Routing in Wormhole Networks

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in the

Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Swaminathan Ramany

December, 1995

DISTRIBUTION NOTICE

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan, Canada

S7N 5A9

ABSTRACT

As software overheads decrease and processor speeds increase relative to network speeds, the performance of multiprocessor interconnection networks will become increasingly important. This motivates design of efficient switching and routing schemes. Wormhole switching is an efficient switching technique wherein messages are pipelined through the network. Networks employing wormhole switching (termed "wormhole networks") are currently very popular as they yield low network latency and have minimal buffer requirements. This thesis is primarily concerned with performance and reliability aspects of wormhole networks. The main goal of this thesis is to identify how routing can be done effectively in these networks.

Two concepts that have been proposed for performance improvements in wormhole networks are "Virtual Channel Flow Control" (VCFC) and "adaptive routing". In the VCFC concept multiple virtual channels (logical channels) share the same physical channel and can be employed as "lanes" with complete freedom in routing choice among them. Many new adaptive routing schemes have also been proposed in recent times which allow the next hop taken by a message to be determined "on the fly" based on local conditions at each router. These adaptive routing schemes have been shown to improve performance for certain traffic patterns and at the same time perform badly for other patterns.

This thesis begins by investigating the question of whether adaptive routing can be made to perform uniformly well for all traffic patterns by proper combination with VCFC. It is shown that although both these techniques are not entirely satisfactory in isolation, a combination of both can provide substantial benefit. Even if adaptive routing techniques can be made to provide good improvements in performance, they are limited by the fact that routing is based only on local conditions at each router. This thesis next investigates the extent to which the local nature of decisions affect routing performance and explores the scope for further improvement. A practical source-directed routing method which uses application-specific knowledge is shown to be a potential approach to achieve these improvements. Finally, this thesis develops a general and cost-efficient fault tolerance approach based on reconfiguration of virtual channels around fault regions. The proposed approach is shown to be capable of tolerating a large set of faults and at the same time providing good performance.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Derek Eager, for countless hours of discussion and guidance without which it would not have been possible to complete this thesis.

Thanks also to my thesis advisory committee members, Carey Williamson, Rick Bunt, Paul Tremblay, Eric Salt and Mary Vernon (External examiner) for their comments and valuable feedback.

Thanks to my mom and dad, my sister, Suja, and my fiancee, Ranjani, for providing moral support during my thesis research.

Thanks to the University and the Department for providing the funding for this research.

Special thanks to Bharadwaj for his help during the final stages of the thesis. Finally, thanks to all my friends in the department who made my stay here in Saskatoon enjoyable.

Contents

List of Figures

Chapter 1

Introduction

Over the last few decades there has been an increasing need for high performance computing. Pollution monitoring, analysis of protein structures, weather forecasting, speech recognition, ozone layer monitoring and petroleum exploration are some of the application areas which need large amounts of computing power [42]. This has resulted in the development of parallel systems typically consisting of tens or hundreds of computing nodes that can execute instructions in parallel. Particularly, parallel systems based on Multiple Instruction Multiple Data (MIMD) architectures have become very popular in recent times. These MIMD systems consist of multiple nodes which can execute independent instruction streams in an asynchronous fashion. These systems can be employed cost-effectively in a wide variety of application domains.

MIMD systems can be classified as either *shared memory multiprocessors* or *message passing multicomputers*. In both these types of parallel systems an interconnection network is used to pass messages containing data or synchronisation information between different nodes. The messages may be sent between processors in a message passing multicomputer or between processor and memory in a shared memory multiprocessor. In both types of systems, the interconnection network is a critical component, as application performance can be very sensitive to network latency and throughput [11]. The interconnection network may also account for a significant fraction of the cost and power dissipation of the machine [24]. Ideally, the interconnection network should deliver messages with low latency and support high throughput. In addition, the interconnection network should also be cost effective, reliable and support a high degree of fault tolerance.

An interconnection network can be defined by its topology, routing, switching and flow control [52]. Topology defines the interconnection between the nodes. Routing determines the path a message will take through the network. The switching and flow control techniques determine how each message is to be broken up into smaller units, allocated network resources, and transmitted on the network. Wormhole switching is a recently proposed switching technique wherein messages are pipelined through the network [24]. This form of switching is very popular in current machines because it can yield low network latency and has minimal buffer requirements. Networks employing wormhole switching are referred to as wormhole networks in the remainder of this thesis.

Two different concepts that have been proposed for performance enhancement in wormhole networks are "Virtual Channel Flow Control" (VCFC) and "adaptive routing". Virtual channels [24] were initially proposed to facilitate deadlock avoidance in torus networks. A virtual channel is a "logical" channel with its own buffer, control and data path, that shares the same physical channel with other virtual channels. In the Virtual Channel Flow Control concept, the multiple virtual channels that are implemented on each physical channel are employed as multiple "lanes" with complete freedom in routing choice among them [21]. Owing to the resulting ability to allow multiple messages in the same physical channel, and thus allow messages to bypass blocked messages, improved performance can be achieved.

Since in wormhole networks a blocked message can hold many resources (buffers and channels) while waiting for other resources, deadlocks are potentially more likely. Most current systems use deterministic routing which permits relatively simple implementation of deadlock avoidance. Deterministic routing, however, allows use of just a single physical path from the source to destination. Thus there is a loss in routing flexibility that can lead to poor performance. Adaptive routing takes advantage of the density of network interconnection. Messages have several routing choices at each node and the path choices are made based on the local information available at the node. Adaptive routing can yield improved performance [22, 31, 33].

This thesis is concerned with performance and reliability aspects of wormhole networks. Of specific interest is the issue of routing in these networks and how it can be done effectively.

1.1 Motivation and Goals

The main goal of this research is to identify how routing can be done most effectively in wormhole networks. Many new adaptive routing algorithms have been proposed recently which promise good improvements in routing performance [10, 12, 13, 15, 19, 22, 26, 33, 37, 47, 60]. However, some studies have shown that adaptive routing algorithms perform badly for certain traffic patterns [31, 33] and that adaptivity is not necessarily a benefit in wormhole networks.

This raises the question, how well does adaptive routing really do when compared against deterministic routing? Can adaptive routing be made to perform uniformly well for all traffic patterns? A previous study on VCFC has shown good improvements in performance with an increase in VCFC, but this study focussed only on a network with deterministically routed traffic [21]. The use of VCFC with adaptive routing is, therefore, a potentially fruitful topic for investigation. Thus, one goal of this thesis is to study adaptive routing performance and, in particular, the interaction with VCFC.

Even if adaptive routing techniques can be made to provide good improvements in performance in comparison with deterministic routing, current techniques are limited by the fact that routing decisions are based only on the "local" conditions at each router. How much does the purely local nature of decisions limit adaptive routing performance, and is there scope for further improvements? If so, what approach should be followed to achieve these improvements? A second goal of this thesis is to quantify how close the performance of adaptive algorithms comes to the best realistically achievable performance. To achieve this goal the best realistically achievable performance should be quantified.

In large networks, faulty nodes and links may become relatively common. This raises questions concerning the types of mechanisms needed to cope with faults and how such mechanisms would impact performance. The fault tolerance schemes proposed in current literature [16, 34] are either significantly expensive or are not powerful enough to be general and efficient. There exists a need for fault tolerance approaches which minimally impact the routing in the absence of faults and at the same time are general and cost efficient in the presence of faults. A third goal of this thesis is the development and evaluation of an approach to fault tolerance that is capable of tolerating a large set of faults and at the same time provides good performance.

1.2 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 presents necessary background information on multiprocessor interconnection networks. Different design and performance issues pertaining to multiprocessor interconnection networks are discussed in this chapter. Chapter 3 provides a survey of the adaptive routing algorithms which have been proposed in the literature. A taxonomy for classifying these adaptive algorithms is also presented in this chapter.

Chapter 4 studies the interaction between VCFC and adaptive routing. Results are presented from a simulation study of deterministic and adaptive routing algorithms with varying usage of VCFC. It is shown here that each of VCFC and adaptive routing may yield unsatisfactory performance only when used in isolation, and that through their appropriate combination uniformly improved performance can be achieved.

In Chapter 5, the limits of realistically achievable routing performance are quantified, and the extent to which adaptive routing approaches these limits is assessed. Simple performance bounds are initially derived so as to allow comparison against absolute performance limits. Then the performance of adaptive routing is compared against a realistically implementable source-directed routing technique which utilises application-specific knowledge to make more informed decisions. The specific source-directed routing method that is proposed can interoperate with either deterministic or adaptive routing, so as to allow these latter techniques to be used as desired when message patterns are unpredictable. The results of this study indicate that adaptive routing techniques perform well when performance is determined by average message latencies. Adaptive routing techniques are found to be less effective, however, in reducing maximum latencies owing to the fact that they utilise only "local" knowledge of the network state. In particular, when the performance is dictated by the maximum message delivery time within a communication phase, the proposed source-directed routing method can yield large improvements in performance in comparison to both deterministic and adaptive routing.

Chapter 6 proposes a new approach to fault tolerance, called "Virtual Channel Reconfiguration" (VCR), based on reconfiguration of virtual channels at nodes adjacent to faults. The proposed scheme allows deadlock-free and livelock-free routing around many topologies of node and link faults as long as "enough" virtual channels are defined on each physical channel in the network (at least 3 for every physical channel in the case of a 2-dimensional mesh). These virtual channels may be used for VCFC and/or adaptive routing when not required for routing around faults. This approach has the advantages of handling a large class of fault topologies, of minimally impacting the network in the absence of faults, and of requiring no additional hardware. A simulation study is also carried out in this chapter so as to assess the performance provided by the VCR approach, and to understand the impact of faults on network and routing algorithm performance. The VCR approach is found to provide very reasonable performance in the presence of faults, while at same time outperforming many approaches in terms of fault tolerance capacity.

Finally, Chapter 7 presents conclusions and describes avenues for future work.

Chapter 2

Multiprocessor Interconnection Networks

Interconnection networks can be broadly classified into two types: indirect networks and direct networks. In indirect networks, communication between any two nodes proceeds through a sequence of intermediate switches. In direct networks each node is connected directly to a small number of other processing nodes by means of point-to-point links and communication between any two non-directly connected nodes must proceed along paths formed by multiple links. The exact connection is defined by the topology of the network. Figure 2.1 shows an example of a mesh type of direct network. Each node has a memory, a CPU and a router. The router connects the node to the rest of the network by means of bi-directional network channels. Direct networks have become more popular in recent times since they can exploit locality in communication patterns, and can thus scale more easily to support large systems [19]. Examples of machines that use direct networks include the Caltech Cosmic Cube [61], Intel iWarp [14], Intel Paragon [36], MIT Alewife [4] and Stanford DASH [44]. Since this thesis is concerned solely with direct networks the focus of

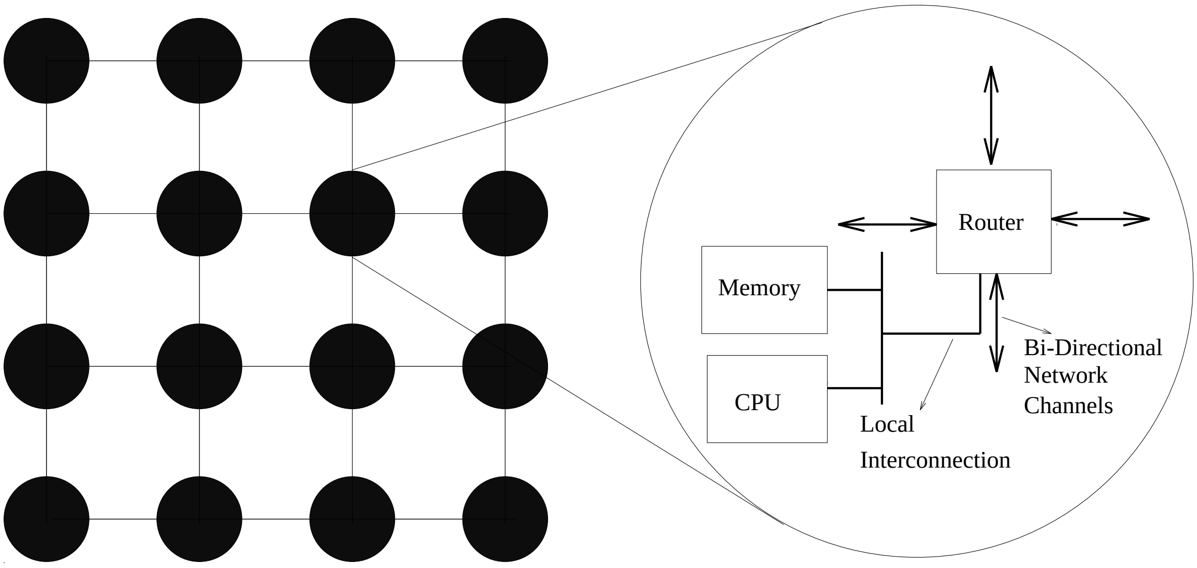


Figure 2.1: An example of a mesh type of direct network showing the typical node architecture

this chapter is on various design and performance issues in these networks.

A direct network design is defined by its topology, switching, flow control and routing [52]. Reliability, functionality, performance, cost, and fault tolerance are important criteria by which designs of networks may be judged. With respect to functionality, one issue is the architectural support (or lack thereof) for collective communication operations such as broadcast and multicast [49]. Sections 2.1 to 2.4 of this chapter consider topology, switching, flow control and routing, respectively, without consideration of the special needs of fault tolerance and collective communications. These issues are considered separately in Sections 2.5 and 2.6, respectively.

2.1 Topology

Topology defines the interconnection between the nodes. Most of the popular direct network topologies fall into the category of k-ary n-cubes or n-dimensional meshes. A

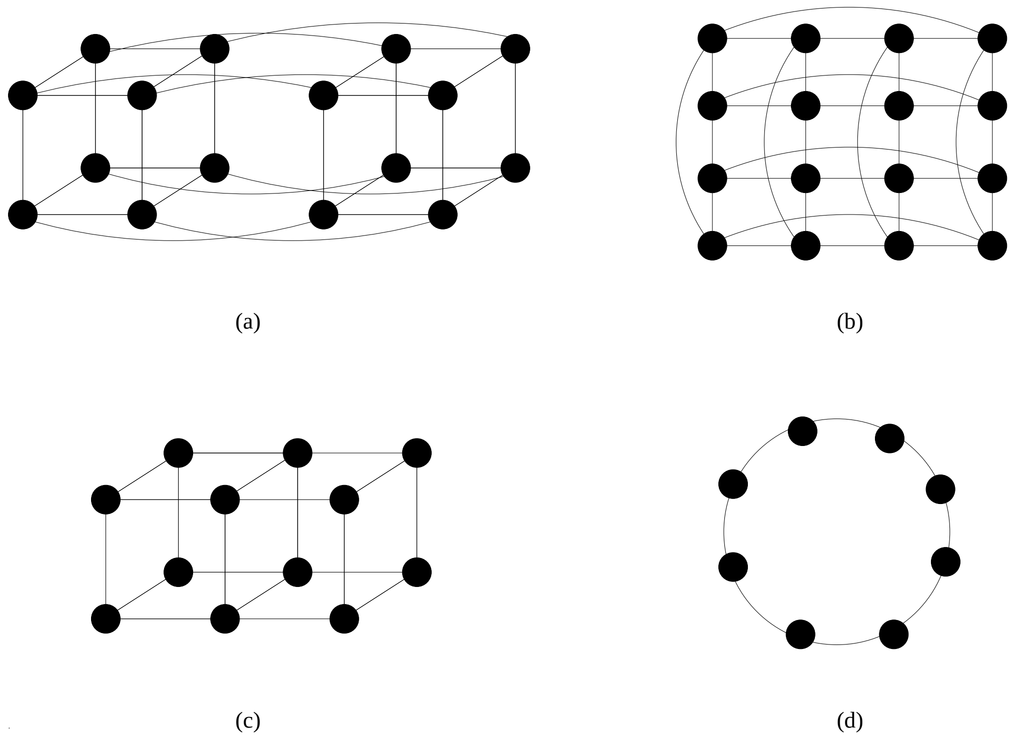


Figure 2.2: Direct network topologies. (a) 2-ary 4-cube (hypercube), (b) 4-ary 2-cube (2D torus), (c) $2 \times 3 \times 2$ 3D mesh, (d) 8-ary 1-cube (ring)

k-ary n-cube network has n dimensions and in each dimension k nodes are connected by point-to-point links as in a ring topology. A n-dimensional mesh differs from a k-ary n-cube in that meshes do not have wraparound links. Figure 2.2 provides examples of different direct network topologies. Note that in k-ary n-cubes all nodes have an equal number of neighbours whereas in meshes there is some asymmetry. Although topologies with uni-directional links are possible, attention is restricted in this thesis to topologies with only bi-directional links, each composed of two (uni-directional) "physical channels" that may be used concurrently.

Most recent systems employing direct network topologies use low dimensional networks (2D mesh/torus, 3D mesh/torus). Low dimensional mesh and torus networks might have significantly less cost and design constraints than higher dimensional meshes and tori and may also utilise physical resources more efficiently. At the same time they have been shown to perform quite well in the presence of communication locality [3, 20], although it appears that very high locality would be required if reasonable performance is to be achieved with large networks [2]. The Intel Touchstone Delta, Intel Paragon [36], Stanford DASH [44] and Symult 2010 use 2D mesh networks. The MIT J-machine and Caltech's Mosaic use a 3D mesh. The Reliable Router [23] chip which is currently being developed is also designed for a 2-dimensional mesh network.

2.2 Switching

Switching concerns the form in which link resources are allocated to messages. The four basic switching policies that may be employed in direct networks are store-and-forward [35], virtual cut-through [38], circuit switching [64] and wormhole switching [24].

In store-and-forward switching, a message destined for a node that is not directly connected to the source node must be received in its entirety at each intermediate node before being forwarded to the next node. Therefore, the "network latency" (delay from the start of transmission of a message at the source until its reception at the destination) is proportional to the distance (in number of hops) between the source and the destination node. In virtual cut-through, in contrast, a message is buffered at an intermediate node only when the desired outgoing channel (or channels if there is routing choice) is/are already in use. In virtual cut-through the network latency, in the absence of contention, becomes largely independent of the distance travelled by the message. This is also the case in circuit switching, in which there is a separate circuit establishment phase during which a dedicated path is constructed between the source and destination nodes. Message transmission then proceeds without delay at the intermediate nodes.

In wormhole switching (sometimes referred to as “wormhole routing” [52]), a message is divided into small fixed-size units termed “flits”; the flit size is often chosen as the number of bits that may be transmitted in parallel on a transmission link. The message passes through the network in a pipelined fashion as shown in Figure 2.3. Header flits contain all the routing information for a message and

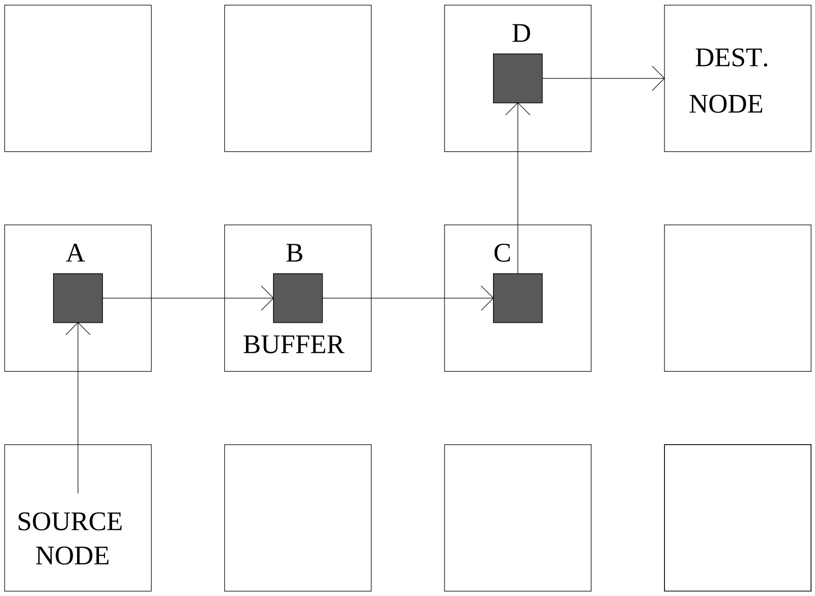


Figure 2.3: Illustration of wormhole switching: Movement of flits from Source to A, A to B, B to C, C to D and D to Destination occurs in parallel

lead the message through the network. When the header flit is blocked (owing to unavailability of an appropriate outgoing channel) all the flits wait where they are, until the header flit can move again. For each link port, only sufficient buffer space for a single flit is required. If additional buffer space is available, some limited movement of flits of a message whose header is blocked may be possible. The major difference between wormhole switching and virtual cut-through is that in virtual cut-through, when a message is blocked there must be sufficient buffer space to store the message locally.

Figure 2.4 compares the time required to deliver a message for the example shown in Figure 2.3, for different switching schemes in the absence of contention. Virtual cut-through is not shown separately as its latency is the same as that of wormhole switching in this case. A wormhole network has low communication latency compared to store-and-forward networks. The network latency for wormhole switching, assuming no contention and switching delays, is approximately $(L_f/B)D + L/B$, where L_f is the length of each flit, L is the message length, D is the distance to be travelled (in number of hops) and B is the channel bandwidth [52]. When $L_f << L$, the distance D has a negligible effect on network latency. Wormhole switching has an advantage over circuit switching also, since circuit switching requires a separate circuit establishment phase. Owing to these advantages, wormhole switching is a popular choice in recent generations of multiprocessors.

2.3 Flow Control

Flow control concerns techniques for dealing with contention by multiple messages for the same channels and buffers. A good flow control policy should reduce congestion, be fair and retain low latency. Flow control techniques are very dependent on the switching scheme employed; owing to the popularity of wormhole switching, that scheme is assumed in this discussion.

Wormhole switching can result in unnecessary blocking as messages wait in the network holding network resources. A flow control technique called Virtual Channel Flow Control (VCFC) [21] has been proposed to reduce blocking in wormhole networks. A virtual channel is a "logical" channel with its own buffer, control and

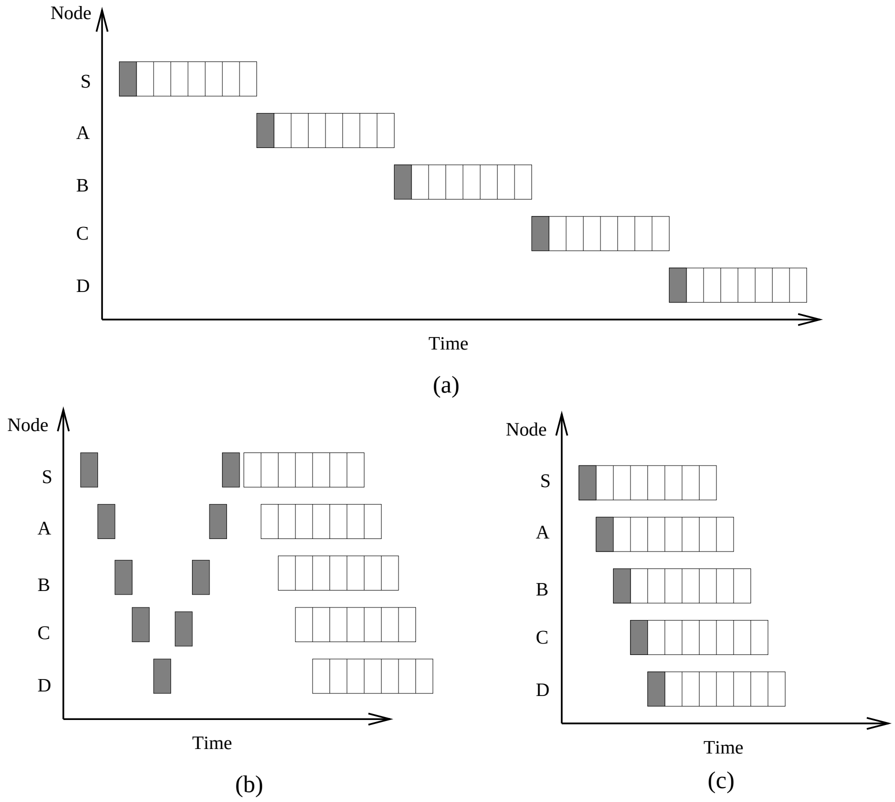


Figure 2.4: Time required to deliver a message from source to destination (for the example in Figure 2.3), for different switching schemes in the absence of contention (a) Store-and-Forward Switching (b) Circuit Switching (c) Wormhole Switching

data path that shares the same physical channel with other virtual channels. The VCFC concept employs virtual channels as multiple "lanes" (with complete freedom in routing choice among them) in a single path. This reduces blocking in the network by allowing messages to bypass blocked messages. It has been shown that VCFC can increase network throughput capacity for a given amount of buffer space [21, 58].

As an example of the operation of VCFC, consider the case shown in Figure 2.5(a), where message B is blocked by message A. This happens since message A is using the physical channel which is required by message B. If this physical channel has two lanes associated with it, then message B would not be blocked by message

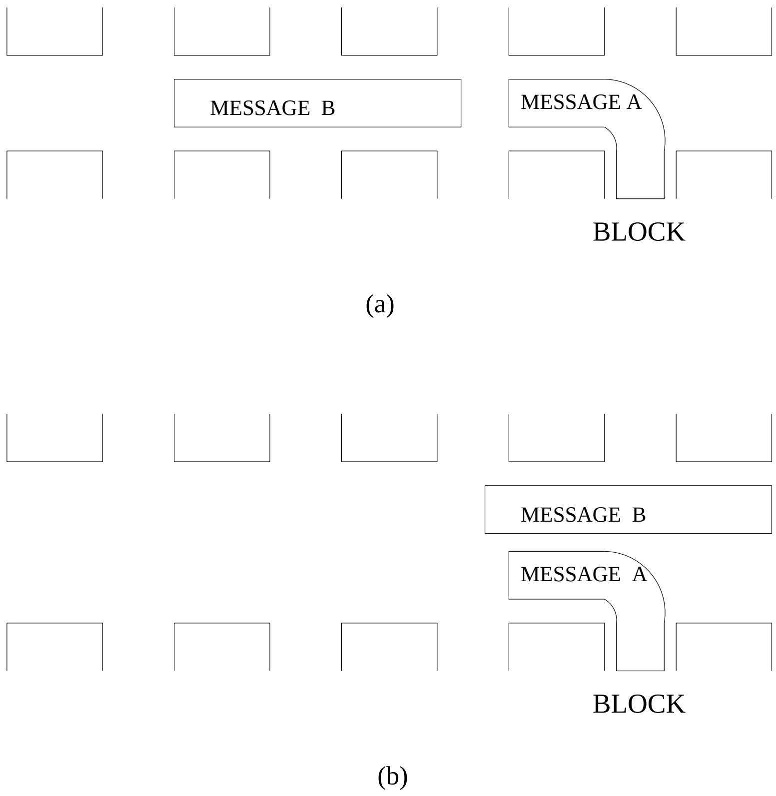


Figure 2.5: (a) Message B is blocked by Message A (b) Message B is no longer blocked due to the presence of extra lanes

A as shown in Figure 2.5(b).

When VCFC is employed, a transmission policy is needed to mediate access to each physical channel. One simple policy is "work conserving round robin", in which unblocked messages occupying virtual channels on the same physical channel are alternatively selected for transmission of a single flit. As another example, one policy that could potentially reduce the variance in message latencies is "oldest flit first", in which those flits belonging to the oldest unblocked message are given transmission priority.

2.4 Routing

A good routing algorithm should have freedom from deadlocks, yield low latency and high throughput capacity, and permit an efficient implementation in VLSI [33]. Deadlock occurs when a message waits for an event that cannot happen. Deadlocks may become possible if messages are allowed to hold some resources while waiting for other resources. Deadlocks are potentially more likely in wormhole networks since blocked messages can hold many resources (buffers and channels) while waiting for other resources. Figure 2.6 illustrates an example of a deadlock due to circular waiting for resources. Here each message is holding a flit buffer and requesting the flit buffer of the next node in a circular fashion.

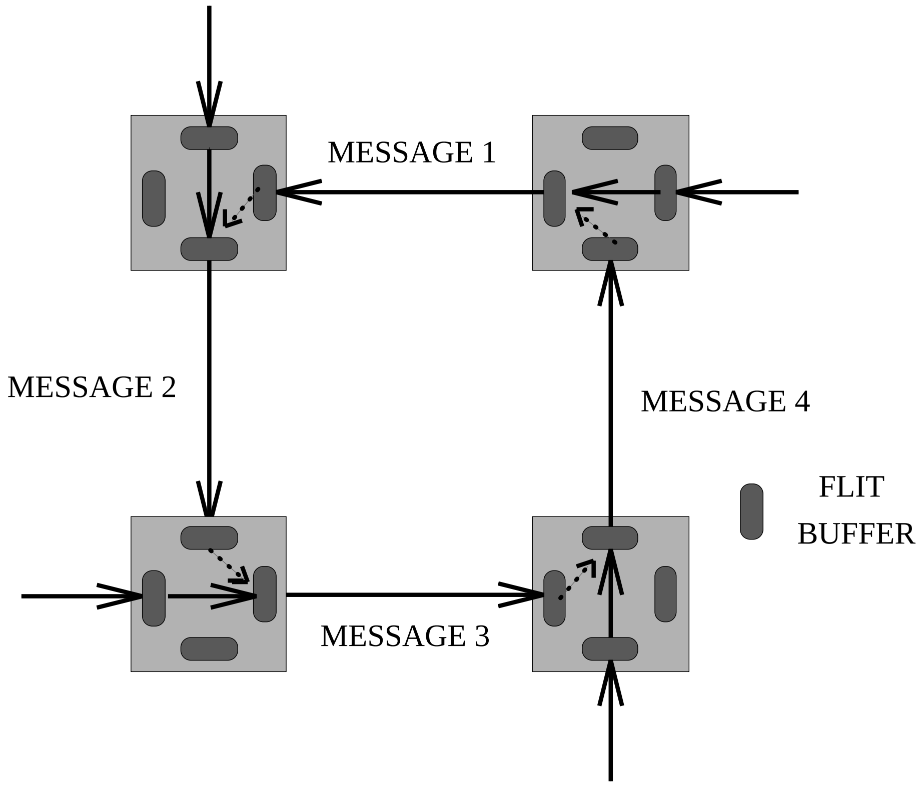


Figure 2.6: Example of a deadlock involving four messages

Deadlocks can be overcome by preemption of messages involved in a deadlock

[40]. The preempted messages can be discarded and retransmitted. This is usually not desirable since detecting deadlocks may be quite expensive and retransmissions can result in large latencies. Another way to overcome deadlocks is to prevent them from happening through appropriate design of the routing algorithm. One such way of preventing deadlock is by establishing an ordering of the network channels, and ensuring that routing is such that a message requests a channel only if the channels already held by the message occur before it in the channel ordering. This is facilitated by using deterministic routing algorithms that choose routes independently of the network conditions. Such algorithms include the XY routing algorithm [52] for 2D meshes, and the E-cube algorithm [63] for binary n-cubes. Deadlock prevention may also be facilitated by adding virtual channels to the network, and thus splitting physical channel resources into multiple virtual channel resources. Deadlock prevention is ensured in this case as long as virtual channels (rather than the physical channels) are used according to some monotonic ordering [24].

2.4.1 Deterministic Routing

Deterministic routing algorithms typically entail routing each message in one dimension at a time until it arrives at the proper coordinate in that dimension before changing over to the next dimension. Ensuring that all messages traverse dimensions in the same order facilitates the establishment of an ordering of network channels that all routings will respect. Deterministic routing is also often referred to as "dimension-order" routing.

One deterministic algorithm is the XY routing algorithm, which is applicable to 2D meshes. The XY routing algorithm sends messages in the X-dimension first and then in the Y-dimension. The routing shown in Figure 2.7 is an example of XY

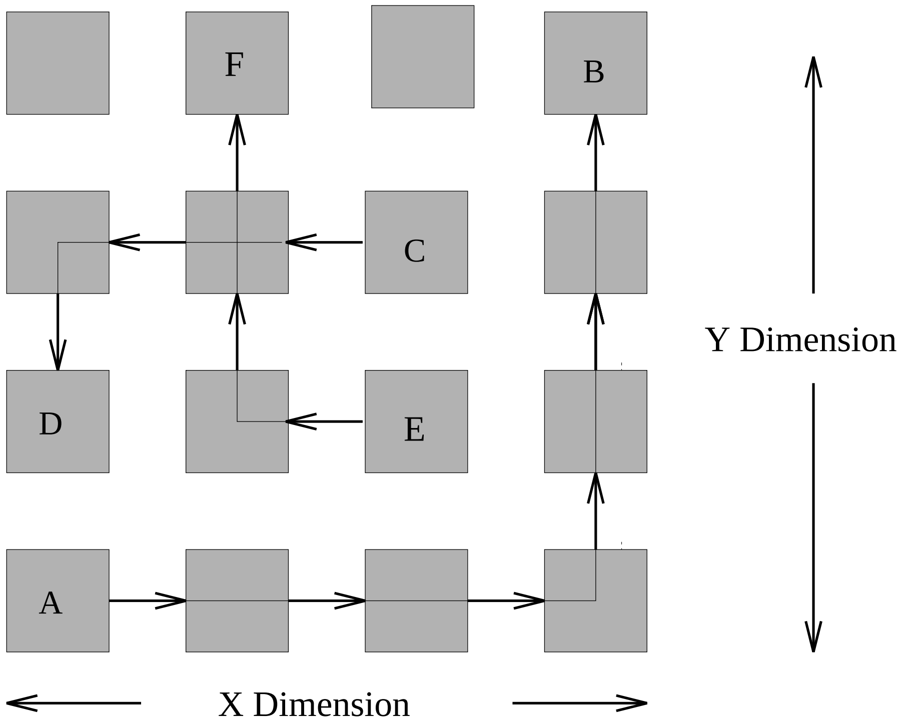


Figure 2.7: Example of XY routing: All messages correct their X coordinate first before correcting their Y coordinate.

routing. It is quite simple and can be extended to n-dimensional meshes also.

The extension of this algorithm for binary n-cubes is called the E-Cube algorithm [63]. Each node is given an address in the form of a n-bit binary number. When a node with address p receives a message with destination address d, it computes $b = p \oplus d$, where $\oplus$ is the bitwise exclusive OR operation. If the result b is zero then the message has reached the destination; otherwise the message is routed along the kth dimension, where k is the position of the rightmost 1 in the binary number b.

For k-ary n-cubes (which have wrap-around links) Dally proposed a deterministic routing algorithm that requires two virtual channels per physical channel [24], termed "high" and "low". Each node is given an address in the form of an n-digit, base-k number. The i^{th} digit gives the position of the node in the i^{th} dimension. Messages are routed in one dimension at a time, in descending order by the dimen-

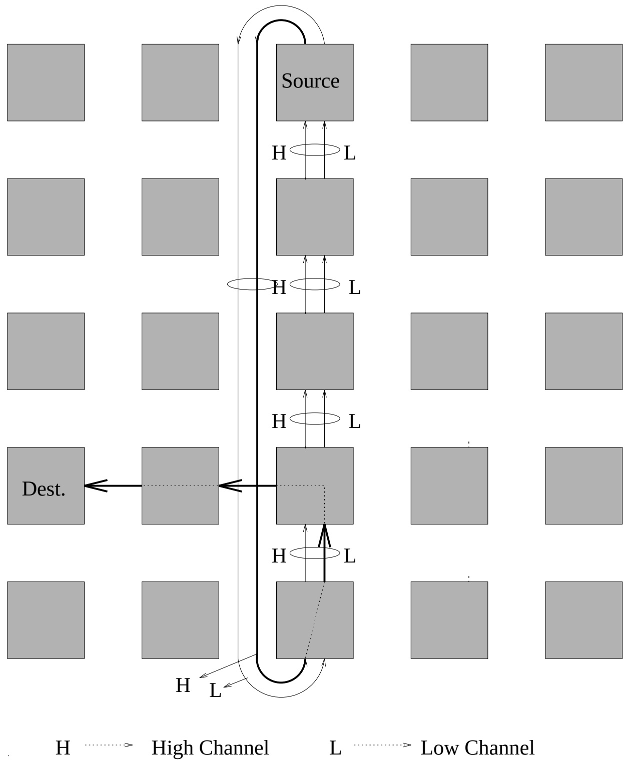


Figure 2.8: An example of deterministic routing in a torus network

sion number. In each dimension n, a message is routed in the high virtual channel if the nth digit of the destination address is greater than the nth digit of the present node's address. Otherwise it is routed in the low virtual channel. (Dally considered networks with uni-directional channels, in which case the direction of routing is fixed.) This scheme ensures that an ordering can be established on the virtual channels that all routings are guaranteed to respect. An example of this algorithm in a 2D torus is shown in Figure 2.8. Here the message takes the high channel until the wrap-around link is taken, after which the low channels are chosen.

2.4.2 Adaptive Routing

Deterministic routing allows use of only a single fixed physical path from the source to destination, even when many paths exist. Thus multiple messages may contend for the same network channel, even when other channels on alternate paths remain idle. The main idea behind adaptive routing is to allow messages to take advantage of the density of network interconnection. When adaptive routing is employed, there can be multiple outgoing links on which an incoming message may be routed, and a choice may be made among these based on their current levels of contention. Adaptive routing has been shown to increase network throughput capacity and decrease average message latencies under many conditions [13, 32, 58].

A disadvantage of adaptive routing in wormhole networks is that deadlocks become possible unless preventive measures are taken. Deadlocks can often be prevented by partially restricting adaptivity [33]. Usually, adaptive algorithms also use multiple virtual channels on each physical channel, for deadlock prevention. In general, adaptive routing involves extra cost and can increase the complexity of the routers [7]. Thus, efficient algorithms that make good use of available resources are essential to the success of adaptive routing.

Many new adaptive routing schemes have been proposed in recent research [10, 19, 26, 31, 37, 47, 60]. These schemes are discussed in detail in Chapter 3.

2.4.3 Source-Directed Routing

Existing adaptive routing schemes might be limited in terms of their ability to improve performance. With adaptive routing the routing decisions are based solely on local knowledge available at each router. One alternative to be considered is source-directed routing in which knowledge about overall network traffic patterns may be employed, when such information is available. In such schemes, message routes that minimise message contention are predetermined and specified in message headers. This should be feasible because often the communication intensive parts of an application have traffic patterns predictable either at compile time [1, 28, 45, 65] or at run time through inspector-executor approaches [59]. Various possible approaches to source-directed routing are discussed in Chapter 5.

2.5 Fault Tolerance

The interconnection network is a critical component in multiprocessor systems; if the network is unusable, so is the entire system. Fault tolerance is therefore an important issue in the design of interconnection networks [53]. In large scale systems, hardware faults may be relatively common, and the network must be robust enough to tolerate such faults by providing routing capability around faulty regions.

The conventional method of making a multiprocessor interconnection network fault tolerant is to add redundant hardware which is used to isolate/tolerate faults to the basic architecture[16]. This sort of approach provides high reliability but at a high cost.

Fault tolerance can also be achieved by using an appropriate adaptive routing algorithm. An adaptive routing algorithm provides fault tolerance by allowing use of multiple paths to the destination, thus providing opportunities for routing around faulty regions. Most adaptive algorithms provide fault tolerance to a limited extent only [55], as the requirements of deadlock prevention typically restrict the usable paths to a small subset of possible paths.

Previous approaches to fault tolerance are discussed in detail in Chapter 6.

2.6 Support for Special Operations

Collective communications operations such as broadcast, multicast, barrier synchronisation and reduction are quite likely in present day multiprocessors. Traditionally, these operations are implemented on top of point-to-point communication primitives. There has been considerable research on the problem of improving the performance of such operations [49].

Two useful hardware features for these communication tasks are multiple injection/consumption channels [9, 49] and intermediate reception capability [54]. Injection/consumption channels are the channels with which a processor can inject/consume a message into/from the network. These channels are also often referred to as *ports*. If a processor has a number of ports equal to the number of outgoing links, then it would be able to inject (or absorb) a message in all of them simultaneously. This would be very helpful in many collective communication operations. Intermediate reception is a feature that would allow a router to deliver an incoming message to the local host while simultaneously forwarding it to another router. This feature would be able to reduce the store-and-forward overhead in collective communications.

2.7 Summary

Direct interconnection networks are currently popular owing to their good scalability. Low dimensional direct networks are particularly preferred currently as they may utilise physical resources more efficiently. In direct networks, wormhole switching is the most popular switching technique since it provides low latency and has low buffer requirements. Virtual Channel Flow Control is a flow control strategy that has been proposed to avoid excessive blocking in wormhole networks. Currently, routing in wormhole switched multiprocessors is often deterministic owing in part to the requirements of deadlock prevention. Many new adaptive routing schemes have been proposed over the last few years that promise to improve performance. These adaptive routing schemes (discussed further in Chapter 3) involve increased cost and complexity and have been shown to improve performance under some conditions. Efficient adaptive routing algorithms which provide consistent improvements in performance are very essential to the success of adaptive routing. One other alternative routing strategy is source-directed routing in which knowledge about overall network traffic patterns may be employed to improve routing performance.

