interleaving between paths is very possible. Inter-leaving of many paths’ operations at a node does not affect any other path except for the competition over bandwidth. Such competition may fail some paths. This is discussed in Section 4.6.7 where, to simplify the presentation, the effect of other paths on a new path is ignored.

The use of Relative addressing of the destination node requires that the distance \( \Delta \) be updated at each node \( n_x \in R_{a,d} \) in order to detect the arrival of a packet at its destination node \( n_d \) (i.e. \( \Delta = 0 \)). If absolute addressing were to be used, \( n_d \) should be used instead of \( \Delta \). This would be especially useful to simplify the forwarding of packets, as no modification of the header along the path would be needed. To simplify comparisons with entries in the routing tables, both \( \Delta \) and \( n_d \) can be used in messages. However, the simulation implementation used does not use this feature.

In addition to the path operations above, there are update messages (of type \( m_{upd} \)) between internal routing layers. These update messages are processed internally (within the node). Update messages are presented in the next chapter (see page 77) and provide for the consistency of information stored in the tables.

4.4.4 Routing Tables

Each output link of each node is associated with an independent routing table (RT). The routing table serves as a state storage for that output link. The contents of each table (and the operations on it) are independent from other routing tables. A typical routing table of \( m \) entries is shown in Figure 4.7.

Searching and updating routing tables can be done in any table-look-up method. As the path set-up operations flow up and down the path, each on a separate routing layer, certain guards are imposed on accessing the tables to maintain consistency and ensure valid contents (i.e. concurrent-read-exclusive-write or CREW). The details of routing layers and routing tables access is shown in the next Chapter 5.
4.5. The Algorithm in CSP

In CSP, the process $N = Node$ of which multiple instances are created has the alphabet:

$$\alpha N = \{\text{Explore, Reject, Acknowledge, Close, SendData, Consume, Generate}\}$$

The last two functions on a message $m_{msg}$ are simply:

$Consume = \text{in}_{in} ? m_{msg} = \text{out}_{out} ! m_{msg}$ and $SendData = \text{out}_{out} ! m_{msg} = \text{in}_{in} ? m_{msg}$.

Packets are either of the type control $m_{con}$ or the type data $m_{dat}$:

$$m_{msg} = m_{con} \text{ | } m_{dat}$$

$$m_{con} = m_{req} \text{ | } m_{ack} \text{ | } m_{req} \text{ | } m_{cts}$$

If $m_{msg} \in \alpha(in_j)$ where $0 \leq j \leq (2k + 1)$, the type of a message $m_{msg}$ is $MsgType(m_{msg})$:

$$MsgType(m_{msg}) = \text{req} \text{ | } \text{req} \text{ | } \text{ack} \text{ | } \text{cts} \text{ | } \text{dat}$$
4.5. The Algorithm in CSP

\[ m_{req} = \{req, n_s, \Delta, b\} \]
\[ m_{rej} = \{rej, n_s, \Delta, b\} \]
\[ m_{ack} = \{ack, n_s, \Delta, b\} \]
\[ m_{ds} = \{ds, n_s, \Delta, b\} \]
\[ m_{dat} = \{dat, out, next, data\_elements\} \]

The alphabet of inputs is:
\[ \alpha(in_j) = \{m_{req}, m_{rej}, m_{ack}, m_{ds}, m_{dat}\} , \text{ for } 0 \leq j \leq (2k + 1) \]

The node's input set is:
\[ \mathbf{i}_N = \bigcup_{i \geq 0} \text{ in}_i \cap m_{msg} \]

Lower-level functions (such as searching and updating the tables) are not represented.

Within node \( N \), the processor element (pe) performs one of the following three functions:

1. Sends messages to \( rm \), that is “injects” messages into the network;
2. Consumes messages that are destined to it; or
3. Performs computations internally (thinks).

Thus the alphabet of a general pe process (called PE) is then:
\[ \alpha PE = \{Inject, Consume, Think\} \]

The PE consumes all messages on its input, and may inject messages into the network at regular or irregular intervals. However, the mean rate of injection is assumed below a pre-specified rate or level determined by the bandwidth of the connection that has been set up.

If no messages have arrived at node \( N \)'s inputs, \( N \) will have no external activity and simply waits doing nothing (while pe also carries on “thinking”!). If more than one message is active
4.5. The Algorithm in CSP

at its inputs, $N$ processes them in a fair manner that prevents starvation. $N$ internally chooses which message $m_{msg}$ to process first (see Figure 4.8):

$$N = ((\text{Think} | \text{Generate}) \rightarrow N) \text{ in}_i^\uparrow m_{msg} (\text{Process}) \rightarrow N)$$

where $0 \leq i \leq (2k + 1)$

![Diagram of states of node $N$](image)

**Figure 4.8: States of node $N$**

Think specifies the state of no external activity of node $N$. While Thinking, $N$ is not necessarily idle, but it can do any internal activity such as internal computations (housekeeping). $N$ must, however, immediately (or within an acceptable delay) detect and respond to an external interrupt (which occurs when a message arrives at one of its inputs). Generate is the node's internal initiation of activities with no timing relation to events at inputs. A typical example of these activities is a path set-up request (i.e. application-level request). Another example is an "active" source node sending data packets along an established path. Generate typically is a source requesting a fresh path set-up after it had enough Thinking. Process is the node $N$'s response to $m_{msg}$ that we shall characterise below. Process may cause $N$ either to produce some messages, or simply to Consume the message. Once $N$ has finished with Process or Generate, it returns to enjoy its Thinking.

Apart from Generate, $N$ is an event driven device that is "triggered" only by events at inputs; that is packets arriving at inputs. Generate After initialisation, $N$ moves to its idle state (i.e.
4.5. The Algorithm in CSP

Each time a packet arrives at \( N \), it does some function, and then goes back to its idle state waiting for the next event. The response of \( N \) upon the events of receiving a control packet \( e_c \) or data packet \( e_d \) is:

\[
N \rightarrow (e_c | e_d) \rightarrow \text{Process} \rightarrow N
\]

The response \( \text{Process} \) of a node \( N \) upon receiving a control packet \( m_{\text{con}} \) or a data packet \( m_{\text{dat}} \) at an input \( \text{in}_j \) is:

\[
\text{Process} \triangleq N \rightarrow \\
(( \text{in}_j \rightarrow m_{\text{con}} | \text{Explore} | \text{Acknowledg} | \text{Reject} | \text{Close} ) ) \\
( \text{in}_j \rightarrow m_{\text{dat}} \rightarrow \text{PassData} ) | \text{Generate} )) \rightarrow N
\]

The functions \( \text{Explore}, \text{Acknowledg}, \text{etc.} \) need to be specified in CSP. Only a brief specification for \( \text{Explore} \) is shown:

- \( \text{Explore} \): Both path finding and construction are performed in one step. Except at \( \eta_b \), where \( m_{\eta\text{req}} \) is internally generated (by the \( p_{\eta\text{eq}} \)), the node \( n_i \in R_{d,\text{ps}} \) internally chooses to either \( \text{Accept} \) or \( \text{LReject} \) the request that is:

\[
\text{Explore} \triangleq \text{Accept} \cap \text{LReject}
\]

where:

- \( \text{Accept} \) specifies that the request has been accepted by a node, and as a result, one new link would be added to the path, and
- \( \text{LReject} \) (i.e. a “local” reject) specifies that a node has “rejected” the request, thus it sends a reject message \( m_{\text{req}} \) back to the node that issued that \( m_{\text{req}} \).

