Pegasus Paper 93-5

Virtual lines, a deadlock free and real-time routing mechanism for ATM networks

Gerard J.M. Smit[1]
Paul J.M. Havinga[1]
Walter H. Tibboel[1]

November 1993

Abstract

In this paper[2] we present a routing mechanism and buffer allocation mechanism for an ATM switching fabric. Since the fabric will be used to transfer multimedia traffic it should provide a guaranteed throughput and a bounded latency. We focus on the design of a suitable routing mechanism that is capable to fulfil these requirements and is free of deadlocks. We will describe two basic concepts that can be used to implement deadlock free routing.

Routing of messages is closely related to buffering. We have organized the buffers into parallel fifos, each representing a virtual line. In this way we not only have solved the problem of Head Of Line blocking, but we can also give real-time guarantees. We will show that for local high-speed networks it is more advantageous to have a proper flow control than to have large buffers. Although the virtual line concept can have a low buffer utilization, the transfer efficiency can be higher. The virtual lines concept allows adaptive routing. The total throughput of the network can be improved by using alternative routes. Adaptive routing is attractive in networks where alternative routes are not much longer than the initial route(s). The network of the switching fabric is built up from switching elements interconnected in a Kautz topology.

1 Introduction

In the past decade we have seen a revolution in VLSI technology. The transmission and switching technologies have rapidly advanced to higher speeds; it is now technically feasible to build networks with gigabit throughput. Field Programmable Gate Arrays (FPGA) is a technology that not only allows gate arrays to be reprogrammed an unlimited number of times, but also has on-chip static memory (the X4000 family of Xilinx). Using programmable elements gives us the opportunity to learn from experiences on prototypes and to adapt the architecture.

We are currently building a prototype network using these off-the-shelf technologies. Our goal is to build a local area network that supports multimedia applications. These applications require not just high transmission speeds, but also small end-to-end latency with little variation, a guaranteed throughput, graceful degradation under heavy workloads, and performance that is both fair and predictable. Low-latency services are necessary for voice and video transfer, process control, remote sensing etc. The data for these services is usually worthless if it does not arrive in time. A video sequence, for instance, must be retrieved at a high and constant rate; frames retrieved too late are no longer useful and can be ignored. An additional problem is the synchronization of different media. These applications require real-time *and* reliable communications where certain strict deadlines must be met.

The bandwidth of many existing networks is by far not enough for distributed multimedia applications. Their throughput and latency is becoming a bottleneck in demanding real-time applications.

---

1. University of Twente, Faculty of Computer Science P.O. Box 217, 7500 AE Enschede, the Netherlands
2. This paper was presented at the Eighth International Symposium on Computer and Information Sciences, November 3-5, 1993, Istanbul, Turkey.

One reason is that these networks use a single shared path for their communication. Point-to-point networks are usually organized as star shaped networks, in which all stations are connected by dedicated links to a central switching fabric. A connection between two stations is established through the switching fabric. The main advantages of point-to-point networks are that they offer an aggregate network bandwidth that can be much larger than the throughput of a single link, and have a high availability by allowing multiple paths. The interfaces in the stations can be simple and low cost (most of the complexities are contained within the switching fabric) and the design is relatively independent of the technology of the physical layer of the links. If there are multiple paths in the switching fabric, the connections will experience a higher availability.

In this paper we present a network that can be used to transfer multimedia traffic with the above requirements.

2 Deadlock free routing mechanisms

Asynchronous Transfer Mode (ATM) has emerged as a leading technique for high-speed packet switching. In packet switched networks, packets are exchanged over communication channels between nodes. In order to provide an arbitrary and fully dynamic connectivity in a static network of nodes, routing mechanisms must be implemented, which provide the propagation of data from node to node. Routing information is contained in each packet, such that it can be routed from source to destination.

A method that is very suitable for ATM packet switching is *wormhole routing*. With this method, all packets of a message are routed on the same path from source to destination. During connection setup a path from source to destination is claimed for that message. Wormhole routing prevents the ordering of packets to be altered.