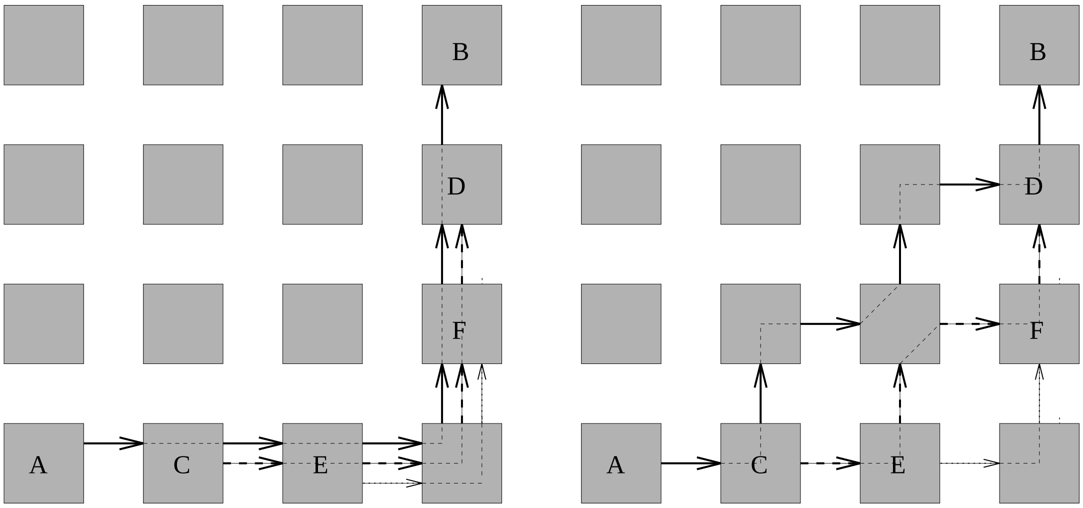
Present day multiprocessor interconnection networks have only limited support for fault tolerance, or for collective communication operations. Ongoing research in these areas has led to a number of different proposals. The fault tolerance schemes proposed in current literature are either significantly expensive or are not powerful enough to be general and efficient. The issue of powerful and cost efficient fault tolerance is examined further in Chapter 6 of this thesis. It is quite likely that one or more of the fault tolerance approaches will be implemented in commercial machines in the near future [53].

Chapter 3

Adaptive Routing

Deterministic routing is inflexible and cannot respond dynamically to network conditions. In many cases, this can lead to poor performance in terms of throughput capacity and network latency [31, 47]. When adaptive routing is employed, a message is routed at each node depending on the relative loadings of the possible outgoing channels. Figure 3.1(a) gives an example of a case in which deterministic routing results in three messages contending for the same channels even when other channels on alternative shortest paths are free. If adaptive routing is employed, for the same traffic pattern, these messages may be spread out so that they use separate physical channels, as shown in Figure 3.1(b). As suggested by this example, adaptive routing may increase throughput capacity (as links may be utilised more evenly) and decrease queueing delays. Adaptive routing may also be used to provide fault tolerance, a subject considered in Chapter 6, by facilitating routing around failed regions.

Adaptive routing algorithms differ in how much adaptivity is provided, in the manner in which deadlock is prevented, and in the manner in which selection is performed between alternate possible routes. With respect to the first of these at-



(a) (b)

Figure 3.1: An example to illustrate the benefits of adaptive routing in which A sends to B, C to D, and E to F (a) With deterministic routing the three messages contend for the same channels (b) With adaptive routing the messages are spread out

tributes, an adaptive algorithm is said to be *minimal* if the set of valid paths allowed by the algorithm includes only shortest paths from the source to the destination. An adaptive algorithm is termed *fully adaptive* if all shortest paths to the destination can be chosen. A *partially adaptive* algorithm allows only a subset of the shortest paths to the destination. In non-minimal routing, messages can temporarily move further away from the destination. Non-minimal routing increases adaptivity, and thus has some potential for providing improved fault tolerance, and, to a lesser extent, improved performance. However, non-minimal routing may create the possibility of livelocks (when routing of a message never leads it to the destination), and may significantly increase the complexity of the router.

Figure 3.2 proposes a classification of different approaches to minimal adaptive routing in networks employing wormhole switching, based on how much adaptivity

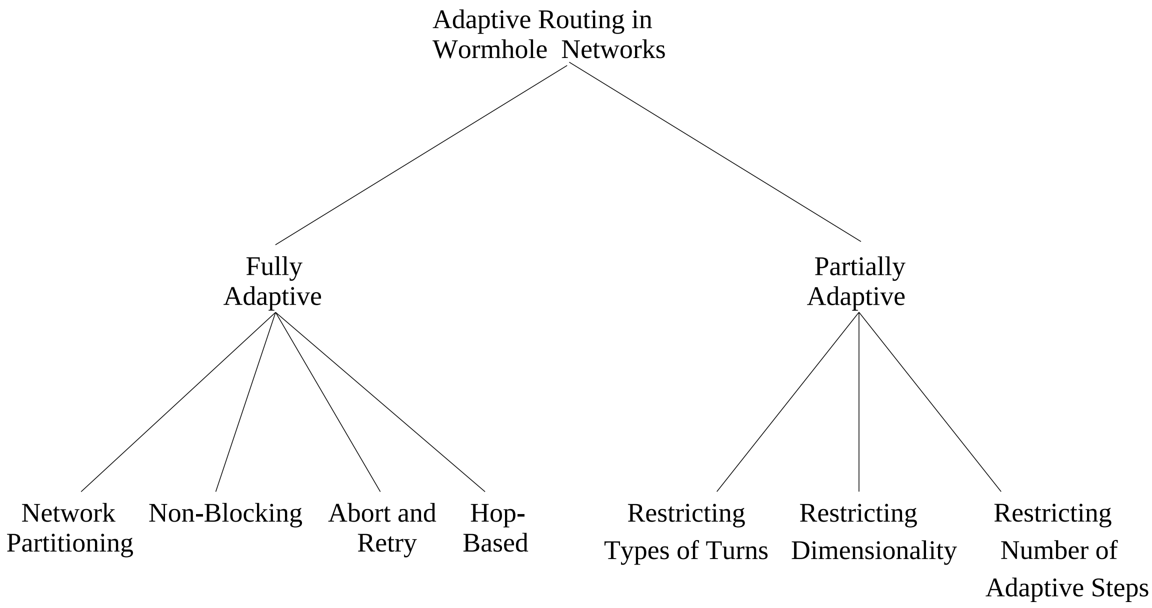


Figure 3.2: A classification of minimal adaptive algorithms

is provided and the means by which deadlock is prevented or otherwise handled. As organised by this classification structure, a variety of routing algorithms are discussed in this chapter. Section 3.1 discusses various fully adaptive schemes. Partially adaptive schemes are discussed in Section 3.2. Some adaptive routing approaches for non-wormhole networks are discussed briefly in Section 3.3. The issues concerning selection among permissible routes are similar for all adaptive routing algorithms, and are considered in Section 3.4. A brief summary of this chapter is presented in Section 3.5.

3.1 Fully Adaptive Routing

In a fully adaptive routing algorithm all shortest paths to the destination can be chosen. The term "path" here refers to a sequence of physical channels rather than virtual channels; among all of the virtual channels on the shortest path full routing freedom is not likely, owing to the requirements of deadlock prevention. This form of adaptivity can potentially provide large benefits. Fully adaptive routing can be provided by many different approaches, differing in how the deadlock problem is dealt with. Four different approaches to fully adaptive routing are discussed in this section.

3.1.1 Network Partitioning Approaches

Network partitioning approaches partition the network channels into independent virtual networks to provide deadlock-free fully adaptive routing. Deadlock is avoided because messages travelling in different directions end up in different virtual networks thereby avoiding circular waiting for resources (a necessary condition for deadlock). Two such algorithms applicable to n-dimensional torus and mesh networks are the Linder and Harden (LH) algorithm [47] and the Jesshope, Miller and Yantchev algorithm [37].

The LH algorithm for 2D meshes requires two virtual channels per physical channel in one of the two dimensions (either X or Y). Supposing that the extra channels are used in the Y dimension, then the algorithm routes all "east-bound" messages using the +X physical channels (left to right) and one set of Y virtual channels and routes all "west-bound" messages using the −X physical channels (right to left) and the other set of Y virtual channels. Figure 3.3(a) is an example of a mesh network with two virtual channels per physical channel in the Y dimension. Figure 3.3(b) shows the channels that alone form the +X network.

Jesshope, Miller and Yantchev [37] propose a fully adaptive algorithm for 2D meshes that partitions the networks into four virtual networks called (+X, +Y), (−X, +Y), (+X, −Y) and (−X, −Y). Thus this algorithm requires two virtual channels per physical channel in both X and Y dimensions. All messages travelling north-east, north-west, south-east and south-west are routed in the corresponding virtual networks. Both the Jesshope *et al.* algorithm and the LH algorithm can easily be extended to the case of k-ary n-cubes. Both the algorithms allow use of any shortest path from the source to destination; the only difference is that the Jesshope *et al.* algorithm employs more virtual channels.

The Jesshope *et al.* algorithm requires 2^{n-1} virtual channels per physical channel for an n-dimensional mesh. The LH algorithm for a n-dimensional mesh requires 2^{n-1} virtual channels per physical channel in the highest dimension, 2^{n-2} virtual channels per physical channel in the next highest dimension, and so on. Thus the number of virtual channels required for both these algorithms increases exponentially with the network dimension. The cost associated with these algorithms, in terms of requirements for virtual channels, becomes prohibitive for higher dimensional networks.

3.1.2 Non-Blocking Approaches

Non-blocking approaches are based on Duato's theory of adaptive routing [26]. This theory allows messages to take routes that are not consistent with any global virtual channel ordering, and that would thus potentially create cyclic dependencies. A condition, however, is that there must be a subset of virtual channels on which routing is restricted so as to obey some global ordering, and such that one of these channels is always permissible as a "next step" for a message at each hop. This theory allows design of adaptive routing algorithms that require only a constant number of virtual channels per physical channel.

Berman *et al.* [10] propose an algorithm for torus networks that is based on

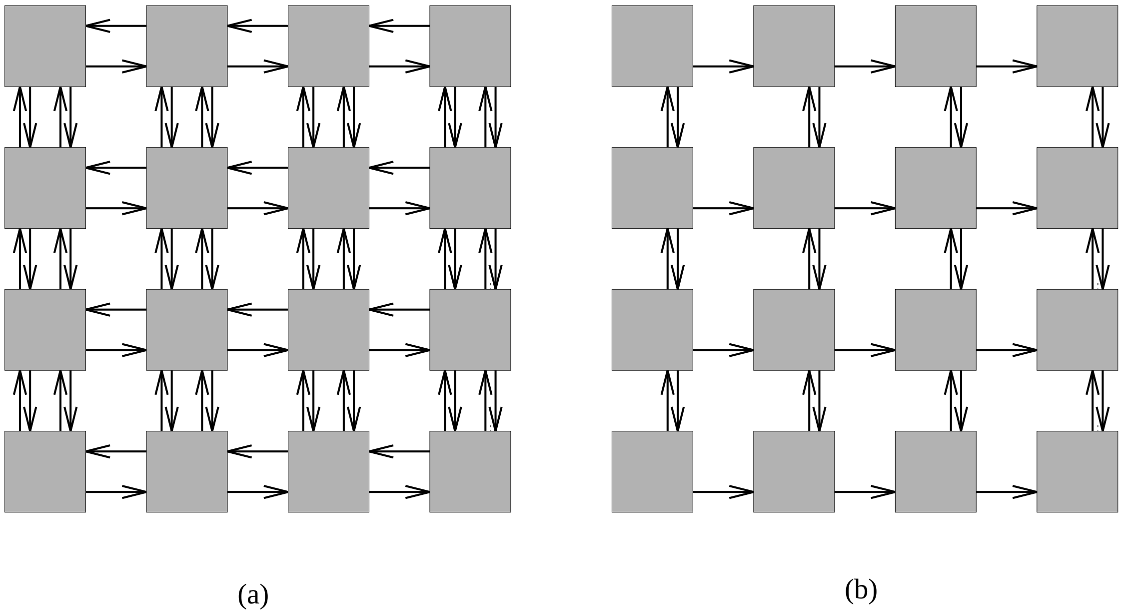


Figure 3.3: Adaptive routing based on network partitioning: (a) A double Y-Channel mesh network (b) Channels constituting the +X network

Duato's theory, termed *-Channels. The virtual channels are divided into star and non-star channels. Messages move in the star channels for dimension-order routing. The non-star channels are used to take transitions not allowed by dimension-order routing. Messages may not block waiting for a non-star channel to become free; if the relevant star channel (in the non-adaptive direction) becomes idle, it must be taken. Thus, this approach is termed a "non-blocking" approach. The *-Channels algorithm for two dimensional torus networks requires three virtual channels per bi-directional link (i.e., per pair of uni-directional physical channels) in the X dimension and five virtual channels per bi-directional link associated with the Y dimension. Similar algorithms have also been proposed by Su and Shin [62], and Schwiebert and Jayasimha [60].

The *-Channels algorithm can be easily modified to suit 2D meshes. In this context, two virtual channels are required for each physical channel in the Y dimension and one virtual channel for each physical channel in the X dimension. Of the two channels in the Y dimension, one channel is the star channel, which is to be used only for dimension-order routing, the other is the non-star channel, which can be used for both dimension-order routing and non-dimension-order routing. If additional channels are added to the topology (for VCFC), they can be classified as non-star channels as the single star channel in the Y dimension is enough to guarantee deadlock freedom. The proof that this modification for 2D meshes does not introduce deadlocks is based on the fact that the *-Channels algorithm is proven to be deadlock-free, if the channel dependency graph of the star channels is deadlock-free [10]. Since only dimension-ordered, deterministic routing (with laning) is permitted on the star channels, this condition is satisfied.

In 2D meshes, when using the *-Channels algorithm, the number of virtual channels available for use by a message on any physical channel in the X dimension is the total number N_x of virtual channels. On any physical channel in the Y dimension, the number of virtual channels available for use by a message taking an adaptive step is $N_y - 1$ (and is otherwise N_y). Although the minimal value of N_y is 2 and N_x is 1, larger values may be used for purposes of VCFC. Comparing the *-Channels algorithm to the LH algorithm under an assumption of N virtual channels per physical channel in both dimensions, note that under the LH algorithm a message has N usable virtual channels on each hop in the X dimension, but only $N/2$ in the Y dimension due to the partitioning of the network into subnetworks. This comparison is illustrated in Figure 3.4.

The success of non-blocking schemes in reducing the hardware cost, in terms of virtual channels, and at the same time providing the benefits of fully adaptive routing, has led to the implementation of a non-blocking algorithm in the MIT Reliable

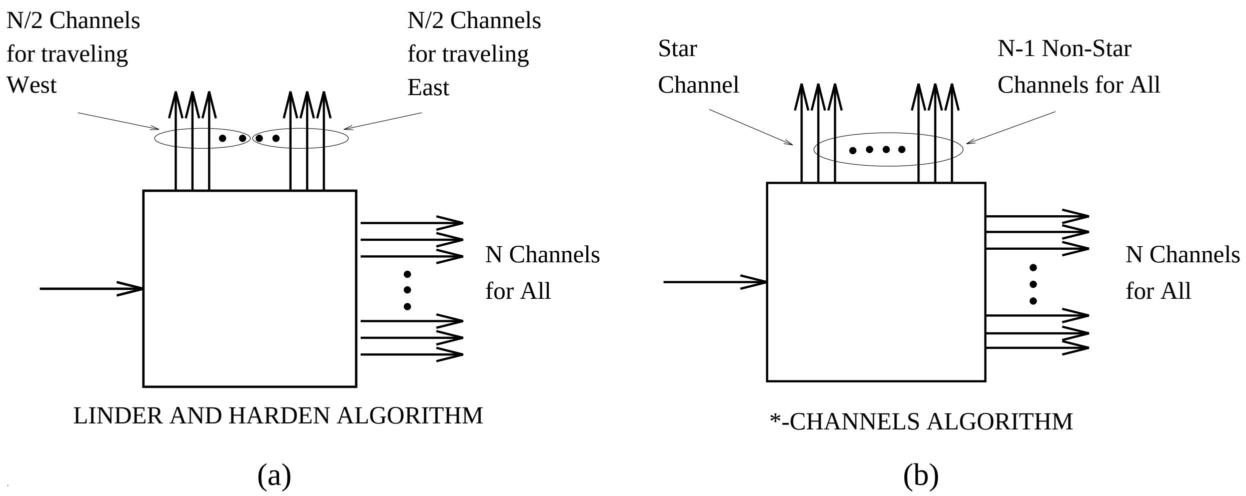


Figure 3.4: Channel choices available for the *-Channels algorithm as compared to the Linder and Harden algorithm

Router [23]. The routing algorithm used in the Reliable Router is quite similar to the *-Channels algorithm. Each physical channel has two star virtual channels, two non-star virtual channels and an extra virtual channel for fault tolerance.

3.1.3 Abort and Retry Schemes

The abort and retry schemes are based on the premise that even if routing was completely unrestricted among all virtual channels on shortest paths, occurrences of deadlock would likely be quite rare. For this reason, it may be reasonable to permit such routing, and when a deadlock is detected (or suspected) abort one or more messages to break the deadlock. Such messages may be retransmitted after a certain period of time. One recently proposed scheme of this type is Compressionless Routing [40].

Compressionless Routing uses backpressure effects from wormhole routing to communicate routing status. A message source keeps track of whether or not the message header has reached its destination. If it has reached its destination a routing deadlock involving the message is not possible. If it has not, and if the message is blocked for a "long" period of time, the source removes the message from the network (and also releases the channels and buffers that the message holds) and tries again later. This ensures deadlock freedom as any deadlock the message was part of would be broken when the message was removed.

The source determines whether or not a message header has reached its destination by counting the number of message flits it has injected into the network. Assuming that each virtual channel has just a single associated flit buffer, if the number of flits injected into the network is greater than or equal to the distance (under minimal routing) to the message destination, then the header must have reached this destination. A case requiring special attention is that in which the length of the message is smaller than the distance it has to travel, as the source cannot now determine whether the message header has reached its destination or not. Under this condition the message is padded with extra flits so that the length of the message becomes greater than the distance it needs to travel.

The advantage of an abort and retry scheme is that it needs just a single virtual channel per physical channel, for any given topology. This makes router design very simple. One disadvantage would be that for networks with large diameter, a lot of bandwidth might be wasted due to the padding flits. Also, the abort and retry scheme might result in high network latency for applications whose routing patterns have a higher probability of deadlock.

3.1.4 Hop-Based Schemes

Hop-Based schemes apply deadlock prevention ideas previously developed for wide area store-and-forward communication networks. The positive hop algorithm [35] is a deadlock prevention technique developed for such networks, in which the number of buffer classes in each node is chosen to be greater than the diameter of the network.

Boppanna and Chalasani [13] have proposed a hop-based algorithm in which each physical channel is provided with d virtual channels, where d equals the diameter of the network plus one. A message uses virtual channel i, at an intermediate node, to complete hop $i+1$. Since this enforces a strict ordering of usage of virtual channels, cyclic waiting for resources is not possible and thus deadlock is avoided. Though hop-based schemes are at least of theoretical interest, the cost associated with the implementation of such a scheme is very high for networks with appreciable diameter.

3.2 Partially Adaptive Routing

The fully adaptive routing schemes discussed above have, for the most part, significant cost associated with their deadlock prevention mechanisms. The motivation behind partially adaptive routing is to reduce the cost of fully adaptive routing and yet at the same time retain as many of its benefits as possible. Partially adaptive approaches do not permit use of all shortest paths from source to destination; instead, message routes are restricted in various ways. These restrictions significantly reduce the resources required for deadlock prevention. Three partially adaptive schemes are discussed in this section.

3.2.1 Restricting Types of Turns

The turn model concept proposed by Glass and Ni [33] allows development of partially adaptive routing algorithms for any given topology without adding any further

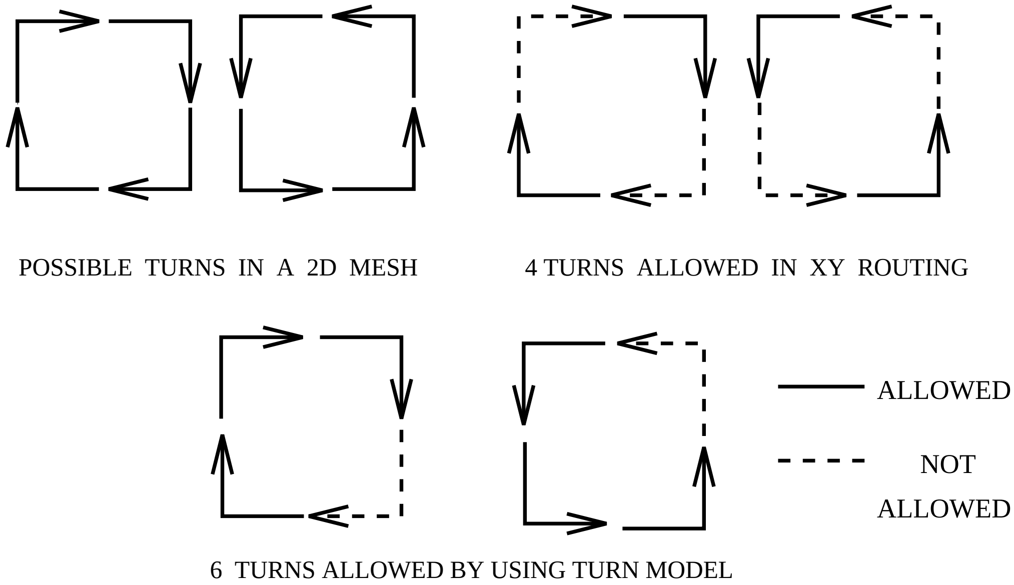


Figure 3.5: Turns allowed by different algorithms in a 2D mesh

virtual channels, by restricting the types of "turns" that a message may take as it is being routed. A turn is defined as a change in direction. The basic concept of the turn model is to prohibit the least number of types of turns necessary to make cyclic dependencies impossible. Consider the case of a 2D mesh. Here there are 8 possible types of turns a message can make as shown in Figure 3.5. XY routing allows only 4 out of these 8 turns. It does not allow a turn from the Y dimension to the X dimension. The turn model weakens this restriction and allows six possible turns. The turn model diagram shown in Figure 3.5 corresponds to the "west-first" [33] algorithm. There are eleven other choices of pairs of turns to prohibit which would also ensure deadlock prevention, although only two other choices are distinct if symmetry is taken into account. Note that although the turn model permits only a subset of all shortest paths from source to destination, it permits use of some non-shortest paths (i.e., non-minimal paths).

The turn model concept can also be applied to virtual channel topologies, such as used in the Jesshope *et al.* algorithm, and in the LH algorithm, to increase adaptivity with respect to virtual channel selection. Although both these algorithms permit minimal fully adaptive routing in terms of the physical topology, routing on the virtual topology is still very constrained. These algorithms prohibit more turns on the virtual topology than are necessary to prevent deadlocks. Since these algorithms are already fully adaptive, the application of the turn model results in a "maximal" fully adaptive algorithm [32]. The turn model might prove to be a good approach to increase the adaptivity of existing algorithms.

The direction restriction model [15] also provides partial adaptivity without requiring any extra virtual channels, again essentially by restricting turns. This model is based on dividing the routing process into two phases. In the first phase, dimensions may be traversed in specified directions (positive or negative). In the second phase dimensions may only be traversed in the opposite directions of the first phase. During each phase any dimension may be chosen for routing at each step, as long as movement in that dimension would permit eventual delivery to the destination under the prescribed routing restrictions. It should be noted that these restrictions in routing could also be derived by the use of the turn model concept. Using this scheme for n-dimensional mesh networks, 2^n adaptive routing schemes can be developed, all of which are deadlock free. The direction restriction model appears to be quite simple in implementation and could provide good performance.

One disadvantage of both the turn model and the direction restriction approaches is that the degree of adaptiveness relative to fully adaptive algorithms decreases as the dimensionality increases. Also, the inherent non-symmetries in the permitted routes may yield poor performance for some traffic patterns. In general, a specific algorithm based on using one of these approaches is better for certain traffic patterns but bad for others. Thus, to achieve uniform gains the routers would have to be programmable in such a way that depending on the application characteristics a particular variation of the model could be chosen.

3.2.2 Restricting Dimensionality

Chien and Kim [19] propose a partially adaptive algorithm called the planar adaptive algorithm, which routes messages adaptively along two dimensions (a plane) at a time. Within each routing plane, messages may use any channel lying on a shortest path to the destination.

The virtual channel requirements of planar adaptive routing are independent of network size and dimension. For an n-dimensional mesh, the requirements are 3 virtual channels per physical channel (although 2 channels would be enough for the 2D mesh) and for a k-ary n-cube the requirements are 6 virtual channels per physical channel in all dimensions. The hardware requirements are thus reduced dramatically in comparison to those required for the LH algorithm owing to the reduction in routing freedom. Note that the number of required virtual channels is not lower than in *-Channels (a fully adaptive algorithm) However, planar adaptive routing can also be extended so as to allow deadlock-free and livelock-free non-minimal routing, as may be required in the presence of faults.

3.2.3 Restricting Number of Adaptive Steps

Dally and Aoki [22] propose a non-minimal algorithm for k-ary n-cubes which has a small requirement in terms of virtual channels. Dally and Aoki define "dimension reversal" as a departure from the dimension order deterministic routing (that is, from a higher ordered dimension to a lower ordered dimension). Their static dimension reversal algorithm needs r virtual channels for each physical channel (in all dimensions), if $r-1$ is the maximum number of dimension reversals (adaptive steps) to be permitted. Each message has a header field that records the number of dimension reversals that it has undergone. If this value is i ($i < r$) then the message can be routed using the ith virtual channel on the desired link. Once the header value equals $r-1$ (i.e., the message has undergone $r-1$ dimension reversals), the message has to be routed deterministically using the rth virtual channel on each link. The parameter r controls the adaptivity of the algorithm.

Dally and Aoki also propose a dynamic dimension reversal algorithm. Here the virtual channels are partitioned into two non-empty classes: adaptive and non-adaptive. Messages in adaptive channels and can suffer any number of dimension reversals until they get blocked by a message that has undergone fewer dimension reversals. Once this happens, the message has to be routed deterministically by using the nonadaptive channels.

The main problem with these approaches is their high complexity. The routing decision at each node becomes complicated due to the necessity for checking the number of dimensional reversals while misrouting. Also, there is a necessity for the message to carry information, in its header, regarding how many dimension reversals it has undergone.

3.3 Some Approaches for Non-Wormhole Networks

Although wormhole switching is currently the most popular switching technology for multiprocessors [53], there has been much research concerning routers based on other switching schemes. Two projects of this kind are the Chaos router [12, 41] and the Ariadne router [5, 29] projects. Both these routers are now in their prototype stage of development.

3.3.1 Chaos Router

The Chaos Router is a non-minimal router that builds on the basic concept of deflection routing [27]. In deflection routing, at each step an incoming packet is paired with its desired outgoing channel and transmitted in the next step. If packets contend for the same outgoing link, all but one are misrouted into some other available link, chosen at random, without being buffered at the node. Livelocks are avoided by employing some priority-based scheme.

The Chaos router uses a virtual cut-through switching scheme. It is a queueing non-minimal router and it differs from deflection routing by having a central buffer at each switch that holds packets awaiting free outgoing links. It relies on a packet-exchange protocol for avoiding deadlock [12]. This protocol ensures that each bi-directional link in the network will not deadlock. The problem of livelock is avoided by randomly choosing a packet to misroute from the queued-up packets. This ensures that given a large period of time any packet would ultimately reach its destination. Thus, the problem of livelock is overcome in a probabilistic sense. Since the Chaos router uses virtual cut-through, network latency is reduced by a large extent if contention is light.

Although randomisation can avoid livelocks, it can result in long delays for certain messages. This situation could become aggravated in the presence of faults.

3.3.2 Ariadne Router

The Ariadne router is based on a variant of circuit switching termed pipelined circuit switching [5]. The header is routed to the destination reserving virtual channels along its path. When the header finally reaches the destination, an acknowledgement flit is returned by the destination to the source node. Once this happens the data flits are sent on the reserved path in a pipelined fashion as in wormhole switching.

The header flit is allowed to backtrack (and release the previously reserved channels) at any point before it reaches its destination, and then to try alternate paths. This allows increased flexibility in setting up a favourable route with the least amount of conflicts/contention. This also provides increased fault tolerance and the ability to tolerate large sets of faults. Deadlock avoidance is also simplified.

The trade-off for these benefits is that setup times can become large, when contention is high. This overhead can be particularly significant when the amount of data to be transmitted is small.

3.4 Path Selection Policies

Adaptive routing offers messages multiple channel choices at intermediate nodes. With adaptive routing multiple messages may also contend for the same choice of channel at any node. Thus an "input selection" policy and an "output selection" policy are necessary to resolve these contentions. The input selection policy [33] chooses a particular message when the header flits of multiple messages contend for the same outgoing channel. The output selection policy chooses a particular channel when the header flit of a message has to choose between multiple eligible outgoing channels. Both these policies together constitute the path selection policy.

One simple input selection policy is the "random select" policy, which selects an input randomly from all contending inputs. Priority-based input selection polices which base their decision on factors such as "the distance already travelled" [33] by the message or global timestamps can provide some improvements in performance, but, at the same time can involve significant cost. Also, in wormhole networks, since messages are likely to include many tens or hundreds of flits, the probability of input contention is relatively small. Thus the input selection policy is not likely to have a large impact on performance (as found later in the experimental study presented in Chapter 4), and simple policies are likely preferable.

The output selection policy, on the other hand, has a large impact on performance. One good output selection policy is the "minimum congestion" policy [21]. The minimum congestion policy picks that physical channel among the eligible outgoing channels that has the most free virtual channels. Another possible output selection policy, proposed in [8], chooses the dimension in which the message has to travel the farthest. This policy is not very well suited for wormhole networks as messages tend to take "zig-zag" paths into the network and can thus end up blocking a large number of other messages. One other possible output selection policy is the 'no turn' policy, where the output choice is the channel in the same dimension of travel. Such a policy may reduce the "zig-zag" nature of paths and reduce blocking.

3.5 Summary

In this chapter a taxonomy for adaptive routing algorithms was presented and different approaches to adaptive routing were discussed. From the performance point of view, a fully adaptive algorithm might be desirable, since partially adaptive algorithms may not provide a consistent improvement in performance. Among the different approaches towards fully adaptive routing, the approaches based on Duato's theory are more promising due to their reduced cost in terms of virtual channels. In the context of low dimensional networks, the network partitioning schemes might also be quite attractive as their costs become prohibitive only for higher dimensionalities.

Chapter 4

A Performance Study of Adaptive Routing

The focus of this chapter is on a simulation study of adaptive routing performed on a 2D mesh network. The primary motivation for such a study is discussed in Section 4.1. The details of the experimental methodology followed in this study are discussed in Section 4.2. Section 4.3 discusses the results of the experimental study. Conclusions are presented in Section 4.4.

4.1 Motivation

Two concepts that have been proposed to improve routing performance in wormhole networks are virtual channel flow control (VCFC) and adaptive routing. Some previous studies have, however, shown that adaptive routing algorithms perform poorly for uniform traffic patterns [31, 33]. Some researchers [13] also claim that adaptivity, full or partial, is not necessarily a benefit in wormhole networks, and they show that certain adaptive algorithms can perform poorly for both uniform and non-uniform traffic patterns. One of the reasons for this poor performance is because adaptive algorithms tend to route messages towards the center of the network which can potentially increase blocking. These results raise a certain amount of doubt as to whether adaptive algorithms should actually be implemented in commercial machines.

VCFC has been shown to improve throughput capacities for uniform traffic, by reducing blocking in the network [21]. In this study, the performance of VCFC for certain non-uniform traffic patterns was investigated. One of the results of this study is illustrated in Figure 4.1. This figure shows the latency versus throughput curve (see Section 4.2) for a 12×12 mesh, for the deterministic XY algorithm, with varying numbers of virtual channels which are exclusively used for VCFC. The traffic pattern considered was a "matrix transpose pattern" in which a node in column i and row j sends messages to the node in column j and row i. It was also assumed that the node in column i and row i (i.e., on the diagonal) sends messages only to the node in column $n-i+1$ and row $n-i+1$. Note that as the number of virtual channels is increased, the average latency values for a given throughput can increase drastically. This occurs due to the increase in the number of messages that could get multiplexed on each physical channel. This leads to the conclusion that adding more virtual channels for VCFC alone can degrade performance for certain non-uniform traffic patterns.

Thus, adaptive routing by itself performs poorly compared to deterministic routing under certain circumstances. Also, employing VCFC alone can degrade performance for certain traffic patterns. The experimental study discussed in this chapter, investigates whether the problems of both these schemes can be mitigated by judiciously combining them. Therefore, the amount of VCFC employed (by adding virtual channels) is varied and its effect on adaptive routing performance is studied.

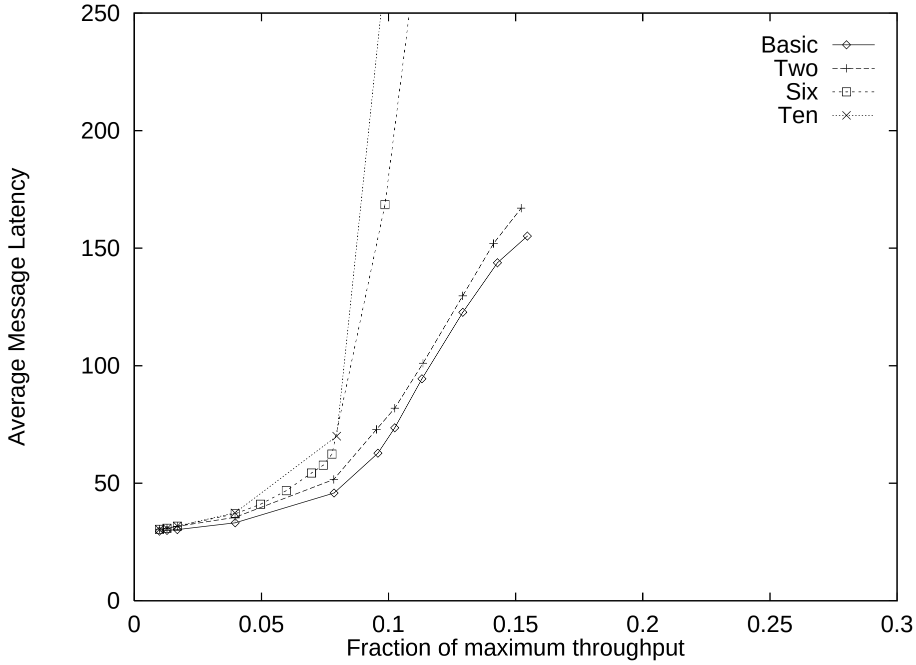


Figure 4.1: Impact of adding virtual channels to the topology exclusively for the purpose of VCFC, for matrix transpose traffic and employing the deterministic XY routing algorithm

4.2 Simulation Methodology

Among the fully adaptive algorithms discussed in the last chapter, the hardware requirements and the algorithm complexity for the Linder and Harden (LH) and *-Channels algorithms were comparatively low for the case of a 2-dimensional mesh network. These algorithms are quite simple to implement and the router complexity is not likely to be very high, in this context. Also, these two algorithms require (at the minimum) only two virtual channels per physical channel in one dimension and one virtual channel per physical channel in the other. The performance of the *-Channels algorithm is generally superior to other known adaptive algorithms [10, 23]. Therefore, the LH algorithm and the *-Channels algorithm are chosen as the two representative adaptive algorithms for the performance study in this chapter (and in the rest of the thesis).

To evaluate the performance of the adaptive algorithms a discrete event network simulator was developed. The simulation was performed at the flit level. The simulator has the flexibility to support various topologies, routing algorithms, selection policies, traffic models, and traffic patterns.

Architectural Details

A 2-dimensional mesh network was considered as the base network in all performance studies. The size of the mesh was set to be 12×12. Experiments were also performed for other mesh sizes, but the results of these experiments did not differ qualitatively from the results of the 12×12 mesh. Each pair of neighbouring nodes in the mesh was assumed to be connected by a pair of (uni-directional) physical channels. The number of virtual channels per physical channel was varied during the experiments. Each router was assumed to be connected to the processor by means of a single consumption channel and a single injection channel, each with the same number of virtual channels as the other physical channels in the network. The flit transfer time between two nodes was set to be 1 clock cycle.

Path Selection and Transmission Policies

As discussed in Chapter 3, the two constituents of the path selection policy are the input selection policy and the output selection policy. For most of the simulations, the input selection policy was chosen as "random select" and the output selection policy was set as "minimum congestion" [22]. With the minimum congestion policy, in case of a tie, it was assumed that the channel in the X dimension is chosen. It is assumed here that the output selection policy is applied before the input selection policy. The physical channel transmission policy, which is employed for multiplexing virtual channels over each physical channel, was set to be "work conserving round robin" among the flits waiting to be transferred. It was found that this policy has better performance compared to other policies such as "oldest flit first", and it should also have a low implementation cost. The impact of some alternative policy choices is examined in Section 4.3.4.

Workloads

The primary workload model considered in this study is an "open" workload model where each processor generates messages at a rate α whenever any virtual channel on its single injection channel is free. The open model has been used in many research studies [9, 12, 13, 26, 33, 40, 39] as it provides an effective means of testing the network performance under a variety of load conditions. In most of the experiments the size of each message was set to be 20 flits because the running time of simulations (for achieving the desired confidence levels) becomes very large with increasing message size. In any case, it was found that increasing the message size to 128 flits, or having two messages sizes (5 and 35) with average still equalling 20, does not affect the qualitative nature of the results. The effect of these alternate message sizes on the results is examined in Section 4.3.4.

The three main traffic patterns that were considered were "uniform" traffic, "matrix transpose" traffic and "hot spot" traffic. With the uniform traffic pattern, a node sends each message to any other node with equal probability. Since the traffic in multiprocessor systems often has some hot spots [56], a mild hot spot traffic pattern was also simulated. Six random nodes were fixed as hot spot nodes and each received three times the amount of traffic as did each of the other nodes. The matrix transpose traffic described earlier was selected as a representative of highly non-uniform traffic.

Performance Metrics

The two metrics that were of interest were the normalised network throughput, T, and the average message latency, l. The normalised network throughput is defined as

$$T = N * L/n^2$$

where N, L and n^2 are the number of messages delivered per unit time, the message length in flits, and the number of network nodes (considering a $n \times n$ mesh), respectively. Since at most $n^2/\ L$ messages can be delivered per unit time (given that each node can consume at most one flit in a clock cycle), $N * L/n^2$ gives the network throughput as a fraction of the maximum possible throughput. (Tighter bounds on throughput for particular traffic patterns are considered in the next chapter.)

Latency was measured by calculating the difference between the time when a message was created and the time the last flit of the message reached the destination. Average latency (measured in clock cycles) was computed by averaging over all messages between all source-destination pairs. A limited number of experiments were also performed with a "single-phase" communication workload model, where completion time of a phase is dependent on the maximum of message latencies. Completion time workloads are considered further in Chapter 5 of this thesis.

4.3 Experimental Results

Simulation experiments were performed for the deterministic XY algorithm and the LH algorithm and *-Channels adaptive routing algorithms, with varying numbers of virtual channels per physical channel. First, the minimal configuration (for the adaptive algorithms) was simulated, where there are two virtual channels for each physical channel in the Y dimension and one for each physical channel in the X dimension. Then, the topology was extended to have 2, 4, 6 and 10 virtual channels in both the X and Y dimensions. For all these cases the simulations were run mainly for the three traffic patterns described in the previous section. When more virtual channels were added to the topology, VCFC was employed (i.e., allowing routing choice among the various "lanes" on a single physical channel) to the maximum extent possible while still guaranteeing deadlock avoidance. The objective was to evaluate the impact of VCFC and how it would affect the performance of adaptive algorithms.

All average latency values were obtained with a confidence interval width of 3%, at a 95% confidence level. Since message latencies are likely to be correlated, batch means were taken (with 1000 messages forming a single batch). Results from the first 5000 clock cycles were discarded in each simulation, so as to remove transient start-up effects.

4.3.1 Performance Under Uniform Traffic

When there are two virtual channels for each physical channel in the Y dimension and one virtual channel for each physical channel in the X dimension, the XY algorithm performs better than the adaptive algorithms at higher loads, while the adaptive algorithms perform slightly better at lower loads. (See Figure 4.2.) This is because, at lower loads, adaptive algorithms provide more paths to the destination and can avoid unnecessary blocking which can occur in deterministic algorithms. At higher loads, when routing is adaptive, many messages take zig-zag routes within the network and block many other messages inside the network. For example, if a message travelling in the Y dimension makes a turn onto a channel in the X dimension, then it blocks all messages that want to travel in the X dimension on the same link. Thus, the greater the number of zig-zag turns, the greater the potential for blocking in the network. This effect has also been observed in previous studies [22, 31].