Notably, upon receiving a message, \( N \) executes some “response” to that event and becomes again ready for another event. The execution time is finite, i.e. it terminates within some time-limit. This feature will be used to argue algorithm properties in the next section.
4.6 Characteristics of the algorithm

4.6.1 Correctness

It is possible to examine the correctness of path functions against the individual requirements. For example, R.1 and R.2 (on page 47) are achieved by the CSP function `Explore`. A low-level code-check of the implementation code would prove that assumption.

Lemma 1: For a given path, the first request \( m_{req} \) at \( n_j \in R_{s,d} \) causes the exploration of all the links that lead to \( n_d \).

Proof: \( m_{req} \) searches the tables of input links. The step vector \( S \) is designed to select all the appropriate directions (shown on page 51).

Lemma 2: Only one virtual path is possible between any pair of nodes.

Proof: Without loss of generality, assume that the three arbitrary-chosen nodes \( n_a, n_b, \) and \( n_c \) are in order on the selected path. Assume also that a subsequent sub-path is developed from \( n_a \) to \( n_c \) through a fourth node \( n_e \). The latest request \( m_{req} \) arriving at \( n_c \) through \( n_e \) is rejected upon implementing `Explore` by \( n_c \). The rejection message is sent back to \( n_a \) through \( n_e \). Rejection iteratively removes links of the sub-path back to \( n_a \). At \( n_a \), another entry of the path prevents rejection to proceed backwards beyond \( n_a \). This process is iterated for every sub-path. Hence, only one path is possible between any pair of nodes \( n_a, n_j \in R_{s,d} \) including the end-nodes \( n_a \) and \( n_d \).

Lemma 3: The algorithm is live-lock free.

Proof: Live-lock occurs when a packet is continuously routed away from its destination. This may occur in some adaptive routing. In this algorithm, routing of packets is deterministic and minimal. Packets can either arrive at their destinations, or become temporarily blocked. Therefore, live-lock is not possible.

4.6.2 Termination

Termination of the algorithm can be looked at on two levels: “local” and “global”. The first one deals with the termination of each function of the algorithm. Global termination addresses the overall mesh activity at various stages of path finding.
4.6. Characteristics of the algorithm

It is essential that any node (and thus the whole network) should respond to any message in finite time, to guarantee termination of the path functions (see the next section). As responses to packets is based on the router state, the functional design of the router ensures that a message (or response to a message) does not repeat itself indefinitely. It has been shown through careful design that no messages can bounce back and forth forever.

4.6.3 Local termination

At the node level, termination of all functions within the nodes is assured by the specification of responses to individual events. The specification of these functions should shows the absence of undesirable repetitions. For example, Explore terminates successfully on completion of one of the following: adding one new link to the path; the rejection of request; or an acknowledgment being initiated by the destination node. Similarly, the remaining functions also terminate successfully.

For a given path, the activity of any node involved in a path set-up ends when that node receives no further set-up control packets belonging to that path. The transit of data packets of that path is not relevant to termination.

4.6.4 Global termination

At the network level, the distributed termination of the algorithm relies on the stability condition of each of its phases. A similar study of distributed termination on a ring topology is presented in [49]. That study shows that a stability condition of each local process is the only condition required for distributed termination. Global termination can be decomposed into a collection of distributed local termination conditions [31]. However, repeated patterns of stable events endlessly "triggering" each other (oscillation) violates global termination. This condition can only occur at the application level, i.e. if a particular source tries continuously to set up a path to a destination that is not currently reachable. The routing table can use the history of previous operations to stop further initiation of failing activities.
4.6. Characteristics of the algorithm

4.6.5 Deadlock

Dijkstra proposed that deadlock occurs only when the following four conditions are all satisfied [50]:

1. Processes can only acquire part of their resources;
2. Processes do not relinquish acquired resources after they requested them, until they have completed their computations;
3. Processes cannot take resources away from other processes; and
4. A circular chain of requests for resources is composed. Each process in the chain requests two or more resources, and at least one of these is also requested by the next process in the chain.

Eliminating at least one of the four conditions (usually 1 and 4) prevents deadlock [22]. Using a graphical representation of a network, an acyclic channel dependency graph is the necessary and sufficient condition for deadlock freedom in non-adaptive routing [34]. Layered meshes with directed planes such as the four-layer MP1 network cannot develop dependency cycles, and are proven deadlock-free [9].

Channel dependencies cycles may occur when packets indefinitely wait for resources. For example, a four processes attempting to simultaneously send messages form a typical cycle. If the routing algorithm does not permit waiting, then dependency cycles cannot develop.

Deadlock can be resolved on a cyclic graph by using virtual channels (extra buffers) ordered in a manner that eliminates cycles [18]. The turn model forbids some message from changing direction as a method to prevent cyclic dependencies [51].

Prohibiting the waiting of requests is essential feature of the exploration. This implies faster route selection, by:

1. It searches up to $k$ directions concurrently; and
2. If some possible paths (or all paths in the extreme) are skipped, the penalty is only failing the search.
4.6. Characteristics of the algorithm

Other types of messages (such as \( m_{\text{ack}} \) and \( m_{\text{cl}} \)) will have to wait. The remaining exceptional case is a cycle formed by such messages (e.g. a cycle of \( m_{\text{ack}} \)). During exploration, busy links are skipped, resulting in the skipping of some possible paths. The exploration is therefore not a complete one. Recall that the algorithm control functions and data routing are all implemented within a bounded region \( R_{a,d} \) and are directed along fixed directions.

The traffic is divided into two types; data and control. Deadlock is addressed within the two contexts:

- **Data traffic**: A large data volume can be split into smaller units that fit into data packets. These data packets are then routed along the assigned path. Competition over outputs is possible. An additional unit that collects data packets from inputs of the node, and then forwards them to the relevant output, simplifies the internal complexity of the router. The servicing of inputs should ensure fairness (i.e. a suitable implementation of internal choice in the routing machine is made). A simplified structure of one router for data traffic and another for control traffic is not an effective design. Hence some level of complexity is unavoidable.

Assuming a layered mesh topology, deadlock in routing data traffic is not possible in the current model and algorithm. The selection of messages for multiplexing onto the output link requires parallel access to the routing tables of other layers. If the bandwidth allocation scheme is used and the data rate of each source is kept below the specified level, congestion is minimised.

- **Control traffic**: The algorithm is designed to function on phases that are performed in steps. Each phase proceeds within the bounded region \( R_{a,d} \) along fixed directions (see Figure 4.6). The search parameters (\( R_{a,d} \) and \( \Delta \)) are determined in advance before any phase starts. No cycles of dependencies can be formed.