Depending on the network topology it is applied to, routing may be subject to deadlock problems. *Deadlocks* occur when we can identify a cyclic buffer waiting graph, for example when none of the packets in the network can advance towards their location, because the necessary queue for that packet on the next node is already full.

In this section we will describe two basic concepts for deadlock free routing. We will show that a virtual line concept suits very well with our demands of a multimedia network, and that it has a low complexity. The main design issue is how to organize the buffers.

2.1 Main concepts

There exists a number of deadlock free routing algorithms. Most algorithms are based on the use of static buffer graphs whose structure ensures that deadlock situations cannot occur. Taking cycle-free buffer graphs will be sufficient to prevent the occurrence of any circularity in resource (i.e. buffer) requests.

A basic concept in deadlock prevention is to guarantee a path in the network. If every message injected in the network follows one of its guaranteed paths, deadlock can not occur since there are no directed loops in the buffer graph. We will describe two main concepts.

- *Class climbing* is a general concept which is often used. When a process (e.g. a task that sends a message from source to destination) has rank *r*, that is, it has already obtained resources of class *r*, it may only request resources of classes higher than *r*. Figure 1 gives an example. There must be sufficient classes of resources (buffers) to satisfy the rank in the network. The number of buffers required per node depends on the maximum number of hops possible in the network, which on its turn depends of the network topology and on the routing properties. Several schemes have been proposed in the literature for constructing buffer graphs and to assign ranks: virtual graphs, valley counting, hops-so-far (hop counting), etc. These approaches can restrict the routing of packets: it reduces the number of paths that a packet may take [Adamo 91]. The length of the path can be longer than the shortest path, such a route implies more latency.
- Another approach is to use a *connection oriented network service*. The basic concept is to guarantee a

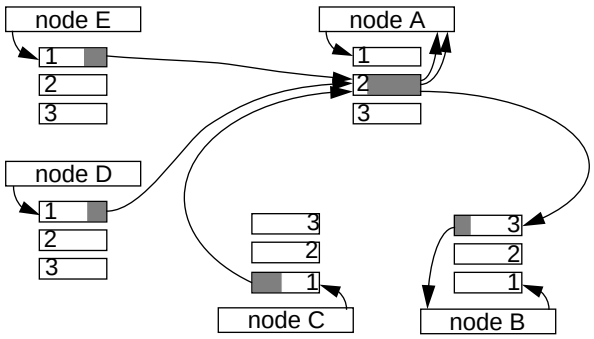

Figure 1: Class climbing concept

path in the network. If every message injected in the network follows one of its guaranteed paths, deadlock can not occur since there are no directed loops in the buffer graph. The method of nosy worms [Whobrey 88] can be used for this approach. During the connection phase, at every node needed to form the virtual communication channel, enough buffers are reserved to support the connection. These buffers will only be released when the connection is terminated. Messages start off by being buffered at their sending node. Each message attempts to establish a connection with the destination node by forming an unbroken path across intermittent nodes. If a block or a failure is encountered along the route the message gives up by recoiling back to the sender, thus avoiding deadlock. If there are alternative routes other routes can be tried. Once a connection is made the entire message is transferred from source to destination, where it is buffered again before it is off-loaded to the receiving system. In fact this is a kind of circuit switching. The main disadvantage of this approach is that is has a low buffer utilization. Circuit switching is often used in combination with virtual channels.

2.2 Virtual lines

An important decision to make is how to organize the buffers. We assume that all buffers are organized as fifos. A dynamic buffer structure, which is commonly used in software solutions, seems not easy to implement in hardware. A more or less static buffer structure is easier to implement. One important problem associated with buffering is the severe throughput degradation due to *Head Of Line* (HOL) *blocking*. Whenever a fifo is filled with data that cannot be forwarded, all the data which enter later will be blocked too.

In this context we have to make a trade-off between performance and buffer utilization. When the buffers are organized as (one or more) large fifos, the efficiency of these buffers can be high, since all buffer space can be used. From a performance point of view a better approach is to use parallel buffers.