When the number of virtual channels per physical channel is increased to six, and ten, it can be seen that the adaptive algorithms perform better than the XY algorithm. (See Figures 4.3, 4.4 and 4.5.) This improvement in performance occurs because the additional virtual channels allows messages to bypass a blocked message. VCFC alleviates the problem of blocking inside the network, which was exacerbated by adaptive algorithms. From Figures 4.2 and 4.3, it can be seen that there is not much improvement in performance with the LH algorithm when the number of virtual channels is increased from one to two for each physical channel in the X dimension. This is because the channels in the Y dimension become the bottleneck, since only one virtual channel of the two allocated to each physical channel in this dimension is available to each message (effectively), due to the partitioning of the network into two subnetworks.

Since the zig-zag paths resulting from adaptive algorithms result in higher blocking inside the network for high loads, a drop in throughput can occur as the generation rate α increases, at very high loads and latencies. (See Figures 4.4 and 4.5.) Since adding virtual channels to the network avoids this blocking to a great extent, the drop in throughput also decreases as the number of channels increases.

Except in the cases of very high loads, the *-Channels algorithm performs better than the LH algorithm because it offers more choices with respect to selection of virtual channels at every node, and loads the network more evenly. However, the improvement in performance obtained by adding more lanes and employing VCFC is independent of the algorithm, and good improvements are observed for both adaptive algorithms. When used with VCFC, for uniform traffic both adaptive algorithms yield performance that is better than that provided by deterministic routing.

4.3.2 Performance Under Hot Spot Traffic

For the minimal topology, with two virtual channels per physical channel in the Y dimension and one per physical channel in the X dimension, there is not much difference between the performance observed under hot spot traffic, and that observed under uniform traffic. (Compare Figures 4.6 and 4.2.) This is due to a bottleneck effect resulting from the lack of additional virtual channels in the X dimension. This effect appears to be stronger than the effect resulting from the traffic becoming mildly non-uniform.

When the number of virtual channels per physical channel is equal to two in each dimension, the performance of all three algorithms under the mild hot spot traffic pattern is slightly worse than that obtained with uniform traffic. (Compare Figures 4.3 and 4.7.) This relative drop in performance increases when the number of channels is four. An explanation for this is that the effect of a lack of virtual channels is no longer a bigger limiting factor than the congestion caused by the hot spot nodes. The relative drop in performance under mild hot spot traffic (in comparison to that under uniform traffic) is substantially less for the adaptive algorithms as compared to the deterministic XY algorithm. This is clearly visible when the number of virtual channels per physical channel is six. (See Figure 4.8.) This is because the adaptive algorithms are much better in spreading the traffic and thus avoiding congestion.

When the number of virtual channels per physical channel is ten (Figure 4.9) the *-Channels algorithm gives the same performance for the non-uniform traffic pattern as it gave for the uniform traffic pattern (e.g., a maximum throughput of about 0.3). This is because the number of virtual channels is sufficient for holding the non-uniform traffic without clogging all virtual channels and thus adaptive routing can be effectively employed to route traffic not destined for the hot spot nodes around links that would otherwise become congested. This is not possible with the XY routing algorithm (which employs VCFC but not adaptive routing).

From the above experiments, it can seen that, if traffic is mildly non-uniform and if the network does not have sufficient virtual channels, deterministic algorithms can do better than adaptive algorithms. This effect disappears once more virtual channels are added to the network. Also, good adaptive algorithms, when used in combination with VCFC, can provide nearly the same performance for mildly non-uniform traffic as for uniform traffic. Thus, overall, the combination of VCFC and adaptive routing appears to be very beneficial.

4.3.3 Performance Under Matrix Transpose Traffic

Since the non-uniformity in the traffic is very high in the case of matrix transpose traffic, the (deterministic) XY algorithm performs poorly compared to the adaptive algorithms even for the minimal topology with two virtual channels per physical channel in the Y dimension and one per physical channel in the X dimension. (See Figure 4.10.) From Figures 4.11, 4.12 and 4.13, it can be seen that adaptive algorithms perform much better than the XY algorithm when the number of channels is increased.

In Figure 4.1, it was shown that the performance of the XY algorithm actually degrades as the number of virtual channels is increased. Figures 4.14 and 4.15 show the impact of adding virtual channels for the two adaptive algorithms separately. Adding virtual channels and performing VCFC is seen to have a favourable effect on the performance provided by the *-Channels algorithm. This difference in behaviour between the *-Channels and the XY algorithm can be explained as follows. Although VCFC allows messages to bypass blocked messages, the message multiplexing that it allows also may dramatically increase the latency of individual messages. Since matrix transpose traffic is highly non-uniform, the load (and hence multiplexing) on certain channels is very high. As the number of virtual channels is increased the load on these specific channels becomes much higher. Since the XY algorithm does not change this unevenness in load, increases in network latency are experienced. On the other hand the *-Channels algorithm spreads the traffic more evenly and thus reduces high multiplexing on any specific channel. Therefore, the negative impact of adding virtual channels is reduced. At the same time, the extra virtual channels help in avoiding blocking inside the network and thus improve the performance. (See Figure 4.14.) When the number of virtual channels is equal to ten, there is a small drop in performance for the *-Channels algorithm. At this point, evidently, the negative impacts of multiplexing outweigh the positive effects of avoiding blocking.

When the number of virtual channels per physical channel is two in both X and Y dimensions there is a drop in performance for the LH algorithm. This is because the negative effect of multiplexing is quite strong due to the extra virtual channel being added to each physical channel in the X dimension, and at the same time no positive effects exist as only a single virtual channel is available to each message in the Y dimension. But when the number of virtual channels is six, there is a slight improvement in performance at low loads. The performance of the LH algorithm is not as good because it does not balance the load on outgoing physical channels at any node, as uniformly as the *-Channels algorithm does and thus the negative effects of adding lanes in this traffic pattern are more pronounced.

In conclusion, adding lanes (and performing VCFC) in the context of a deterministic algorithm can severely degrade performance for certain traffic patterns. This degradation in performance can be avoided by employing certain adaptive routing algorithms (e.g. *-Channels) that spread the traffic more evenly and provide lower multiplexing.

4.3.4 Impact of Other System, Policy, and Workload Assumptions

Effects of some alternative system, policy and workload assumptions on the results obtained so far are discussed below. All these experiments were performed using the *-Channels algorithm as the representative adaptive algorithm.

Two variations of output selection policies were considered. In the first variation, the output selection policy chooses the dimension in which the message has to travel the farthest [8]. Due to the zig-zag nature of this output selection policy, the resulting performance was poorer compared to the minimum congestion policy used previously. The other variation was to choose the dimension which has the least load.[1] There was a slight improvement in performance, in some cases, by using this policy. In general, however, the trends of the results obtained were not altered when the output selection policy was changed.

Two different physical channel transmission policies were tried. One was the "oldest flit first" policy, and the other was the "switch when necessary" policy.

[1]This is different from the minimum congestion policy since here the choice is based on the number of active virtual channels (both star and non-star channels) in each dimension.

In the "oldest flit first" policy, the flit belonging to the message that has spent the longest time in the system, among the unblocked messages requiring use of a physical channel, is transmitted first. The "switch when necessary" policy transmits the same message as long as possible, before it encounters a gap in the message,[2] and then switches to the next message in a round robin fashion. The basic idea of this policy is to minimise the occurrence of gaps in messages. Figures 4.16 and 4.17 show the performance obtained using these policies for the *-Channels algorithm for two different traffic patterns. From these graphs notice that at lower loads both the "oldest flit first" and "switch when necessary" policies perform slightly better than the round robin policy. This could be because these two policies tend to increase the probability that successive flits transmitted over a channel are from the same message. This can reduce the average multiplexing factor and thus improve mean latency. At high loads, however, round robin does better. For the XY deterministic algorithm (which employs just VCFC), the behaviour is quite similar as seen in Figures 4.18 and 4.19. The relative performances of the XY algorithm and the *-Channels algorithm were not affected considerably.

In general, message sizes can vary by a large amount depending on the system and application under consideration. In all the experiments reported previously in this chapter, the message size was fixed at 20 flits to keep simulation running times at manageable levels. To understand the impact of message sizes on the results, simulations were also performed with larger messages. The message sizes considered in this study (20 and 128) were believed to be representative of message sizes in current multiprocessors. Smaller messages sizes (say less than 5 flits long),

[2] Gaps may occur between successive flits in a message due to blocking and multiplexing in the network.

though unrealistic, may yield different results. Figures 4.20 and 4.21 show results from experiments with a message size of 128 flits, 6 virtual channels per physical channel, and two different traffic patterns. Comparing these figures with Figures 4.12 and 4.4, the relative performances of the XY algorithm and the *-Channels algorithm are seen to be largely unaffected by the change in message size. (The Y axis of Figures 4.20 and 4.21 is scaled 6.2 times to accommodate the increase in message size by the same amount).

Experiments were also performed in which the size of each message was either 35 or 5 flits, with equal probability. Note that the average size of a message is 20 flits, matching the size used in previous experiments. In this context the "round robin" policy does better than "oldest flit first" for both uniform and matrix transpose traffic as shown in Figures 4.22 - 4.25. This is because the "round robin" policy does not make small messages (flit size = 5) wait behind larger ones (flit size = 35). Again, the relative performance of the XY algorithm and the *-Channels algorithm was not affected much by this change in the workload.

Experiments were performed with a variant of the Linder and Harden algorithm in which the network is assumed to have twice the number of virtual channels in the Y dimension as in the X dimension. The motivation behind such a scheme is to equalise the number of virtual channels in each dimension of each of the subnetworks employed by the algorithm. Figure 4.26 compares the performance with 8 virtual channels per physical channel in the Y dimension and 4 virtual channels per physical channel in the X dimension ("LH-Unequal") to that with 6 virtual channels per physical channel in both X and Y dimensions ("LH-Equal") under uniform traffic. The former yields marginally better performance only under light load.

Experiments were performed for other traffic patterns, including the center reflection pattern. The center reflection pattern is an artificial pattern which acts as a good stress test on the bisection bandwidth of a network. In the center reflection pattern, each node (i, j) in a $n \times n$ mesh sends a message to node $(n-1-i, n-1-j)$ (assuming the nodes are indexed from (0,0) to $(n-1, n-1)$). This pattern is quite notable, due to the fact that adaptive routing performs relatively poorly when the number of channels is minimal [33]. This is due to the fact that all messages have to cross the links in the central bisection of the network and adaptive routing pushes messages more to the center, thus crowding the links closer to the center, and using the links near the network edges poorly. Whereas, XY routing, which corrects the X coordinate of a message first, results in a very good load balance on the bisection X links. With additional virtual channels and the use of VCFC, the performance of adaptive routing improves considerably, as links become more evenly utilised. The performance of the *-Channels algorithm is compared with that of the XY algorithm in Figures 4.27 and 4.28, with center reflection traffic, for cases where each physical channel has two and six virtual channels respectively.

Finally, an alternate workload model was used instead of the “open” workload model. The alternate model that was considered is a “single-phase” communication model, which captures a single phase of communication in the network in which every node (or a subset of nodes) has a message to send to some other node. The performance metric of interest with this model is the time until all these messages are received, assuming that the transmission is initiated simultaneously, thus completing the communication phase. The completion time is determined by the maximum of message latencies in contrast with the open workload model where performance is evaluated according to the average message latency.

With the completion time workload model the performance improvement of adaptive routing over deterministic routing was more modest. Figure 4.29 shows the performance of the *-Channels adaptive algorithm and the XY algorithm for general traffic patterns and Figure 4.30 shows performance for a matrix transpose pattern. The results for general traffic patterns are averages over a large number of patterns in which each node generates a single message that is destined to any other node in the mesh with equal probability. As the figures show, addition of virtual channels and employing VCFC on these workloads typically improves performance, as with the open workload model. However, the magnitude of the improvements to a large amount depend on the traffic pattern. Note that VCFC mainly improves throughput capacity under the open workload model. The completion time metric reflects both throughput capacity and messages latencies, so the gains investigated here owing to VCFC differ in this context.

In summary, the alternate system and policy assumptions investigated here do not appear to change the qualitative nature of the results.

4.4 Conclusions

In this chapter the results of an experimental study of adaptive routing was reported. The amount of VCFC was varied, in the context of two different adaptive algorithms, and its effect on performance was studied.

This study has confirmed that employing adaptive routing with a minimal channel topology (no extra virtual channels) can result in poor performance for uniform and some non-uniform traffic patterns. Furthermore, using VCFC alone (using virtual channels to provide extra lanes, but not also to allow adaptivity in routing) can severely degrade performance for certain traffic patterns. However, when VCFC is combined with adaptivity in routing, these effects can be mitigated and good performance can result. There are complex interactions between VCFC and adaptivity. Adaptive algorithms perform well once VCFC is employed and VCFC gives good benefits with adaptive algorithms. The main conclusion of this study is that both adaptive routing and VCFC an indeed be beneficial but their real benefit is observed when they are employed together.

This study also shows that a particular number of virtual channels per physical channel are needed (approximately six for the 2D networks in this study) for good performance. Investing more channels beyond this returns only diminishing benefits and can even hurt performance due to effects of traffic multiplexing. Thus the choice of the number of virtual channels in the network should be made relatively carefully. The algorithms that were studied do not appear to have high overheads and the router complexity for their implementations is also likely to be low. This leads to the conjecture that adaptivity can be provided at a reasonable cost and would be beneficial to employ.

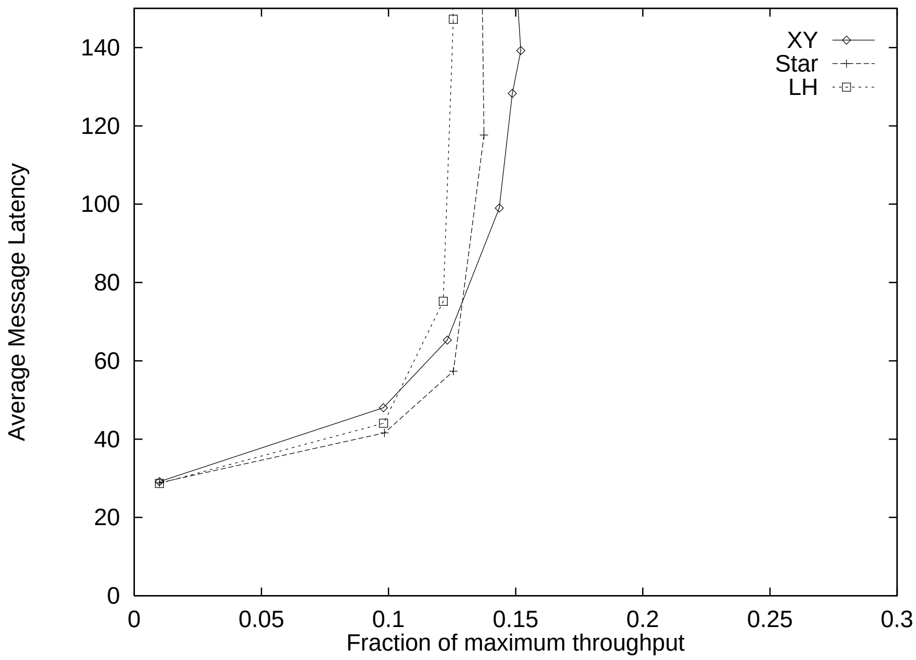


Figure 4.2: Comparison of routing algorithms for uniform traffic with 2 virtual channels per physical channel in the Y dimension and 1 in the X dimension

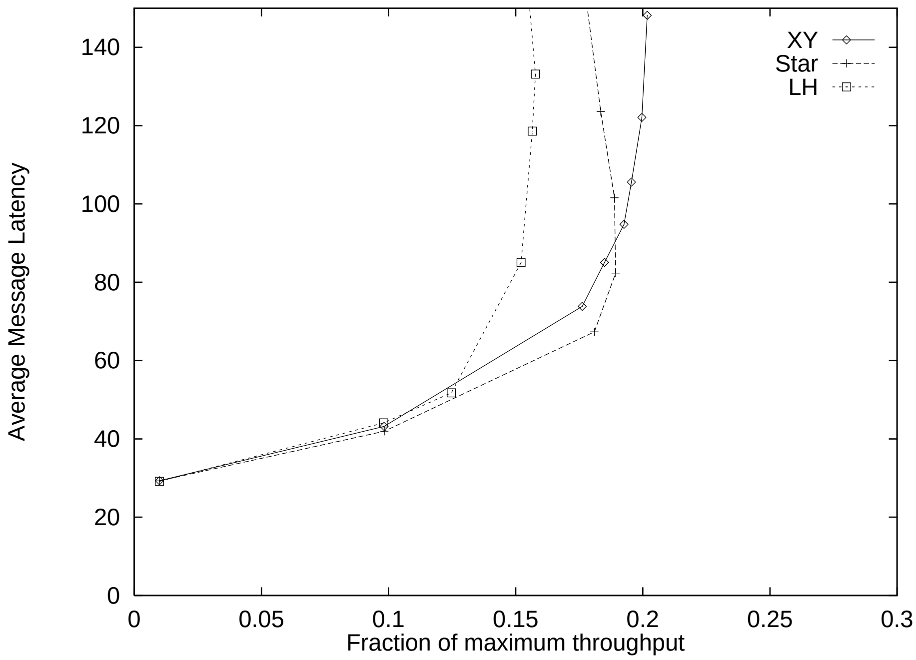


Figure 4.3: Comparison of routing algorithms for uniform traffic with 2 virtual channels per physical channel in the X and the Y dimensions

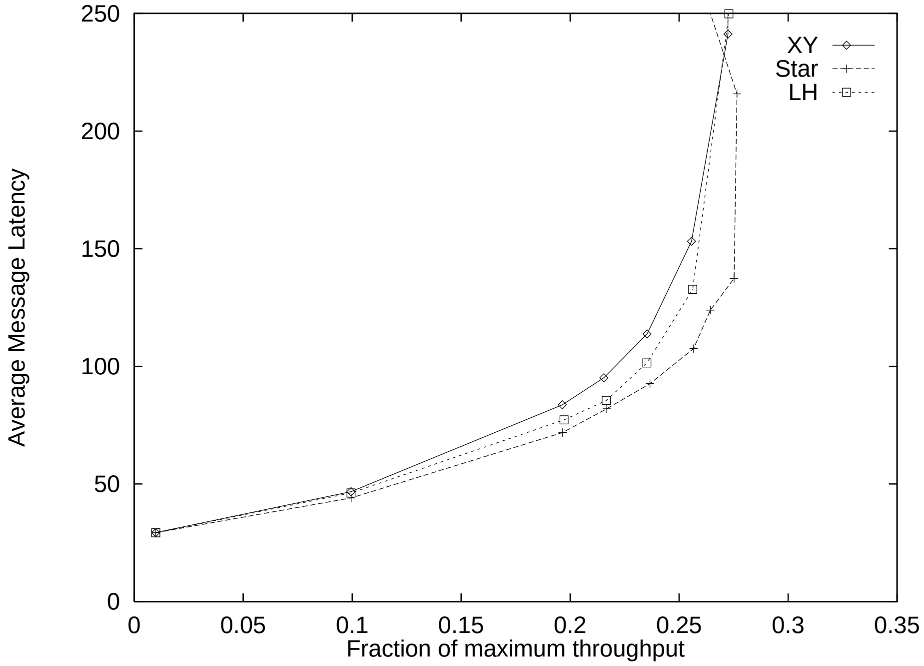


Figure 4.4: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel in the X and the Y dimensions

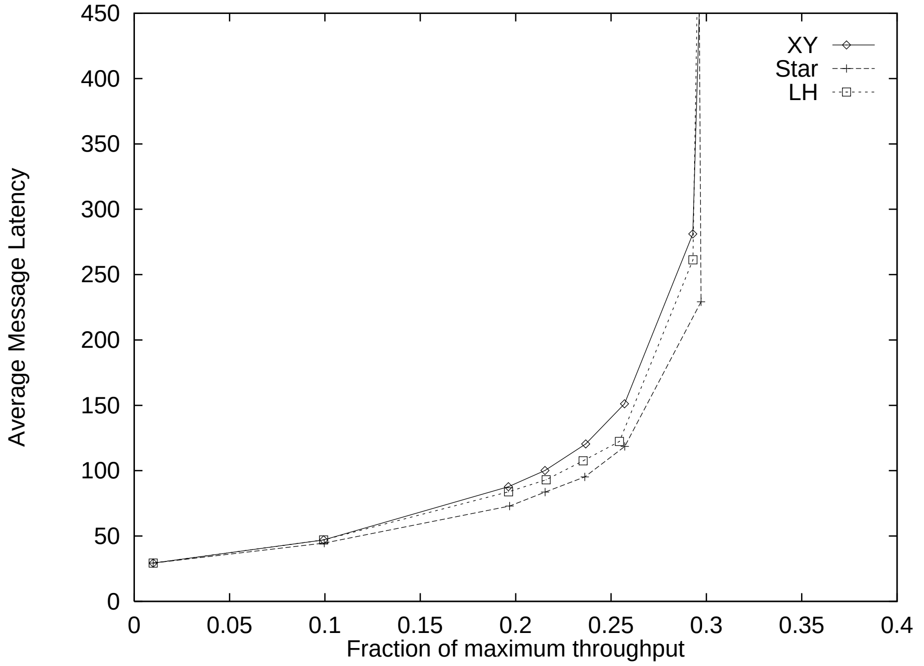


Figure 4.5: Comparison of routing algorithms for uniform traffic with 10 virtual channels per physical channel in the X and the Y dimensions

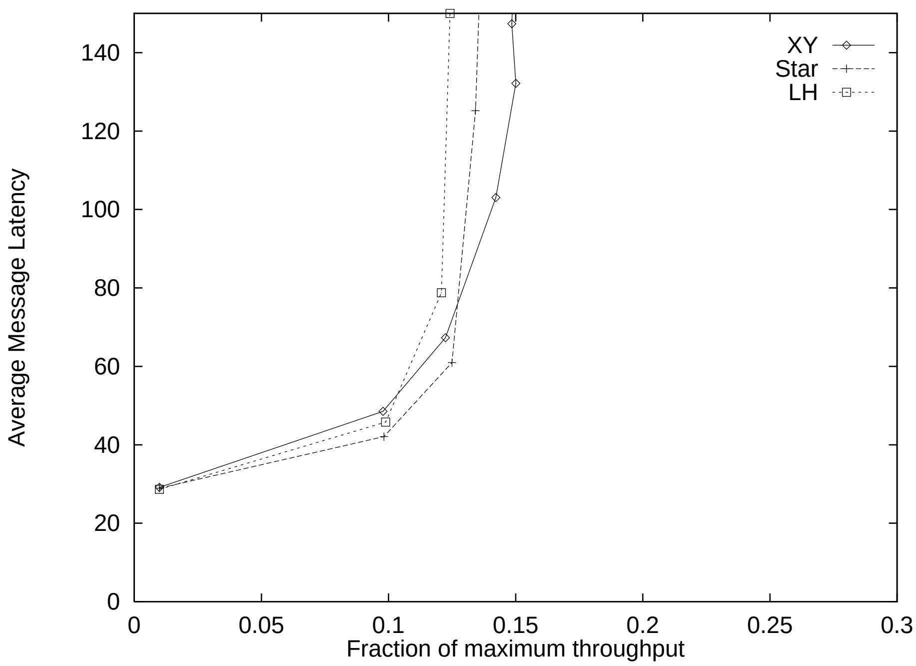


Figure 4.6: Comparison of routing algorithms for mild hot spot traffic with 2 virtual channels per physical channel in the Y dimension and 1 in the X dimension

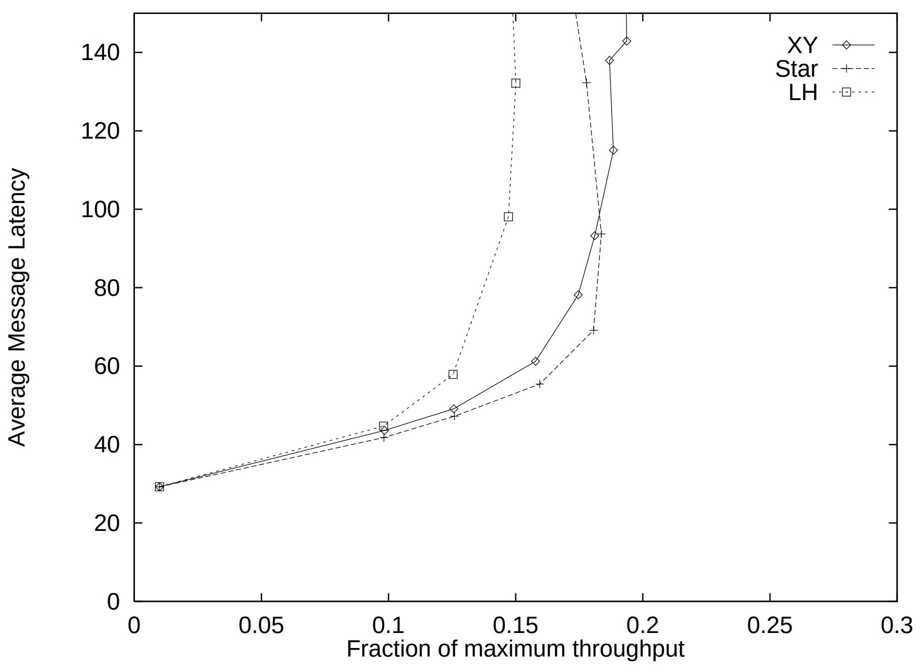


Figure 4.7: Comparison of routing algorithms for mild hot spot traffic with 2 virtual channels per physical channel in the X and the Y dimensions

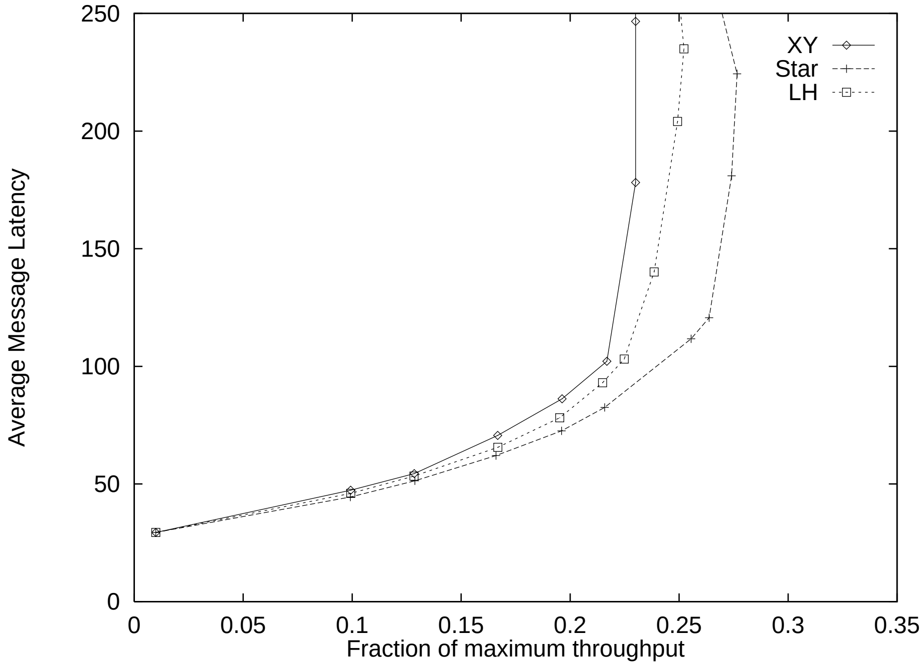


Figure 4.8: Comparison of routing algorithms for mild hot spot traffic with 6 virtual channels per physical channel in the X and the Y dimensions

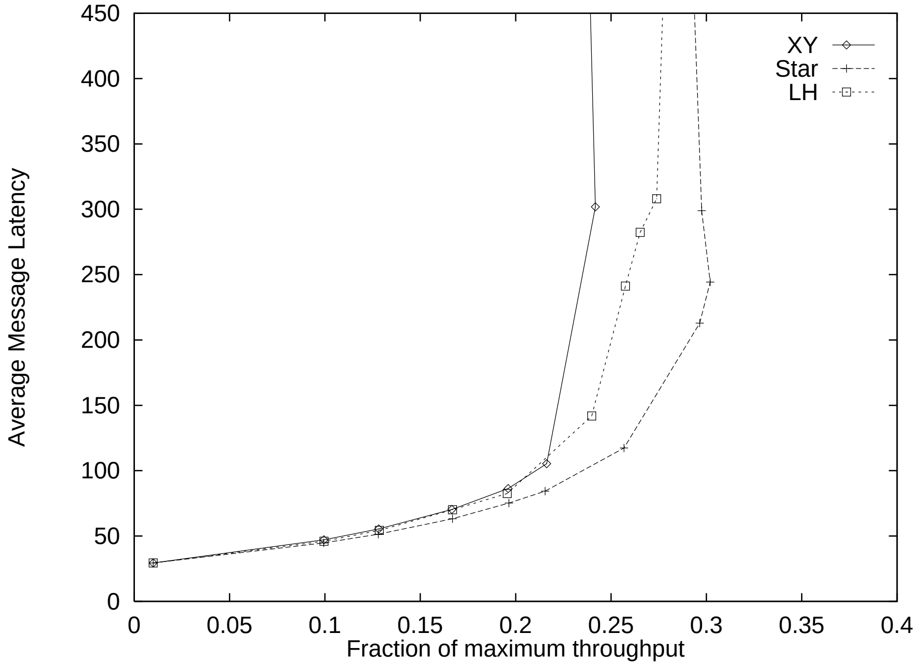


Figure 4.9: Comparison of routing algorithms for mild hot spot traffic with 10 virtual channels per physical channel in the X and the Y dimensions

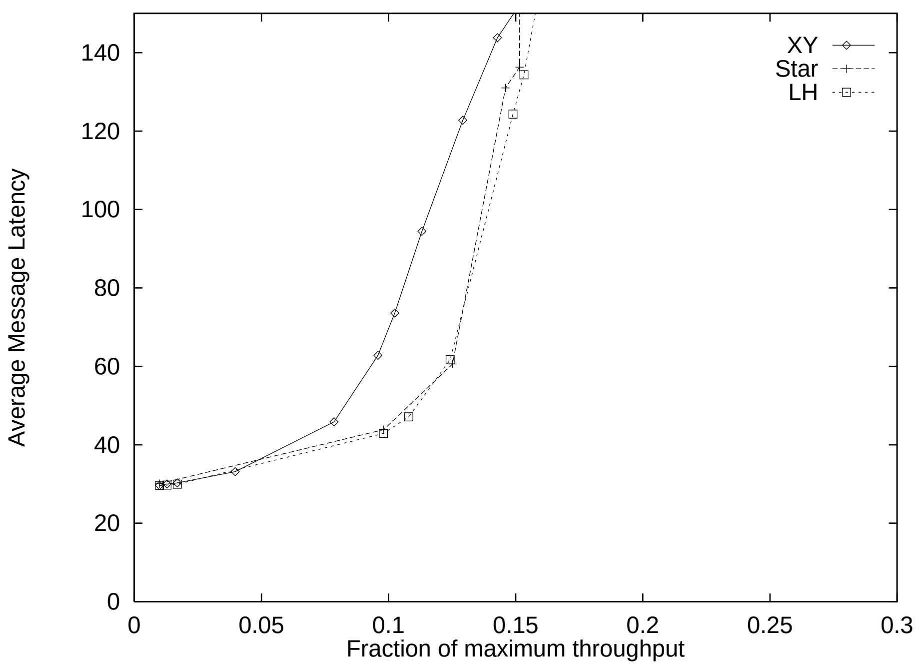


Figure 4.10: Comparison of routing algorithms for matrix transpose traffic with 2 virtual channels per physical channel in the Y dimension and 1 in the X dimension

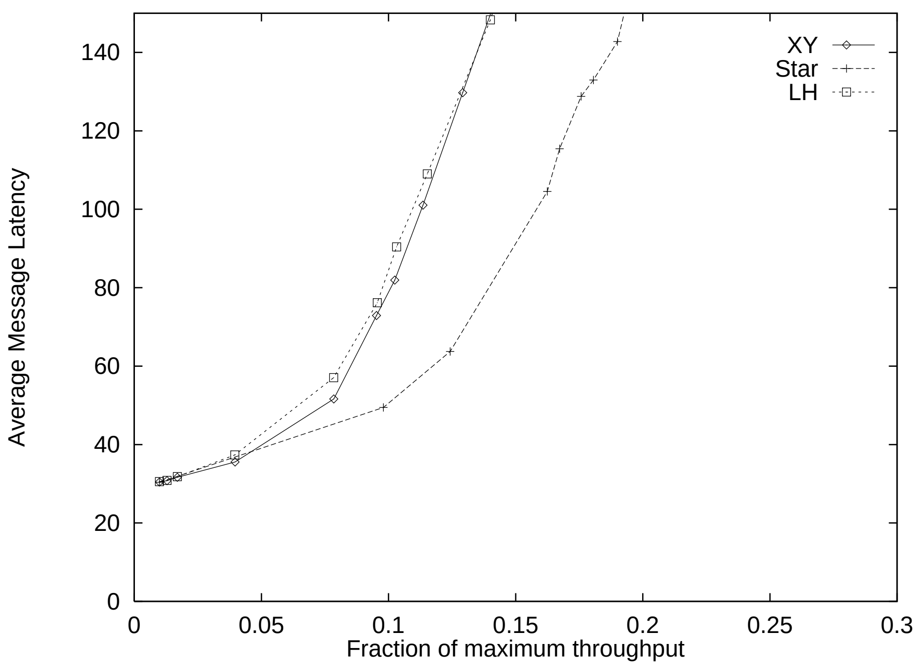


Figure 4.11: Comparison of routing algorithms for matrix transpose traffic with 2 virtual channels per physical channel in the X and the Y dimensions

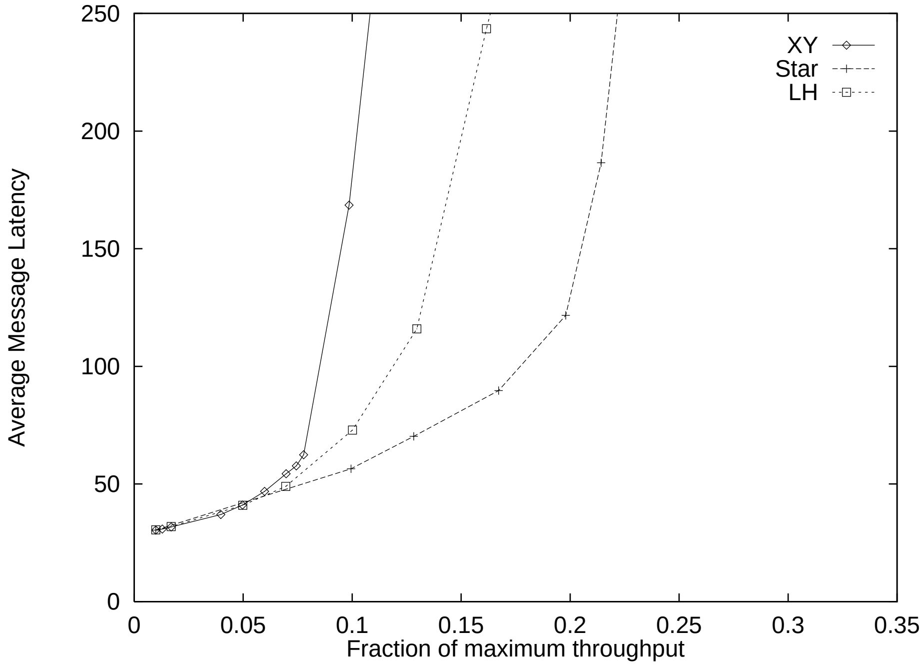


Figure 4.12: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel in the X and the Y dimensions

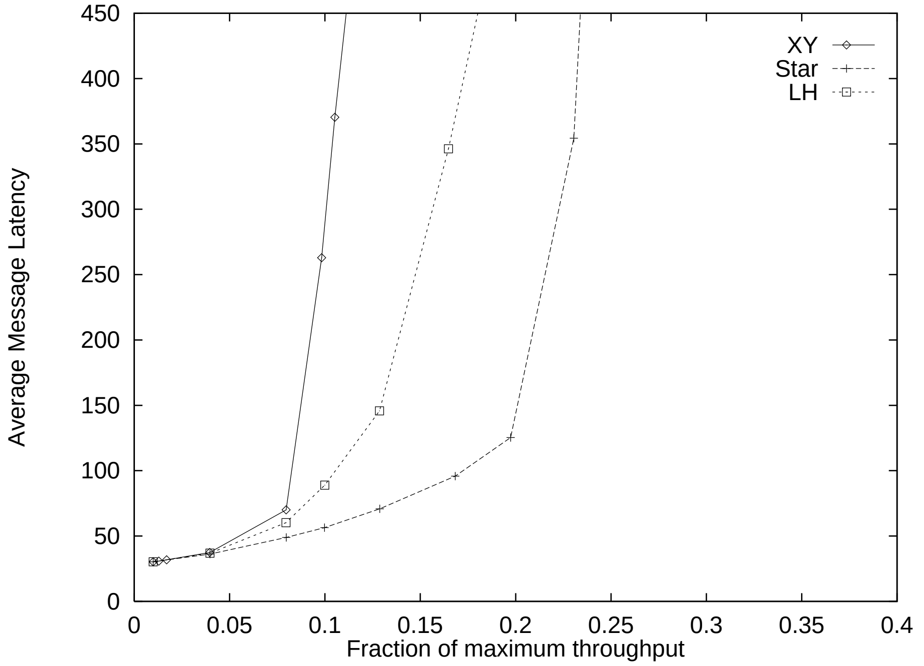


Figure 4.13: Comparison of routing algorithms for matrix transpose traffic with 10 virtual channels per physical channel in the X and the Y dimensions

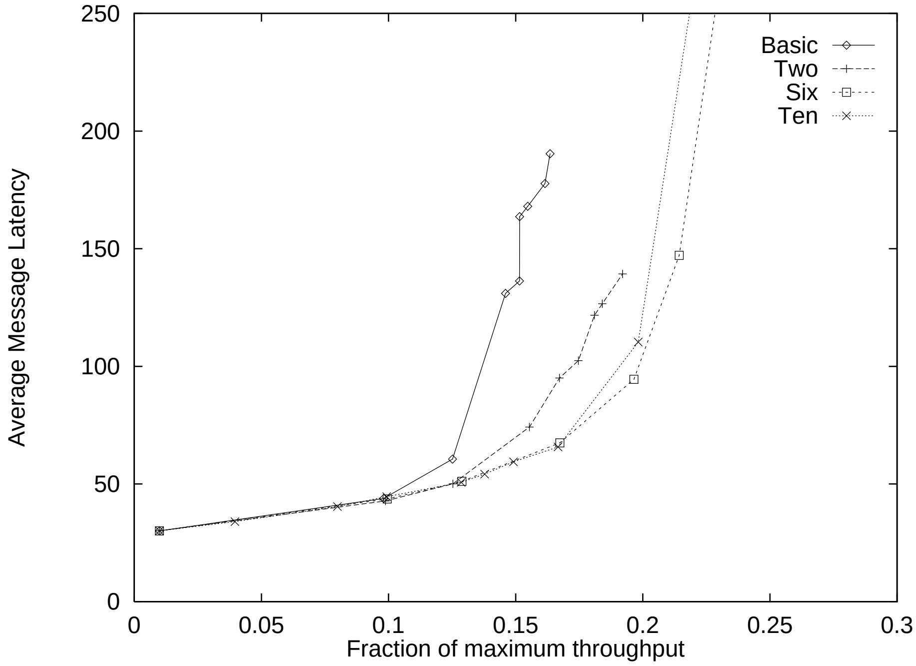


Figure 4.14: Impact of adding virtual channels to the topology for matrix transpose traffic and employing the *-Channels algorithm

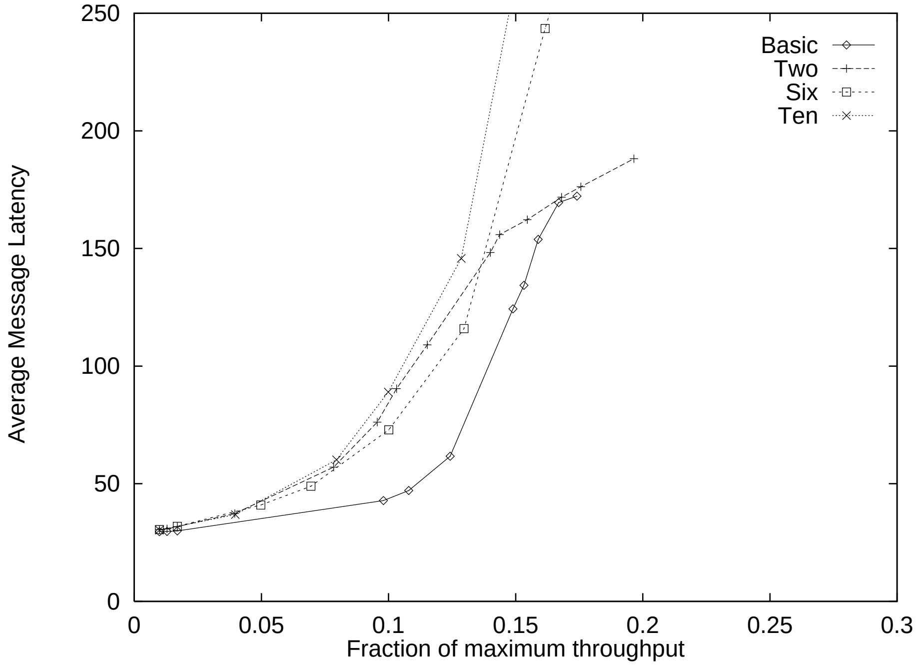


Figure 4.15: Impact of adding virtual channels to the topology for matrix transpose traffic and employing the LH algorithm