**Lemma 4**: The algorithm is deadlock free.

**Proof**:

1. The algorithm implements non-adaptive messaging. The search for a path is performed along directions that are determined according to the distance \( \Delta \). Along any \( j^{th} \) axis
4.6. Characteristics of the algorithm

(where $0 \leq j \leq k + 1$), at most one direction for the search is permitted. In a mesh, bidirectional paths along each axis are essential to complete a cycle.

2. The acknowledgment message also follows the same pattern, but as a single message along a certain path (in the opposite direction to the request), it has no cycles.

3. Data traffic also follows the specified path resulting from the successful set-up of a path, and that path has no cycles.

4. Indefinite waiting is not allowed at any node, hence cyclic dependencies cannot develop. Apart from competition over bandwidth, the traffic of different virtual paths (data and control) is independent.

From above, the algorithm is deadlock free.

4.6.6 Performance

There are two types of delay: i) the delays of path set-up $t_s$ and closure $t_c$, and ii) the data transfer delay $t_d$.

- **Path set-up latency:** Path functions are performed on a link-by-link basis. Thus the linear distance $d$ (the number of steps or links) from $n_a$ to $n_d$ is the sum of the distance along the $k$ dimensions, that is:

Assume that:

- $t_x$, $t_a$, $t_r$, and $t_c$ are the corresponding latencies of exploration, rejection, acknowledgment, and closure functions respectively;
- $t_1x$, $t_1a$, $t_1r$, and $t_1c$ are the corresponding single step exploration, acknowledgment, rejection, and closure latencies of the above functions respectively; and
- $t_{pr}$ and $t_{rp}$ are the latencies of a path function injection and consumption of messages. These latencies specify latencies on pe-rm and rm-pe links respectively.

Then, worst case latencies are:

- The time $t_s$ for a successful path set-up is: $t_s = (t_1x + t_1a) \times d + t_{pr} + t_{rp}$
4.6. Characteristics of the algorithm

- The time $t_s$ for a failing search path set-up is: $t_s = (t_1 + t_r) \times d + t_p + t_r$. This assumes that rejection is generated unluckily at the furthest $pe (pq)$; and

- The time $t_c$ for path closure simply is: $t_c = t_1_c \times d + t_p + t_r$

- **Data transfer latency**: The time taken by a data packet from its departure from $pe_0$ until it reaches $pe_d$ is: $t_d = t_1_d \times d$.

Like any store-and-forward routing algorithm, the latencies are functions of $d$, that is $O(d)$. The time elapsed for sending data ($SendDatda$) is application dependent, hence it is irrelevant.

The analysis above indicates that the locality of communicating tasks is an important factor for latencies. In general, the distribution of an application's tasks (i.e., the placement) onto nodes that are close together is essential to increase performance.

4.6.7 Competition on bandwidth

As the sharing of a link's bandwidth among paths is possible, concurrent searches for different paths across this link are also possible. Their searches proceed with no synchronisation. Competition between searches over the bandwidth of this link may occur. It might cause some (or all) of these searches to fail. Simulation is required to measure the effect of competition on failures of path-finding searches.

As the search for a path progresses, more amounts of bandwidth is reserved on links, causing extensive reservation of bandwidth that may never be utilised. This pattern of unused bandwidth decreases as the network become loaded with paths. Therefore, one may argue that the algorithm may perform better in busy network conditions. Conversely, many paths might fail to be allocated due to a temporary lack of bandwidth.

4.6.8 Development issues

The following is a list of issues for further development of the algorithm:

- The path-finding region could be extended to include links and nodes within a $k$-neighborhood similar to those of fault tolerant algorithms (c.f. [52]).
4.7. Summary

A fully distributed path-finding and routing algorithm has been presented. It uses a bandwidth allocation method that has been developed to minimise congestion around hot spots. The algorithm also allows bandwidth reservation according to a first-served-rest-rejected rule to minimise path set-up delays. The next chapter examines a functional design of a router that implements this algorithm, along with simulation issues in another chapter.
Chapter 5

Design of the Router

5.1 Introduction

This chapter describes the design approach used for building a full functional model of the router. The model simulates a router that incorporates the load-balancing and path-finding algorithm in a 2-D mesh topology (shown in Figure 4.3). Occam processes were used to simulate the functionality of the design. The simulations were carried out using the Kent Retargetable Occam Compiler (KRoC) environment that runs under Linux on a single processor [53]. Extra care was taken to effectively simulate parallel interaction of the nodes within the multiprocessor network (see Sections 5.4.2 and 6.3.2 below).

Implementing the router in hardware becomes a matter of compiling the Occam code into silicon, using one of Occam-to-FPGA tools that are becoming available commercially or academically, such as those described in [54], [55], or [56]. Justification of the design decisions are also presented.

The following sections describe the design requirements, and the functional structure of the router. A summary of some alternative design options is then presented. A model for the processing element is also outlined for completeness.

5.1.1 Design considerations

The following is a list of essential issues for the design of the router:
5.2. Router Structure

As described in the previous chapter (Section 4.3 on page 44), there are three components in each node (see Figure 5.1):

1. The *Router*, which is composed of four identical blocks for routing messages to other nodes as well as to the local processor. These blocks are called "routing layers",

2. The *State Store* and its management via a pair of "routing servers" that uses local messages to interrogate or to update the routing tables. Each of the servers is dedicated to supporting two routing layers, and

3. The *Interface* block that provides for the flow of messages between various blocks within the router. This includes forwarding messages between various layers, and between these layers and the local processor.

The following sections describe these components in more detail.

5.2.1 Routing Layers

Four identical routing layers are used to route messages across to and from other nodes and to/from the local processor \( p_0 \) (via the *interface block*). These four layers are assigned to four virtual layers for routing messages in all possible directions without introducing deadlock. This architecture is re-used from previous designs such as the MP1 routing chip [9].
Due to the exploration, route selection, and route closure in the algorithm, path set-up involves two-directional messaging across the region $R_{b,d}$ (see Section 4.4.3 on page 49):

- **Forward** messages (i.e. $m_{req}$, $m_{cls}$, and $m_{dat}$) down the path or the prospective paths; and
- **Backward** messages (i.e. $m_{req}$ and $m_{ack}$) back along the path.

This two-way messaging implies coupled functionality of pairs of layers. Hence, routing layers need to function in pairs (see Figure 5.2):

- The first pair is composed of layers $X+Y+$ and $X-Y-$, called the **top layer** and the **bottom layer** respectively; and
- The second pair which is composed of layers $X+Y-$ and $X-Y+$, also called the **top layer** and the **bottom layer** respectively.
5.2. Router Structure

Processing of requests at the server should not be interleaved to maintain deadlock freedom [57] (see also the next section). A request made by a routing layer can only be answered after the server has completed its action on a previous request. Consulting the routing server eventually slows down the operation of layers. To improve the speed of interaction each layers and server interaction, each layer maintains a local copy of the routing table that is kept in the routing server. Each layer interrogates the routing server and updates its local copy of the routing tables accordingly. The local copy is used for looking up exit links for data packets instead of consulting the routing table server. This minimizes bottlenecks and allows for the two layers to proceed independently, hence increasing the overall speed of operation. Control packets, however, still require updating the original table and the local copy.

Therefore, the two layers may compete on accessing the server. Eventually one request is held
back until the processing of other request is completely finished.