A switch may organise the input buffers by associating them with *virtual lines*, i.e. a connection between sender and receiver. Virtual networks, implementing a number of virtual lines on one physical link, was first introduced as a technique to avoid deadlocks in networks. Virtual lines decouple allocation of buffers from allocated lines by providing multiple buffers for each channel in the network. A blocked message, even one that extends through several links (e.g. in case of wormhole routing), blocks only one virtual line and can be overtaken by messages in other virtual lines. Figure 2 shows as an example the line buffers of 5 nodes.

Dally [Dally 92] showed that virtual networks increase the connectivity of networks and have performance advantages as well. The disadvantage is that buffer utilization is low. However, as the cost of memories is fairly low, it seems more important to realize an efficient data transfer, than to have a high buffer utilization.

Since the total available buffer space is divided into many small buffers, we need to have an efficient *flow control* mechanism to prevent buffer overflow. In ATM flow control has been omitted. Bandwidth is allocated in such a way that statistically the amount of buffering is sufficient most of the time.

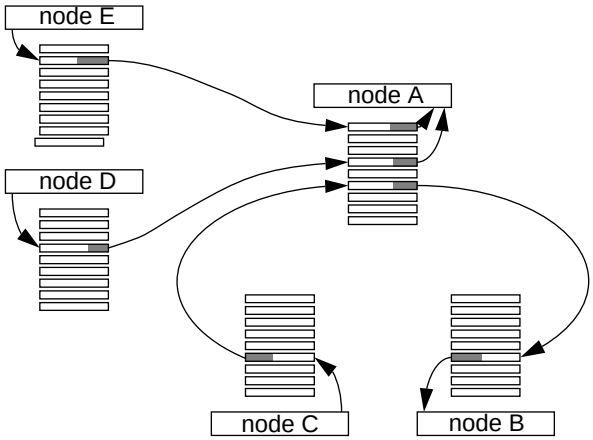

Figure 2: Virtual lines concept

Cells that arrive at an input port with full buffers are discarded; evidently not a subtle method, but necessary to avoid deadlock. This lack of flow control has been adopted for simplicity reasons, it is very difficult to implement a flow control over high speed links bridging large distances. In a local network environment however, such as within a switching fabric, it *is* possible to perform a proper flow control. The main advantage of an efficient flow control is that there is no longer need for huge buffers that are capable to deal with the high speed networks. Moreover, even with large buffers it is not possible to prevent loss of cells. With speeds of 100 Mb per second even large memories will be filled fast. Adding the number of buffers (lines) is a more efficient use of storage than adding depth to a buffer.

2.3 Conclusion

The most important difference between the two concepts is the buffering. The class climbing concept uses large buffers that can be used by all connections, whereas the virtual lines concept uses small buffers in which each buffer belongs to one connection. Other properties are:

- The number of *simultaneous connections* using virtual lines is restricted to the number of buffers. The class climbing concept however supports an unlimited number of connections.
- There is an *automatic workload distribution* due to the nosy worms protocol. If the shortest route contains a contented node, the algorithm will automatically select another (node disjoint) route.
- Both methods are *deadlock free*. When all virtual lines of a link from a route from source to destination are already claimed, the nosy worms protocol bounces back to the source, thereby avoiding deadlocks. The source will then try another route.
- The virtual lines have a *bounded latency*. The class climbing concept cannot give any bandwidth guarantees.
- The *buffer utilization* of virtual lines is poor. The memory is split up into many lines. The lines can only use their allocated memory, even though other lines have buffers available. However, the virtual line concept has not the intention to solve local bandwidth problems of a high-speed network with large memory storage, because even large memories will be filled fast. The buffer utilization of class climbing is high, but suffers from HOL blocking.

3 The Rattlesnake port controller

In this section we will give some details about the prototype Switch, called Rattlesnake, we are currently designing. The switch consists of a number of *switching elements* interconnected via bi-directional links. The switching elements are configured in a Kautz network topology [Kautz 68] because

of its valuable properties. Particularly, Kautz graphs have a small diameter, a fixed and a small degree [Imase 86].