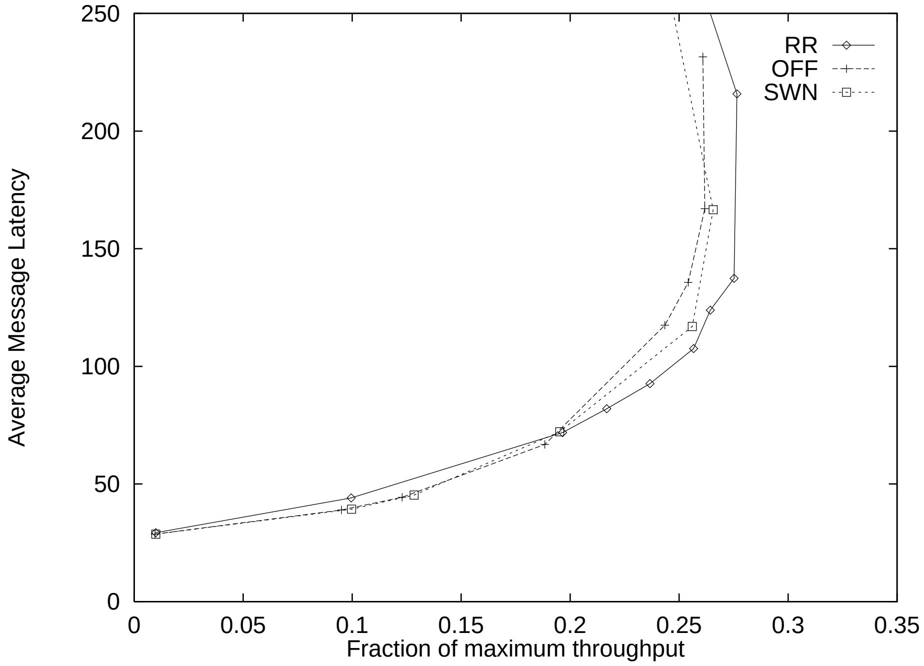


Figure 4.16: Impact of various transmission policies on the performance of the *-Channels algorithm for uniform traffic

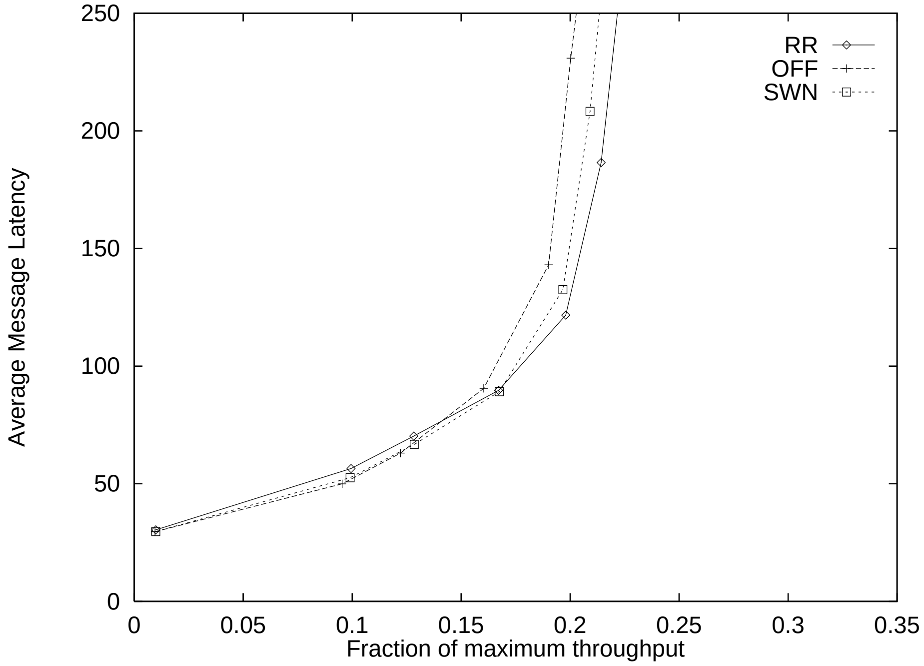


Figure 4.17: Impact of various transmission policies on the performance of the *-Channels algorithm for matrix transpose traffic

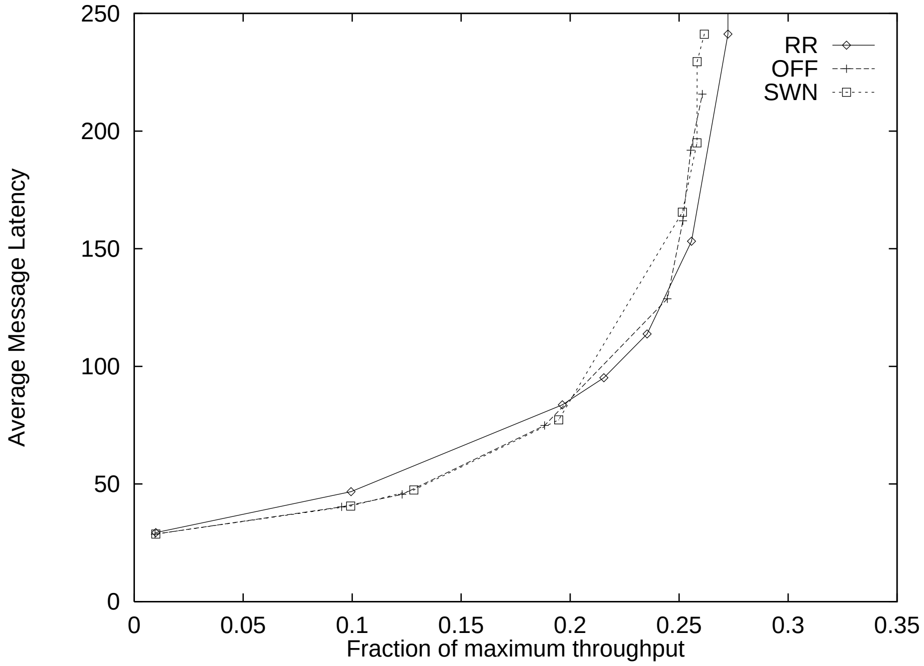


Figure 4.18: Impact of various transmission policies on the performance of the XY algorithm for uniform traffic

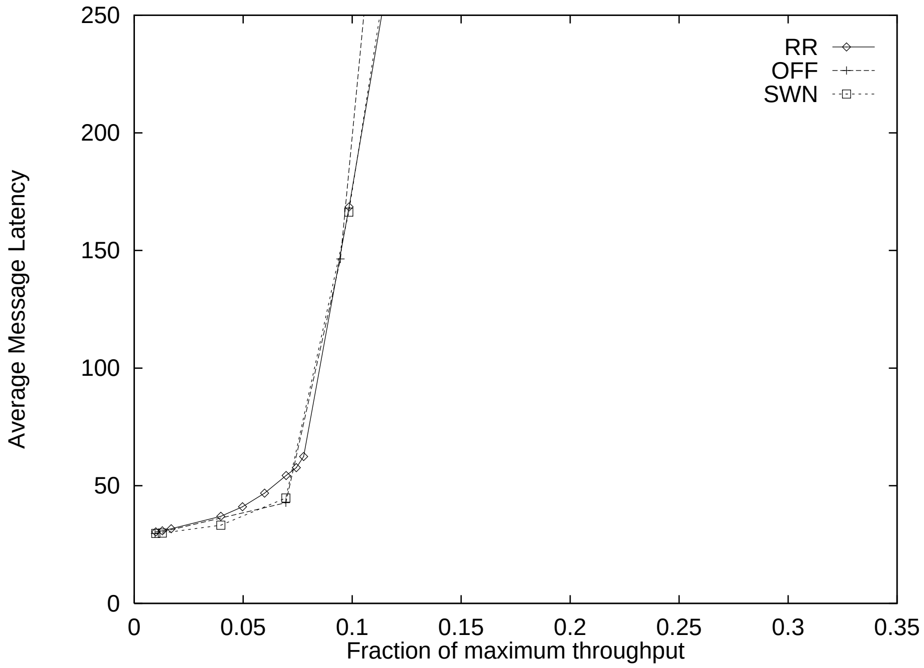


Figure 4.19: Impact of various transmission policies on the performance of the XY algorithm for matrix transpose traffic

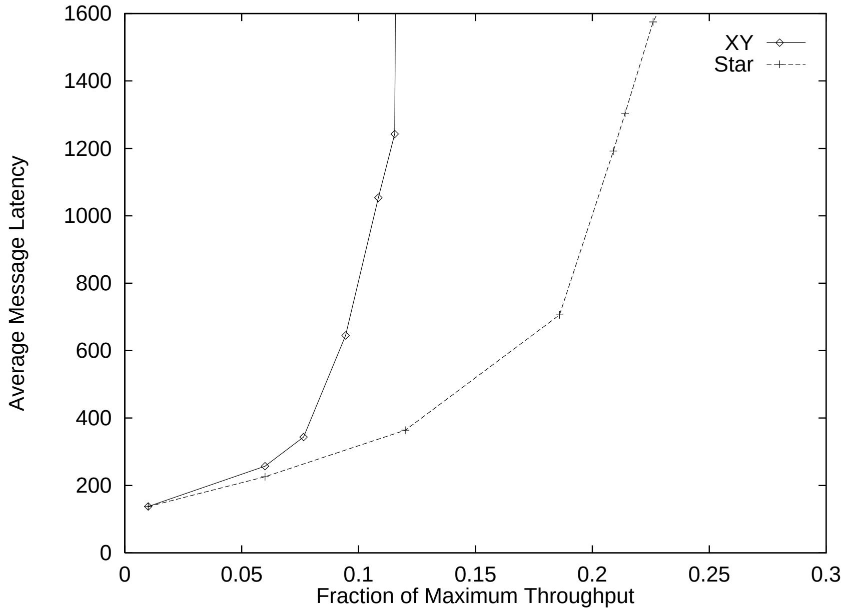


Figure 4.20: Comparison of the *-Channels algorithm and the XY algorithm, for matrix transpose traffic, with larger message sizes (128 flits)

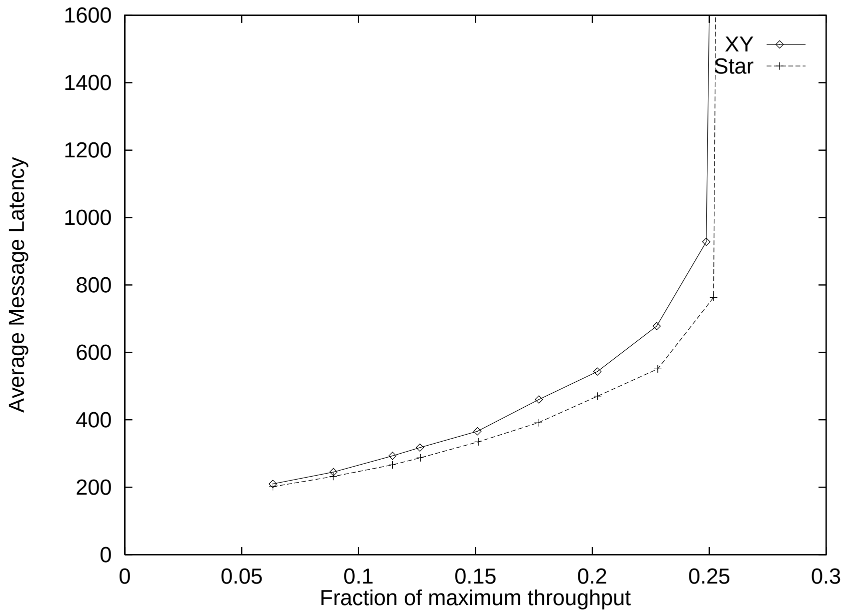


Figure 4.21: Comparison of the *-Channels algorithm and the XY algorithm, for uniform traffic, with larger message sizes (128 flits)

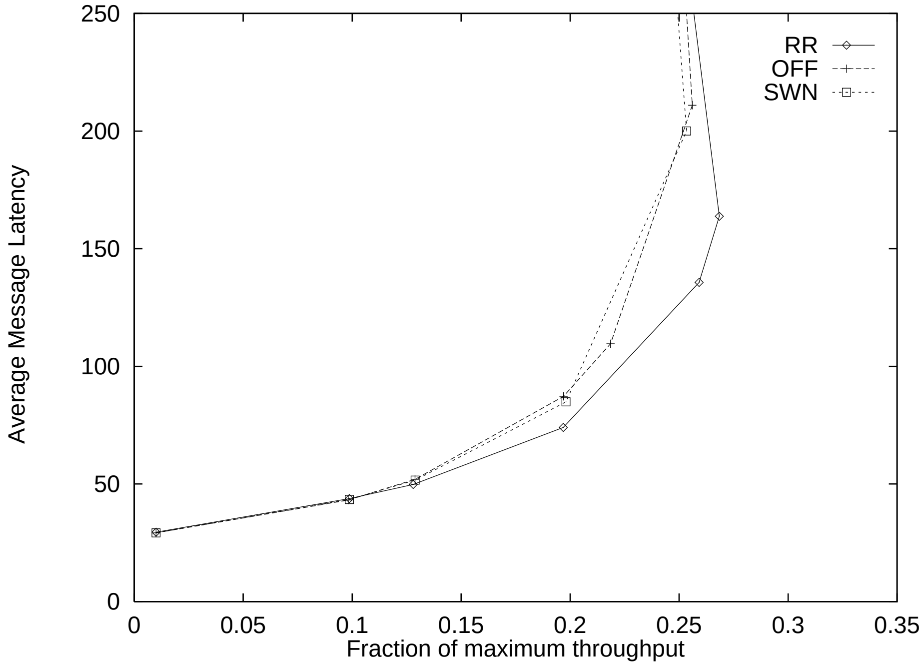


Figure 4.22: Performance under a bimodal message size distribution, using the *-Channels algorithm, for uniform traffic

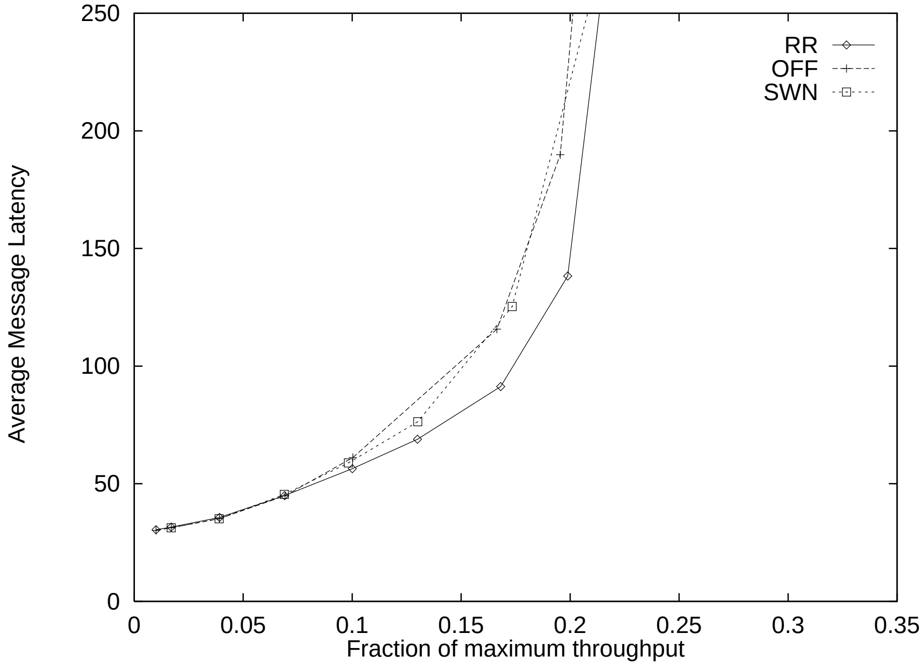


Figure 4.23: Performance under a bimodal message size distribution, using the *-Channels algorithm, for matrix transpose traffic

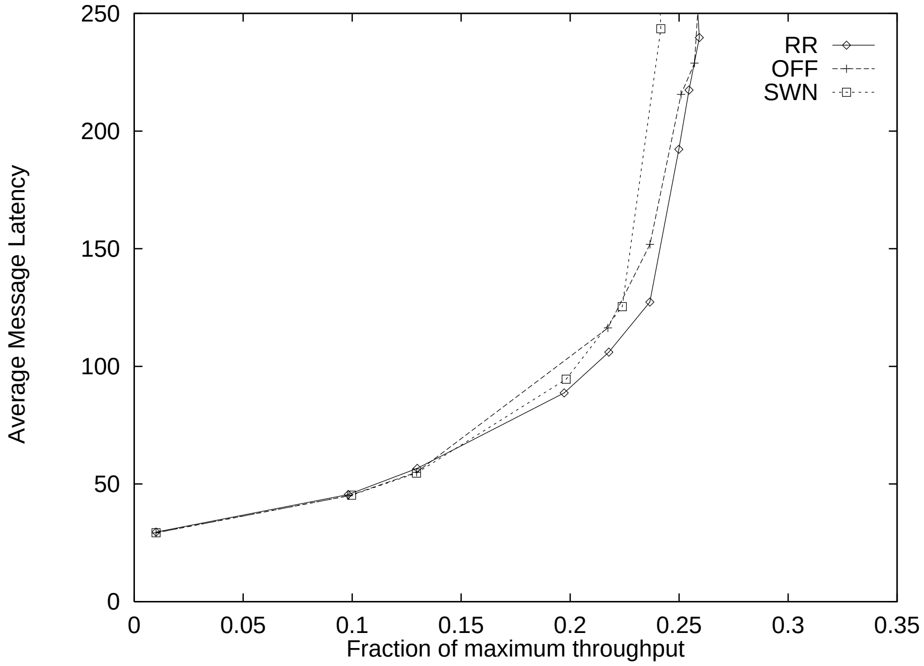


Figure 4.24: Performance under a bimodal message size distribution, using the XY algorithm, for uniform traffic

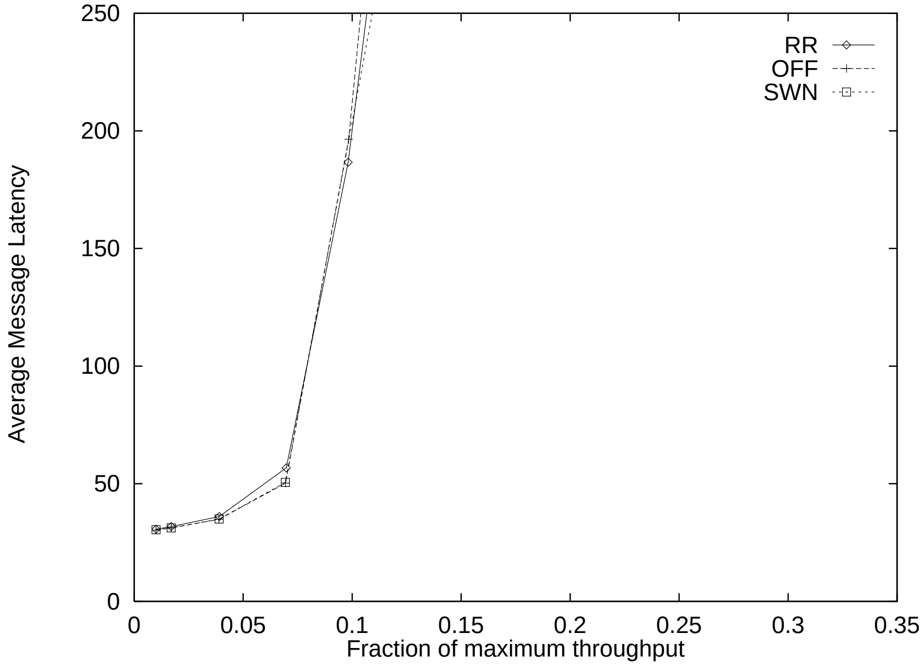


Figure 4.25: Performance under a bimodal message size distribution, using the XY algorithm, for matrix transpose traffic

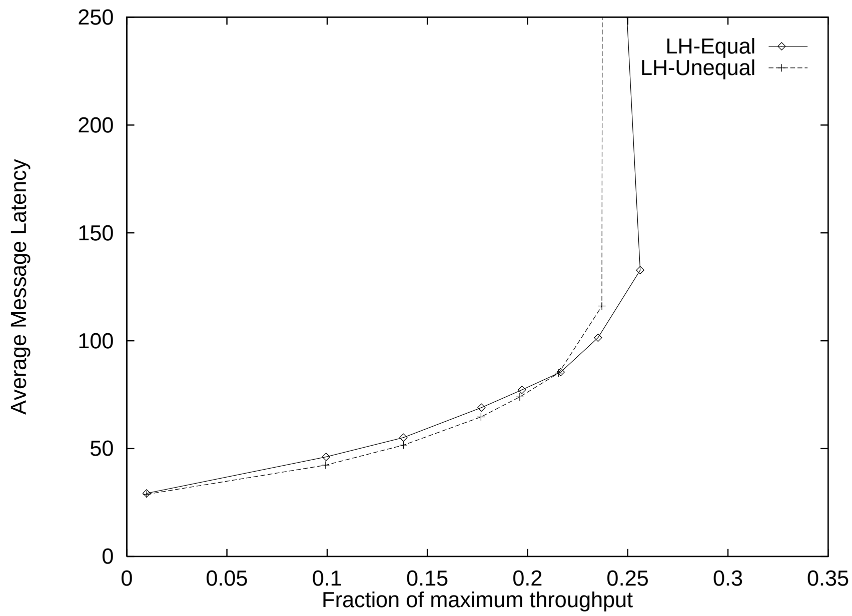


Figure 4.26: Two variations of the Linder and Harden Algorithm

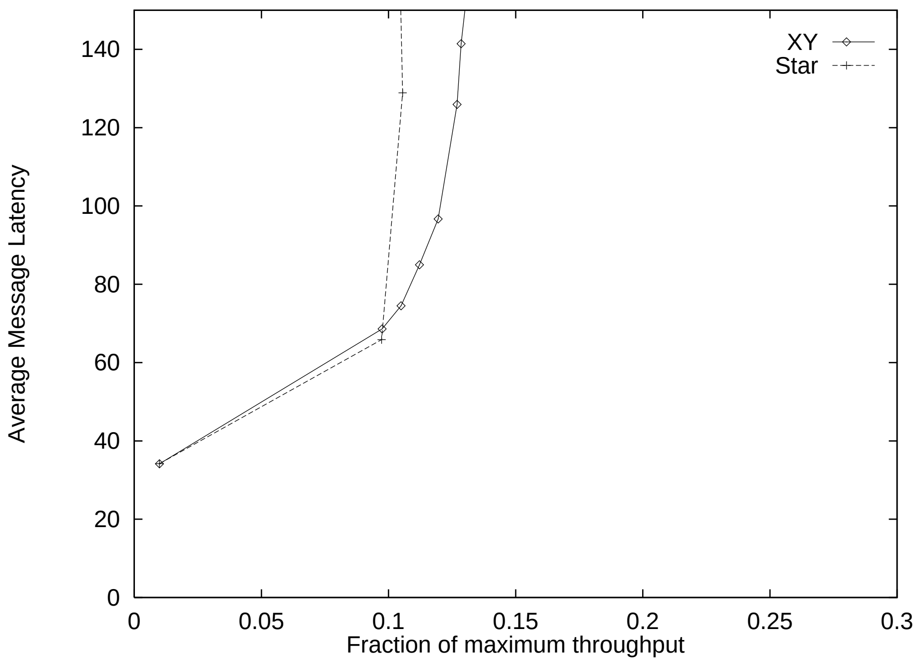


Figure 4.27: Comparison of the *-Channels algorithm and the XY algorithm for center reflection traffic with 2 virtual channels per physical channel in the X and the Y dimensions

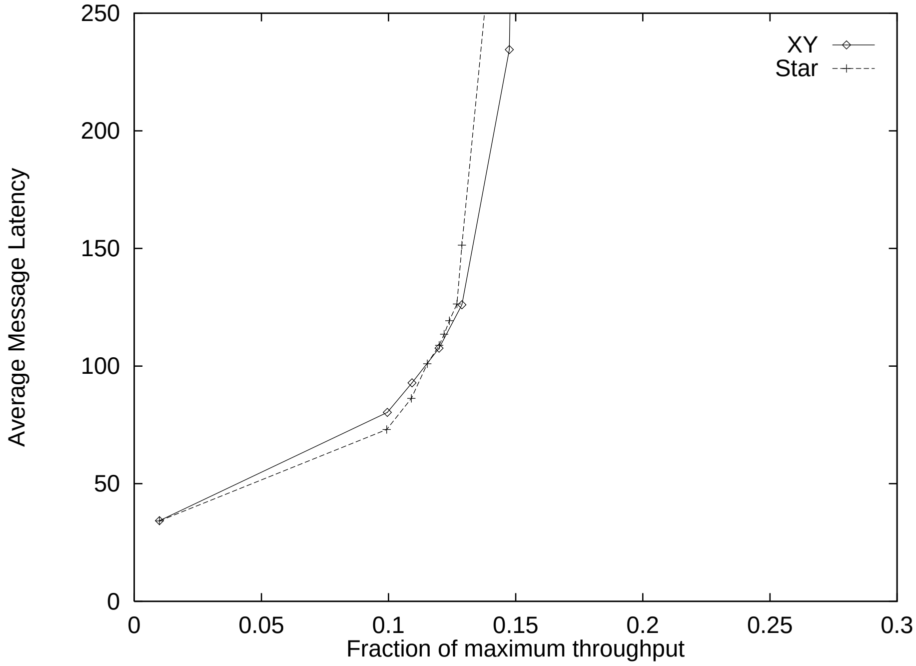


Figure 4.28: Comparison of the *-Channels algorithm and the XY algorithm for center reflection traffic with 6 virtual channels in the X and the Y dimensions

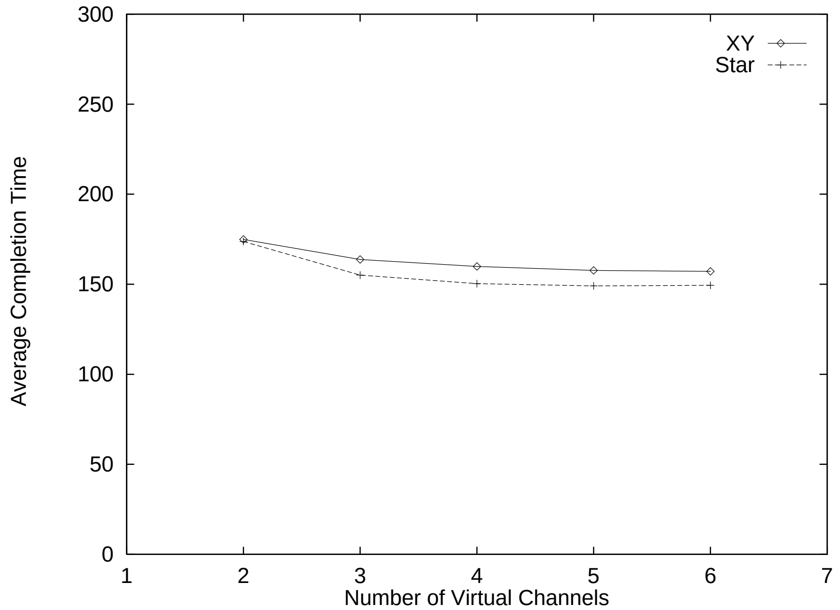


Figure 4.29: Average completion times for the XY and the *-Channels algorithms, for general traffic patterns

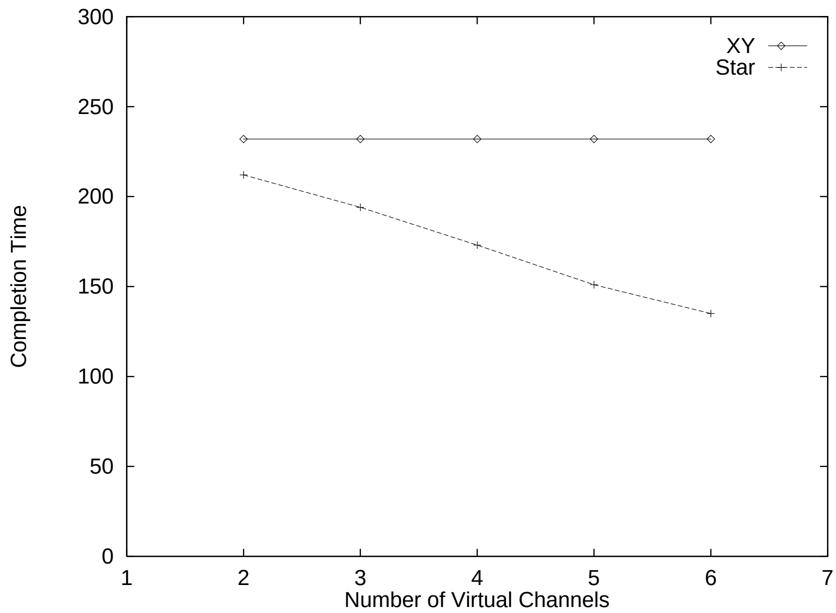


Figure 4.30: Completion times for the XY and the *-Channels algorithms, for a matrix transpose traffic pattern

Chapter 5

Quantifying Achievable Routing Performance

As shown in the previous chapter, adaptive routing techniques when combined with VCFC can significantly improve average message latency and throughput capacity. The question remains, however, as to whether much better network performance might be possible or whether there is in fact not much room between the performance offered by adaptive routing, and the best that is achievable. In this chapter, this question is answered in two parts: initially by comparing the performance provided by adaptive routing to the bounds on *maximum* throughput, *minimum* average latency and *lowest* completion time, and then by comparing adaptive routing against a *realistically achievable* source-directed routing technique that utilises application specific knowledge to make more globally-optimal routing decisions.

The rest of this chapter is organised as follows. Section 5.1 describes bounds on performance and compares them to the performance values obtained with adaptive routing. Section 5.2 discusses various source-directed routing approaches. Section 5.3 presents a new source-directed routing method. An experimental study is performed to compare deterministic and adaptive routing schemes against source-directed routing in Section 5.4. Finally, in Section 5.5, conclusions are presented.

5.1 Bounds on Performance

The network throughput is an important performance measure in interconnection networks. In Section 5.1.1, various bounds on the maximum network throughput in interconnection networks are discussed. Asymptotic bounding analysis is then used to derive additional throughput, as well as latency bounds, in Section 5.1.2. Section 5.1.3 compares the performance values obtained with adaptive routing to the theoretical bounds. Finally, Section 5.1.3 deals with completion time bounds for various traffic patterns.

5.1.1 Upper Bounds on Network Throughput

Four different upper bounds on network throughput can be defined as follows.

Absolute Consumption Bound

Each node in the network can consume at most k flits per clock cycle, where k is the number of consumption channels present at any node. If L denotes the average length of a message in flits and N_{nodes} denotes the number of nodes in the network, $N_{nodes} \times k/L$ determines the maximum number of messages that can be consumed (i.e., the maximum throughput) per clock cycle.

Since at most four physical channels can come into any node in a 2D mesh, increasing k beyond four would not be of any benefit. This thesis focuses on the case $k = 1$.

Hot Spot Bound

For non-uniform traffic, throughput is limited by the consumption rate of the most severe hot spot node (i.e., that node which receives the greatest portion of the total traffic).

Let ρ denote the proportion of traffic destined to the most severe hot spot node. Then

$$T \times \rho \times L \leq k,$$

where T denotes network throughput (in messages per clock cycle) and as before L denotes the average message length, and k denotes the number of consumption channels for each node in the network. Thus throughput is limited by $k/(\rho * L)$.

Capacity Bound

Let N_{links} denote the number of physical channels in the network and P denote the average path length (number of hops) for the traffic pattern under consideration. Since each physical channel can transmit at most one flit per clock cycle, $N_{links}/(L \times P)$ yields an upper bound on the network throughput.

Bisection Bound

This bound is determined by the number of physical channels that would be cut when the network is partitioned into two parts, and by the portion of traffic that must cross these channels. The number of physical channels that connect the two parts would determine how many flits can be transmitted per clock cycle from one part to another. Letting T denote the network throughput, C the average number of crossings per message of the cut physical channels and N_{cut} the number of cut physical channels,

$$T \leq (1/C) \times N_{cut}/L$$

This bound holds for every possible partitioning of the network; by choosing to cut "bottleneck" channels as determined by an analysis of the traffic pattern under consideration, the tightest bound of this form can be obtained.

□

The bound that most constrains the range of feasible throughput is the minimum of these four bounds. Out of the four bounds discussed here, only the consumption bound is independent of the traffic pattern. For this reason, it was chosen as a normalising factor (for throughput) in the performance study presented in Chapter 4. If there are no specific hot spots, both the hot spot bound and the consumption bound would be equal to $n^2 \times k/L$. In the presence of hot spots, the hot spot bound would always be tighter than the consumption bound. P/N_{links} represents the average number of crossings of a link (over all links in the network) and, if the tightest of the bisection bound is chosen, C/N_{cut} represents the maximum over all bisection cuts of the average number of crossings of a link belonging to the cut. This implies that C/N_{cut} would never be less than P/N_{links}. Therefore, the bisection bound ($N_{cut}/(C \times L)$) would always be at least as tight as the capacity bound ($N_{links}/(P \times L)$) irrespective of the traffic pattern.

In what follows, the hot spot and bisection bounds are analysed for the three different traffic patterns considered in Chapter 4 (uniform, matrix transpose and hot spot). A 2D mesh of size $n \times n$ is considered; here N_{nodes} is n^2 and N_{links} is $4 \times n \times (n-1)$.

For uniform traffic the destination node for any message in the network can be any node except the source node. The proportion of traffic crossing a bisection cut through the center of the network would be $(n^2/2)/(n^2-1)$. For large n this is approximately equal to 1/2. Thus the value for the bisection bound would be approximately $(4 \times n/L)$. For all $n > 4$ $(4 \times n/L)$ is less than $n^2 \times k/L$. Therefore, the tightest bounding value for uniform traffic for the case of interest is $(4 \times n/L)$ (approximately).

With matrix transpose traffic, since each node receives and sends just one message, the hot spot bound is equal to the consumption bound $(n^2 \times k/L)$. A bisection cut along the network diagonal yields the tightest bisection bound. Since all traffic crosses the bisection cut, $C = 1$ and N_{cut} is equal to $4 \times (n-1)$. Thus, the bisection bound evaluates to $4 \times (n-1) \ / \ L$. For all values of n, this bound is tighter than the hot spot consumption bound.

For the hot spot traffic pattern considered in Chapter 4, the hot spot bound is calculated as follows. Attention is restricted solely to the case of $n = 12$. Since there are six hot spot nodes each of which receives 3 times as much normal traffic as does every other node, the proportion of traffic going to a hot spot node would be $3/(138 \times 1 + 6 \times 3)$. The hot spot is thus $52k/L$, which is larger than the bisection bound $(4 \times n/L)$ for $n = 12$. With regard to the bisection bound, note that the hot spot nodes are spread randomly through the network, so the bisection cut would on the average result in three hot spot nodes in each part. Thus the tightest bound is the bisection bound, as for uniform traffic.

It is interesting to note that even in the presence of mild hot spots the bisection bound dominates and limits the performance in 2D mesh networks. When the hot spot bound and the bisection bound are equated, the value of ρ becomes $k * C/N_{cut}$. For uniform traffic with evenly allocated hot spots, $C = 1/2$ and $N_{cut} = 2n$, which further yields $\rho = k/(4 \times n)$. Thus as long as any hot spot node does not receive more than the fraction $k/(4 \times n)$ of the total network traffic, the maximum possible network throughput (assuming uniform other traffic) is not limited by the consumption capacity of hot spots. This is consistent with the experimental results in the previous chapter where a mild hot spot traffic gave performance similar to uniform traffic when enough channels were present in the network to reduce unnecessary blocking. Note that increasing the number of consumption channels (k) can increase the value of ρ. Thus it can be inferred that having more consumption channels (up to a maximum of 4 in 2D meshes) may help to reduce the effects of severe hot spots.

5.1.2 Bounds Based on Asymptotic Analysis

Asymptotic analysis is a simple bounding technique that is based on consideration of the (asymptotically) extreme conditions of heavy loads and light loads [43].

Consider the average message latency with no contention in the network, termed here the "base latency" and denoted by R_{min}. For wormhole switching this is equal to $L+(D-1)$ where L is the message length in flits, and D is the average distance (in number of hops) to be travelled by a message. For uniform traffic, for example, in a $n \times n$ mesh, average path length can be shown to be $2 \times n/3$. Thus, for a value of L = 20 and $n = 12$, as employed in Chapter 4, the base latency equals 27 cycles.

In terms of the average number of messages in the system N, N/R_{min} yields an upper bound on throughput by Little's law [43]. From the previous throughput analysis an upper bound on throughput for uniform traffic, for example, was shown to be equal to $(4 \times n/L)$. Thus, incorporating the above asymptotic analysis yields an upper bound on throughput, for uniform traffic, of $\min(4 \times n/L,\ N/R_{min})$. The value of N required to apply this bound will be measured during simulation experiments, as will be shown in the next subsection. Bounds for other traffic patterns are similarly derived.

Consider now bounding average latency again in terms of N. The average message latency has to be greater than or equal to the base latency R_{min}. Again applying Little's law to the maximum throughput value, the average latency has to be greater than N/T_{max} for any value of N. Thus a lower bound on average latency is max(R_{min},N/T_{max}).

The maximum throughput can also be bounded in terms of the injection rate (α) at any node in the network. The maximum possible throughput in terms of α is $n^2 \times \alpha$, yielding a throughput bound of min($n^2 \times \alpha$, $4 \times n/L$).

5.1.3 Comparison with Performance Offered by Adaptive Routing

The bounds obtained by the above analysis are shown in Figures 5.1, 5.2 and 5.3, along with the simulation results for the *-Channels adaptive algorithm, with two different numbers of virtual channels per physical channel, for uniform traffic on a 12×12 mesh. The average number of messages in the system was measured from the simulation experiments. The throughput on the Y axis in Figures 5.1 and 5.2 is represented in a normalised form by dividing it by n^2/L, so as to be consistent with the graphs presented in Chapter 4.

As shown in Figure 5.1 the throughput achieved with the *-Channels algorithm increases with increasing N, and then after reaching a peak it starts dropping. This is because for large values of N the contention in the network becomes high. This results in unnecessary blocking for all messages which in turn reduces the throughput. This situation is greatly improved once more virtual channels are added as seen in this figure. The N/R_{min} bound shown in Figure 5.1 is not really achievable since it assumes minimum latency for all messages in the network. An alternative bound on throughput is the $n^2\alpha$ bound shown in Figure 5.2. Here note that the throughput is quite close to the maximum possible throughput, until higher loads are reached.

The bounds on latency as given in Figure 5.3 indicate that there is some room for improvement over the performance provided by adaptive routing when the average number of messages in the system is small. For a large number of messages in the system the room for improvements in latency is quite modest. As before the addition of virtual channels has resulted in good improvements.

The results with matrix transpose traffic were quite similar in nature as can be seen in Figures 5.4, 5.5 and 5.6. In general, one cannot expect the bounds to be closely matched by real system performance owing to the extreme assumptions made during the derivations. Thus, these results suggest that there may not be realistic opportunities for very large improvements in performance beyond that offered by adaptive routing (excepting, possibly through use of a load control strategy to avoid the sometimes observed sharp drop-off in throughput under overload conditions).

5.1.4 Completion Time Bounds

In this section the time to complete a single communication “phase” is of interest. This type of workload was briefly considered in Section 4.3.4. Completion time depends to a large extent on the traffic pattern being considered. Determination of the lowest possible completion time requires a complete knowledge of the traffic pattern. In this section a bound on completion time is initially derived for the completely predictable matrix transpose pattern.