### 5.2.2 State-Store Servers

As each pair of layers need to access the state-store in an ordered manner, the state-store servers in each layer are implemented as a single server. The server responds to its two clients (the two layers) to guarantee un-interrupted read-modify-write cycles and to prevent deadlock between the server and its clients. This technique is based on the work presented in [57]. The client and the server are denoted by "c" and "s" respectively on the channels in Figure 5.3 and the following figures.

Thus, a pair of "routing table servers" are dedicated for the two layers, one server for each complementary pair of layers. Each server acts as a "state-store server" that provides for a storage of virtual path information in the routing tables, for state change/update, and for the ordering of operations on the routing tables (see Figure 5.3). The last is to provide for the correct order of access/update operations on the routing tables.

The overall access-update operations to the store should maintain consistent information following each operation. Hence the integrity of every path is maintained.

The routing layers consult the routing table servers to select an output link and other information for each message. This activity cannot be done in parallel with other lookup operations to maintain client-server operation and avoid deadlock. A contrasting parallel implementation is shown in [58].
5.2.3 Interface Block

The interface block allows for messages to flow from one layer to its complementary layer, and to/from the local processor, i.e. for message consumption and injection from/to the network respectively. The interface block is shown in figure 5.4, and is composed of similar smaller blocks. These smaller blocks accept messages and then forward them to an output. Hence, these small blocks are store-and-forward devices. These blocks are:

- Four $1 \times 2 S\text{-}Dupl$ (i.e. duplexor) blocks accept tags followed by a message at their inputs, then it forwards the messages to one or both outputs depending on the value of the tag.

![Figure 5.4: Interface Block](image-url)
• Seven $2 \times 2$ Mux (i.e. multiplexer) blocks are basic multiplexers that accept messages from either input, then forward them to the output; and

• Four BB blocks are "bubble-buffers" (i.e. two-stage buffers) that prevent deadlock between layers. Each BB only accepts a message if the two buffers are empty at the same time. Hence a complete cycle of full buffers is prevented, avoiding deadlock. This technique is based on the bubble router presented in [46].

### 5.3 Processing Element

A functional model of the processing element (PE) is depicted in Figure 5.5. The model is not part of the algorithm design, and is only used for the purpose of the simulation runs.

The PE accepts messages from the router and forwards them to the user's screen and/or to the file system. It also accepts commands (in the form of messages) from the user keyboard and/or from the file system. It is assumed that these commands do not contradict with each other to allow the user to make some useful use of the simulator. Messages that are sent from the router to the file system are optionally stored in a set of log files, one file for each router, for later analysis.

The PE includes the following components (see Figure 5.5):

• The Up-Buffer (UB) allows the consumption of messages. According to the mode of operation in place, it accepts messages from the router then forward them to the user's screen and/or the file system. The mode of operation is chosen by sending a message to the UB;

• The Down-Buffer (DB) allows the injection of messages into the network via its corresponding router. The DB accepts messages from the user, the transmitter, and/or the file system;

• The Injection-Buffer (IB) allows injection of packets only when the router is ready; and

• The Overwriting-Buffer (OWB) accepts a single-byte message and overwrites the stored value. This value is used as an indicator to the message that should be reported to the logging file system. All other messages are simply discarded.
5.3. Processing Element

Figure 5.5: Functional diagram of the processing element
Client-server operation is maintained between these blocks as indicated in the Figure 5.5.

5.4 Correctness of the design - the large picture

5.4.1 Deadlock

Figure 5.6 shows the structure of the four routing layers. Extra care was taken on the design of the interaction between these processes according to the client-server method in [57].

As backward messages are required to update the local copy of the routing table in the comple-
5.4. Correctness of the design - the large picture

In the context of the design, each layer needs to pass messages to its complementary layer (see also update messages on page 77). A two-stage buffer is used to guarantee deadlock freedom. The two-stage buffer keeps a buffer always empty similar to the bubble principle used in [46]. The consequence is that a message may be held behind an empty buffer. This eventually slows down message forwarding.

A full dependency cycle between any two nodes is hence avoided. The two nodes could be adjacent or far apart. A simple cycle is depicted in dashed arrows in Figure 5.6.

5.4.2 Fairness

A routing layer may accept messages from either input; along the x-axis or along the y-axis. Fair selection between input messages is achieved at the layers to allow fairness among inputs. An arbiter is used to control the admission of packets at layers in a first-in-first-out (FIFO) fashion (see Figure 5.7). However, messages arriving at the same time slot (i.e. with an equal time stamp) are treated in an alternating priority fashion to ensure fairness. The Occam PRI ALT mechanism is used for this purpose. Messages sent to the routing-table server (and the server's responses) do not require time-stamps. A single time-stamp diagram that illustrates the client-server interaction method (as in [57]) is shown later in section 6.3.2 (see page 86).

![Figure 5.7: Time-stamping of messages at the routing layer](image)

Figure 5.7: Time-stamping of messages at the routing layer
5.4.3 Update messages

Update messages are internal messages that are exchanged between complementary layers to maintain consistent contents in the routing tables. Update messages are routed from each layer to its complementary layer via the interface block (see page 72). Figure 5.8 below shows pairs of complementary layers with a typical sequence of events (i.e. messages) that can cause flow of update messages.

![Diagram of update messages](image)

Figure 5.8: Update messages following backward messages

In (a) of the Figure 5.8, a Reject message arrives at the layer. After consulting the routing server, that layer produces an Update message that is sent to its complementary layer via the interface block. The Update message contains updated pointers.

In (b), the Reject message is terminated at the node after initiating the Update message because of the presence of another "live" sub-path.

In (c), an Acknowledgment message progresses up the path after initiating the Update message. A similar pattern also occurs in (d) except that a "live" sub-path is found in the Y+ direction. This sub-path is then closed via the complementary layer (through a Close message).
5.5 Summary

A full description of a possible implementation has been introduced through the presentation of functional models. Implementing these functional blocks is possible in different ways. One approach is to compile the Occam code into FPGA as suggested earlier.
Chapter 6

Simulations and Results

6.1 Introduction

This chapter is include description of the simulator used and associated tools, latency and performance results, and comparison with competitive routing scheme.

Occam-2 is used for the simulations through usage of software (e.g. KRoC [53]). The simulation models are coded using Occam processes and have been tested using KRoC under the Linux environment. Special care was given to the timing and scheduling to effectively simulate parallel processes in single processor environment.

Several simulation runs were carried out to show the advantages of the path finding algorithm. Another simulation run was also done using minimal adaptive routing to show comparisons with a good competitor.

The following chapter is the last chapter and lists the conclusions of this work.

6.2 The scope of simulations

6.2.1 Latency measurements

The router model is a double buffered device, that is there are input buffers and output buffers attached to each port (i.e. to each link). This choice provides the extra buffering needed to
simplify the simulator implementation.