Each switching element is connected to a port controller (called *Snake Control*). The external links of the port controller are point-to-point links (> 100Mbit/s) using the ATM transfer mode. These links are connected to a (work)station or a server. The port controller has local memory in which the ATM cells coming from and going to the serial link are buffered.

The communication architecture should provide a guaranteed throughput and a bounded latency. Therefore it must be able to establish *real-time* connections. We realize this with bandwidth reservation using virtual channels. A connection between two arbitrary stations is made via two or more switching elements in the switching fabric. A message generated by a source station travels through these switching elements to reach a destination station. The switching elements forward messages from an input link to an output link, as directed by the destination address in each message header.

3.1 Rattlesnake routing and buffering

The port controller is connected to a Switching Element (SE) [Smit 92] that supports real-time connections with other SEs. A SE has 3 input links, and 3 output links; each link contains 16 small buffers. These buffers are used to implement 16 *virtual channels* on one physical link. Note that the term 'channel' is directly related to the physical connection between SEs, whereas the term 'line' is related to the connection between port controllers. In a Switching Element only a few flits are buffered. A *flit* is the smallest unit of data and is in our case 32 bits.

When establishing a connection from source to destination, each SE assigns an input channel and an output channel for this circuit. To find a connection from source SE to destination SE a *wormhole routing* mechanism is used. Wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. As soon as a node examines the header flit of a message, it selects the next channel on the route and begins forwarding flits down that channel. The path of a logical connection is set for the duration of a connection. This means that messages of a certain connection will always follow the same path through the switch.

In the design of the Snake Controller we also use a kind of wormhole routing, called *virtual line routing* [Tibboel 93]. In the Snake Controller packets (i.e. ATM cells) are buffered. The buffer space is located in off-chip memory. This memory is divided into a number of buffer queues, called *virtual lines*. The Snake Controller uses the virtual channels of the Switching Element (SE) to make real-time connections to other Snake Controllers (SC). The virtual lines are put on top of the virtual channels of a SE. So on each virtual channel between SEs there can be several virtual lines from a SC. The circuit switched connections between SCs may be to the neighbour SE, but it may also traverse several SEs before reaching the next SC. In this way it is possible to create a highway between SCs with a high mutual communication load.

A path from source to destination is established by reserving and linking the virtual lines in each intermediate SC on the path. The packets are transferred on a hop-by-hop basis from SC to SC until the destination is reached. Each cell is first received by a SC in its full length at each intermediate SC between source and destination, and then it is transmitted to the next SC. If a failure or a blocking condition is encountered along the route the set-up message gives up by recoiling back to the sender, thus avoiding deadlock. Another (node disjoint) route can be tried[1]. This leads to an adaptive routing mechanism that routes around congestions and faults. Adaptive routing is attractive in networks where alternative routes are not much longer than the first route(s). The averaged hop lengths in our prototype network of 108 nodes in a Kautz topology of the first, second and third route are respectively: 3.5, 4.9 and 5.2 hops [Tibboel 93].

By this combination of circuit switching and packet switching we have created a *hybrid transfer mode*. We have combined the real-time circuit switched network between SEs with the flexibility of packet switching between SCs. Applications generating hard real-time data are able to reserve a

1. The probability of a connection message reaching its destination depends primarily on the chances of finding an unbroken chain of channels. [Smit 93] gives simulation results showing that these chances are improved by using a low diameter network (e.g. Kautz), using alternative routes, and by using virtual channels.

circuit switched connection from source to destination. Less demanding applications, like file transfers, may use the packet switched component of the transfer mode. The hybrid switching mode appears to be particularly attractive for networks that have to support dissimilar traffic of both continuous and bursty types generated by heterogeneous users, such as networks for multimedia traffic. A circuit switched connection can be used to satisfy the stringent real-time and latency constraints for continuous traffic, and the packet connection can be used for non real-time traffic.