Since all messages start at approximately the same time, completion time de-

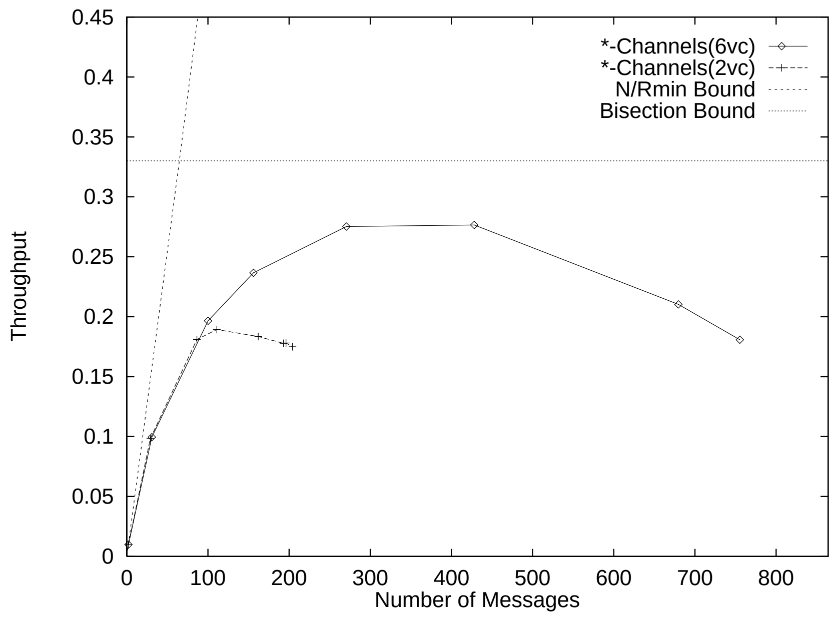


Figure 5.1: Upper bounds on throughput, and the throughput provided by the *-Channels algorithm, as a function of the average number of messages in the system, for uniform traffic pattern.

pends largely on the maximum multiplexing faced in the network. With the transpose traffic pattern (as defined for a single communication phase), all nodes in the mesh (except the diagonal nodes) generate messages which go across the diagonal physical channels. Therefore the physical channels on the diagonal would have the highest multiplexing factor. Figure 5.7 shows the channels that would be used by messages going from the lower triangular section of the mesh to the upper triangular. There are $n \times (n-1)/2$ nodes below the diagonal, which is equal to 66 for a 12 by 12 mesh. These nodes generate messages which must each use one of the $2 \times (n-1)$ physical channels (which is 22 for a 12 by 12 mesh) to traverse the diagonal. This might lead one to conclude that the traffic can be balanced such that no physical channel need carry more than three messages. If this is the case then physical channels L1 and L2 (shown in Figure 5.7) would have to carry at least five messages for the two corner diagonal physical channels to have three messages each.

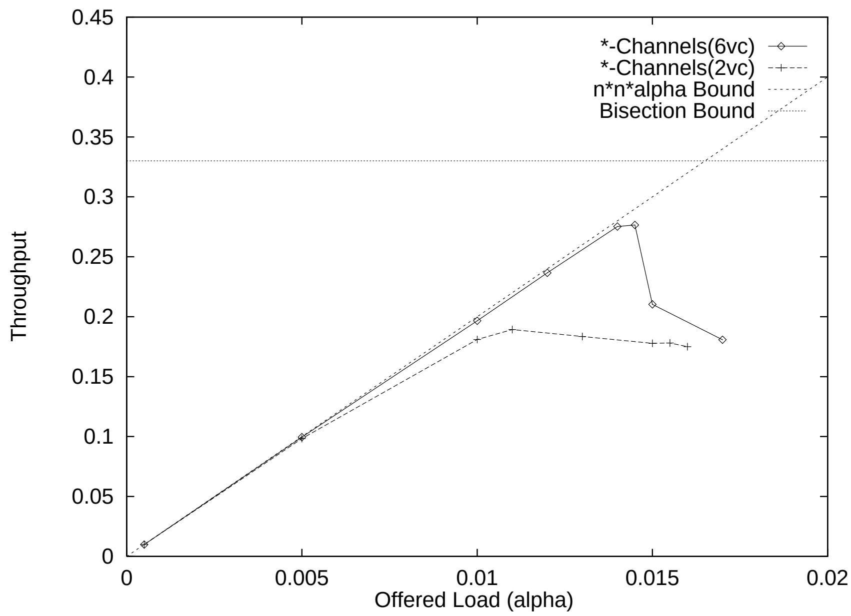


Figure 5.2: Upper bounds on throughput, and the throughput provided by the *-Channels algorithm, as a function of the offered load, for uniform traffic pattern.

Thus the maximum multiplexing becomes five instead of being three. Whereas, if some center physical channels are allowed to carry four messages each, then the corner physical channels can carry two each, and then the maximum multiplexing can be as low as four messages per physical channel. This would mean that the completion time is at least $4 \times L$, where L is the message size in flits. This bound is not strictly achievable, however, as messages still have to be fully received at their final destination after crossing the physical channel with maximum multiplexing before the phase is completed. The completion time bound is shown along with the experimental values observed for adaptive routing and deterministic routing for the matrix transpose traffic pattern in Figure 5.8.

Completion time bounds for other traffic patterns can be similarly derived by analysing the bottleneck physical channels and determining the multiplexing factor on these physical channels. To do this more efficiently, a program was devised which

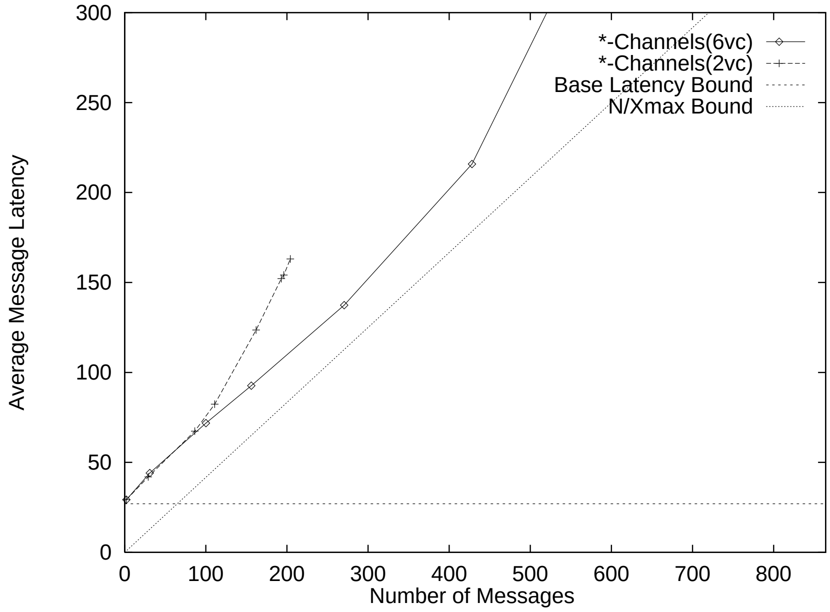


Figure 5.3: Lower bounds on latency, and the latency provided by the *-Channels algorithm, as a function of the average number of messages in the system, for uniform traffic pattern.

would partition a mesh network at various points (typically at every column and row) and determine the number of messages that would have to go across these cut physical channels to the other side (given the knowledge of the traffic pattern). The maximum of these values would give the lower bound on the maximum contention level for any physical channel in the network. This program also considers the maximum number of messages to be consumed by any node in a communication phase. The largest of these two values gives the maximum multiplexing factor.[1] Once the maximum multiplexing (M) is known, the completion time would be $M \times L$. Figure 5.9 shows the results of this completion time bound calculation

[1]It is assumed here that the program would be able to find the maximum contention faced on any physical channel by analysing different row and column bisections. But it might so happen that the row or column bisection may not yield the correct picture and some other subset of physical channels may constitute the bottleneck. For this reason, if the program makes a poor estimate, the bounds would not be very tight. However, the experimental study discussed later in this chapter shows that this bound is indeed quite tight and can approached very closely.

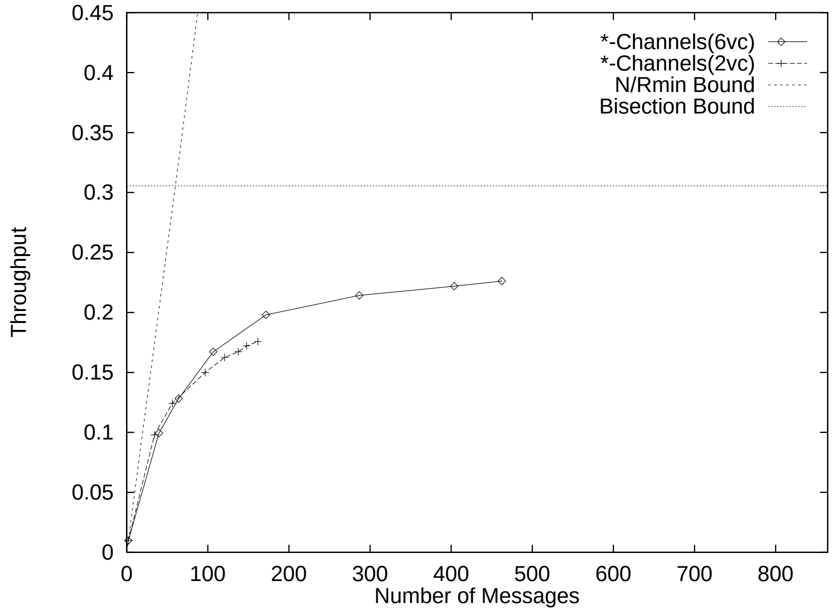


Figure 5.4: Upper bounds on throughput, and the throughput provided by the *-Channels algorithm, as a function of the average number of messages in the system, for matrix transpose traffic pattern.

for a general traffic pattern (considered briefly in Section 4.3.4. and more fully in Section 5.4). Note that, as with matrix transpose traffic, there is a large gap between the completion time provided by adaptive routing, and the lower bound.

Similar analysis for other traffic patterns also shows a large gap between adaptive routing and the lower bound. Although it is possible that this may simply reflect the looseness of the bounds, it is also possible that some more substantial defect in adaptive routing is being illustrated. Routing decisions in these policies are based only on the local conditions at each router, rather than on global knowledge of network traffic patterns. Over a number of message transmissions, it is likely that at least one message will be badly routed, resulting in a larger maximum multiplexing value, and thus a larger completion time.

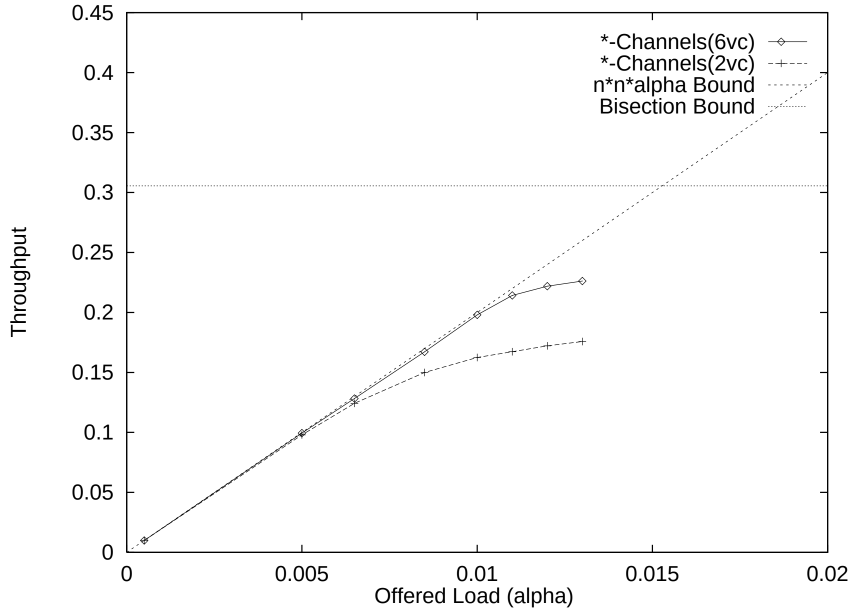


Figure 5.5: Upper bounds on throughput, and the throughput provided by the *-Channels algorithm, as a function of the offered load, for matrix transpose traffic pattern.

5.1.5 Summary

In this section the performance of the *-Channels adaptive algorithm was compared against the maximum throughput and minimum latency bounds. It is quite possible (though unlikely) that some other adaptive algorithm might provide better performance than the *-Channels algorithm and thus change some of the conclusions in this section. The *-Channels algorithm was chosen as the representative of adaptive algorithms since its performance has been shown to be generally superior to other algorithms.

The bounding analysis suggests that adaptive routing techniques (when combined with VCFC) may perform reasonably well in terms of maximum possible throughput and average latency. On the other hand, adaptive routing schemes may not perform as well on workloads in which completion time is governed by maximum

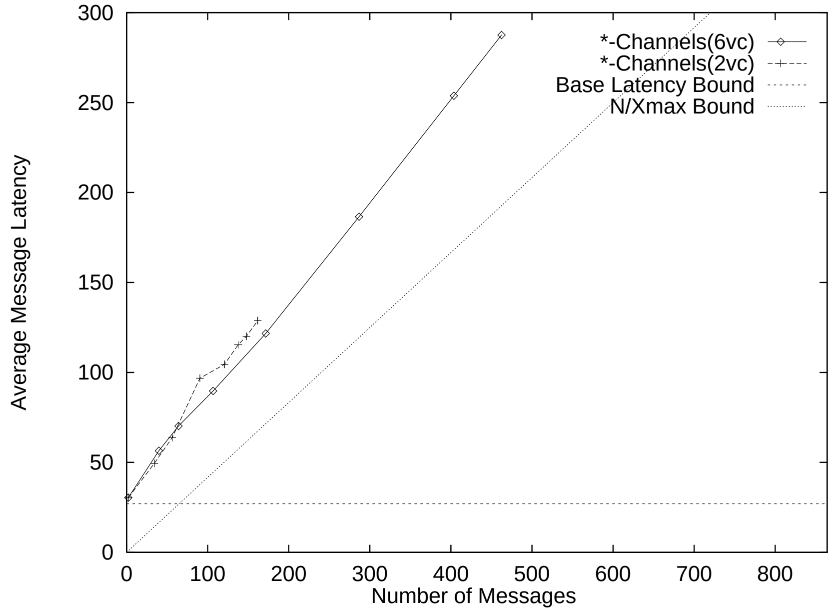


Figure 5.6: Lower bounds on latency, and the latency provided by the *-Channels algorithm, as a function of the average number of messages in the system, for matrix transpose traffic pattern.

message latencies. There may be significant room for improvement in completion times achieved by adaptive and deterministic algorithms for these workloads. The question as to whether much better network performance can be achieved with other, more sophisticated routing techniques, is studied further in the next few sections by considering source-directed routing methods.

5.2 Source-Directed Routing

In the reminder of this chapter, the limits of realistically achievable network performance are quantified by considering source-directed routing methods, under the favourable assumption of full predictability of traffic patterns. In source-directed routing, message routes that minimise message contention are determined in advance, and specified by the source in the message header. In several common cases

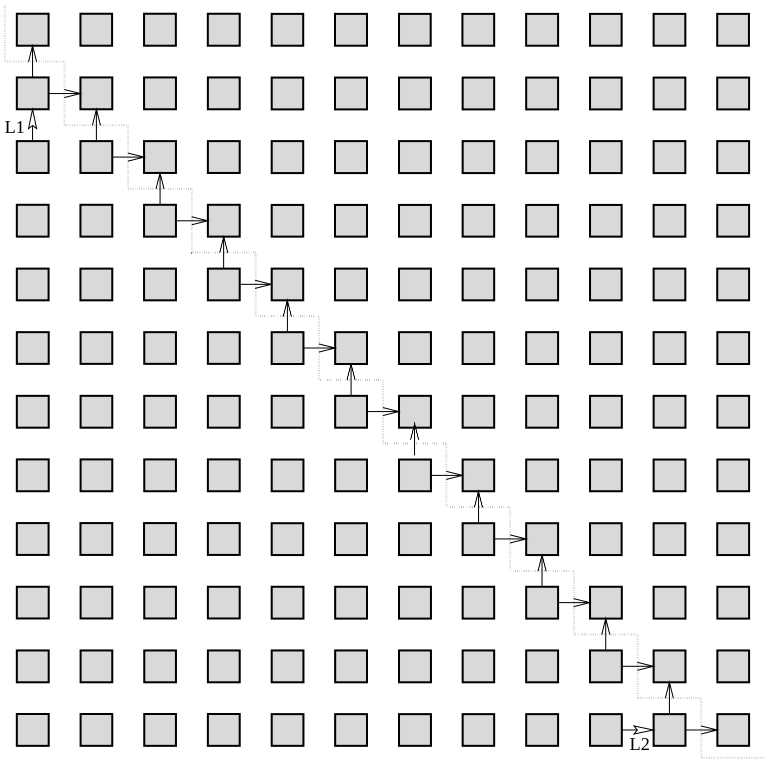


Figure 5.7: All messages cross the diagonal of the mesh

the most communication-intensive parts of an application have traffic patterns predictable either at compile time [1, 28, 45, 48, 65], or at run time through inspector-executor approaches [59], and thus this pre-determination should be possible in these cases. Thus, source-directed routing schemes can potentially act as good indicators of realistically achievable performance.

The design of a good source-directed routing algorithm is in itself an interesting problem. As full advance knowledge of routing patterns will not *always* be available, it is desirable that the source-directed routing method be inter-operable with conventional deterministic or adaptive routing techniques. Specifically, not only should the network be capable of carrying non-source routed traffic, but it should be possible for the network to carry both source routed and non-source routed traf-

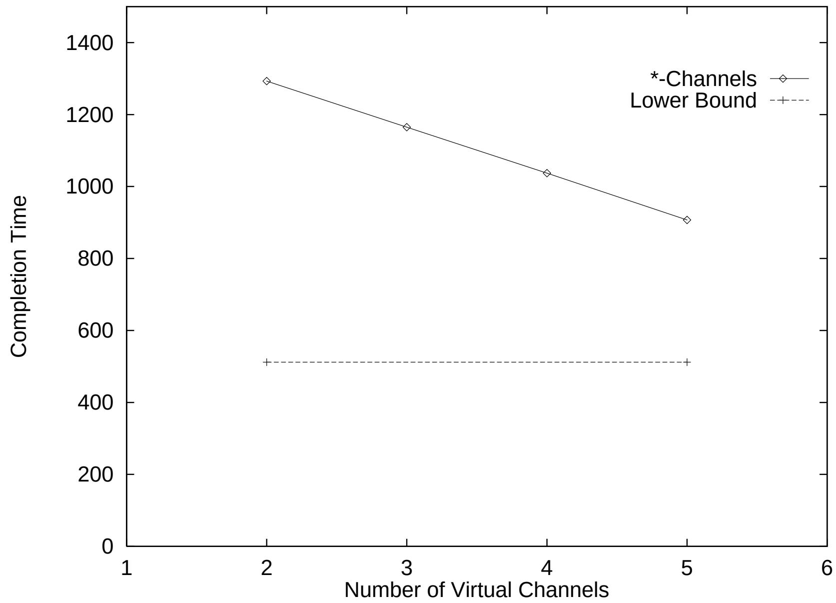


Figure 5.8: Bounds on completion time for matrix transpose traffic pattern

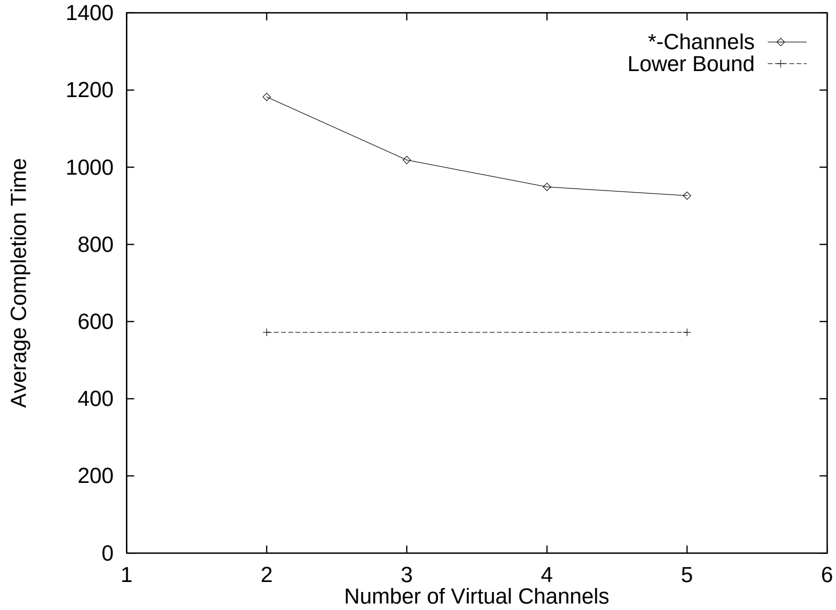


Figure 5.9: Bounds on average completion time for general traffic patterns

fic simultaneously. An important implication of this requirement is that deadlock prevention in source-directed routing cannot be handled by "compiling" it into the chosen routes, as there may be other traffic on the network that was not predictable at the time the routes were chosen.

Previous research on source-directed routing has included the development by Zhong and Lo of an algorithm that determines good message routes, on a mesh topology, for a specified input traffic pattern [65]. The algorithm labels communication channels with weights proportional to the probability of usage, assuming that all shortest paths between a source-destination pair are equally likely to be used. Once the channels are labelled appropriately, messages are considered in increasing order of the number of routing choices each has, and assigned to paths with the least probable congestion. As paths are chosen, the weights of their physical channels are increased. For deadlock avoidance, an approach similar to that employed in the LH adaptive routing algorithm is employed, requiring a number of virtual channels per physical channel that is exponential in the dimensionality of the mesh. Performance comparisons are made against deterministic routing only.

Abali and Aykanat [1] have considered the use of source-directed routing in machines like the IBM SP1. They argue that a large class of applications (including, for example, iterative solvers for systems of equations) can benefit by employing source-directed routing schemes, even if the knowledge of traffic patterns is non-temporal in nature (i.e., only the pattern of message exchanges is known, but not the relative timings at which messages are generated). They show that their source-directed routing can greatly reduce congestion in multistage interconnection networks.

Finally, source-directed routing has been employed in Intel's iWarp multiprocessor system [14]. The iWarp system has a logically reconfigurable network in which a uni-directional connection called a *pathway* can be set up between any two nodes. Once a pathway is created, it is used for all communication between the two nodes. The pathways are set up by the source nodes based on the application's expected traffic patterns. The issue of deadlock avoidance in iWarp is left to the programmer/compiler to handle, and the co-existence of source-routed traffic with non-source-routed traffic is not possible.

In the next section, a specific source-directed routing method is proposed that can interoperate with either deterministic or adaptive routing, so as to allow these latter techniques to be used as desired when message patterns are unpredictable. The proposed method imposes minimal additional router cost beyond that required for the deterministic or adaptive routing technique that is otherwise employed. In Section 5.4, the performance of this method is studied on 2D mesh networks using simulation, comparing it to both deterministic and adaptive routing on a number of traffic patterns.

5.3 Proposed Source-Directed Routing Scheme

Implementation of source-directed routing requires both hardware mechanisms to carry out such routing, and an algorithm to be executed at compile time (or possibly run time) to determine good routes. In Section 5.3.1, the hardware implementation is discussed ignoring temporarily the problem of deadlock prevention. The issue of deadlock prevention is considered in Section 5.3.2. Finally, Section 5.3.3 considers the task of route determination.

5.3.1 Implementation Overview

Ideally, a source-directed routing method should require little in the way of additional hardware beyond that required for the base deterministic or adaptive routing method, as employed for the non-source routed traffic. This goal is attained by making use of the base routing method in the source-directed routing implementation.

In the method proposed here an arbitrary number of intermediate points through which the message must travel, according to the chosen route, are specified in header flits of the message. This is illustrated in Figure 5.10, where it is assumed that a single flit can accommodate the specification of one intermediate point. Routers forward each message to the node specified in the message's leading header flit, according to the base routing scheme. The router at each of the specified intermediate nodes strips off the leading header flit and continues forwarding the message, rather than delivering it to the attached node. A single bit in each header flit can be used to determine if it is the last such flit. This is necessary to differentiate an intermediate node from the destination node. The "address" of each intermediate node, and of the destination, can be specified by any scheme (e.g., as an absolute address or as an X-offset and Y-offset). A non-source-routed message would not have any intermediate points specified in its header, just the destination, but such traffic can easily co-exist with source-routed traffic as both the schemes follow the same base routing strategy.

The hardware resources necessary for the implementation of source-directed routing would very much depend on how deadlock prevention is implemented. Since source-directed routing is implemented on top of the base routing scheme, the selection of certain intermediate points can potentially lead to deadlock (as the message is not stored and forwarded at any of the intermediate points).

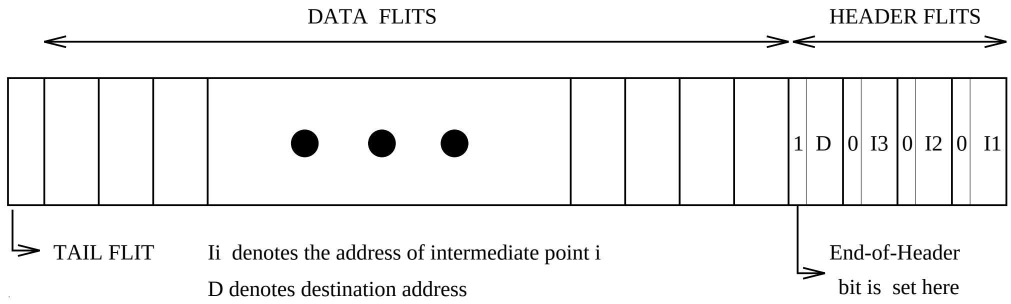


Figure 5.10: An example of a message with three intermediate points

5.3.2 Deadlock Prevention

If source-routed traffic is to be able to co-exist with non-source-routed traffic, then deadlock prevention cannot be "compiled in" simply by choosing intermediate points so that there are no routing cycles possible among the source routed messages. This might not be desirable in any case, owing to the restrictions it would place on allowable routes. Instead, it is necessary to extend the network topology as is done in adaptive routing algorithms, through use of virtual channels, so as to permit the route determination algorithm flexibility in choice of routes, while ensuring freedom from deadlock. Of course, these topology extensions may already be in place if the base routing scheme is adaptive.

One possibility is to use the virtual channel topology employed in the LH adaptive routing algorithm. This would permit the use, in source-directed routing, of any routes that lie on shortest paths between source and destination. Note that although the base routing strategy could be adaptive with such a topology, it need not necessarily be adaptive. (Use of deterministic routing would avoid the cost of router logic for input selection, output selection, and adaptive choice, as required for adaptive routing.) A problem with such a topology, however, is that the number of virtual channels required per physical channel is exponential in the dimensionality of the mesh. This is inconsistent with the desire for an implementation of source-directed routing with minimal additional hardware cost.

An alternative solution is to employ the virtual channel topology used in the *-Channels algorithm. As with the LH topology, this will permit the use, in source-directed routing, of any routes that lie on shortest paths between source and destination. Again, the base routing strategy employed with such a topology certainly could be adaptive, but need not be. The principal advantage over the LH topology is that only two virtual channels are required per physical channel, in each dimension but one (in which no additional virtual channels are required), for a mesh of arbitrary dimensionality. Further, for the 2D mesh (in which the virtual channel requirements are the same), better performance was observed with this topology than with the LH topology, as will be seen in Section 5.4.

There is, however, a significant consequence to the use of the *-Channels topology for source-directed routing. When using the *-Channels topology, a non-star channel must be taken whenever a message is required to take a step other than that which it would have taken from that router if deterministic routing was employed. Consider a 2D mesh, for example, in which deterministic routing proceeds by first routing along channels in the X dimension, and then finally along channels in the Y dimension. In this case, when employing source-directed routing, a non-star channel must be taken whenever the next node in the specified path is due "north" or "south" of the current location (and thus requires use of only Y dimension channels), and that next node is not the final destination. (Note that there would be no point in specifying such an intermediate node if the final destination was also due "north" or "south". Thus, one has to eventually turn "east" or "west", implying that the movements "north" or "south" at this time violate the dimension ordering used in deterministic routing.) Recall, however, that in *-Channels, messages may not block waiting solely for a non-star channel; if all non-star channel(s) in the desired direction are occupied by other messages, and if the relevant star channel (in the non-adaptive direction) is idle [2], it must be taken. Therefore, when using the *-Channels topology, there is no guarantee that it will be possible to precisely follow the path preselected by the source. In such a case, the header flit specifying the respective intermediate node must be discarded, and routing of the message must then be done (according to whatever base routing strategy is followed, either deterministic or adaptive) to the next node in the specified list.

Experimentally, it was observed that such cases occur only under very heavy load, and even then, are very rare. Further, it may, in fact, be beneficial to deviate from the preselected route in such cases, as it may be an indication that load conditions in the network are somewhat different (then what was expected), at least at that particular time instant. Experiments were done with a modified strategy in which some blocking for use of non-star channels was allowed (one can prove that deadlock freedom is still preserved, as long as the message holding the desired non-star channel is travelling in the same direction as the blocking message), and did not find this strategy to be generally useful.

[2] One difficulty here is that it is necessary to determine what the non-adaptive direction is. In the example given, for the 2D mesh, one would need to know whether the next intermediate node, if any (and thus the destination), lies to the "east" or to the "west" of the current location. Also, for higher-dimensional meshes, there exists the problem of even knowing whether or not a desired step is in an "adaptive" direction. A general solution to both of these problems, for a n-dimensional mesh, is to add n bits to each of the specifications of an intermediate node in the message header. For all but the last dimension that is followed with deterministic routing, one of these bits would be used to indicate whether further travel in that dimension will be required, once the respective intermediate node is reached. The remaining bit indicates the direction of travel, in that dimension which would be followed next with deterministic routing, once the respective intermediate point is reached. For a 2D mesh, in fact only this latter bit is needed.

5.3.3 Route Determination

The determination of suitable routings for the collection of messages within a particular application communication phase may be performed either at compile-time, or at run time as part of an inspector-executor [59] approach, depending on when the required information is available. In this work, only the case of compile time route determination is considered, thus the primary concern is the quality of the routes produced by an algorithm, rather than the problem of minimising the algorithm execution time.

The algorithm that is proposed here uses the "affected area" concept of Zhong and Lo [65]. The affected area of a message is defined to consist of all of the physical channels that could be included in a minimal path from the source to the destination node. In a 2D mesh network, for example, the affected area for a message travelling from a source node $\{x_1, y_1\}$ to a destination node $\{x_2, y_2\}$ consists of all physical channels leading towards the destination node (and away from the source node) in the rectangle formed by the nodes $\{x_1, y_1\}$, $\{x_1, y_2\}$, $\{x_2, y_2\}$, and $\{x_2, y_1\}$. The algorithm proceeds by first labelling all physical channels in the network with a zero weight. Then, for every message, the weights of all physical channels that are in its affected area are incremented by one. The result of applying this procedure with a sample set of three messages is shown in Figure 5.11.

After the channel weights have been determined, the messages are ordered according to some criteria, and then each, in turn, is assigned a route. The route chosen is one among the possible routes that has the least maximum weight on any physical channel, as motivated by the fact that message delivery time is going to be largely determined by the maximum multiplexing factor encountered along the message path. Among all possible routes that share this least maximum physical

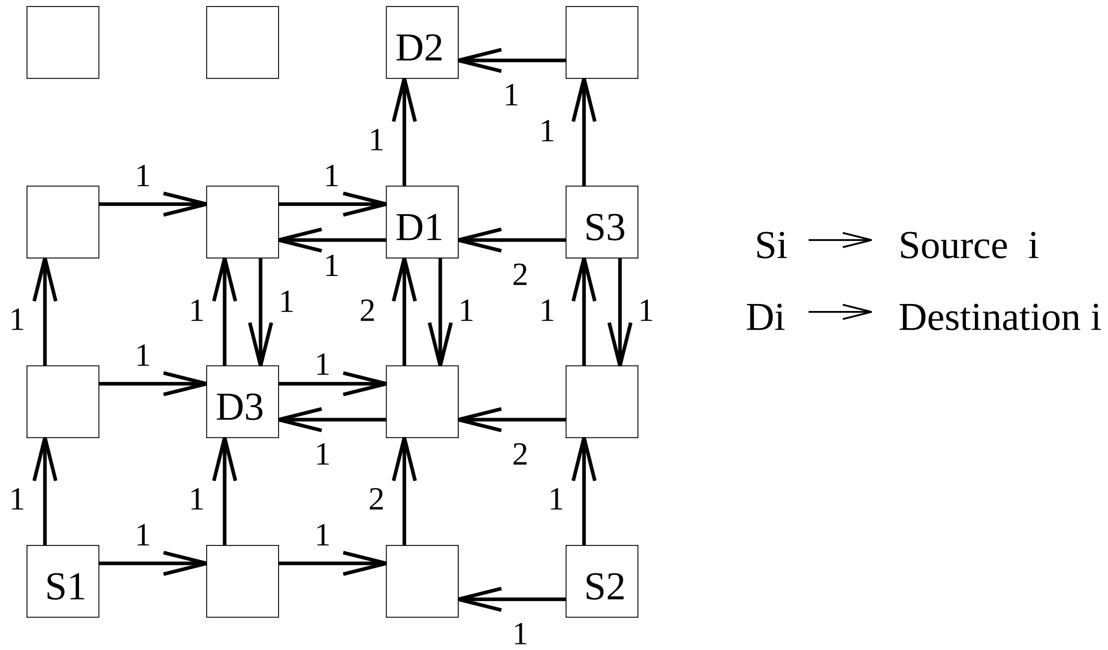


Figure 5.11: Channel labelling with a sample set of three messages, on a 2D mesh network; only physical channels with non-zero weights are shown

channel weight, the one selected is the one with the least number of turns, as motivated by the fact that a path with fewer turns will require the specification of fewer intermediate points, and thus would result in a smaller header overhead for the route specification. (In addition, paths with fewer turns may result in less blocking within the network.) Any shortest path algorithm applicable to acyclic graphs may be used to find a path with the lowest maximum physical channel weight. The search for paths can be made quite efficient by appropriately pruning a search tree, based on the already calculated lowest maximum physical channel weight, and the number of turns.

Once a path is chosen for a message, the weights of all the physical channels in its affected area are reduced by one (cancelling out the earlier incrementation by one), and then the weights of the physical channels on the chosen path are increased by a very large number (say 100). This increase is necessary, because when considering

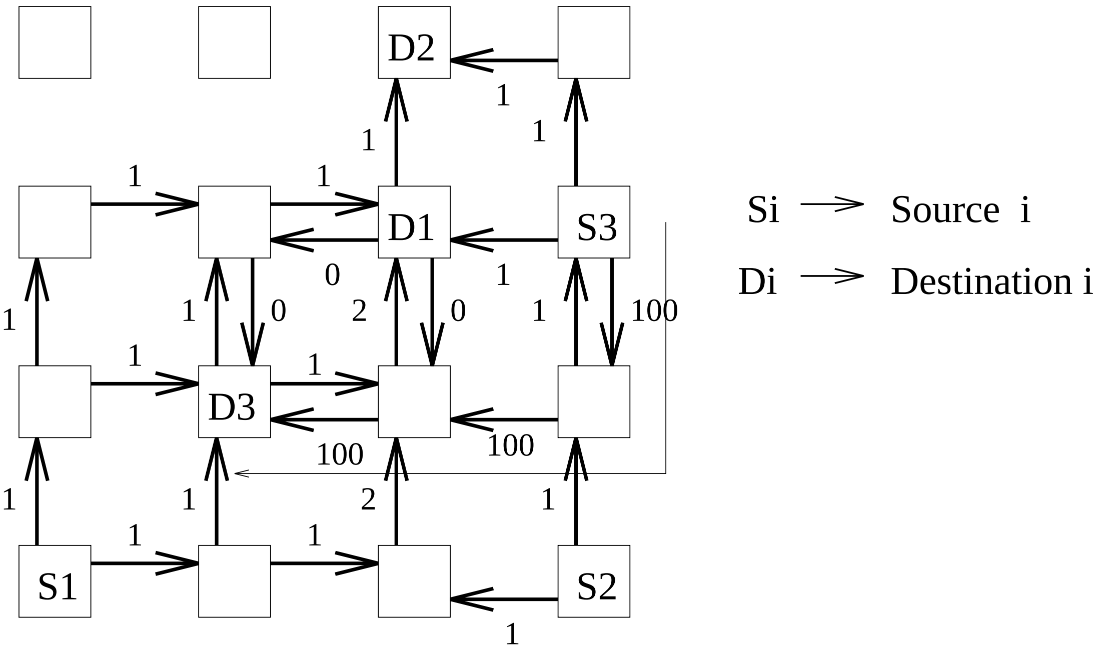


Figure 5.12: An example of updating physical channel weights upon route assignment; from Figure 5.11, the message from S3 to D3 has been considered first, and a path has been selected for it with least maximum weight.

multiple possible paths, the one that has the least number of messages already assigned to it should, in general, be chosen. The Zhong and Lo algorithm, which uses the same general concept of affected area, does not, however, differentiate between a chosen path and a potential path very clearly, as their algorithm increases the weight of the chosen path by just one. Experiments performed with various algorithm variations suggest that this can lead to uneven link loads and higher maximum message latencies. Figure 5.12 shows an example of how weights are changed by the algorithm proposed here once a message has been assigned a path.

Three different message orderings for route assignment were tried in this experimental study: in increasing order of the distance from source to destination, in decreasing order of the distance from source to destination, and in increasing order of the number of paths from source to destination, or "freedom" in the terminology of Zhong and Lo [65]. (For a 2D mesh network, if the difference between the X coordinates of source and destination is dx and the difference between the Y coordinates is dy, then the freedom value is $C_{dx}^{(dx+dy)}$.) No one ordering was found to yield uniformly better or poorer results; in fact, for each ordering there were cases in which it yielded substantively better results than the other two. Thus, the algorithm chosen for further use actually assigns routes three times, using each of the three orderings, and then picks the best overall route assignment out of the three. The metric used in this selection is the maximum multiplexing level (i.e., the maximum number of messages whose routes include the same physical channel), and if this is not sufficient to differentiate between route assignments, then the number of physical channels on which that maximum is attained. (And if even this is not sufficient to differentiate between route assignments, then the next highest level of multiplexing is examined.) This selection criterion was found to correlate well with the resulting network performance. Figure 5.13 show the percentages of times that each ordering was selected, according to the above criteria, in experiments (described in Section 5.4) with general traffic patterns, on a 12×12 mesh, with various numbers of nodes generating messages.

The complexity of the route determination algorithm can be analysed as follows. Let M denote the number of messages and P denote the number of physical channels in the network. The initial labelling would require $O(MP)$ time and the sorting would take $O(Mlog(M))$ time and finally choosing a path can be done in $O(MP)$ time [65] by using shortest path algorithms for acyclic graphs. Thus the final complexity of the route determination algorithm is $O(M(P + log(M)))$. Empirically, it was found that the running time of the route determination algorithm, as employed for the experiments, was typically about 2 seconds on a DEC Alpha

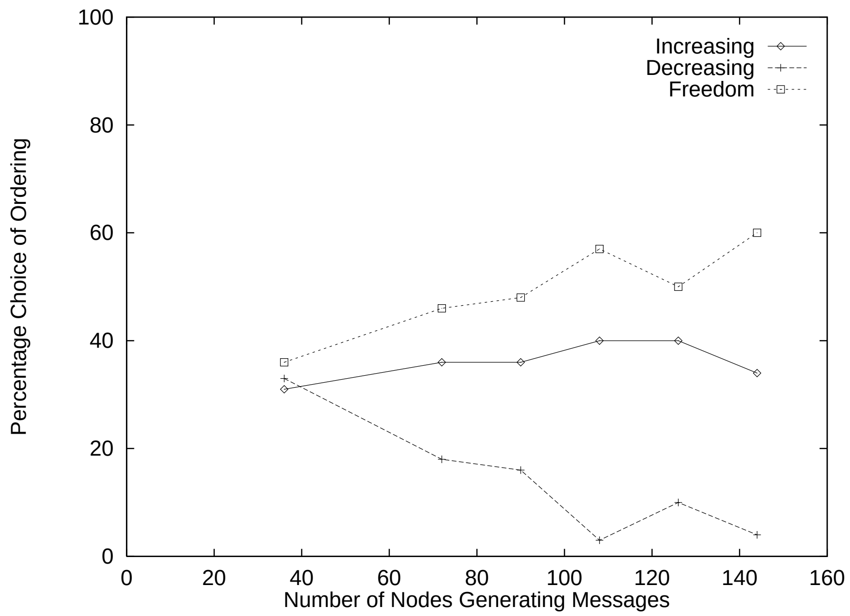


Figure 5.13: Frequency of choice of each message ordering by the route determination algorithm, for general traffic patterns on a 12×12 mesh, with various numbers of nodes generating messages

3000 workstation.

5.4 Experimental Study

Simulation experiments were carried out to compare source-directed routing to deterministic and adaptive (specifically *-Channels) routing, with respect to the performance achieved. Section 5.4.1 describes the methodology that was followed in these experiments. Section 5.4.2 presents the principal results, focusing on completion times for various patterns of message exchanges. The performance measure of average message latency is considered in Section 5.4.3. Experiments with hybrid traffic patterns are considered in Section 5.4.4. Finally, in Section 5.4.5, the impact of various algorithm, architecture, and workload variations is considered.

5.4.1 Methodology

Experiments were performed using a flit-level network simulator that has the flexibility to support various topologies, routing algorithms and traffic patterns. As in Chapter 4, a 2D mesh network of size 12×12 was assumed for this study.

Each pair of neighbouring nodes in the mesh was assumed to be connected by a pair of (uni-directional) physical channels. Each router was assumed to be connected to the processor by means of a single consumption channel and a single injection channel, each with the same number of virtual channels as the other physical channels in the network. The number of virtual channels per physical channel was varied during the experiments. The flit transfer time between two nodes was set to be 1 clock cycle. Finally, the size of each message (excluding the header) was fixed as 128 flits in most of the experiments. Since the simulation running times for the experiments performed in this chapter are not very long, larger message sizes could easily be considered. Experiments were also performed with a message size of 20 flits, as used in the experiments reported in Chapter 4, and these are reported in Section 5.4.5.