The following is a list of types of latency that could be used in simulation measurements:

- **Injection** latency can be seen at the injection buffers in the routers that are associated with the processors. This latency is the time taken from the moment the local processor commits the message for injection until it is accepted by the router;

- **Waiting** latency is incurred by messages inside the routers. This latency is mainly caused by messages waiting in the input buffers that are associated with neighbouring router nodes. Thus this is the latency incurred by messages en-route while route selection is being made. This latency also includes the time taken to access (search and/or update) the local tables to look up the output link (or links) to be used;

- **Blocking** latency occurs at nodes which are waiting for a resource or a link to be freed. It is the time taken by a message while stored at an output buffer in the router because of blocking at the next node. This latency is different from the waiting delay in that a message is not allowed to move forward because the input buffer at the next node is full. The waiting delay is incurred by a message at a router while processing other requests and after being accepted into the input buffer;

- **Delivery** latency is the time taken by a message to be consumed by the local processor. This latency is similar to the injection latency shown above. This time is application-dependent and is caused by local processors executing for extended periods of time; and

- **Forwarding** (or routing) latency is the time taken by the router to examine a message, select a course of action, and send a response to the appropriate output buffer.

The internal latencies of blocks within nodes are not simulated nor measured. For example, accessing the routing tables is assumed to take one clock cycle in a hardware implementation. In reality, it may require a number of cycles. Hence, lower-level simulations for such delays have not been performed.

### 6.2.2 Targeted simulations

A typical simulation follows one of the following methods:
6.2. The scope of simulations

- *Spatial* simulations look at the effects and interaction between virtual paths (e.g., competition between paths);

- *Temporal* simulations examine latencies, as well as network stability when paths come to life or are destroyed;

- *Topological* simulations look at the use of virtual paths in a physical layer. A “virtual” topology then could be dynamically reconfigured out of (or mapped onto) the fixed physical topology. In theory, these measurements could also be extended to included dynamic selection of alternative connections (i.e., routes);

- *Routing* simulations focus on the efficiency of creating virtual paths and removing them over time under various traffic patterns. Comparative measurements would also evaluate the algorithm against others.

A combination of more than one method can be helpful to evaluate multiple-parameter measurements. A typical example is the effect of latency variations in relation to competition of paths over bandwidth.

6.2.3 Traffic models and patterns

Traffic models describe the temporal flow of messages within the network. The most common models are [59]:

1. *Uniform traffic*: the rate of transmission of messages fluctuates randomly. This is the most common model in computer-based simulations due to: (i) it simplifies the analysis of the results, and (ii) traffic becomes less bursty at intermediate nodes due to their even message distribution.

2. *Bursty traffic* represents situations where nodes become active for short periods of time and remain almost inactive in between.

3. *Steady traffic* is a simple fixed rate message flow.
Traffic patterns describe the spacial flow of traffic within the network, i.e. where the messages flow within the network. The following is a list of common traffic patterns. More variants of these patterns can be found in [60], [46], and [61]:

1. *Random uniform distribution (RUT)*: all destinations including the source are equally likely.

2. *Hot-spot*: refers to situation where many nodes prefer to communicate with one node (called hot-spot). Another variant is the 4X Hot-Spots where ten randomly selected nodes are distinguished. Destinations are chosen randomly such that the distinguished nodes are four times more likely to be chosen than the undistinguished nodes.

3. *Correlated*: refers to the situation where traffic flow is grouped, with each group belonging to a different connection. This mostly concerns traffic flow for different connections. This is similar to hot spots in that certain areas of the network become concurrently loaded, but messages passing through a layer do not share the same end-node.

4. *Isolated*: refers to situations where traffic flow is largely scattered (i.e. in contrast with correlated).

5. *Complement*: is a permutation where each node sends messages to the node of complementary indexes (i.e. by complementing bits of the index)

6. *Transpose*: is a permutation where each node sends messages to its opposite node in respect to mid-range-indexed node or "central node".

7. *Bit Reversal*: is a permutation where each node sends to a node whose index contains bits in the reverse order to the transmitting node.

8. *Shuffle*: is a permutation where each node sends messages to a node with a shuffled index.

### 6.3 The Simulator

A multiprocessor with a 2-D mesh topology is size $8 \times 8$ nodes has been chosen for use in the simulations. The chosen size is sufficient to represent larger sizes without loss of generality.
6.3. The Simulator

Similar studies also used the same mesh size (e.g. [45]). A larger size would only consume resources and extend simulation times.

The Occam code simulates the routing algorithm on a 2-D mesh topology using asynchronous exchange of messages in Occam [62], [63], [64], and [65]. The code size was about 3,500 lines.

A small packet size is used throughout all the simulations. The packet size is fixed at 12-bytes. Using larger packets does not affect the latency simulations as the whole packet is transferred at once in store-and-forward routers. The transmission time would eventually increase in routing larger packets.

6.3.1 Limited-rate source

A source of data packets is attached to each processor. It is simulated using a generator process that has an upper limit on its output rate (depicted as $T_x$ in Figure 6.1).

The internal structure of the limited-rate source (i.e. $T_x$) is shown in Figure 6.2. The simulator uses a single source at each node that is capable of generating data packets for a single path only. Multiple sources can be adapted to expand the simulator capabilities. The $T_x$ model consists of three blocks: the Responder (RX), the Data Generator (DG), and the Rate Generator (RG). These blocks work as follows:

- The Responder (RX) provides a response to some of the path operation requests as follows:
  - it responds to a Request message by sending an Acknowledge message to the network via the data generator (DG) block shown below. This response is triggered by the arrival of the Request at the destination node.
  - it responds to an Acknowledge message by sending a data packet template to the data generator. This response is triggered upon the arrival of the Acknowledge at the source node, and
  - it responds to an Acknowledge message by sending the value of the requested bandwidth $B_r$ to the rate generator. The last two items are performed in parallel.
Figure 6.1: Data packets source with bandwidth limit
6.3. The Simulator

Figure 6.2: A model for the generator of data packets

- The **Data Generator** (DG) works as follows:
  
  - it forwards the *Acknowledge* message from the RX to the network,
  
  - it accepts a command for the rate generator in the form of a signal that increments the local time value, and as a trigger to send the data packets (see next item).

  - it uses the data packet templates to compose data packets. A template is a complete data packet (i.e. includes the look-ahead information) its time stamp is set to zero. Each data packet is updated with a "packet sequence number" to make it identifiable for tracing the packet through the network and to capture its progressive timing. It also sends data packets to the network only if the value of the command was *True* (T) and correctly generates their sequence number within each stream.

    The local time, identical to the global network time, is added to the packet (as a time stamp) to allow timing analysis later.

- The **Rate Generator** (RG) uses a random number generator to send signals to DG. Each time a clock tick arrives, it sends either a *True* (T) or a *False* (F) command to the DG. It produces a random number of uniform distribution. It then compares the number with
the bandwidth. It sends the True command to the DG.

### 6.3.2 The timing of events

Timing of events under a single-processor *KRoC* environment requires time-stamping of messages. A time-stamp block is used for each input in the routing layers. Figure 6.3 shows a diagram for a time-stamp block.

![Diagram of time-stamp block](image)

**Figure 6.3:** Fairness between inputs of a node

Thus a global clock is distributed through the mesh of nodes and processes in the *KRoC* simulation to measure time and synchronise certain events. However, this global clock does not violate the asynchronous nature of the mesh operation. It is purely for time-stamping purposes in the time-sliced *KRoC* environment. The client-server style is again re-used as in [57].