3.2 Snake Control architecture

Figure 3 shows the global view of a node. Each node contains a switch, a Snake Control and two

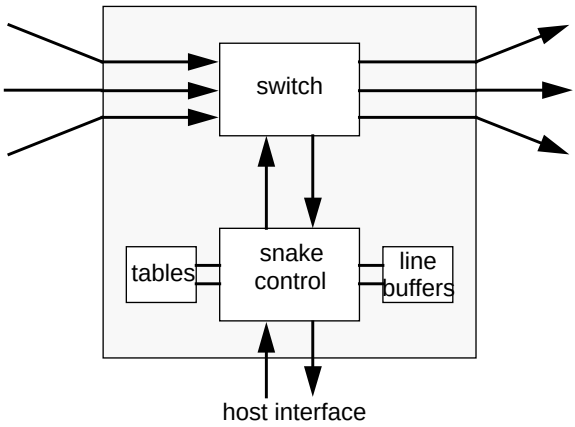

Figure 3: structure of a network node

memories. One memory is the line memory, where data of the network can be stored. The other memory is the table memory, which is used for administration storage.

The communication between the host and its network node is based on ATM cells. When a host transmits an ATM cell to the node, the port controller will store the cell in the virtual line queue that was reserved for the ATM virtual channel (indicated by the VCI in the header of the cell). A virtual line connection between port controllers is put on top of the virtual channels of a SE. Therefore a Virtual Line Identifier (VLI) contains 3 elements: outgoing link number, virtual channel number and the line number on the channel.

The Snake Controller splits up ATM cells coming from the stations into flits and transmits them to its Switching Element. A *flit* consists in our case of 32 bits data and 8 bits identification. The identification consists of a 4 bits virtual channel number and 4 bits type. In addition to this data path, there is a 1 bit reverse status path that can be used to implement a flow control mechanism. The flow control can prevent loss of cells due to buffer overflow not just within the SE, but also within the SCs.

Figure 4 shows the internal organisation of a Snake Controller. It has 640 parallel *line queues*. These queues are used to implement the virtual lines. When establishing a connection from source to destination, each SC assigns a line queue for this circuit. The routing information (i.e. the selected outlink, the virtual channel to be used and the virtual line) is stored in a *mapping table*. Each entry in the table contains the new virtual channel number, the selected outlink and the line on the virtual channel. This information is used for the connection with the next SE. Each entry in the mapping table has only local significance and identifies the local virtual line translation.

The receiving Snake Control receives the flit data and the corresponding virtual channel and the inlink number. This information is used to index the table to generate the new virtual line. This line is used to select the required line queues.

3.3 Scheduling

The scheduler of the sending Snake Control has the most important and difficult task: it must select a flit of an ATM cell that must be transmitted to the next SE.

On one virtual channel the SC will transmit whole ATM cells only. A complete cell is transmitted,

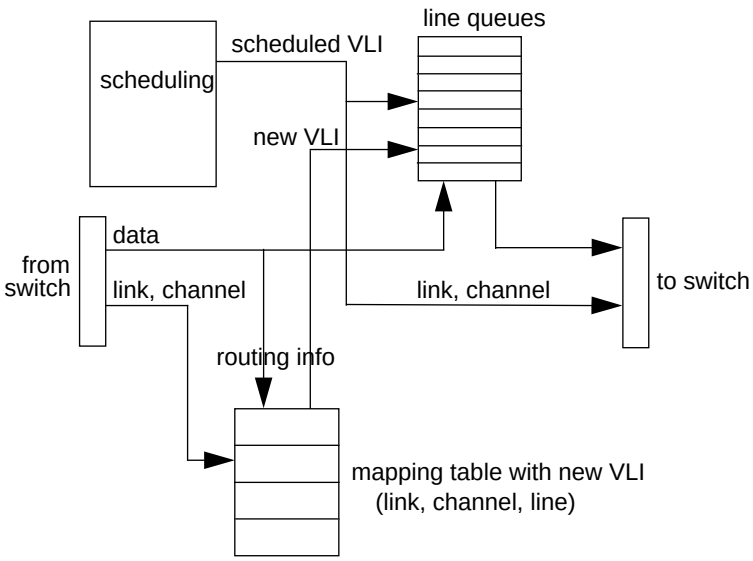

Figure 4: Structure of the port controller

interleaved with cells of other channels, before the next line of the same channel will be scheduled. The current implementation of the scheduler uses a simple round robin mechanism to select a line that contains an ATM cell. The scheduler is the most critical part of the SC design with respect to the implementation limitations in time and space. In the current implementation it must choose from 80 virtual channels, and within each virtual channel it must schedule 8 lines. The scheduling problem is split up into two layers. First a virtual channel for a link is scheduled, and secondly a line of that channel is scheduled.