In this study a "single communication phase" workload model (introduced in Chapter 4) is used as the primary workload model. In this context, source-directed routing should provide near optimal performance, and would provide a basis against which the performance of adaptive routing, and the scope for further improvements beyond this performance, can be judged. "Multiphase" and open workload models are also considered to a limited extent. Both randomly selected "general" traffic patterns and sample specific patterns that have been employed in previous studies of routing algorithm performance were used in the experiments conducted here. In the general patterns, some fraction of the nodes in the mesh (typically all of them) generate a single message that is destined to each other node in the mesh with equal probability. Simulations were performed for sufficient instances of these general patterns, such that the average of the completion time values, and the average message latencies, were obtained with a confidence interval width of at most 3% at a 95% confidence level.

The principal traffic pattern for which results are presented is matrix transpose, in which each node (i, j) for $i \neq j$ sends a single message to node (j, i). Two other patterns for which results are presented to varying extents are center reflection and bit reversal. (The latter somewhat modified from its standard description owing to the size of mesh considered) In the center reflection pattern, each node (i, j) in a $n \times n$ mesh sends a message to node $(n-1-i, n-1-j)$ (assuming the nodes are indexed from $(0, 0)$ to $(n-1, n-1)$). In the bit reversal pattern, each node (i, j) sends a message to the node (m, n) such that the binary representation of m is the reversal of that of i, and that of n is the reversal of that of j. As the mesh size is not a power of two, this pattern is slightly modified by taking the "reversal" of 3 as 7, the reversal of 7 as 11, and the reversal of 11 as 3.

The source-directed routing method described in Section 5.3 is employed with a *-Channels topology and with a deterministic base routing algorithm. In Section 5.4.4, however, some results for source-directed routing using an adaptive base routing algorithm and for source-directed routing using a LH topology are shown. In all cases, it is assumed that intermediate points are specified in the header so as to fully determine the desired route. Thus, the distinction between an adaptive and deterministic routing base only affects the source-routed traffic when an intermediate point must be skipped, owing to an inability to take an "adaptive" step, as might occur when employing the *-Channels topology.

5.4.2 Completion Time Comparisons

Figure 5.14 compares the performance of adaptive and deterministic routing algorithms against that of source-directed routing for a matrix transpose traffic pattern. The lower bound on completion time derived earlier in this chapter is also shown. Note the large gap between the completion time provided by adaptive routing, and the lower bound. This gap results from the inability of adaptive routing to avoid making at least a few routing "mistakes" (choices of route that diminsh performance), owing to its use of purely local knowledge in routing decisions.

Source-directed routing mostly closes this gap by making use of global knowledge of traffic patterns. It yields near optimal performance when there are at least four virtual channels per physical channel. Results for general traffic patterns are shown in Figure 5.15 (32 nodes generating messages) and Figure 5.16 (all 144 nodes generating messages). Note that the performance improvements offered by adaptive routing, at least in terms of the communication phase completion time, are very small for these general patterns. Source-directed routing, in contrast, yields near optimal performance, excepting for the case of heavy load (Figure 5.16) and only two virtual channels per physical channel. In this case, there is an unavoidably large amount of blocking that occurs in the network, and all routing methods perform similarly. Note that the specific traffic patterns correspond to heavy load conditions as well (as they involve most if not all nodes generating messages), and thus a similar degradation in the performance of source-directed routing is also seen for them, to varying degrees, for low numbers of virtual channels. It is important to keep in mind that, as illustrated by Figure 5.15, this degradation does not occur for more moderate loads.

Figures 5.17, 5.18, and 5.19 show detailed results corresponding to the average results shown in Figure 5.16, for general traffic patterns, and four virtual channels per physical channel. The figures show, for each routing algorithm, the number of instances of the general traffic pattern (among 100 such instances in total) such that the completion time for that pattern fell into each of the histogram intervals. It is interesting to note that regular peaks appear with a spacing of about 128 cycles (the message size). This is evidence that completion time is closely linked to the maximum multiplexing faced on any physical channel; if that maximum multiplexing is increased by a factor of l, for example, then the completion time is often in the neighbourhood of $128l$. Source-directed routing does well at minimising the maximum multiplexing, and thus the maximum message latency, and thus the completion time of the communication phase.

To illustrate this last point, Figures 5.20, 5.21, and 5.22 show the distribution of message delivery times for deterministic, adaptive, and source-directed routing, respectively, for the matrix transpose traffic pattern and four virtual channels per physical channel. Note that with source-directed routing, the distribution is sharply truncated, with a high peak at a little over 500 cycles, and no messages with higher latencies. With deterministic routing, in contrast, messages experience widely varying latencies, and the maximum message latency (and thus the completion time) is much higher. Adaptive routing certainly improves the average delivery time, but is unable to avoid a few long latencies (specifically, 19 message latencies are greater than 800 cycles), thus inflating the completion time of the communication phase.

Results for the center reflection traffic pattern are shown in Figure 5.23. This traffic pattern is distinguished by the fact that deterministic routing has been shown to yield close to optimal performance, while adaptive routing algorithms have been shown to perform badly with small number of virtual channels [32]. As shown in the figure, source-directed routing performs as well as is possible. Results for the bit reversal traffic pattern are shown in Figure 5.24; in this case, results quite similar in nature to those for the general traffic patterns are obtained.

5.4.3 Average Message Latencies

Previous studies of routing protocol performance have typically emphasised the average message latency, rather than the time to complete a communication phase consisting of a number of message exchanges. To illustrate the distinctions between these two measures, Figure 5.25 shows average message latencies for the matrix transpose traffic pattern. Comparing these results to those in Figure 5.14, it can be seen that adaptive routing yields much better performance (in comparison to source-directed routing) with respect to average message latency, than it does with respect to completion time. This is explained by the fact that the time to complete a communication phase depends on the maximum message latency, rather than on the average, and is therefore sensitive to even just a few routing "mistakes". Source-directed routing yields its largest benefits (in comparison to both adaptive and deterministic routing) with respect to completion times, rather than with respect to average latencies. In fact, the average latencies that are observed in the experiments with general traffic patterns are only modestly better (by about 20 %) than those observed with deterministic or adaptive routing, even though the average benefit with respect to completion time (as shown in Figures 5.15 and 5.16) is large. Figure 5.26 shows similar performance for bit reversal traffic.

Average latency can more directly determine performance in asynchronous computations in which multiple communication phases may overlap. This was studied in experiments with multiple phases of matrix transpose or bit reversal communication.

Recall that in each of these traffic patterns, each node involved in communication sends and receives exactly one message within a communication phase. In the multiple phase experiments, each node is assumed to compute for some fixed length of time after receiving the message generated for it in one phase of communication, after which it sends its message for the next phase of communication. Note that there is no global synchronisation that keeps nodes together in terms of which phase of communication/computation each is at, and thus different phases of communication may overlap.

Figures 5.27, 5.28, 5.29, and 5.30 show results for the total completion time and average message latency of 30 asynchronous phases (including both communication and computation), with matrix transpose and bit reversal traffic patterns. These figures show a very good correlation between the total completion time of thirty phases, and the average message latency. Note also that as the compute time becomes large, the completion time is increasingly dominated by computation alone, and thus the differences between the three routing algorithms in terms of this completion time become small.

Source-directed routing was also tested with an open workload model in an experimental setup similar to that in the previous chapter. Figure 5.31 compares the performance provided by source-directed routing (both on an adaptive base and a deterministic base) against that with *-Channels adaptive routing and deterministic routing for a matrix transpose traffic pattern. This figure illustrates that source-directed routing yields performance similar to that of adaptive routing in this context and does not provide any additional benefits. Experiments with the open workload model further strengthens the conjecture that adaptive routing yields performance close to the best that could be realistically achieved, when performance is measured by average message latency.

5.4.4 Hybrid Traffic Experiments

Experiments were also performed with workloads in which a percentage of the traffic was unpredictable. In these experiments, initially, some percentage of source-destination pairs are known and route determination is performed based only on this known traffic. Other nodes generate random traffic in which each node is the destination node with equal probability. Figure 5.32 shows the improvements (in terms of completion time), for a general traffic pattern, that the source-directed routing algorithms can provide over adaptive and deterministic routing as the knowledge of the traffic pattern increases. A similar experiment was also performed for a workload in which a percentage of source-destination pairs communicate with a known matrix transpose type of traffic. The remaining nodes, as before, engage in random communication. The results of this experiment are shown in Figure 5.33.

These experiments show that as the knowledge of the traffic pattern increases, the gains become quite substantial.

5.4.5 Impact of Other System, Workload, and Policy Assumptions

A number of experiments were performed with various algorithm and architecture variations; some of the results are described here. First, source-directed routing was implemented with a LH topology rather than with a *-Channels topology. As Figure 5.34 indicates for the case of general traffic patterns, the performance with a *-Channels topology was found to be superior. (Note that the virtual channels in the Y dimension, with the LH topology, must be split into two groups reserved for "eastbound" and "westbound" messages respectively; thus, for this topology only cases of even numbers of virtual channels were considered.)

Second, experiments were performed with source-directed routing (using the *-Channels topology) as implemented on an adaptive rather than deterministic routing algorithm base. As shown in Figure 5.35 for the case of general traffic patterns, this resulted in small improvements in performance only when the network load was heavy and there were only two virtual channels per physical channel. Under such conditions, it is more frequently necessary to deviate from the preselected route, owing to an inability to take an "adaptive" step. Thus adaptivity in the base protocol may be more frequently exercised.

Third, fewer intermediate points in the message headers were specified, so as to achieve a blend of global route determination and local adaptivity. In the experiments that were performed here, this hurt performance rather than helped. It might be thought that one advantage might be to reduce header overhead, but in fact, header overhead was not found to be a significant factor. Since the route determination algorithm attempts to choose a message route (among those with the same maximum multiplexing factor) with the fewest turns, on average messages required the specification of very few intermediate points. For the general traffic patterns, for example, the average number of intermediate points was found to be about 1.6.

Finally, the message lengths were found to have some impact on the performance improvements offered by source-directed routing. With shorter messages, the gains are slightly less as is illustrated in Figure 5.36, which shows performance results for general traffic patterns, with a fixed message size of 20 flits. (Compare to Figure 5.16.) Experiments were also performed with hybrid message sizes. These did not appear to adversely impact source-directed routing, as long as the route determination algorithm was modified so that the weights assigned to each physical channel are proportional to the length of the message. For example, in the initial channel labelling, if message lengths are bimodally distributed between 20 flits and 100 flits, the weighting could be 1 for each 20 flit message and 5 for each 100 flit message.

5.5 Conclusions

This chapter considered the question of how much room for improvement there may be in the network performance provided by (locally) adaptive routing policies. This issue was addressed in two ways. Initially, bounds were derived on maximum achievable performance, in terms of throughput, latency, and completion time. These results suggest that there may be only a moderate room for improvement (by about 20%) beyond the performance provided by adaptive routing, with respect to average latency and throughput. In terms of completion time it appears that there may be a large scope for improvement.

Next, limits on realistically achievable performance were quantified by considering source-directed routing methods, under the favourable assumption of full predictability of traffic pattern. A specific source-directed routing method was proposed, of interest in its own right, and its performance was studied in comparison to that of adaptive and deterministic routing in a set of simulation experiments on 2D meshes.

Simulation results support the hypothesis that there is only a moderate room for further performance improvement beyond that offered by (locally) adaptive routing, when performance is dictated by average message latency. When performance is determined more by maximum message latencies (such as may be the case when application performance is limited by the time to fully complete a message exchange phase), it appears that adaptive routing techniques are less effective, and that there is large scope for improvement.

Use of source-directed was observed to improve performance by upto 50 %. The specific method proposed here can interoperate with either deterministic or adaptive routing, so as to allow these latter techniques to be used as desired when message patterns are unpredictable. Further, it imposes minimal additional router cost beyond that required anyway for the deterministic or adaptive routing technique that is otherwise employed.

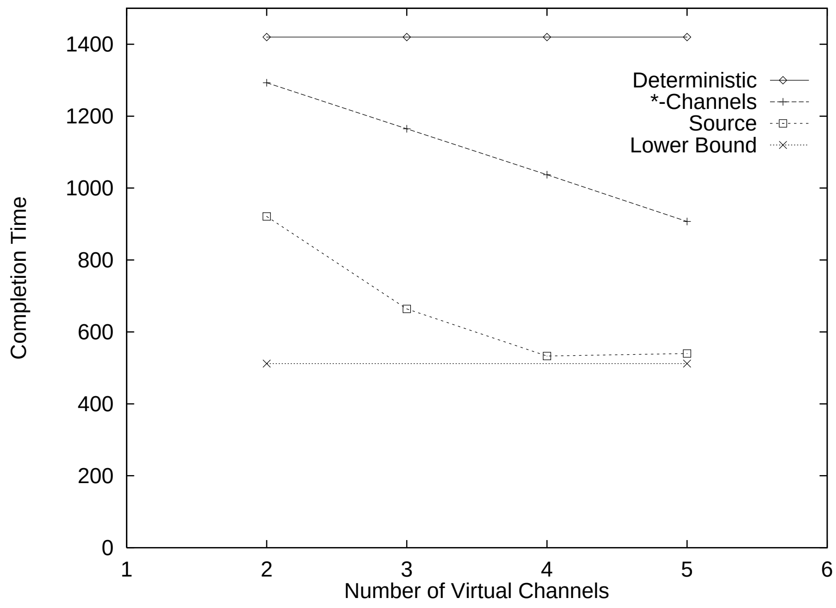


Figure 5.14: Average completion times for deterministic, adaptive and source-directed routing algorithms, for matrix transpose traffic pattern

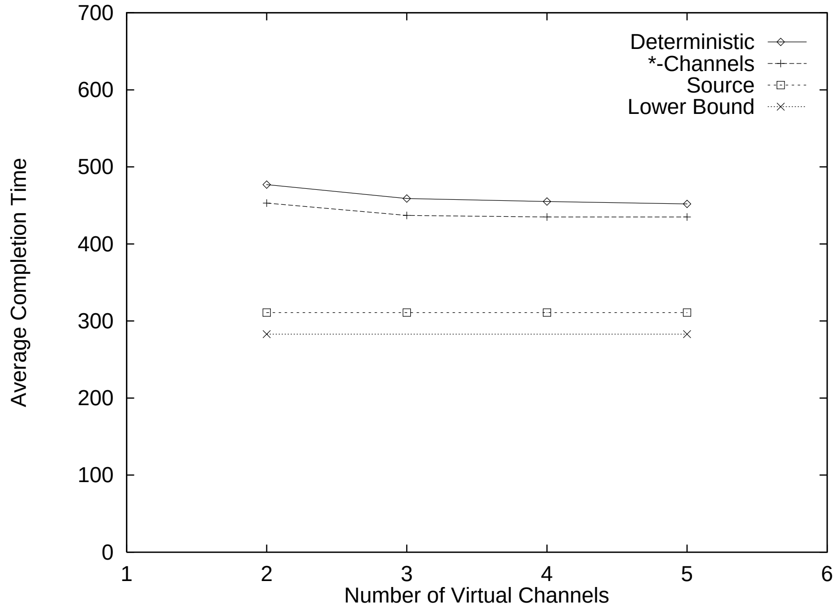


Figure 5.15: Average completion times with deterministic, adaptive and source-directed routing algorithms, for general traffic patterns, with only 32 nodes generating messages

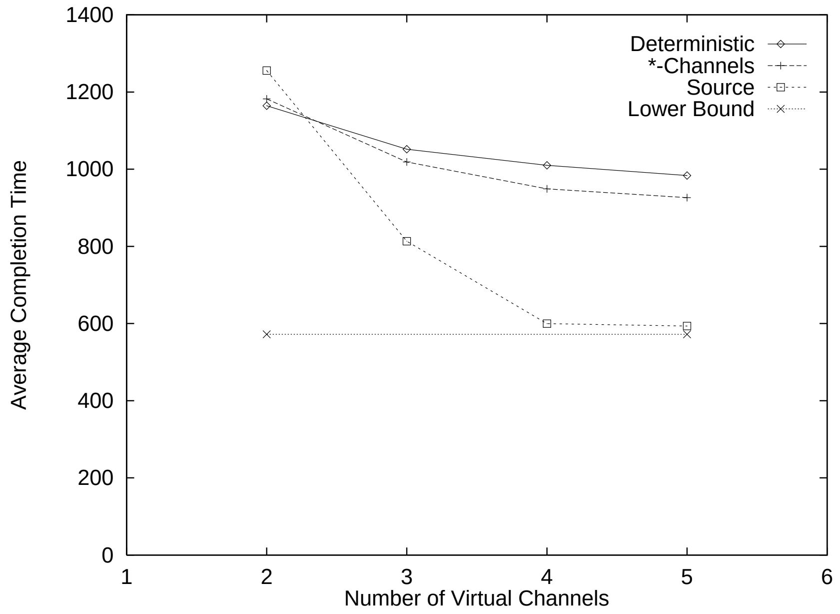


Figure 5.16: Average completion times with deterministic, adaptive and source-directed routing algorithms, for general traffic patterns, with all 144 nodes generating messages

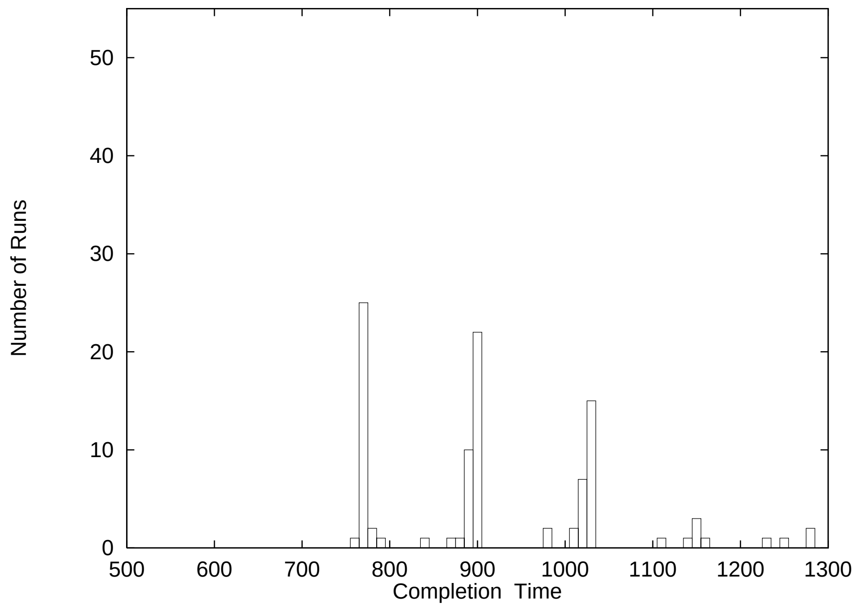


Figure 5.17: Histogram of completion times with deterministic routing

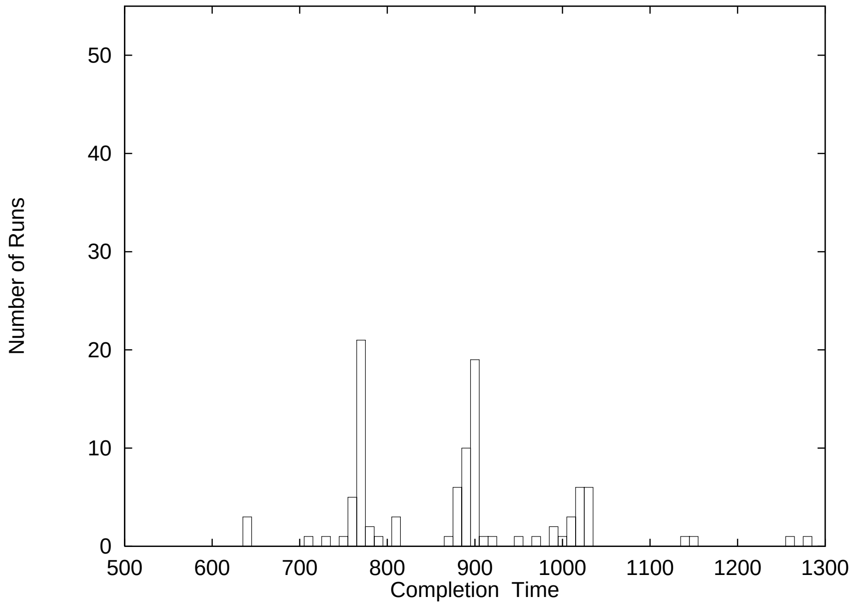


Figure 5.18: Histogram of completion times with the *-Channels adaptive routing algorithm

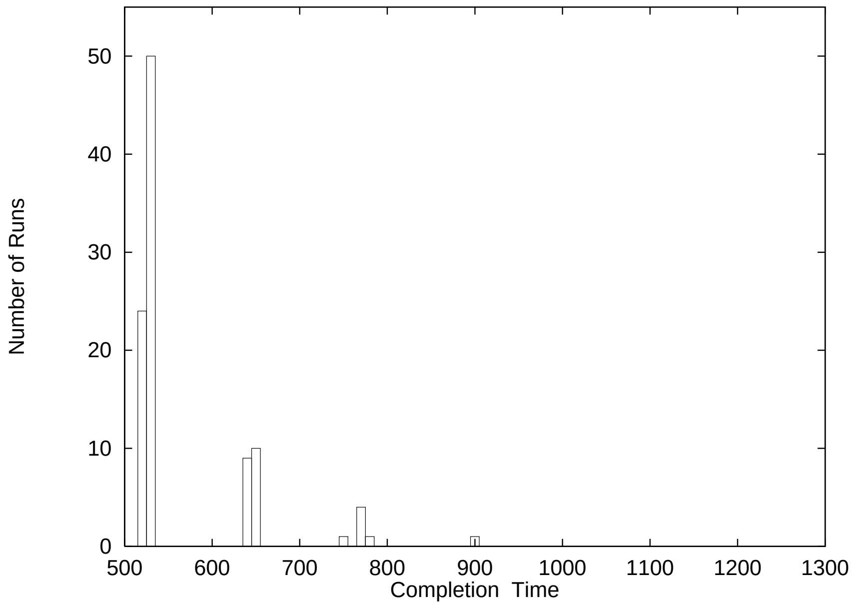


Figure 5.19: Histogram of completion times with the source-directed routing algorithm

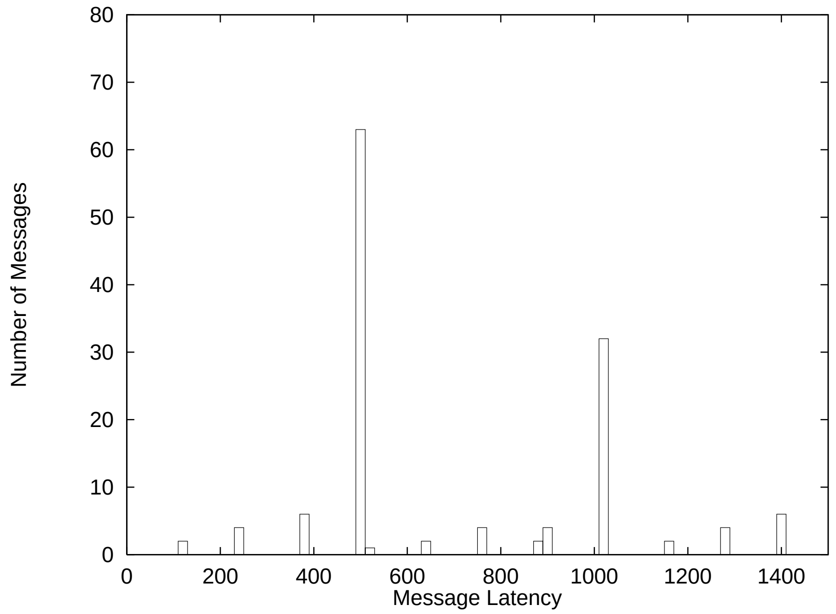


Figure 5.20: Histogram of message latencies with deterministic routing, for matrix transpose traffic pattern

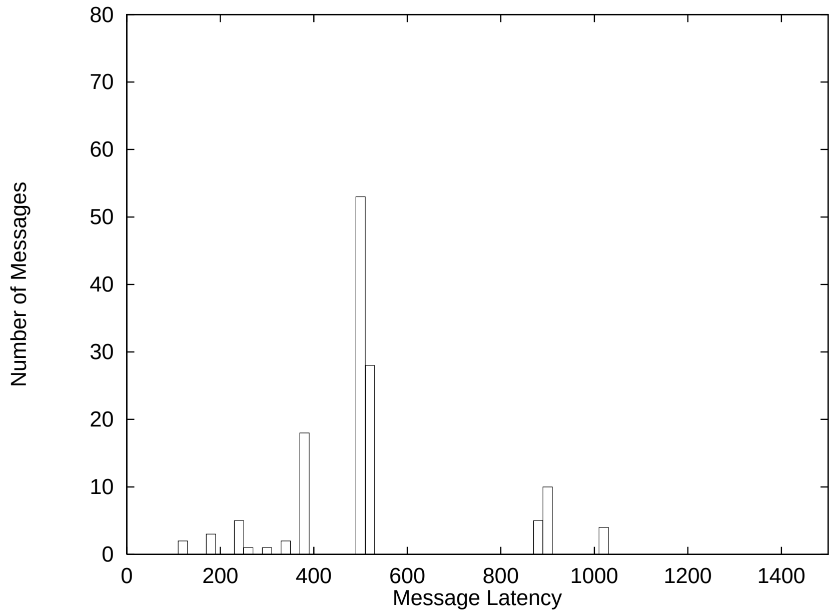


Figure 5.21: Histogram of message latencies with the *-Channels routing algorithm, for matrix transpose traffic pattern

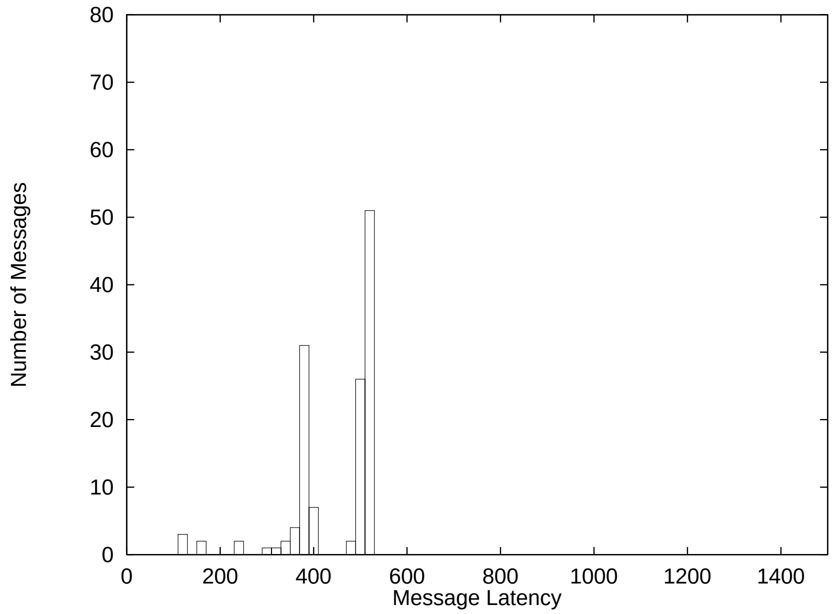


Figure 5.22: Histogram of message latencies with the source-directed routing algorithm, for matrix transpose traffic pattern

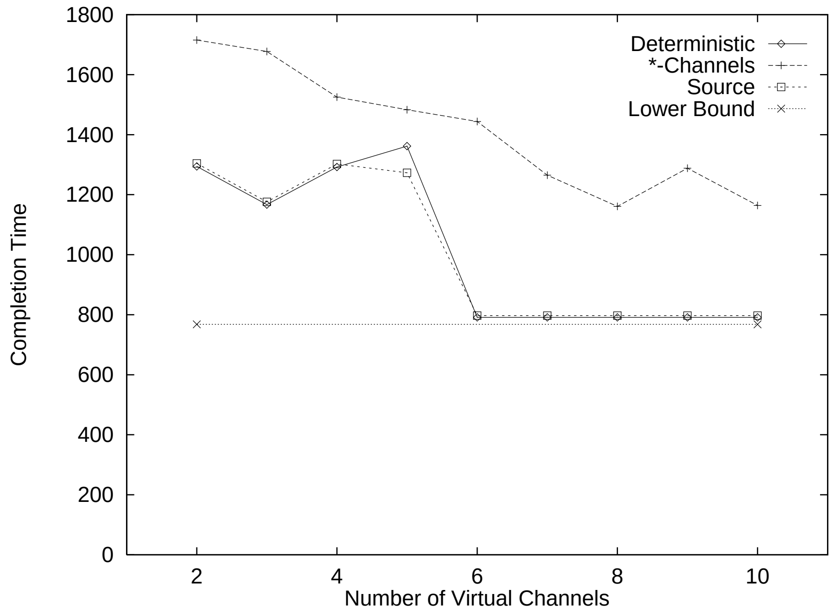


Figure 5.23: Completion times for deterministic, adaptive and source-directed routing algorithms, for center reflection traffic pattern

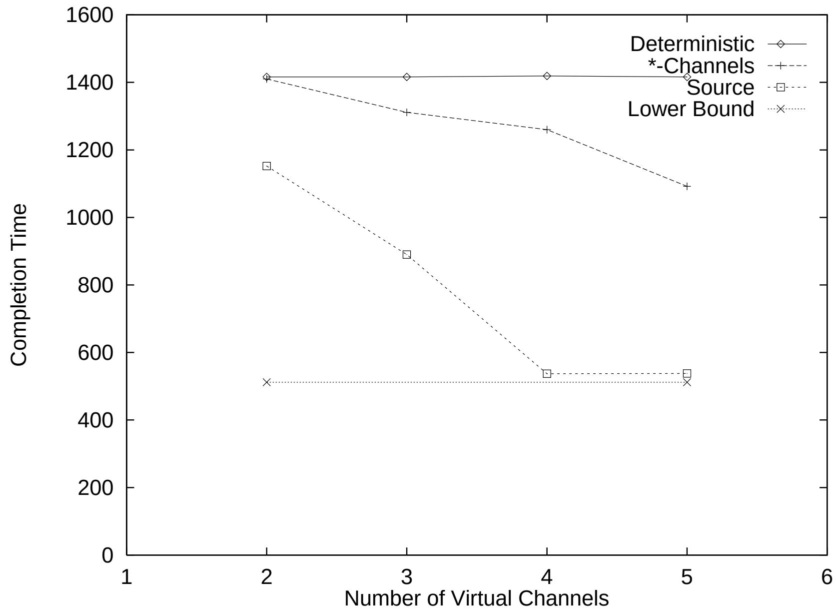


Figure 5.24: Completion times for deterministic, adaptive and source-directed routing algorithms, for bit reversal traffic pattern

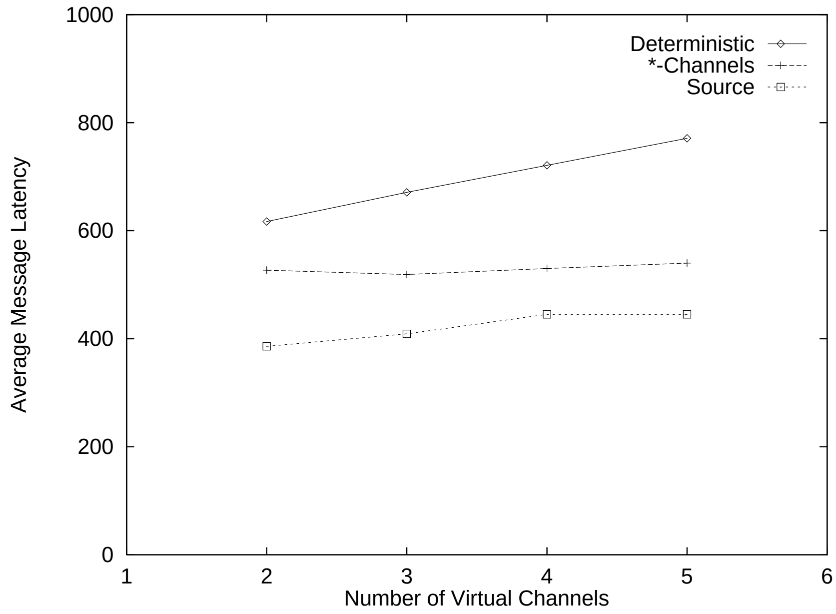


Figure 5.25: Average message latencies with deterministic, adaptive and source-directed routing algorithms, for matrix transpose traffic pattern

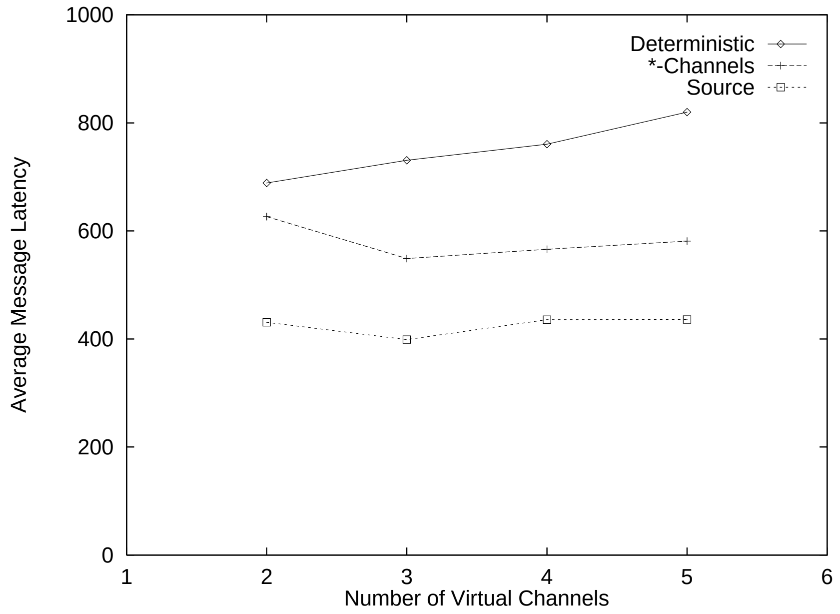


Figure 5.26: Average message latencies with deterministic, adaptive and source-directed routing algorithms, for bit reversal traffic pattern

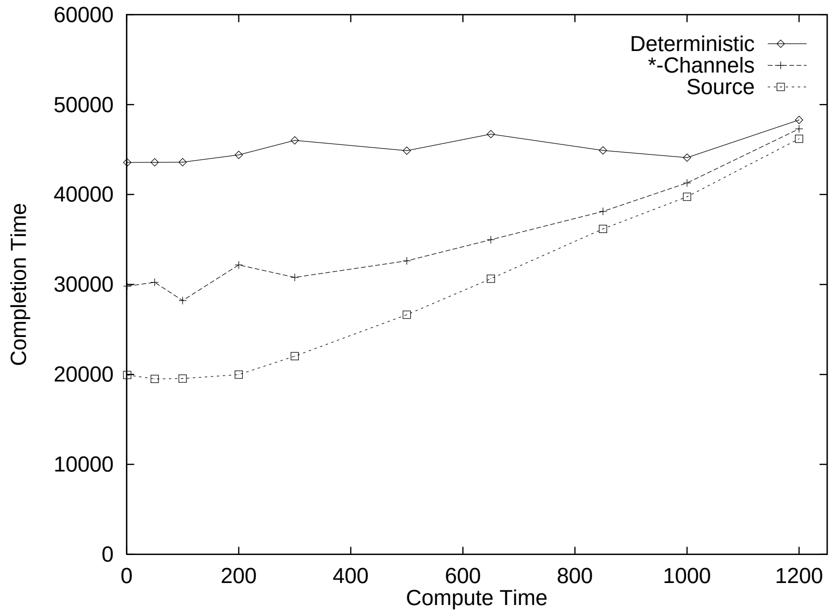


Figure 5.27: Completion times with deterministic, adaptive and source-directed routing algorithms, for thirty asynchronous bit reversal message exchange phases

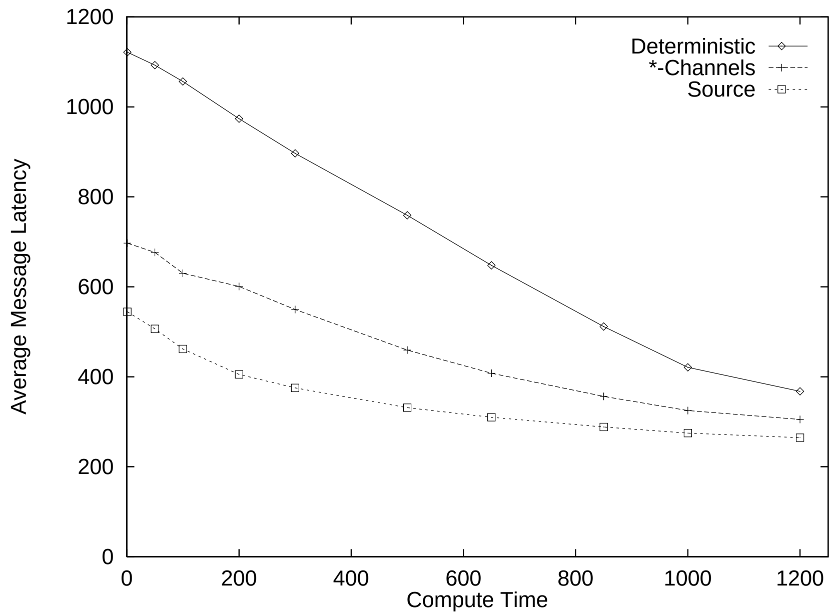


Figure 5.28: Average message latencies with deterministic, adaptive and source-directed routing algorithms, for thirty asynchronous bit reversal message exchange phases

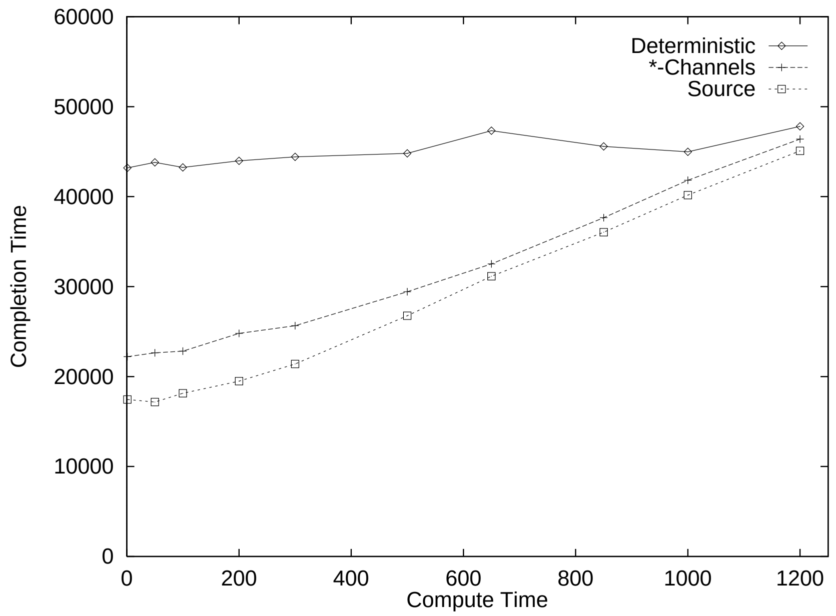


Figure 5.29: Completion times with deterministic, adaptive and source-directed routing algorithms, for thirty asynchronous matrix transpose message exchange phases

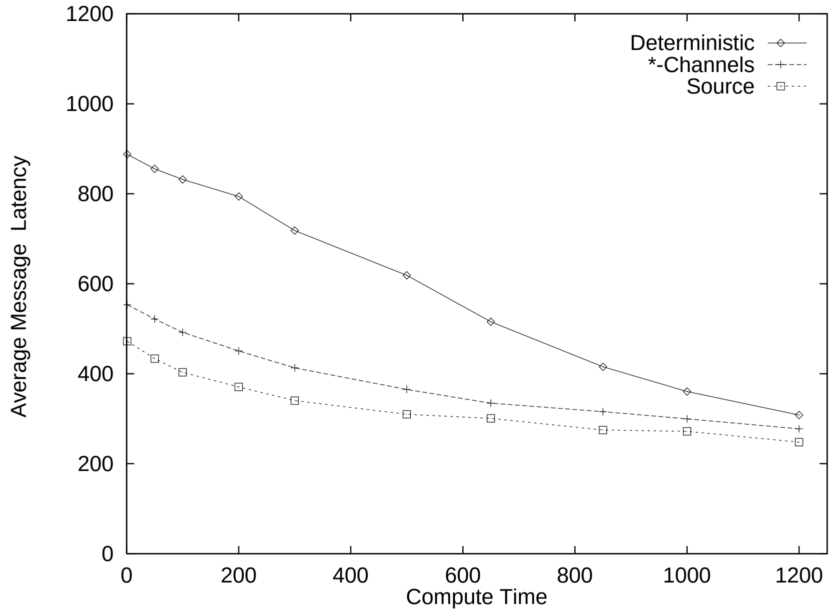


Figure 5.30: Average message latencies with deterministic, adaptive and source-directed routing algorithms, for thirty asynchronous matrix transpose message exchange phases