It is important to note that the time-stamp block forces all routers to advance in lock-step fashion, i.e. they advance at the same time. This common clock acts as a “barrier” that allows each router to process packets in equal time intervals, which emulates a clocked hardware implementation. Thus the chain of routers along an axis acts like a pipeline for advancing packets. Events between clock ticks are not time-controlled. These events may follow any possible sequence permitted by the Occam scheduler.
Also, the local time value at each router advances at the same rate allowing common and correct timing of events across the whole network.

Accessing the routing table is not included in the simulated time. This assumption does not affect the the overall measurements as such. If hashing were to be used then a fixed delay would normally be added to packet processing at each node. If another table search method were to be used, then a variable delay time would be added instead. Considering that the size of the routing tables is small (e.g. 10 entries is assumed in Section 4.4.4), then this delay is not substantial. A typical figure might be 1-2 clock cycles per each read or write operation. A read/write cycle would consume double the time (i.e. 2-4 clock cycles). The first cycles of each access operation could be incorporated into the other routing operations.

### 6.3.3 Message logging facility

The simulator is designed with a facility at each node for message reporting. A byte-wide bit pattern is used to turn on or off the reporting of different types of messages. Copies of these messages are sent to the file system via the local processor to be logged in files for later analysis.

For example, by setting a data masks bit to one and the rest of bits to zero, it is possible for every node to report data messages only. Then each node sends a copy of every data message it has accepted at every input. These copies contain the original message with the following added information:

- A record of the time of arrival to allow timing analysis. This record is the time stamp shown in Section 6.3.2,
- A flag that is set to indicate that this message is a copy. This is necessary to distinguish copies from original messages that are destined to the local processor, and
- An indication to the input link that the message arrived along.

The first feature above is used in the performance measurements in Section 6.6.
6.4 Results of simulations

As indicated in Section 6.3.3 above, the simulator logging facility was used to measure the performance of the routing algorithm in various circumstances. As each message includes a sequence number, it is possible to trace messages within the network one by one.

6.4.1 Measurement method

The following method is used to extract the results from individual log files:

- Merge all log files into one large log file. As each message includes a record of where it was recorded, i.e. a reference to the node;
- Sort the contents of the total log file according to several keys, such as type of message, the node's reference, and time stamp;
- Calculate time differences between messages by comparing time-stamps in messages. Selection of messages at successive nodes reported within a specific time slot. The time slot is selected according to time-stamp values that fall between start and end time-limits; and
- Averaging of several results is then used when needed, particularly in obtaining performance figures.

Overall performance measurements cover the whole mesh, while latency is a local feature of few nodes, a path or more, or a sub-mesh. The following sections include latency and performance results.

6.5 Latency results

A series of experiments were carried out to measure latencies in control packets and data packets. These experiments were carried out in an empty mesh; except for the single path in question, there were no other packets flowing within the network. The following sections detail the results of these measurements.
6.5. Latency results

To obtain the average latency per node, messages were copied to the logging files according to the nodes that reported them. The difference between time stamps for a particular message at successive nodes represents the latency for each message at each node.

6.5.1 Path Request latency

Path request latency is shown in Figure 6.4 as a 3-D graph. Assuming a source at node (0,0) in the mesh, then the latency associated with each path request to every destination node is plotted. The latency incurred by messages from the source to itself (i.e. distance = 0) is not a valid experiment and is assumed to yield zero latency.¹

The path request latency is found to be 12 clock cycles per node. This figure represents the router processing cycle. It also equal to the number of internal buffers is used along the message flow within the router from an input to an output. This figure is related to the actual implementation of the simulator.

¹Zero distance is not a valid, hence latency is assumed to be zero throughout the following sections.
6.5. Latency results

This latency is a linear function of the distance, a typical feature of serial store-and-forward operation. In the case of 2-D mesh, the Manhattan distances are used (i.e. the distance = \(d_x + d_y\)). The first request message to arrive at the destination initiates the acknowledgment phase. Hence, it also determines the total latency. Further requests that flow within the region \(R_{s, d}\) that is bounded by both the source and the destination nodes will have no effect on this latency figure as it will be cancelled and all associated sub-paths will be closed.

If the time taken to access the routing table is to be considered, then a higher latency would be measured. The latency figure will be increased by 2-4 cycles per node. However, the latency will remain a linear function of the combined distance after that shift.

The ripples appear on parts of the graph (e.g. at \(x=4, y=2\)) are caused by possible errors in timing measurements due to missed clock cycles. The timing block at the inputs of each router works as a clocked pipeline. In the simulation of parallel processes on a uni-processor system, the scheduler may allow execution of process at a stage \(i\) before its predecessor process \(i - 1\). The stage \(i\) would have wasted that clock cycle. Indeed, a large number of runs of the same experiment would eventually give a clear 3-D surface with no ripples.

6.5.2 Path Acknowledgment latency

A linear function of distance is also measured in the path acknowledgment latency and measurements is also a linear function of distance as shown in Figure 6.5.

Path acknowledgment latencies include two components that are due to the two types of messages used to set up the paths:

- **Acknowledgment** messages from one node to the previous one in the path, and
- **Update** messages within the nodes, from one layer to its complementary layer.

The routing tables are accessed once during the processing of acknowledgment messages. Update messages simply send a copy of the path pointers and informations about a measure of the available bandwidth to the complementary layer. These two messages are composed in sequence. That is, only after the acknowledgment message is processed, the update message is composed and sent. Hence, higher latencies are incurred during the acknowledgment phase.
6.5. Latency results

Figure 6.5: Latencies of path acknowledgment message in unloaded 8x8 mesh

Again the ripples that appear in this graph are measurement errors as indicated in previous section. The path acknowledgment latency is measured at 12 clock cycles per node. These errors are symmetrical with errors in previous section indicate an element of error is carried forward from previous measurements of path request latency. Averaging of results of a large repetition would eventually smooth these errors down.

6.5.3 Path Set-Up latency

By combining the two latencies together, the path request latency and the acknowledgment latency, the graph in Figure 6.6 is obtained. As each of the two latencies is linear, the combined latency also remains linear at 24 clock cycles per node.

The calculations here were mathematically obtained from the previous two measurements of latencies by simple scalar addition. The increase of latency due to accessing routing tables at nodes will have to be doubled (i.e. 4-8 clock cycles per node), since there is one increase during the path request phase, and another during the acknowledgment phase.
6.5. Latency results

6.5.4 Path Closure latency

Similar calculations for path closure operations were also carried out in similar manner. The resulting plot is shown in Figure 6.7 and also indicates a latency of 12 clock cycles per node. The closure message also updates the routing tables by removing the deleted entries that belong to the closed path. These entries are assigned as empty. The available bandwidth is also adjusted accordingly.