- The *channel scheduler* is implemented with a fifo of 80 deep. The fifo contains channels that have an ATM cell in a line queue. Every time an ATM packet is received from either the host, or from another port controller, the channel will be put on top of the fifo. Such a fifo can be implemented with on-chip memory.
- The *line scheduler* uses a bit array to administrate the status of the lines, indicating which lines are active of that channel. This status together with the current line gives the next active line. This line is stored in a lookup table containing the currently active lines of a channel.

The scheduler will select one line and read one flit from the queue. This line will be offered to the SE or to the host. When the flit is destined for the SE and it has not yet transmitted the end of the ATM cell, the line will be put on top of the fifo again. A scheduled line that is destined for the host will transmitted a whole ATM cell.

3.4 Connection setup

The connection setup procedure requires connections on two layers. Firstly it requires the circuit switched connection between SEs. These connections are used as point-to-point connections between SCs. Such a connection in general reaches the neighbour SE only, but it may pass several SEs. The circuit is used to transfer the packets of various virtual lines. Hard real-time connections can have there own private circuit from source to destination. The circuit switched connections are *static*: once they are set up, they remain intact for a long period. The wormhole routing mechanism is used to reserve buffers in a SE.

The second connection layer is a packet switched end-to-end connection between the source and destination of the message. The packets are transmitted on a hop-by-hop basis through one or several SCs before reaching the destination. The packet switched connections are more *dynamic*: the connection will be released after a message transfer is completed. In this way the line buffers are released and can be used for other connections. The packet switched connections also use the wormhole routing mechanism. This requires that each node of the path has a free line. Only when all the nodes of the

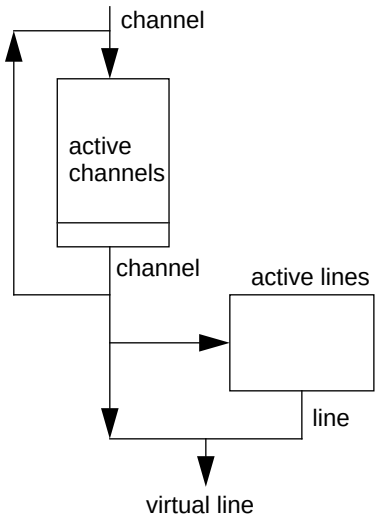

Figure 5: Scheduling

path can reserve a virtual line, a connection can be established. When a node has no free line, this node must inform all the preceding nodes which did reserve a virtual line, so that they can cancel the reservation. Only the last node of the connection can return a status indicating that the connection is established.

To find a path from source to destination we take advantage of the self-routing property of Kautz networks. The host which will setup a connection must generate a connection setup request cell. This cell contains the route to be followed, called the routing tag. The consecutive digits of the routing tag are interpreted stage-by-stage. If this connection setup effort fails because one of the links of the path is already complete reserved, the host will try an alternative route.

The path of a logical connection is set for the duration of a connection. This means that messages of a certain connection will always follow the same path through the switching fabric until it receives a *release* cell. The SC will transmit the release cell to the other SCs down stream.

3.5 Implementation

The architecture of the port controller uses several memories. These memories are used to implement the lookup tables, the scheduling fifos, the line queues, and some local administration. Most of the basic elements of the SC operate concurrently to meet the timing requirements needed to handle the high speed bandwidth rates. The transmission of a cell to the external link operates simultaneous with the reception of a cell from the link. Likewise, the transmission and reception of flits to and from the SE also occur simultaneously. Simultaneous access to tables and fifos is required to achieve these operations. One shared memory would create an enormous bottleneck that can only be solved using a wide data path, or very high speed memories. A better way is to use separate memories. We make extensive use of the distributed memories inside the Xilinx X4000 FPGA. Due to the little capacity of these memories they can only be used to implement several small tables. The large tables and the line queues are implemented using two external memories.