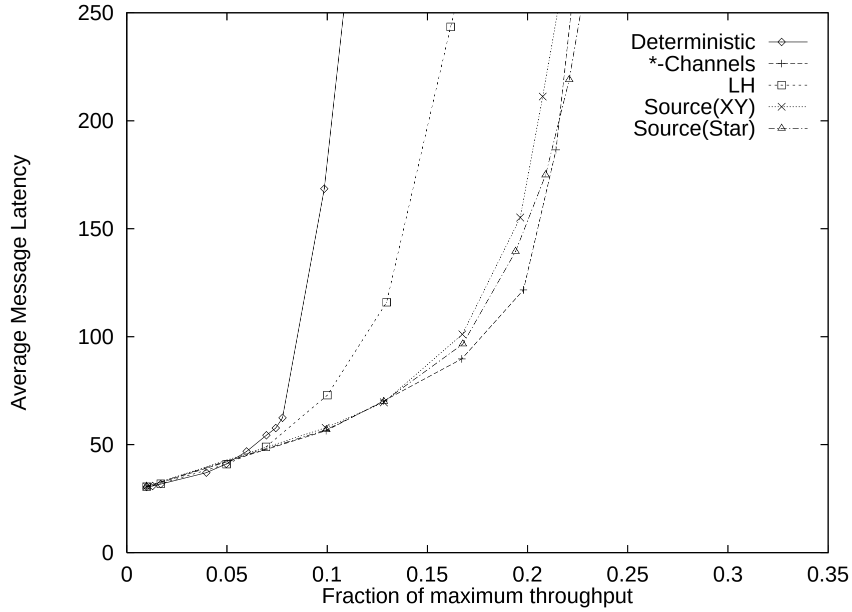


Figure 5.31: Comparison of deterministic, adaptive and source-directed routing algorithms for a matrix transpose traffic pattern with 6 virtual channels in the X and Y dimensions

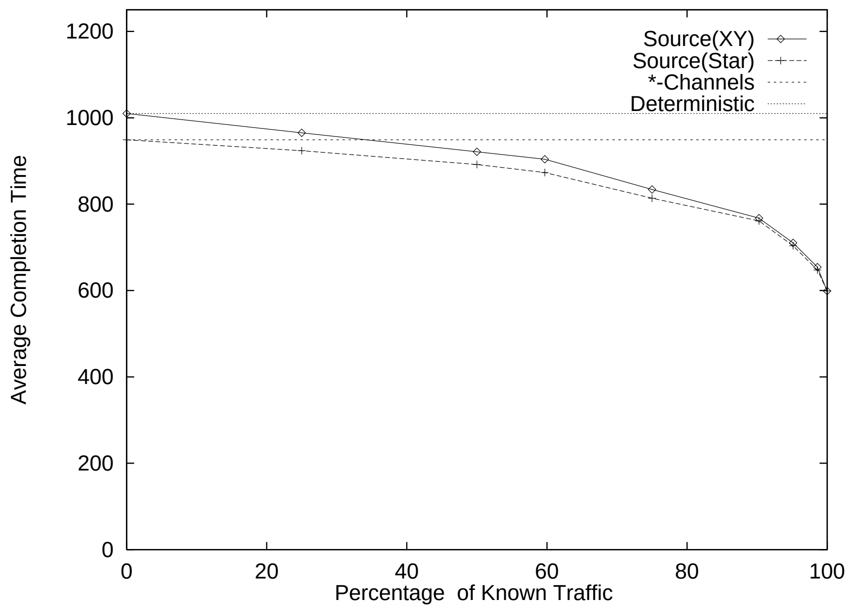


Figure 5.32: Performance of source-directed routing, for general traffic patterns, with only a percentage of traffic known in advance

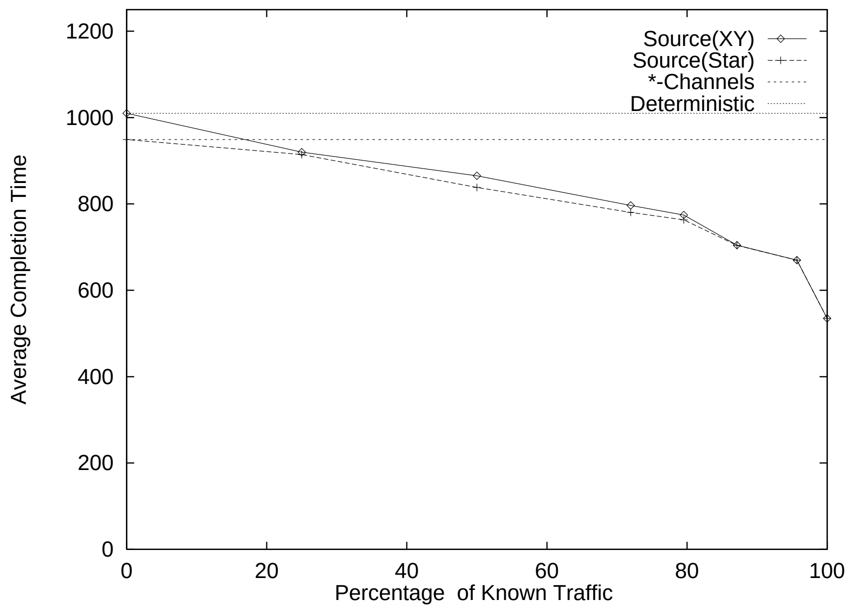


Figure 5.33: Performance of source-directed routing, for a matrix transpose traffic, with only a percentage of traffic known in advance

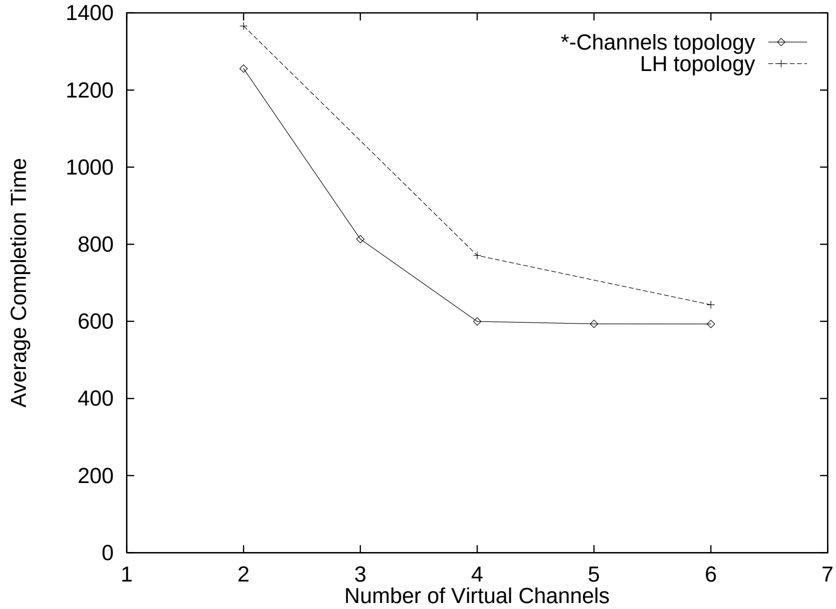


Figure 5.34: Average completion time with source-directed routing on a Linder and Harden topology in comparison to that on a *-Channels topology, for general traffic patterns

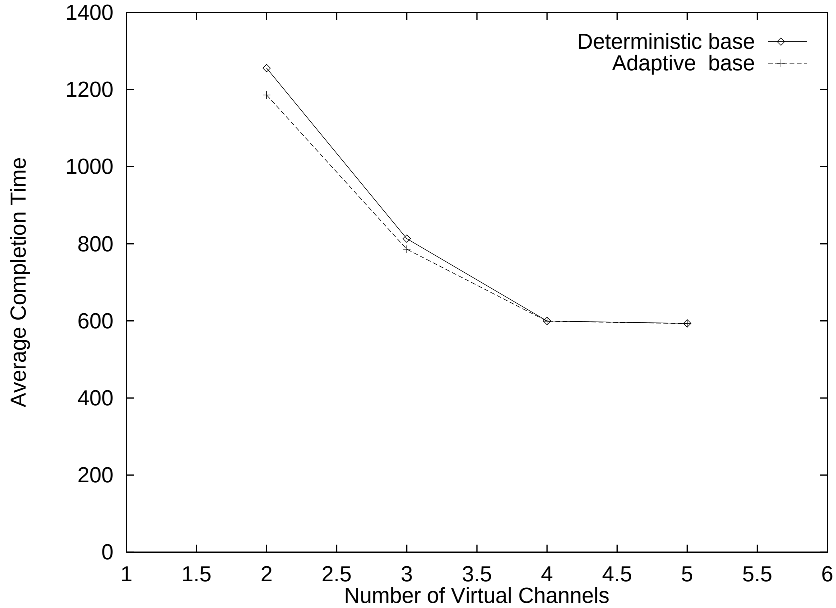


Figure 5.35: Average completion time with source-directed routing on an adaptive routing base in comparison to that on a deterministic routing base, for general traffic patterns

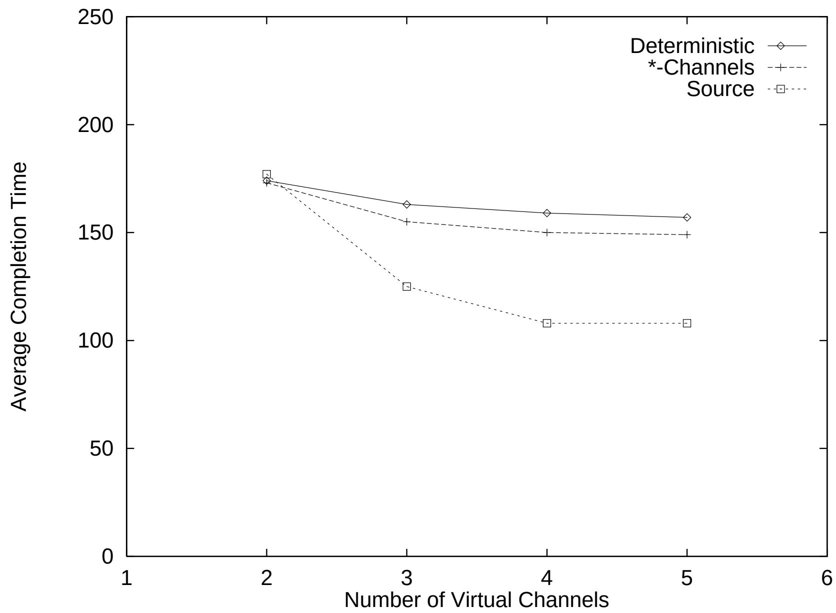


Figure 5.36: Average completion times with deterministic, adaptive and source-directed routing algorithms, for a message size of 20 flits, and general traffic patterns

Chapter 6

Fault Tolerance

The interconnection network is a critical component of a multiprocessor system as it plays a major role in the performance and reliability of the entire system. In large scale and massively parallel systems, hardware faults could become relatively common, and the network must be robust enough to provide routing capability around faulty regions. This motivates the design of mechanisms for coping with interconnection network faults.

Future networks are likely to have multiple virtual channels for every physical channel for the purposes of VCFC and/or adaptive routing. In this chapter, a new strategy is proposed for making such networks fault tolerant, through reconfiguration of their virtual channels when faults occur. The proposed scheme allows deadlock-free and livelock-free routing around various node and link faults as long as "enough" virtual channels are defined on each physical channel in the network (at least 3 for every physical channel in the case of a n-dimensional mesh). These virtual channels may be used for VCFC and/or adaptive routing when not required for routing around faults. No additional hardware in terms of extra physical channels is required.

The rest of this chapter is organised as follows. Section 6.1 discusses the proposed approach to fault tolerance (termed "Virtual Channel Reconfiguration" (VCR)) for the case of 2D mesh networks employing deterministic routing. The use of the VCR method with adaptive routing and VCFC is outlined in Section 6.2. Section 6.3 discusses how the VCR approach can be extended to work for higher dimensional meshes, and tori. In Section 6.4, other approaches to fault tolerance are discussed and compared with the VCR approach. Section 6.5 describes a performance study of the proposed approach. Conclusions are presented in Section 6.6.

6.1 Virtual Channel Reconfiguration

In this section, an approach to fault tolerance termed Virtual Channel Reconfiguration is presented, for the context of a 2D mesh network employing deterministic routing. Informally, the idea behind this approach is to reconfigure the virtual channels at the nodes surrounding each fault, so as to permit deadlock-free routing around it.

Node and bi-directional link faults are treated. In the description below link faults are primarily considered, as a node fault can be treated as a special case of multiple link faults where all the links associated with a particular node have failed. The set of faulty nodes and links constitute the fault topology of the network. In theory, fault topologies can be arbitrarily shaped (for example, like a maze) and getting past such topologies in an effective manner might be very difficult, and may not even be possible. Designing a fault tolerance scheme for such complex cases might prove to be too expensive (in terms of resources required) and wasteful, as such cases are very unlikely to occur in practice. At the same time, it is necessary that the fault tolerance approach be able to handle large sets of simple/complex faults that are more likely to occur.

Thus, the primary motivation in this chapter is to design a fault tolerance scheme that is simple and cost effective, and at the same time powerful enough to handle a large set of fault topologies. Since both node and link faults are likely in interconnection networks, the fault tolerance scheme should be sufficiently general to handle both of these. Of interest here are schemes based on reconfiguration of virtual channels; for such an approach to be cost effective the requirement in terms of number of virtual channels should be low. Also, there should not be a necessity to specially tag/mark messages at any node. Use of tagging may significantly increase router complexity. Further, routing at intermediate nodes should employ only knowledge of the "direction of travel", as in conventional deterministic or adaptive routing, rather than knowledge of the precise destination address. It is also desirable that pre-defined routing options exist at each router (selectively enabled as necessary when faults occur), rather than a large number of options that are dependent on the precise fault topology. Finally, the fault tolerance scheme should give good performance in the presence of faults and should not impact performance in the absence of faults.

It has been found in this work that a particular class of fault topologies can be handled efficiently; these topologies are called "routing-convex" (r-convex) topologies (a formal definition is presented in the next subsection). The VCR approach is sufficiently powerful to handle multiple faults with r-convex topology. Previous approaches to fault tolerance [17, 19] typically handle fault topologies of rectangular shape. Rectangular faults form only a small subset of the possible r-convex fault topologies.

The VCR approach assumes that all node/link faults are detectable. It is also assumed that failures are relatively long-lived, and that therefore the cost of a network reconfiguration process (following the detection of a fault) is acceptable. It is also assumed that all nodes are informed about faults during the reconfiguration phase; in particular, it is assumed that following reconfiguration no messages will be destined to faulty nodes.

6.1.1 Permitted Fault Topologies

Informally, r-convexity can be understood by the following illustration. Consider the examples shown in Figure 6.1. This figure shows four different fault topologies and illustrates how messages can possibly get past these fault topologies. (It is assumed here that there are no operational links within the shaded regions.) In Figure 6.1(a), the fault zone (the shaded area in the figure) is rectangular in shape and a message travelling in the X or Y direction could get past this fault zone without needing to take a step in a direction opposite to its desired direction of travel. Similarly, the fault topology shown in Figure 6.1(b) also does not require a message to take a step in a direction opposite to its direction of travel. In contrast, the fault topology shown in Figure 6.1(c) may require a message travelling in the X direction to take a step back (at some point) in the X direction to be able to get around the fault zone. Similarly, a message travelling in the Y direction and encountering the fault in Figure 6.1(d) may need to take a step back in the Y direction to be able to get around the fault zone. These latter two fault topologies have a concavity in their shape that requires "backward" steps. It appears that permitting backward steps may greatly increase routing complexity, owing to the difficulty of ensuring freedom from deadlock under such circumstances. The regions shown in Figures 6.1(c) and (d) are not r-convex, while those in Figure 6.1(a) and (b) are.

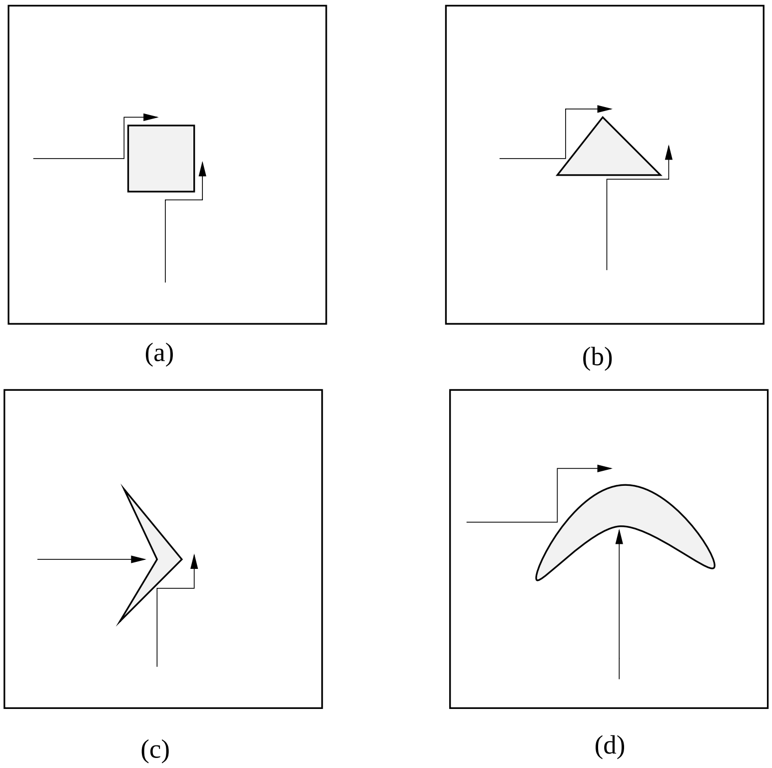


Figure 6.1: Illustration of different types of fault topologies; shaded areas indicated failed regions

R-convexity can be defined formally as follows. In a $n \times n$ mesh, any node can be denoted by a tuple (X, Y), where X is the column number and Y is the row number with respect to the left hand bottom corner, which is considered as the origin. A link L connecting nodes (X_1, Y_1) and (X_2, Y_2) is represented by four coordinates (X_1, Y_1, X_2, Y_2). As links are by definition bi-directional, the same link can also be alternatively represented as (X_2, Y_2, X_1, Y_1). All X channels running from left to right are called $+X$ channels and all X channels running from right to left are called $-X$ channels. Similarly, all Y channels running from bottom to top are called $+Y$ channels and all Y channels running from top to bottom are called $-Y$ channels.

A fault topology (consisting of one or more node and/or link faults) in a 2D mesh network is defined as r-convex if and only if it satisfies the following two conditions

:

- **Condition I :** any "working" column (i.e., a column with at least one working node) is reachable from any working node, by either using $+X$ and/or $\pm Y$ channels (without using any $-X$ channels), or using $-X$ and/or $\pm Y$ channels (without using any $+X$ channels),

  and

- **Condition II :** within each working column any working node is reachable from any other working node in the same column by either using $+Y$ and $\pm X$ channels, or using $-Y$ and $\pm X$ channels.

To understand what types of topologies are permitted by the above r-convexity definition, the concept of a "fault region" is useful. A fault region is defined by a collection of adjacent node and/or link faults. In this section the concept of fault region is used primarily to illustrate r-convexity.

Fault regions in a 2D mesh can be classified into X regions, Y regions and compound regions (formed by overlapping X and Y regions). X fault regions are those which comprise adjacent faulty X links and similarly Y fault regions are those which comprise adjacent faulty Y links. Two X links (resp. Y links) are defined to be adjacent if they are attached to a common node or if they connect two adjacent nodes in the same column (resp. row) to two other adjacent nodes in an adjacent column (resp. row). In more formal terms, the X link $(X_i, Y_j, X_i + 1, Y_j)$ has four adjacent links, namely, $(X_i, Y_j + 1, X_i + 1, Y_j + 1)$, $(X_i, Y_j - 1, X_i + 1, Y_j - 1)$, (X_i+1, Y_j, X_i+2, Y_j) and (X_i-1, Y_j, X_i, Y_j). Similarly the Y link (X_i, Y_j, X_i, Y_j+1) has four adjacent links, namely, $(X_i+1, Y_j, X_i+1, Y_j+1)$, $(X_i-1, Y_j, X_i-1, Y_j+1)$, $(X_i, Y_j + 1, X_i, Y_j + 2)$ and $(X_i, Y_j - 1, X_i, Y_j)$. The adjacency property is used transitively to define regions. In other words, if A and B are adjacent faulty links, and B and C are also adjacent faulty links then the three links A, B and C all belong to the same fault region. A X and a Y region are defined to be overlapping if they share at least a single node in common. A collection of overlapping X and Y fault regions is called a compound fault region. Note that if two compound regions overlap then a single compound region is formed. Network boundaries require special consideration. For the purpose of illustrating the permitted fault topologies using the fault region concept, the left and right boundaries of the mesh can be considered to constitute X fault regions, and the top and bottom boundaries constitute non-intersecting Y fault regions.

Consider simple fault regions that are not part of a larger fault region. In particular, consider an X fault region. A message from any working node can use $\pm Y$ channels to get above or below the fault region and can then bypass the fault region by using either $+X$ or $-X$ channels. This implies that a single X fault region cannot prevent a message from reaching any working column. Also X fault regions cannot prevent a message from any working node (in any working column) from reaching any other working nodes in the same column. Thus, a simple X fault region satisfies the r-convexity requirements. Consider a simple Y fault region. A message from any working node destined to another node in the same column can use $+X$ or $-X$ channels to bypass the fault region. This implies that a single Y fault region cannot prevent a message from reaching any working column. Also, a message from any working node can use the $+X$ or $-X$ channels to get left or right of the fault region, if necessary, and then use the $+Y$ or $-Y$ channels to reach any row, and can finally reach its destination by once again using the $+X$ or $-X$ channels (if such channels are used initially to move to the left or right of the fault region.) Thus, a simple Y fault region also satisfies the r-convexity requirements.

Next consider compound fault regions. The only such regions that violate the r-convexity requirement are shown in Figure 6.2. All these regions violate the r-convexity requirement in some way. They require either a step back in the X dimension (for a message travelling in the X dimension) or a step back in the Y dimension (for a message travelling in the Y dimension). As a specific example, consider a message needing to travel from node A to node B as illustrated in Figure 6.2 (c). Such a message cannot reach node B (in the same column) by using either $+Y$ channels and $\pm X$ channels, or using $-Y$ channels and $\pm X$ channels. Thus the fault topology in Figure 6.2 (c) violates the r-convexity requirement. Figure 6.2 shows X and Y fault regions of oval shape for purposes of clarity. Also, the compound fault regions shown in Figure 6.2 can themselves be a part of a much larger compound region. A network that does not have any of the fault region types shown in Figure 6.2 as a part of its fault topology would satisfy the r-convexity requirement.

6.1.2 Overview of Virtual Channel Reconfiguration

This subsection briefly outlines the VCR approach. The concepts discussed here are presented more formally in the next few subsections.

In the VCR approach, once faults are detected, the network's fault topology is checked for the r-convexity requirement. If the fault topology is found to be non-r-convex, then certain nodes are marked as unreachable (and the associated links are marked as failed) so that the fault topology is converted into an r-convex fault topology.

Once the r-convexity of the fault topology is ensured, the virtual channels at

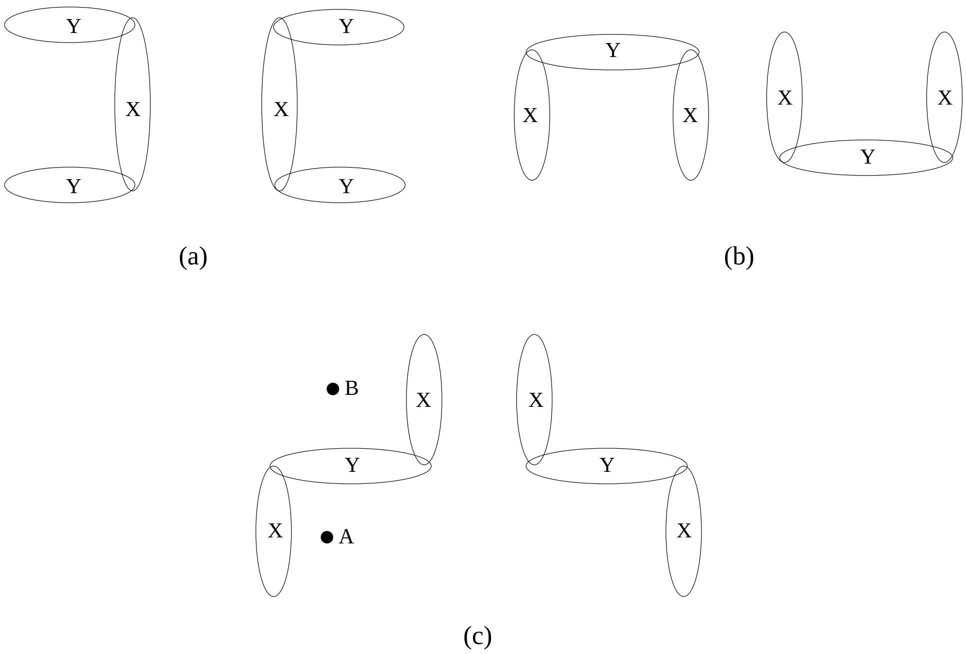


Figure 6.2: Compound fault regions that violate the r-convexity requirement

nodes adjacent to faults are reconfigured, according to the local fault structure. In the absence of faults in the network, messages may be routed using the XY algorithm (use of the VCR approach with adaptive routing is considered in Section 6.2). When faults are present in the network, the routing algorithm uses the special channels configured at nodes adjacent to faults to get around the faults without the possibility of deadlock or livelock. Once faults are bypassed, normal deterministic routing is resumed.

In summary, the VCR approach consists of three major steps:

- checking whether fault topologies obey the r-convexity requirement; if not, converting non-r-convex topologies into r-convex topologies by marking some nodes as unreachable, and the associated links as failed,

- reconfiguring the virtual channels at nodes adjacent to faults,

  and

- using these reconfigured channels and the modified routing algorithm at nodes adjacent to faults.

The process of ensuring r-convexity is described formally in Section 6.1.3. The reconfiguration process wherein the virtual channels are reconfigured at nodes adjacent to faults is described in Section 6.1.4. The routing algorithm is described in detail in Section 6.1.5.

6.1.3 Ensuring r-Convexity

R-convexity is ensured if the conditions I and II defined earlier are satisfied. The complexity of the algorithm to ensure these two conditions can be analysed as follows.

Condition I requires that any working column is reachable from any working node either by using only $+X$ and/or $\pm Y$ channels, or by using only $-X$ and/or $\pm Y$ channels. Condition I would be satisfied if every node in a working column can reach every other node in its two neighbouring columns either by using $+X$ and/or $\pm Y$ channels or by using $-X$ and/or $\pm Y$ channels. In every column there are $root(N)$ nodes and at most $root(N)$ minimal paths to any other node in the next column. The maximum path length to any node in the neighbouring column is also $root(N)$. So the worst case time complexity to implement Condition I, for a single column, would be $O(N * root(N))$. Since there are $root(N)$ columns in the network, the total time complexity for checking Condition I is $O(N^2)$.

Condition II requires that within each column any node can reach any other working node by using either $+Y$ and/or $\pm X$ channels or $-Y$ and/or $\pm X$ channels. Condition II would be satisfied if every node can reach its immediate working neighbour(s) in the same column by using either $+Y$ and/or $\pm X$ channels or $-Y$ and/or

$\pm X$ channels. Note that the path taken to reach the nearest working neighbour would involve only nodes which are immediately next to the fault. This is because, if each column satisfies the r-convexity requirement then it would be possible to reach the immediate working neighbour by passing through nodes immediately next to the fault. Thus, if the neighbouring node is distance d away (due to d node faults in between), a path to reach that node can involve examining at most $O(d*root(N))$ nodes. Also, the number of node pairs that need to be examined per column is only $O(root(N))$ (since only immediate working pairs are considered). Therefore, the time complexity to check all pairs in a single column is at most $O(N*d)$. Since there are $root(N)$ columns in the network, the total time complexity for the whole computation is $O(N*d*root(N))$. Since two nodes can be at a distance of at most $root(N)-1$ apart in a single column, the final time complexity for ensuring Condition II is $O(N^2)$. Thus both condition I and II can be ensured by having an algorithm whose complexity is at most $O(N^2)$.

When the fault topology does not satisfy the r-convexity requirement, then some nodes need to be marked as unreachable (and the associated links treated as failed links) so that the resultant topology becomes r-convex. If a node does not satisfy condition I, that node would be considered unreachable, and the node with its associated links would be marked as non-working. If condition II is not satisfied between any two working nodes in a column, then one of the working nodes would be marked as unreachable, and the node with its associated links would be marked as non-working. The choice of which node to mark unreachable might be based on some simple heuristic, such as which node has the greater number of non-working links. After such changes, the topology of faults needs to re-evaluated. Therefore, r-convexity checks, and marking of unreachable nodes, should be performed iteratively until no further change occurs to the fault topology, in which case the topology is r-convex.

Examining every node in determining r-convexity is wasteful when the network has few faults. A simpler algorithm to check for r-convexity can be devised by using the notion of fault regions described in Section 6.1.1. If faults are classified into regions then r-convexity can be ensured by just examining nodes in the vicinity of fault regions alone (instead of examining the whole network). One such approach is described below.

Consider a fault region, possibly compound, that is not part of a larger fault region. Construct a minimal rectangle that encloses the region. The first condition requires the following: for every working node (i, j) inside the bounding rectangle there should be a path of working links consisting solely of $+X$ and $\pm Y$ channels or solely of $-X$ and $\pm Y$ channels, to every "working" Y column that intersects the region. Such a path may employ working links immediately adjacent to the fault region as well as inside it. The second condition requires that in every column with a node or link fault (belonging to the fault region under consideration) any working node (inside the bounding rectangle) should have a non-faulty path to every other working node in the same column (and inside the bounding rectangle) consisting solely of $+Y$ and $\pm X$ channels or solely of $-Y$ and $\pm X$ channels. Again, such a path may employ working links immediately adjacent to the fault region as well as inside it.

A network will have an r-convex fault topology if and only if every fault region in the network satisfies the above two conditions. These conditions ensure that the conditions required for r-convexity are satisfied around the fault region and in other places they would be trivially ensured due to the absence of faults. Corresponding to the reduction in the size of region to be examined, the computation time of the algorithm would also be reduced.

Since complex fault cases are likely to be rare, having a much simpler algorithm to check for r-convexity might be desirable. Two simpler conditions are presented below which are shown to be sufficient, but not necessary, for r-convexity. Use of these conditions implies that sometimes topologies that are indeed r-convex will be classified as non-r-convex, but a non-r-convex topology will never be classified as r-convex.

Consider a fault region, possibly compound, that is not part of a larger fault region. The first condition requires that for every working node inside the fault region, there should exist a straight line path of working links out of the bounding rectangle in at least two directions ($X+$, $X-$, $Y+$ or $Y-$). This condition is described more formally as follows. Let the predicate $F(X_1, Y_1, X_2, Y_2)$ be true if link (X_1, Y_1, X_2, Y_2) is one of the faulty links belonging to the fault region under consideration or if the link is a non-existent link that would be beyond the mesh boundary. For any working node (i, j) within the bounding rectangle, two out of the following four conditions should hold:

$\forall\ k$, s.t. $0 \leq k \leq j - cmin$, $\neg F(i, j-k, i, j-k-1)$,

$\forall\ k$, s.t. $0 \leq k \leq cmax - j$, $\neg F(i, j+k, i, j+k+1)$,

$\forall\ k$, s.t. $0 \leq k \leq i - rmin$, $\neg F(i-k, j, i-k-1, j)$,

$\forall\ k$, s.t. $0 \leq k \leq rmax - i$, $\neg F(i+k, j, i+k+1, j)$.

Since the above condition requires a check of straight line paths, an algorithm to check for r-convexity would take only $O(K * P)$ time, where K is the number of nodes inside the minimal rectangle and P is the average length of the path out of the minimal rectangle.

An X direction (either $X+$ or X-), if any, in which there is a non-faulty straight line path out of the bounding rectangle is called an "open-X" direction and similarly a Y direction, if any, in which there is a non-faulty straight line path out of the bounding rectangle is called an "open-Y" direction. The second condition to be checked is that, in every column containing at least one Y link fault within the fault region, any two working nodes should have an identical open-X direction.

The proof that these two simpler conditions ensure r-convexity is as follows. When a message is correcting its X coordinate its movement can be blocked only by an X fault. Since it is travelling in the X direction, at any node where it is blocked there should be at least one open-Y direction that it can take and continue until it reaches a node without an X fault (by the first condition described above). While travelling in the chosen open-Y direction it cannot be blocked again by a Y fault belonging to the same compound fault region as it would also violate the open-Y definition. As well, it cannot be blocked by a Y fault belonging to some other region because that would mean that the Y fault regions overlap with the X fault region (by definition of "overlap") and would therefore be a part of the same compound fault. This would mean that the message would ultimately get an opportunity to correct its X coordinate at some node without taking a step back in the X direction. By induction, such a step should be possible in any column. So the message would ultimately correct its X coordinate. Similarly, while correcting the Y dimension the first condition would ensure that a message would always get an opportunity to correct its Y coordinate without a need to step back. An additional problem of not being able to correct any steps taken off in the X dimension while correcting the Y dimension can occur during the Y correction phase. The second condition would guarantee that such a thing would not happen by ensuring that for every faulty

Y link the associated good nodes on both sides have the same open-X directions. Thus, messages never need to step back in the dimension that is being corrected, and therefore, the above two conditions ensure r-convexity.

In conclusion, r-convexity can be ensured by different approaches. Classifying faults into regions helps in providing a simpler implementation for the r-convexity checking algorithm. However, the reconfiguration scheme and the routing algorithm do not require that faults be classified into regions. They just require that the fault topology in the network satisfies the r-convexity requirements.

6.1.4 Reconfiguration Process

In the reconfiguration process, only the nodes adjacent to faults must react to a fault by reconfiguring one or more of their virtual channels. The reconfigured channels act as topology extensions, permitting messages to be routed around faults, in a manner preserving the deadlock freedom properties of XY routing. So as to minimally impact the network, reconfiguration is done so as to minimise the maximal virtual channel requirements for each link. Reconfiguration is done as follows.

Consider a case where a working node has a single faulty X link (plus possibly a faulty Y link). Then one of the extra lanes in the Y direction that would allow the message to reach any column is reconfigured as an X_{sp} channel. Since the fault topology is r-convex, such a Y direction has to exist (either $Y+$ or $Y-$). If possible, for performance reasons, the X_{sp} channel could be configured in both Y directions. If a node has two faulty X links then two channels in the Y direction(s) are reconfigured to be X_{sp} channels.

Consider a case where a working node has a single faulty Y link (plus possibly a faulty X link). In that case an X direction (either $X+$ or $X-$) has to exist such that a message can reach the first working node in the direction of the failed link. A special channel Y_{sp1} is reconfigured in the $X+$ and/or $X-$ direction as necessary. Again for performance reasons, both could be configured, if possible. For every Y_{sp1} channel configured between node (i, j) and node (k, j), $k = i \pm 1$, due to a faulty Y link $(i, j, i, j-1)$, configure a corresponding Y_{sp2} channel such that it connects node $(k, j-l)$ and node $(i, j-l)$, where l is the minimum $l > 0$ such that the link is not faulty. For every Y_{sp1} channel configured between node (i, j) and node (k, j), $k = i \pm 1$, due to a faulty Y link $(i, j, i, j+1)$, configure a corresponding Y_{sp2} channel such that it connects node $(k, j+l)$ and node $(i, j+l)$, where l is the minimum $l > 0$ such that the link $(k, j+l, i, j+l)$ is not faulty. Since condition II is satisfied by the fault topology, nodes on both sides of the Y fault should have common X directions through which they could communicate and therefore configuration of the above channels would always be possible. If a working node has two faulty Y links, then two Y_{sp2} channels are configured.

The following examples illustrate the r-convexity checks, disabling of nodes and reconfiguration of channels for two different cases. In these examples it is assumed that faults are classified into regions for the purposes of r-convexity checks.

Example 1

Consider the 6×6 sub-mesh shown in Figure 6.3 (a). Only the faulty links in the network are shown (by means of highlighted lines) in this figure. The faults shown in the figure can be classified into three regions (labelled A, B and C). Region A is a compound fault region composed of two X fault regions and a single Y fault region (which consists of faults in two adjacent columns). Region B is a simple X fault region, and Region C is a compound fault region with an overlapping X fault region and a Y fault region.

In this example it is assumed that condition I and condition II are iteratively checked, in that order. Figure 6.3 (b) shows the nodes that would be marked unreachable after one iteration. Nodes (2, 3) and (2, 2) are marked unreachable as they have three faulty links and thus violate condition I. Once these two nodes are marked unreachable the two operational links attached to these nodes are also marked as faulty. When condition II is now checked, nodes (1, 1) and (1, 3) which lie in the same column are found to not share a common X direction by which they could communicate. Thus one of them is disabled. In this example it is assumed that node (1, 1) is disabled. For fault region B no nodes need to be marked unreachable. For fault region C node (5, 1) has to be marked unreachable as it has two faulty links, and also lies in the left mesh boundary which is considered to constitute an X-fault region.

Once nodes are marked unreachable the fault regions can grow in size as do regions A and C (see Figure 6.3 (c)). More iterations of r-convexity checks do not produce any further change in region C, while two more nodes from region A are marked unreachable in the second iteration due to their failure to satisfy condition I. After two iterations of checks, there is no change in any of the fault regions. Figure 6.3 (c) shows the final shape of the fault regions, which are r-convex. Figure 6.3 (d) shows how special channels are now reconfigured around the fault regions. Note that if a node is adjacent to two X faults it can have two special X_{sp} channels configured in the same physical channel. In this example and in subsequent examples it is assumed that channels are configured in both directions where possible, for performance reasons. In this figure, corresponding Y_{sp1} and Y_{sp2} channels are shown with the same number.

□

Example 2

Figure 6.4 shows another example of how r-convexity is ensured and how channels are reconfigured for a mesh with node faults. With a node fault all associated links belonging to the faulty node are marked faulty. Figure 6.4 (a) shows a mesh with node faults and the nodes that would be marked unreachable after r-convexity checks. In this example condition II is satisfied by all columns. Figure 6.4 (b) shows how the channels are reconfigured around the fault region. Again, corresponding Y_{sp1} and Y_{sp2} channels are marked with the same number.

□

THEOREM 1 : *For a 2D network with a r-convex topology, the reconfiguration process requires at most 3 virtual channels on each physical channel in the network (two of these to be reconfigured as "special").*

Proof : Any working node can have at most two faulty links due to the r-convexity requirement. If it has two faulty X links then two special channels are reconfigured on any physical channel (going out of the node) in the Y direction. If it has two faulty Y links then at most two special channels are reconfigured on any physical channel in the X direction. If it has one faulty Y link and one faulty X link one special channel is reconfigured in both X and Y directions. Thus any physical channel requires at most two special channels and one regular channel.

□

Although Theorem 1 specifies the minimum required number of channels, for performance reasons, more special channels could be configured if available in both directions depending on the fault. This issue is considered in Section 6.2.