6.5.5 Data Forwarding latency

This is shown in Figure 6.8. It again follows a similar pattern to close messages. Data packets also access the routing tables to utilise the look-ahead feature. The latency was again measured at 12 clock cycles per node.
6.5. Latency results

Figure 6.7: Latencies of path closure message in 8x8 unloaded mesh

Figure 6.8: Latencies of data packets in an 8x8 mesh with single path
6.6. Performance results

6.5.6 Overall control latency

This is shown in Figure 6.9, and it was calculated adding Set-up, Acknowledgment, and Closure latencies. Also plotted on the same graph are:

- Processor response latency to request message at the destination node. The latency represents the time from accepting a request message to the time an acknowledgment message arrives at network (i.e. at the associated router). This latency is measured as a fixed delay of 12 clock cycles.

- Processor response latency to acknowledgment message at the source node. This is a measure of how quickly the processor starts injecting data packets into the network. It is the time from receiving the acknowledgment message to the time it injects the first data packet into the network (i.e. at the associated router). Based on the simulator model and the processor model shown above in Section 6.3.1, this latency was measured at 23 clock cycles.

The last two latencies are also included in the graph, but are not added to the overall latency. It is noticeable that the Acknowledgment latency shows a small rise at distances 4 and above. This shift occurs when the acknowledgment message involves closure of any redundant sub-paths that remain “live”.

6.6 Performance results

Three traffic patterns were used to evaluate the performance of the PFA algorithm. Running the simulations under these patterns provides sufficient evidence of the algorithm functionality and the router performance.

Throughput measurements require the measurement of the amounts of injected traffic and of the accepted traffic. The simulator uses the synchronised messaging of Occam. Once a process is committed to a channel communication, it can only proceed after the communication is completed. In other words, once a message is assigned to an output channel, the message must be transferred. The same discussion applies to input channels.
Therefore, the simulator suspends execution (i.e. blocks) until any message communication has succeeded. The only solution to avoid lockups is to provide an extra flow-control channel in the opposite direction. This control channel guards the original channel to grant or deny message transmission. Committal on this extra channel is guaranteed to succeed all the time. This adds extra complexity to the simulator. Furthermore, all injected messages were accepted by the network and were communicated. The applied load is the same as the accepted load, thus throughput measurements were not implemented as in [46].

The message propagation delay per node is measured after the set-up of paths is completed. Hence, the latency results shown below do not include set-up delays.

1. Isolated traffic

Isolated traffic (IT) is used to simulate several paths that are largely independent of each other. However, nodes can serve as a source on a path, and a destination on another. The results of the isolated traffic measurements are shown in Figure 6.10.

From Figure 6.10, the increase of the requested (and achieved) bandwidth from 10% to
6.6. Performance results

Figure 6.10: PFA latency in 8x8 mesh with isolated traffic

100% leads to a near-linear increase of latency from 17 to 22 clock cycles. The rise of latency implies that busy routers held messages in input buffers longer at heavier loads. This is realistic feature because un-clocked processes between time-stamping blocks along the pipeline became busier (see section 6.3.2). The results are also close to the single path experiment. They give statistical indication of realistic conditions. The single path case in otherwise an empty network is a rather ideal situation.

2. Random uniform traffic

A group of paths with randomly chosen end-nodes were used to simulate PFA latency with Random uniform traffic (RUT). The chosen paths may share end points as well as intermediate links. Shared end-nodes are a source for one path and a destination for another. The bandwidth of all paths are set to vary from 10% to 100%, i.e. the generator sends packets at random intervals at and average rate of 10 to 100 times in each 100 clock cycles. Obviously this is to demonstrate the full range.

In practice, reaching the full bandwidth leaves no room for control traffic. However, measurements started after all paths are set-up and transmission of packets started. The
results are shown in Figure 6.11.

![Figure 6.11: Latencies of in PFA with random uniform traffic](image)

The chosen paths were established at a bandwidth of 10%. The bandwidth was then increased on all the paths until it reached the full bandwidth of the involved links.

From Figure 6.11, an increase of requested bandwidth on random uniform traffic selection of individual paths up to the full bandwidth leads to almost similar results of isolated traffic. This provides evidence that virtual paths are independent from each other.

3. Correlated traffic (CT)

To highlight the bandwidth management in PFA, hot-spots were created in the mesh. These hot spots were then loaded with correlated traffic (CT). A similar traffic pattern was again used later on a similar mesh but with adaptive routing for comparisons of latencies (see Section 6.7).

A sample of correlated traffic pattern is shown in Figure 6.12. The sub mesh was loaded with three traffic patterns belonging to three paths. The three paths were created before propagation measurements were recorded. Then all of the paths were loaded with large number of packets to allow the measurement of latencies in steady-state traffic flows. The
three paths were created using the following sequence to force traffic flow into shared areas:

(a) Path P1 from node A to node H at the maximum bandwidth (i.e. bw=100%);
(b) Path P2 from node A to node E (via nodes B, C, and D) at half of the maximum bandwidth (i.e. bw=50%); then
(c) Path P3 from node F to node G (via nodes B, C, and D) at half of the maximum bandwidth (i.e. bw = 50%).

Clearly the choice of end-nodes of P3 forces the selected route through the nodes B, C, and D. Also P2 and P3 share the bandwidth equally, i.e. the links B-C, and C-D are equally shared between P2 and P3. Without forcing the above configuration, PFA would possibly choose different routes. PFA does not allow sharing of links with a total
requested bandwidth exceeding 100%.

After these three paths were set-up, then large number of packets were sent through each path. The latency is measured and averaged for among chunks of packets. The chunk size was determined by limitations in software tool that was used in the calculations.

![Image: FPA latency vs bandwidth in 8x8 mesh - correlated traffic](image)

Figure 6.13: Node propagation delays when using PFA with correlated traffic

It can be seen in Figure 6.13, that correlation between different paths sharing some links does not produce dramatic changes. The traffic belonging to the two paths is independent. Paths interact at set-up time because of competition over bandwidth leading to failure to establish some paths.

6.7 Competitive comparisons

*Minimal adaptive routing* (MAR) was chosen and simulated for comparisons and evaluations of the path-finding algorithm (PFA). The choice of MAR is made because of:

- it offers high performance,
6.7. Competitive comparisons

- it uses the store-and-forward operation, i.e. similar to the PFA, and
- its route selection is dynamic, i.e. according to traffic conditions.

When there is contention between packets over output channels, blocking is inevitable in store-and-forward (SAF) routers. Thus buffering comes into play to hold blocked messages. Each output channel can serve one input channel in any time slot (clock cycle), hence buffering becomes inevitable - it allows blocked messages to wait until their output channel is available. Possible configurations to accommodate this are: output buffering, input buffering, combined input-output buffering. More variants of these configurations are also shown in [59] and [46].

The SAF router used here is based on the Sequential Input Crossbar (SIC) model that is used in the Cray T3E [66]. Another "bubble" version called Adaptive bubble SIC was also simulated in [46]. The SIC-based model is shown in Figure 6.14 below. It consists of a cross bar and a set of input buffers. The cross bar is controlled by an internal arbiter.