The prototype printed circuit board with one network node contains 2 Xilinx PG4010 FPGAs to implement the Switching Element and the Snake Control, and two memories of 2 Mbytes connected to the Snake Control. A switch has 3 input and 3 output links with each 9 wires. These wires can be selected uni-directional or bi-directional. The interface with the node computers is provided using TAXI links[1].

This prototype board allows us to do experiments with and performance measurements of several transmission modes, bandwidth reservation, scheduling, routing for non real-time traffic, link proto-

1. We use the Fairile transmission board designed by the University of Cambridge.

cols, multicast etc. We use VHDL as a design tool and a VHDL synthesizer from VIEWlogic [VIEWlogic 90] to generate the configuration code for the Xilinx chips.

4 Conclusion

In this paper we have presented the architecture of a network suitable for hard real-time multimedia applications. This ATM network uses a modified nosy worms protocol for setting up connections. Once a connection is established it guarantees a bounded latency. We make use of virtual lines to improve the performance and to avoid communication deadlocks.

The buffering of the messages is a main topic. We have organized the buffers as parallel fifos, each representing a virtual line. In this way we have not only solved the problem of HOL blocking, but we can also give real-time guarantees. We have shown that for local high speed networks it is more advantageous to have a proper flow control than to have large buffers. Although the virtual line concept can have a low buffer utilization, the transfer efficiency can be higher.

The virtual lines concept allows adaptive routing. The total throughput of the network can be improved by using alternative routes. Adaptive routing is attractive in networks where alternative routes are not much longer than the first route(s).

A prototype of the switching fabric will be implemented with standard FPGAs. We use two external memories and several on-chip memories to implement the various tables and fifos. This allows us to access the memories concurrently and meet the timing requirements. The design of the switching fabric with FPGAs makes it possible to experiment with switching mode, routing strategy and scheduling policy in a multimedia environment.

5 References

[Adamo 91]
Adamo, J.M.: "*Minimal, adaptive and deadlock-free routing for multiprocessors*", Laboratoire de LIP-IMAG, Ecole normale superieure de Lyon, France, June 1991.

[Dally 92]
Dally W.J.: "*Virtual-Channel Flow Control*", IEEE transactions on parallel and distributed systems, Vol. 3, no 2, March 1992, pp 194-205.

[Imase 86]
Imase M., Soneoka T., Okada K.: "*A fault-tolerant processor interconnection network*" (original in Japanese); translated in Systems and Computers in Japan, Vol. 17, no 8 pp 21-30, 1986.

[Kautz 68]
Kautz W.H.:"*Bounds on directed (d,k) graphs. Theory of cellular logic networks and machines*", AFCRL-68-0668 Final report, pp 20-28, 1968.

[Leslie 93]
Leslie I.M., McAuley D., Mullender S.J.: "*Pegasus - operating system support for distributed multimedia systems*", ACM SIGOPS Operating System Review, Vol. 27, no 1, Jan. 1993, pp 69-78.

[Smit 92]
Smit G.J.M., Havinga P.J.M.: "*The architecture of Rattlesnake: a real-time multimedia network*", Proceedings 12th IFIP World Congress, September 1992, pp 578-584

[Smit 93]
Smit G.J.M., Havinga P.J.M.: "*Performance analysis of routing algorithms for the Rattlesnake network*",

Proceedings MASCOTS '93, January 1993, pp 155-160

[Tibboel 93]
Tibboel W. H.: "*Virtual lines, a deadlock free and real-time buffer allocation mechanism for an ATM-network*", Ms Thesis, department of Computer Science, University of T wente, July 1993.

[Viewlogic 90]
"*VHDL-Designer User's Guide*", VIEWlogic Systems Inc., April 1990.

[Whobrey 88]
Whobrey D.: "*A communications chip for multiprocessors*", Pr oc. CONPAR 88, pp 464-473, 1988.

[Xilinx 91]
"*The Programmable Gate Array Data Book*", Xilinx Inc., 1991.