![SIC block diagram](image)

(a) SIC block

![Modified SIC layer](image)

(b) Modified SIC layer

Figure 6.14: SIC-based model used in simulations

The SIC version shown above works under the control of the arbiter. The arbiter executes an indefinite loop in round-robin fashion with four steps [46]:

1. Select and active input;
2. Check status of the requested outputs;
3. Select one output of those in the previous step; then

4. Activate the crossbar and route the packet across.

This adapted version is a realistic representation of the original version used in [46]. To maintain similarity with the PFA router, the four-layer structure was also maintained, eliminating the need for "bubbles" in the SIC. The SIC model was simulated under similar traffic patterns in Section 6.6 to maintain accuracy. Figure 6.14 shows the performance of two routers: PFA and SIC.

![Figure 6.14: Comparisons of PFA and SIC-based routers](image)

The same network model was used in this experiment, except the PFA router was replaced by the SIC router. The SIC-based network also used the four-layer architecture to maintain consistency between the set of results. The same paths (i.e. combinations of end-nodes) and bandwidth selections were used above in PFA RUT experiments (in Section 2). However, the rate source in the SIC experiment was modified to allow variable message rate with the same paths, i.e. keeping the same combinations of source-destination nodes. This variation of the rate is not valid under the PFA algorithm requirement specifications (see R.5 in Section 4.4.1).

From Figure 6.15, the comparison between PFA and SIC shows:

- SIC performs better than PFA at smaller traffic loads. Using the PFA for routing light loads leads to inefficient utilisation of the network resource due to path management...
overheads (e.g. state store updates, exploration, etc.). As the network is not too busy, this inefficiency is not crucial. However, sudden changes in traffic load may expose the slow responsiveness of the network.

- Latency increases more notably at higher loads (above 50%). The PPA performs at least 30% better at full load. Considering that time set-up is not included, the 30% figure should be scaled down to compensate the initial delay for the path set-up. If the duration of the path life time is far greater than the set-up delay, the this delay is negligible.

- SIC latency seem to approach a stable value at higher loads. This does not appear to follow results shown in the literature (e.g. [46]). This means that the simulator does not force extra traffic into the network more than network can handle. This is due to the committal nature of Occam messaging, which was explained earlier in Section 6.6 above.

6.8 Further development issues

These issues demonstrate additional areas for development. Further interesting issues can be found in [58] and are not included or simulated here.

1. Multiple message lengths
   By adding another class of messages with a larger size, it is possible to divide these messages into smaller chunks (such as a 12-bytes chunk size) and transmit them using adaptive routing. Then the total bandwidth available should be reduced by a pre-calculated amount to provide enough bandwidth to the new class. Routing of the rest of messages is unchanged.

2. Bandwidth negotiation
   It should be possible to dynamically adjust the reserved bandwidth along a certain path according to traffic conditions. An update message would be required to pass through the entire path to announce the change.
This would have the advantage of aggressively using the available bandwidth for short periods. There should be a policy on predicting the changes in bandwidth requirements. At busier times, the reserved bandwidth could be smoothly reduced accordingly.

Negotiation of bandwidth could also be carried out at path set-up. The acknowledgment message may carry back the actual available bandwidth that could be served at that particular moment. It remains up to the source node whether to use the “offered” bandwidth or to close the connection because the requested bandwidth was not available.

3. Algorithm optimisation

The algorithm involves extra overhead due to information exchange for managing virtual paths. In some occasions, it would be more efficient to use “traditional” routing without the need to use virtual paths. For example, a source wishing to send a token, or few packets to a destination could imply use dimension-order routing. A policy would be required to choose when to use virtual path and when not.

A candidate policy is to allow the source to make the choice of using virtual paths or not. This choice would be based on one or more of the following factors:

- The data size: a threshold for size of data would be required to compare against. A policy that adopts this method optimises the network performance under different (fixed or changing) traffic patterns. As an example, one may choose the following policy:
  - Light-load Any amount of data to be transferred that is below a pre-specified level would always be routed through the network using adaptive routing,
  - Heavy-load Any amount of data to be transferred that is above a pre-specified size would always be sent using the PFA algorithm presented earlier, and
  - Medium-load data would be transferred using either of the above two methods. As an example, in heavy traffic conditions, medium-load data could be transferred according to PFA. But the same size of data could also be routed using adaptive routing when not much traffic is flowing through the network.

This method adds extra complexity to the choice of the routing method. The threshold could be selected according to some pre-selected level of traffic in
6.9. Summary

A good simulation of router design that implements path-finding algorithm has been presented and shown to run with no deadlock. The performance analysis shows that the path-finding algorithm and the router structure provide competitive solution to congestion based on bandwidth reservation, virtual paths, and guided routing.

The next chapter lists the conclusions of this research project.
Chapter 7

Conclusions

Conclusions are grouped into three areas:

7.1 The Path-finding Algorithm

1. A routing algorithm has been developed to achieve route discovery and selection in k-D meshes. The combination of bandwidth management and virtual paths in multiprocessor networks has been researched.

2. The combination of route discovery (or sub-mesh exploration) and path selection at the same time has been researched. This technique has been developed and utilised in multiprocessor networks.

3. The path-finding algorithm performs better than adaptive routing in certain conditions such as correlated traffic. The concurrent nature of the exploration does allow fast route discovery. Exploration process also terminates in finite time.

4. Packet ordering is guaranteed in our algorithm due to the serial nature of the transmission on one path. For comparison, re-ordering of packets at their destination would be required in adaptive routing because packets could follow different routes with various delays.
5. The route selection process discovers routes around congested areas in the network. This feature also coincides with fault tolerance in avoiding "un-suitable" areas within the network because these areas are either busy or congested.

7.2 The implementation

1. A complex functional model was created to demonstrate a router with the path-finding algorithm described. A hierarchical structure was created to simulate the desired operation. The functionality of each internal block has been defined and simulated to demonstrate the consistent operation of the model.

7.3 Performance

1. The path-finding algorithm performs well in busy network conditions with random uniform traffic distribution. This is due to limiting the injection of messages into the network to prescribed (and negotiated) levels. In a less busy network, or in low load conditions, the overall utilisation drops to extensive exploration. This drop does not affect performance because the demand on resources is low at light loads.

2. The results also show that routing in path-finding reaches up to 30% percent higher performance than the adaptive routing at network loads above 60%. The two perform equally at the 60% load level. At lighter loads the adaptive routing wins by about 50%. This is particularly true when there is competition between adjacent or overlapped traffic flow patterns.

7.4 The simulations

1. Occam offers a simple and efficient tool for modeling parallel architectures and networks. This tool bridges the high-level specifications and the lower-level implementation. Message passing provides for asynchronous behavioral modeling. A physical implementation of the modelled router could be achieved via compiling the code into silicon by using an Occam-to-FPGA compilers.
7.5 Other issues

2. A new tool for timing concurrent events in parallel system is introduced. This timing approach has not been reported in the literature.

3. Simulation of parallel processes using a single-processor system has been used utilising the timing method above. This technique is combined with Occam's "Fair ALT" construct to achieve a true simulation of parallelism.

7.5 Other issues

1. The simulation tool that was used (i.e., KRoC) provides efficient and sufficient platform for functional modeling for true parallel designs at no cost.

7.6 Final summary

This work presents an algorithm for guided packet routing in multiprocessor networks. The algorithm combines path-finding with virtual paths to achieve bandwidth management.
Bibliography


[65] INMOS. *The transputer applications notebook: system and performance*.