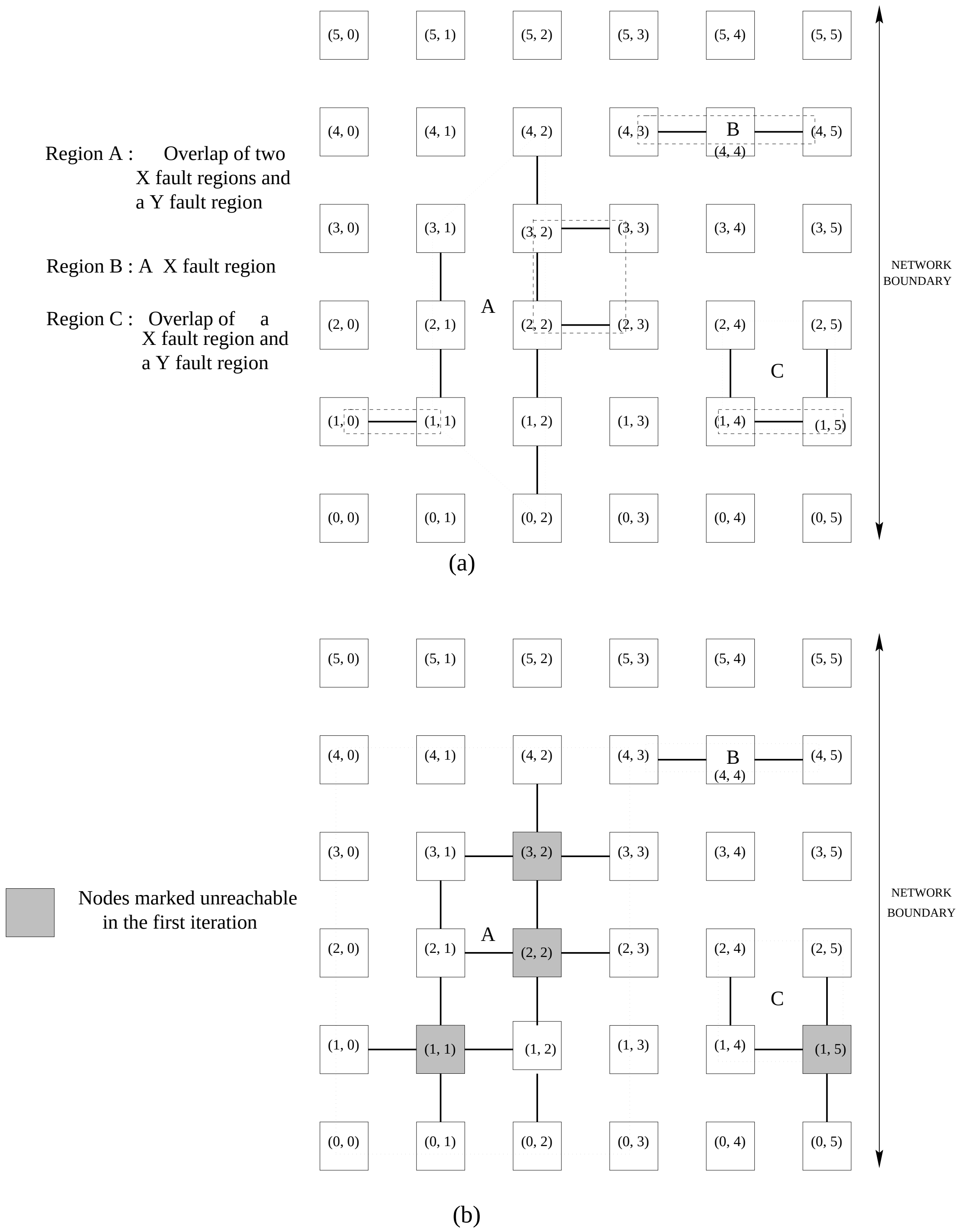
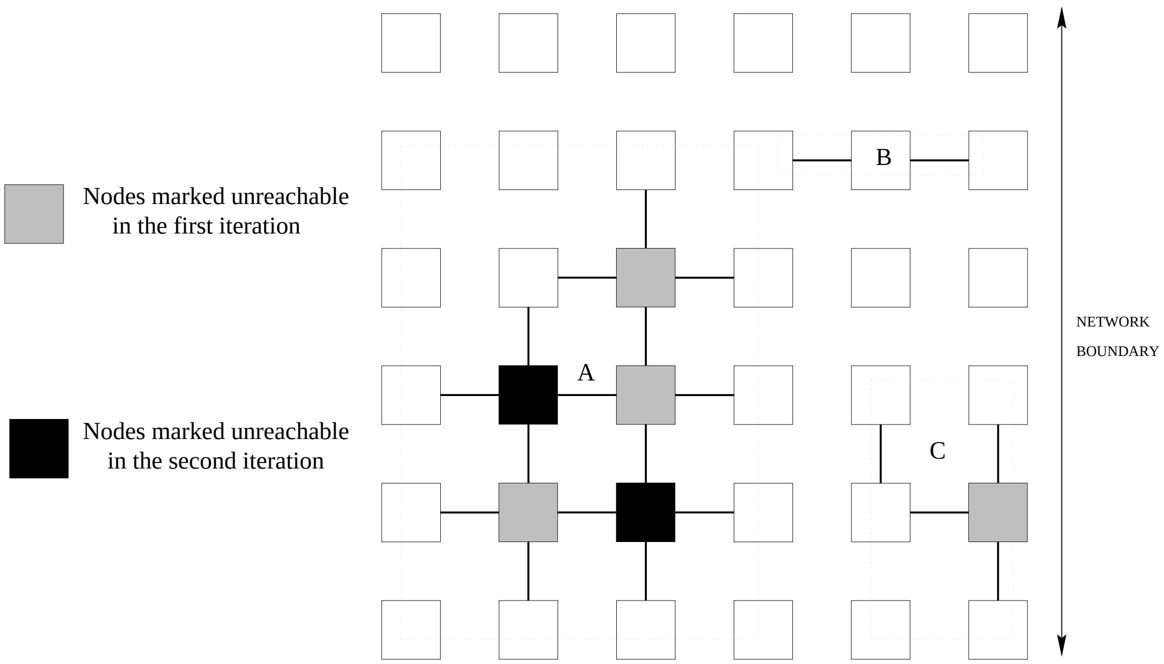
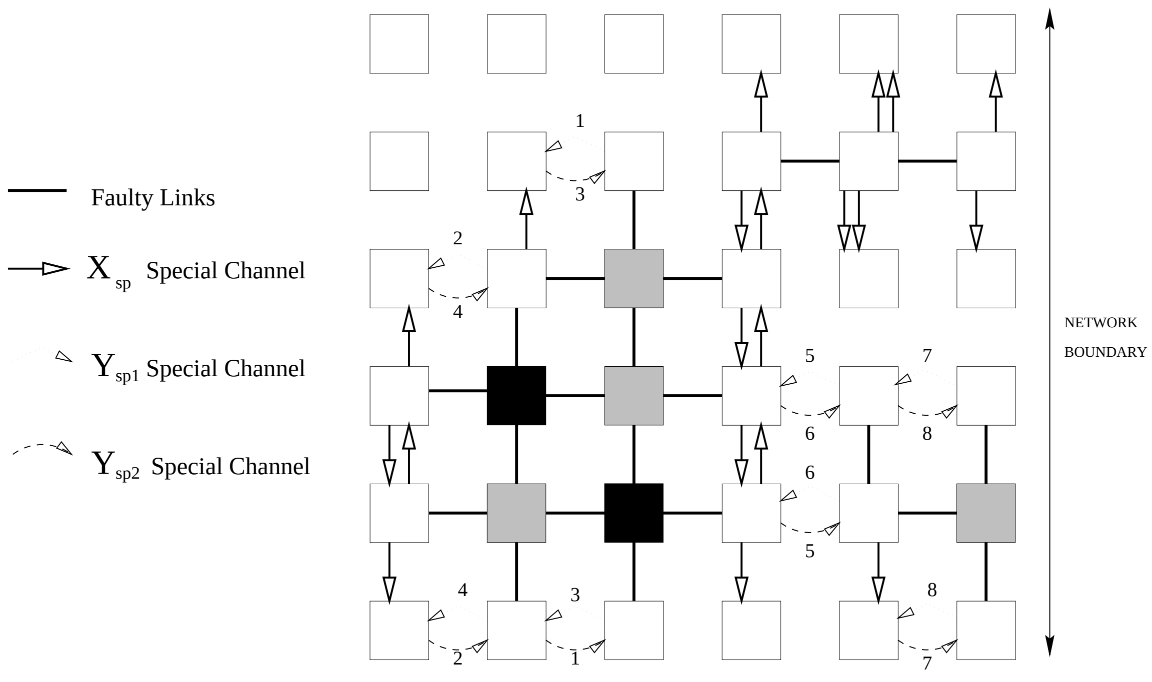


Figure 6.3: Reconfiguration of a Network with Link Faults: (a) Different fault region classifications; (b) Nodes that are marked unreachable after one iteration of r-convexity checks



(c)



(d)

Figure 6.3: (continued) (c) Nodes that are marked unreachable after two iterations of r-convexity checks; (d) Reconfigured channels around the fault region (each corresponding special Y channel is marked with the same number)

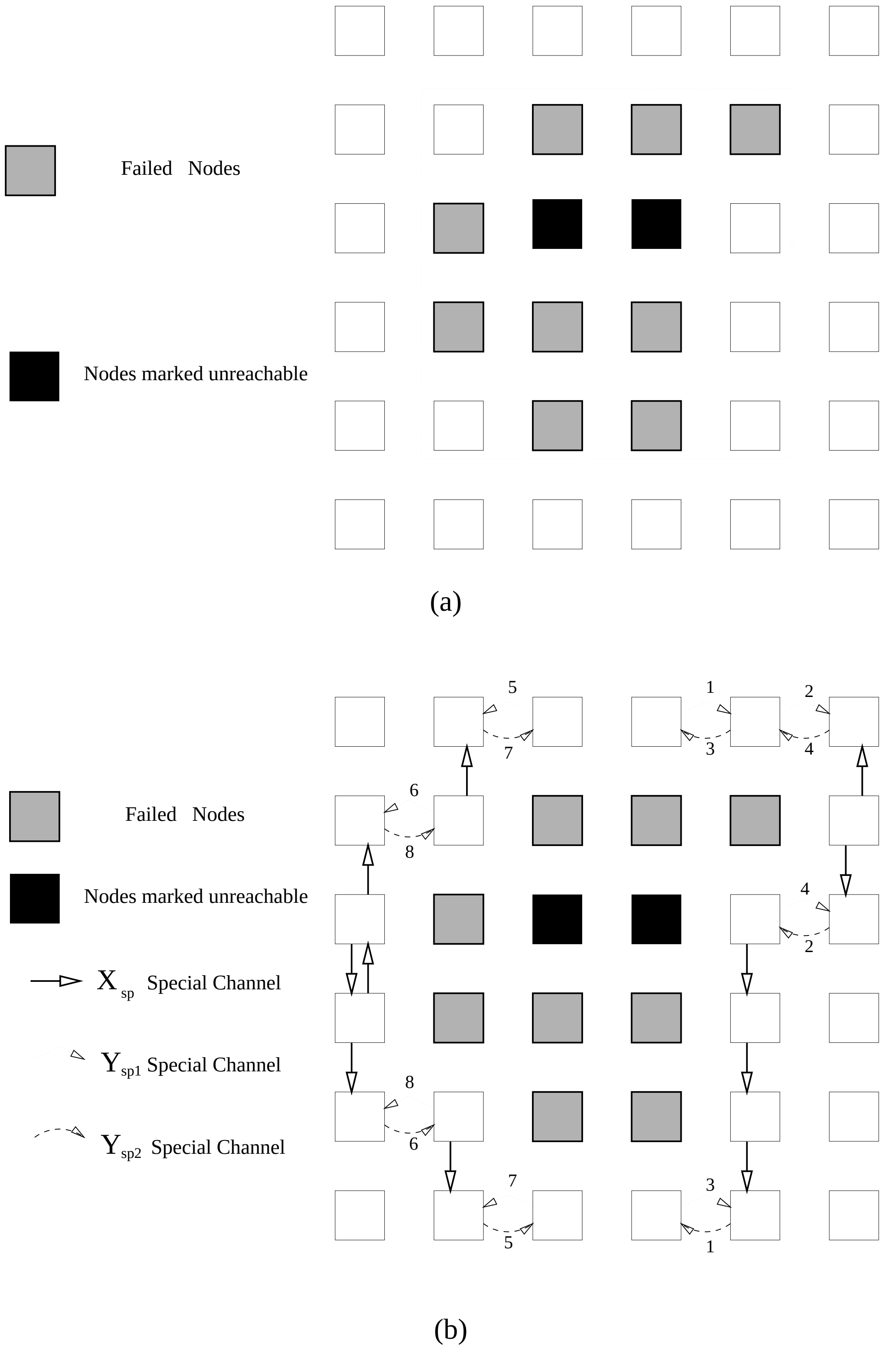


Figure 6.4: Reconfiguration of a Network with Node Faults (a) Nodes that are marked unreachable; (b) Network with reconfigured channels

6.1.5 Routing Algorithm

In the absence of faults, there are no special channels and the routing algorithm is the standard XY routing algorithm. Only when faults are encountered would there be special channels present, for use by the routing algorithm.

The routing algorithm in the presence of faults can be broadly described as follows. The router at each node first determines whether the X coordinate of a message is completely corrected or not. If it is uncorrected the message should be routed in the X dimension. If the next X link the message should be routed to is faulty, however, then the message is routed in the Y dimension by using an X_{sp} channel. If only one choice of X_{sp} channel exists, it is chosen. Otherwise, an arbitrary choice among the X_{sp} channels may be made, or alternatively, the X_{sp} channel in the direction in which the later corrections for the Y coordinate are likely to be minimum may be chosen [1]. The router at the new node that receives this message again attempts to route the message in the X dimension. This is possible if the required link in the X dimension is not faulty. If this link is also faulty, then the message is again routed using a X_{sp} channel, and so on, until a node is reached at which the message can finally correct its X coordinate. Since the fault topology is r-convex such a node has to exist. Note that if such a node exists in only one direction due (for example) to the intersection of the fault with the boundary then the configuration algorithm would have created the X_{sp} channels in this direction only (as seen earlier in Figure 6.4 (b)). Since the fault topology is r-convex, the X coordinate will ultimately be corrected.

[1] One possible heuristic : If $rmax$ and $rmin$ are the maximum and minimum Y coordinates of the bounding rectangle, $rdes$ is the Y coordinate of the destination node, and j is the Y coordinate of the current node, choose the $X_{sp}+$ channel if $rmax + rmin - j \leq rdes$, else choose the $X_{sp}-$ channel. Alternatively, an even simpler heuristic is to simply route in the direction in which the Y coordinate must eventually be corrected.

Once the X coordinate of a message is corrected, routers at subsequent nodes will attempt to correct the Y coordinate of the message. If the next link the message has to take in the Y dimension is faulty, then the message is routed in the X dimension by using a Y_{sp1} channel. If a choice of Y_{sp1} channels exists an arbitrary choice can be made among these as both should lead the message to the destination (due to the r-convexity requirement). The router at the new node again attempts to route the message in the Y dimension. This is possible if the required link in the Y dimension is not faulty. If it is faulty, the message continues to traverse Y_{sp1} channels until a node is reached at which the message can finally be routed in the Y dimension. Again, since the fault topology is r-convex, such a node has to exist. Once a message starts correcting its Y coordinate again, the routers at every node can identify that the message has moved away from its corrected X coordinate due to faults, as the message will be travelling in the Y dimension with an uncorrected X coordinate, in violation of the XY routing algorithm. The router at each node will try to route the message through a Y_{sp2} channel (which will re-correct the X coordinate of the message), if one exists. Thus, a message that has taken K steps from the corrected X coordinate using the Y_{sp1} channels, will take exactly the same number of steps in the X dimension using the Y_{sp2} channels. Once the X coordinate is corrected, the routing algorithm will attempt to correct the Y coordinate using the normal Y channels. (Note that since the fault topology is r-convex, the message can never go past the Y coordinate of its destination.) Thus, ultimately the X coordinate and the Y coordinate will be corrected completely.

The above routing algorithm can be described more formally by listing all possible switchings from the input channels to the output channels at any node in the network. Table 6.1 illustrates the switching allowed at any node in the network.

Table 6.2 illustrates under what conditions these switchings are allowed. Table 6.2 uses the notation $next_d(i,j)$ to refer to the next link direction (to be corrected) in dimension d at node (i,j). A "Don't Care" entry means that the boolean condition need not be checked.

In Table 6.2 there are four boolean conditions that determine, for each input channel, the output channel to which an incoming message should be switched. Thus for each input channel, there are 16 possible cases, based on which output channel can be selected. Note that Table 6.2 deals with all 16 cases for every input channel except for Y_{sp2} and Y. The scenarios not covered in Table 6.2 are:

- A message that has arrived from a Y_{sp2} channel into a node (i,j) has an uncorrected X coordinate and a corrected Y coordinate, and $next_X(i,j)$ is faulty. $next_Y(i,j)$ may or may not be faulty.
- A message that has arrived from a Y channel into a node (i,j) now has a corrected Y coordinate but its X coordinate is not corrected, and $next_X(i,j)$ is faulty. $next_Y(i,j)$ may or may not be faulty.
- A message that has arrived from a Y_{sp2} channel into a node (i,j) has uncorrected X and Y coordinates, and both $next_X(i,j)$ and $next_Y(i,j)$ are faulty.

Since the fault topology is r-convex, none of the above three scenarios can occur. Figures 6.5 (a) and (b) show how the routing algorithm chooses paths around the different faults, for the examples illustrated earlier in Figures 6.3 and 6.4, respectively.

6.1.6 Deadlock and Livelock Freedom

THEOREM 2 : *The proposed routing scheme for 2D meshes is deadlock-free.*

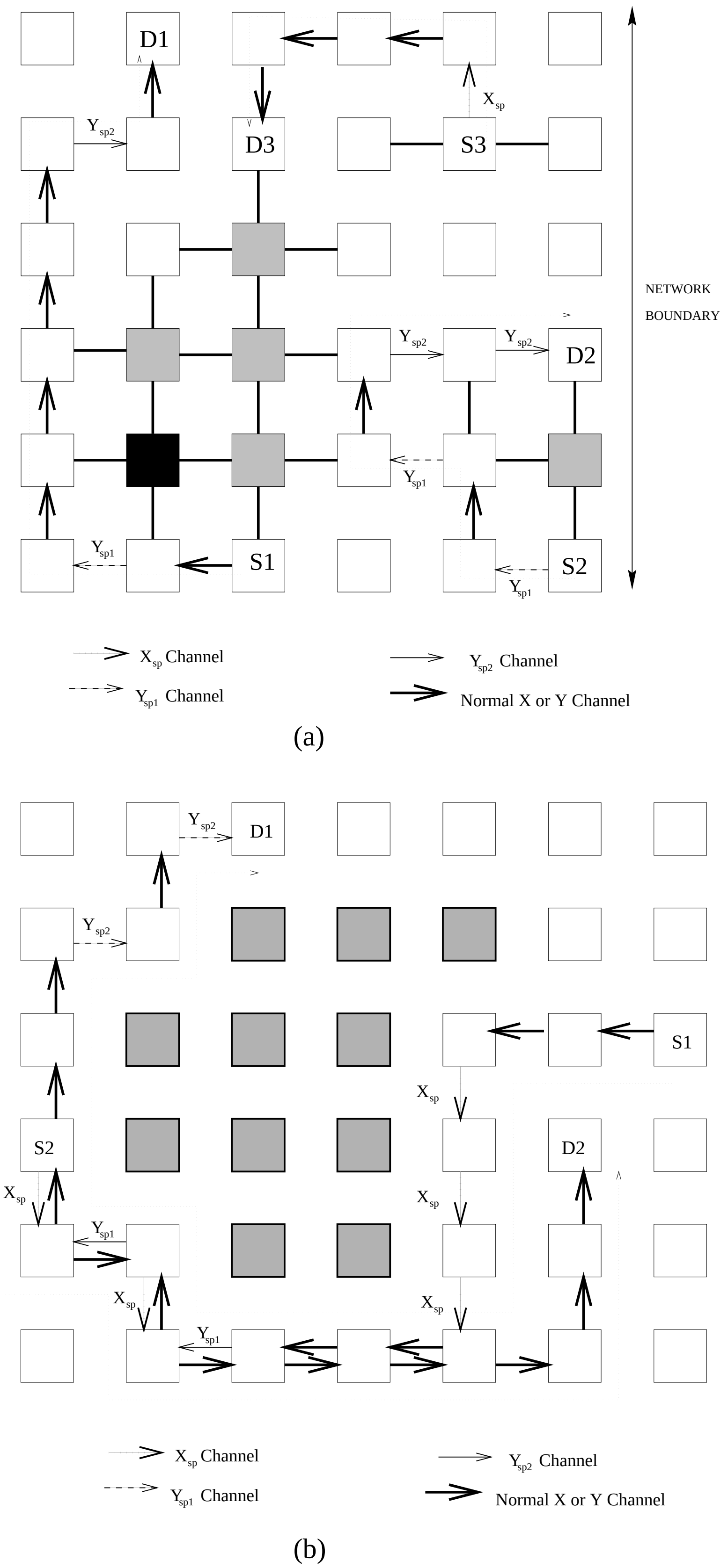


Figure 6.5: Examples of paths taken by different messages

Proof : The proof of deadlock freedom relies on the fact that a routing algorithm is deadlock free if the channels in the network can be numbered such that the algorithm routes messages along virtual channels with strictly increasing or decreasing numbers [24].

The channel numbering employed here ensures that channels are traversed in strictly increasing order of channel numbers. Each channel number is a triplet (x, y, z) where x is the most significant component and z is the least. The numbering of nodes and channels in the network is done as follows. All nodes have increasing X coordinates from left to right and increasing Y coordinates from bottom to top. The injection channels have all component values of their numbering set to $-\infty$ and the consumption channels have all component values of their numbering set to $+\infty$. The numbers of all regular channels is such that the numbers of channels in the Y dimension are higher than those of all channels in the X dimension. All virtual channels that are used for VCFC purposes on a given link (see Section 6.2) are assigned the same number. All X_{sp} channels in any column, always have numbers less than the numbers of the regular channels in the X dimension connecting to the column in the direction of the fault, and greater than those from the adjacent column. Similarly, the Y_{sp1} channels have numbers that are less than the numbers of the regular Y channels connecting to the row in the direction of the fault, and greater than those connecting from the other adjacent row. Finally, the Y_{sp2} channels have numbers that are greater than the number of the regular Y channels connecting from the row in the direction of the fault, and less than those connecting to the other adjacent row. Figure 6.6 shows the numbering of channels at an arbitrary node in the network. (Note: not all channels shown may actually be present, depending on the pattern of faults.) An example of numbering in a mesh with two node faults (forming overlapping X and Y fault regions) is shown in Figure 6.7.

Note that in Figure 6.6, Y_{sp1} channels are shown for the case where they connect nodes $(X, Y), (X-1, Y)$ to nodes $(X+1, Y), (X, Y)$ respectively. If the Y_{sp1} had, instead, connected nodes $(X+1, Y), (X, Y)$ to nodes $(X, Y), (X-1, Y)$ respectively, then the Y_{sp1} channels going out of node (X, Y) would be numbered (for example) $(n, y+1, -x+1)$ and $(n, m-y+1, -x+1)$. Similarly, the Y_{sp2} channels coming into node (X, Y) (for example) would be numbered (n, y, x) and $(n, m-y+1, x)$ if they connected from the node $(X-1, Y)$ rather than from node $(X+1, Y)$.

Deadlock freedom is shown by the fact that for any arbitrary node in the network, the allowable switchings between channels are always from channels with lower numbers to channels with higher numbers. For all the possible switchings listed in Table 6.1, the numbering in Figure 6.6 implies that channels are always traversed in increasing order. Thus, the algorithm is deadlock-free.

□

THEOREM 3 : *The proposed routing scheme for 2D meshes is livelock-free.*

Proof : The proof of livelock freedom requires showing that any message is guaranteed to make definite progress towards its destination at any node.

Consider an arbitrary message, destined to a working (and not marked unreachable) node. Recall that it is assumed that following reconfiguration no messages will be destined for failed/unreachable nodes. If the message does not encounter any faults on its path from source to destination then it would surely reach its destination as XY routing is livelock-free and deadlock-free.

Suppose the message encounters a fault on its path. When the message encounters a fault, it can either be in the phase where it is correcting its X coordinate, or in the phase where it is correcting its Y coordinate. If the message encounters the

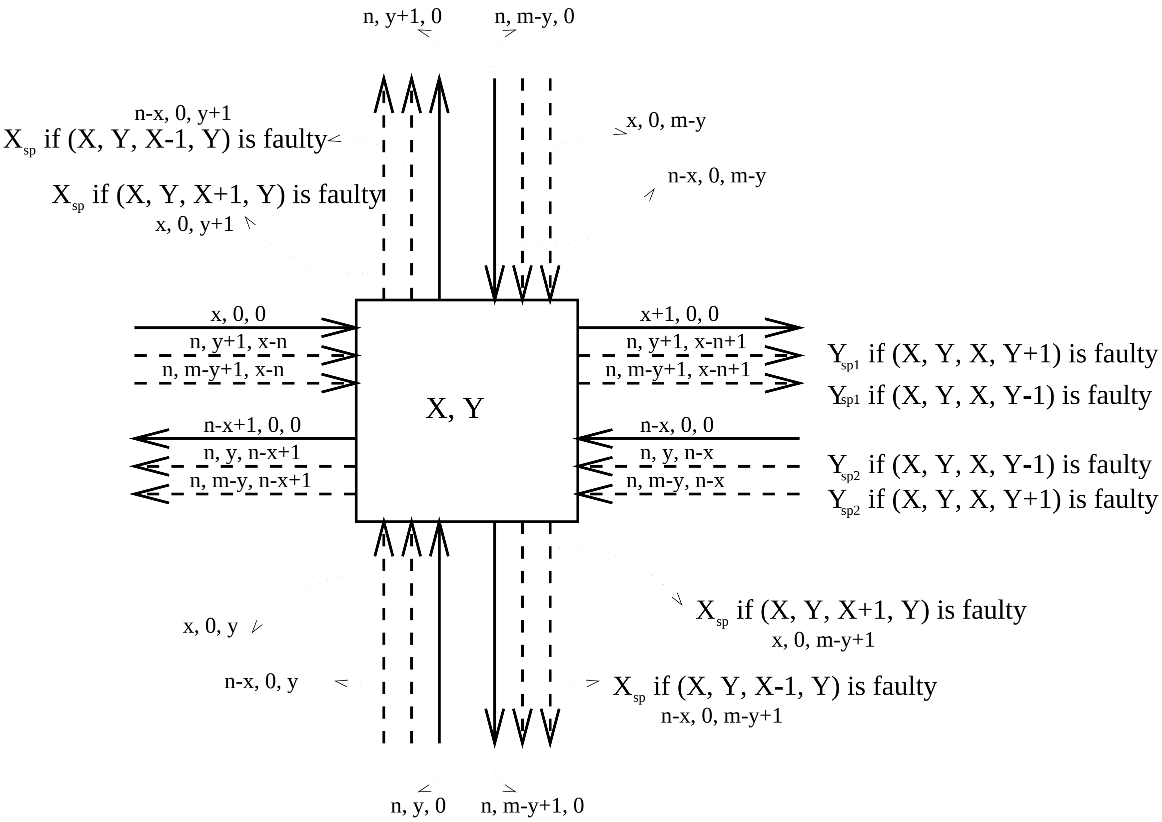


Figure 6.6: Channel numbering at any node in the network

fault while correcting its X coordinate, it chooses the X_{sp} channels at successive nodes until it reaches a node where it can continue the XY routing. Since such a node has to exist because of the r-convexity of the fault topology, the message ultimately corrects its X coordinate. Thus there is definite progress when faults are encountered during the X correction phase.

If the message encounters a fault while correcting its Y coordinate, it moves away from the correct X coordinate (by taking the Y_{sp1} channels). The configuration algorithm and the routing algorithm, however, guarantee that the message would choose a Y_{sp2} channel at a later stage for every Y_{sp1} channel chosen, thus re-correcting the X coordinate. Further, at least one step in the Y dimension (in the proper direction) must be made between a movement along a Y_{sp1} channel and

Table 6.1: Switching allowed by the routing algorithm at any node.

Input channels	Output channels					
	X	X_{sp}	Y	Y_{sp1}	Y_{sp2}	Consumption
X	√	√	√	√		√
X_{sp}	√	√				
Y			√	√	√	√
Y_{sp1}			√	√		
Y_{sp2}			√	√	√	√
Injection	√	√	√	√		

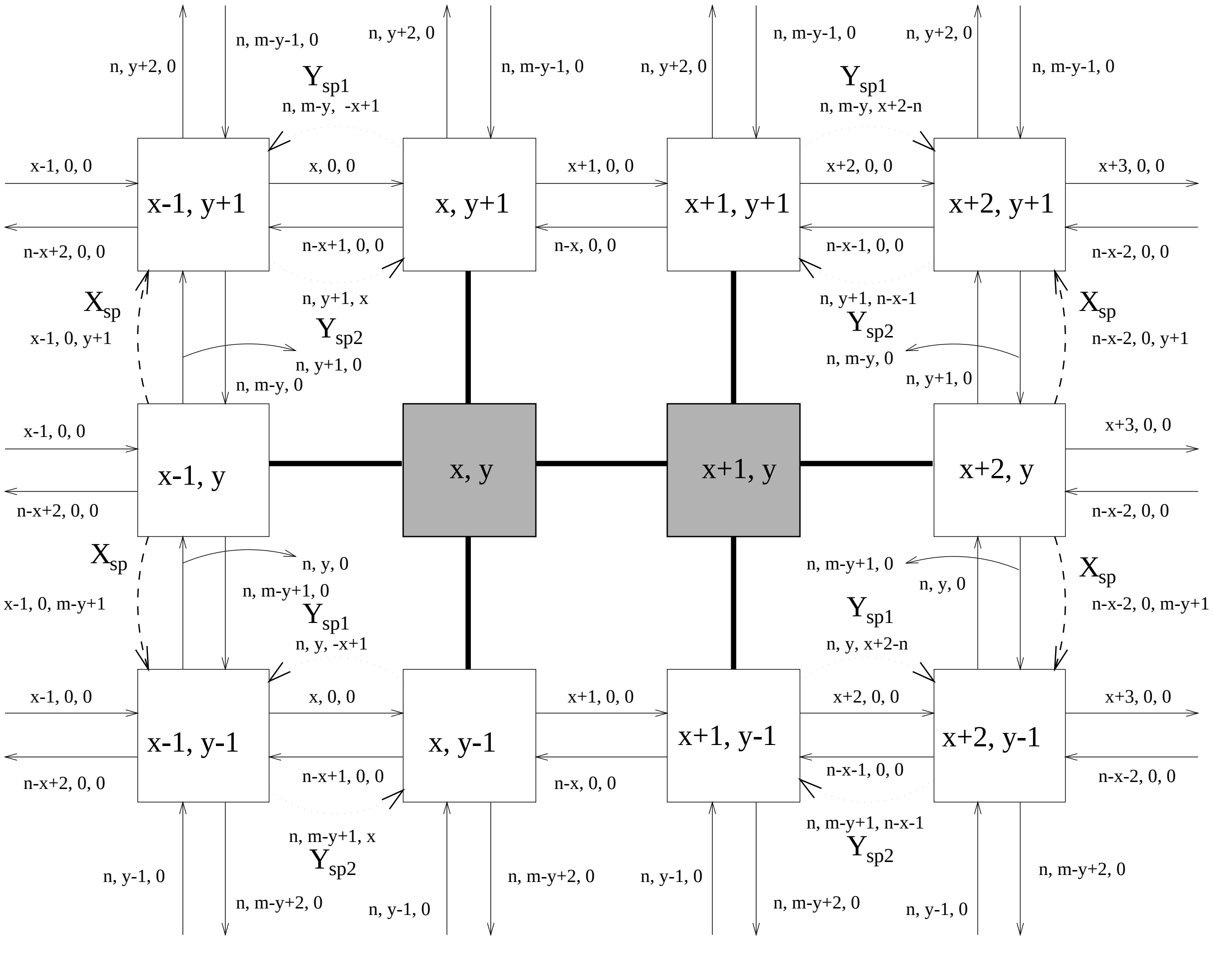


Figure 6.7: Channel numbering around two adjacent node faults

Table 6.2: Conditions under which the channel switchings listed in Table 6.1. can occur

Channel Switchings	Followed if			
	X Corrected ?	Y Corrected ?	$next_X(i,j)$	$next_Y(i,j)$
X to X	No	Don't Care	Not Faulty	Don't Care
X to X_{sp}	No	Don't Care	Faulty	Don't Care
X to Y	Yes	No	Don't Care	Not Faulty
X to Y_{sp1}	Yes	No	Don't Care	Faulty
X to Cons.	Yes	Yes	Don't Care	Don't Care
X_{sp} to X	Don't Care	Don't Care	Not Faulty	Don't Care
X_{sp} to X_{sp}	Don't Care	Don't Care	Faulty	Don't Care
Y to Y	Yes	No	Don't Care	Not Faulty
Y to Y	No	No	Faulty	Not Faulty
Y to Y_{sp1}	Yes	No	Don't Care	Faulty
Y to Y_{sp1}	No	No	Faulty	Faulty
Y to Y_{sp2}	No	Don't Care	Not Faulty	Don't Care
Y to Cons.	Yes	Yes	Don't Care	Don't Care
Y_{sp1} to Y	Don't Care	Don't Care	Don't Care	Not Faulty
Y_{sp1} to Y_{sp1}	Don't Care	Don't Care	Don't Care	Faulty
Y_{sp2} to Y_{sp2}	No	Don't Care	Not Faulty	Don't Care
Y_{sp2} to Y	Yes	No	Don't Care	Not Faulty
Y_{sp2} to Y	No	No	Faulty	Not Faulty
Y_{sp2} to Y_{sp1}	Yes	No	Don't Care	Faulty
Y_{sp2} to Cons.	Yes	Yes	Don't Care	Don't Care
Inj. to X	No	Don't Care	Not Faulty	Don't Care
Inj. to X_{sp}	No	Don't Care	Faulty	Don't Care
Inj. to Y	Yes	No	Don't Care	Not Faulty
Inj. to Y_{sp1}	Yes	No	Don't Care	Faulty
Inj. to Cons.	Yes	Yes	Don't Care	Don't Care

a movement along a Y_{sp2} channel. Finally, since the fault topology is r-convex, a message cannot go past the Y coordinate of its destination. Thus faults cannot obstruct the correction of the Y coordinate.

In conclusion, every message is guaranteed to make progress towards its destination as ultimately the X and Y coordinates would be corrected. Therefore, the proposed routing scheme is livelock-free.

□

6.2 Employing VCFC and Adaptive Routing with Virtual Channel Reconfiguration

Because of the potential benefits that can be achieved with VCFC, networks may have more virtual channels than what is minimally required by the VCR approach. This raises the issue of how such extra virtual channels could be divided up among various types of special and normal virtual channels.

One possible solution would be to allocate a minimal set of virtual channels for implementation of the VCR approach, and to allocate all the remaining virtual channels to a shared pool from which a virtual channel could be used freely by any message (without blocking for it, as blocking for such a shared channel could cause deadlock). Such a scheme would be feasible as long as the router at any node, on receiving a message from one of the virtual channels from the shared pool, could identify what type of virtual channel the message would have chosen, (e.g., special channel *vs* regular channel) had it not taken a shared channel. This would be necessary to guarantee correct choice of virtual channels (e.g., special channel *vs* regular channel) in future routing. If this identification is possible, then such a scheme would not cause deadlock or livelock (by Duato's theory [26]) as blocking would not be allowed for the shared virtual channels. To do this, however, messages would need to be tagged to identify whether they require use of special channels. Since three different types of messages can enter a node from any direction (due to the minimum requirement of three virtual channels in any direction) one would need to have at least two bits of information in the header of each message for unique identification. Thus having a shared pool of virtual channels would require that messages be tagged before being routed and that these tags be changed appropriately at each node, as each message progresses through the network.

To avoid this cost, one simple solution would be to partition the extra virtual channels among the different traffic types statically, using some form of equipartition. For example, consider the case shown in Figure 6.8 where traffic entering a node from three different directions needs to share virtual channels in two outgoing directions due to a faulty Y link. If the two outgoing links (referred to as out(1) and out(2) in the figure) each have six virtual channels, for example, then two lanes for Y_{sp1} traffic could be created on each outgoing link for the traffic going around the Y fault. This partitioning would give an equal number of virtual channels (four in this example) for each type of traffic (traffic avoiding the Y fault, traffic moving in the $X+$ direction, and traffic moving in the $X-$ direction) and thus might ensure that the benefits of VCFC are spread uniformly, while at the same time not requiring the cost of tagging.

Another issue that arises is that of how adaptive routing could be used in conjunction with the VCR approach. One algorithm that appears to be ideally suited for combination with the VCR approach is the *-Channels algorithm. The *-Channels algorithm obtains deadlock freedom by ensuring that there is a network of star

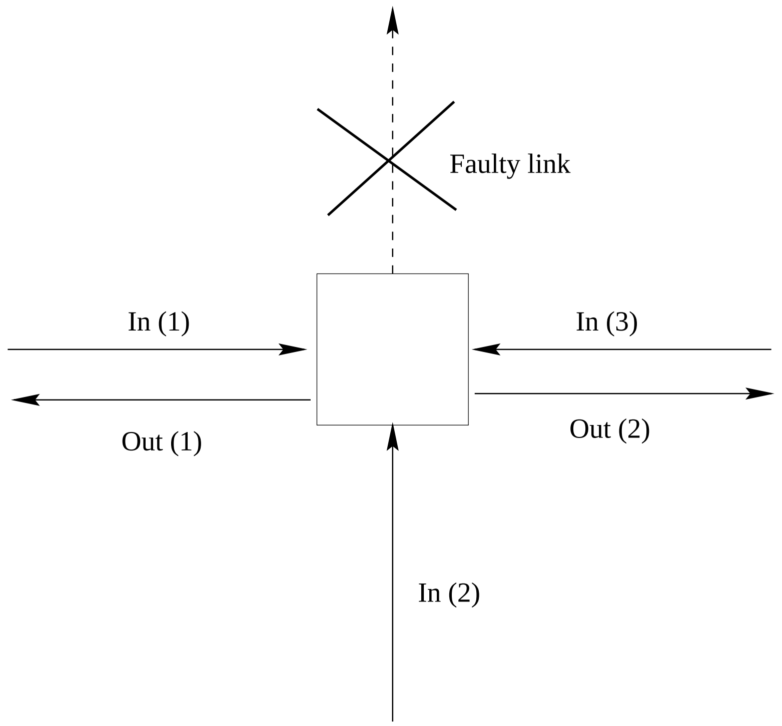


Figure 6.8: Example in which traffic from three incoming directions must share virtual channels in two outgoing directions, due to a faulty link

channels on which routing is deterministic and thus deadlock-free, and exercising adaptivity only on non-star channels. Since the VCR approach makes a network employing deterministic routing deadlock-free, even in the presence of faults, it can easily be used in conjunction with the *-Channels algorithm. With such a scheme, under fault-free conditions all channels not needed for the star channel network, under the *-Channels algorithm, would be classified as non-star channels. When faults occur the non-star channels are used for reconfiguration, using the VCR approach to fault tolerance. For example, in the case of a 2D mesh, the VCR approach requires at least 3 virtual channels per physical channel in the Y dimension. Under fault-free conditions these could be allocated on each physical channel as one star channel and two non-star channels. The two non-star channels could be used freely for taking adaptive steps or even deterministic steps (as a part of the VCFC mechanism). When faults occur one or both of these non-star channels would be converted into special channels to get around the faults. Again, equi-partitioning could be used to allocate channels to different traffic types, if more than the minimal number of virtual channels were present.

If adaptivity is permitted on all links, then when faults are present (and the VCR approach is used), a router cannot identify whether an incoming message in a Y channel is a message that has been misrouted to avoid a faulty Y link or whether it is a message taking an adaptive step. It is necessary to do such identification since such messages must take Y_{sp2} channels when encountered. Previously, with deterministic routing, such a problem did not exist as any message coming through the normal Y channels with an uncorrected X dimension could only be a message which has been misrouted to avoid a fault. With adaptive routing, messages with an uncorrected X dimension can arrive at any node through the non-star Y channels. This would suggest that message tagging is necessary.

In fact, however, it may not be desirable to have adaptivity at nodes immediately surrounding a fault, as messages may take adaptive steps into the fault zone. Also, having adaptivity with certain types of faults can lead to possibilities of deadlock unless routing tables are changed to take care of these special conditions. For example, in Figure 6.9 a deadlock occurs because a message travelling from node S to node D takes an adaptive step at node A, and due to the fault at node B takes a special channel back to node A, and eventually ends up back at node B, but with all special channels from B to A occupied by its own flits, thus causing it to wait for itself.

One good solution is to not permit adaptivity (i.e., not have the non-star chan-

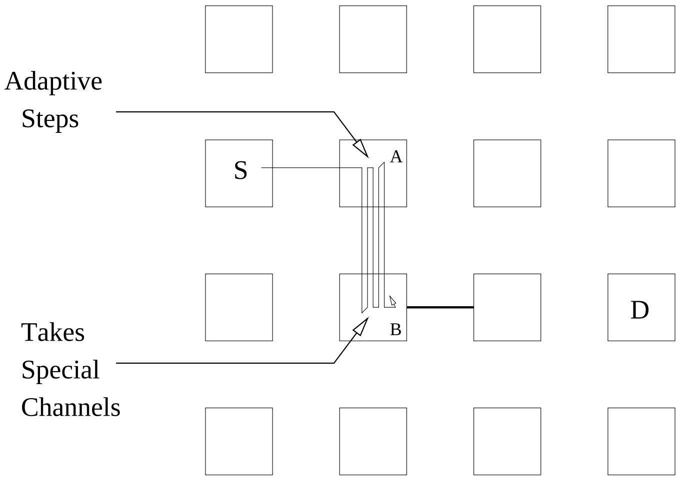


Figure 6.9: An example of a livelock that could occur due to improper combination of adaptive routing and the VCR approach

nels) on all links on which special channels are configured. The non-star channels would then be available only on links which do not have any special channels. Ignoring all adaptive channels, all the other channels are numbered properly and are deadlock free. Also, as before, waiting for adaptive channels is not permitted at any node. Thus, this modification is certainly free of deadlocks. To see that the problem with identification of messages misrouted to avoid a faulty Y link is satisfactorily addressed, note that if such a message would be present on a particular Y channel, then that link must have a special channel configured on it (a X_{sp} channel), thus adaptivity would be disabled on that link.

6.3 VCR Approach on Higher Dimensional Networks

The VCR approach can be extended to work on three dimensional and higher dimensional meshes. For the sake of clarity, such extensions are discussed initially for a three dimensional mesh network and then similar extensions for higher dimensional meshes are presented. Finally extensions to torus networks are explored.

6.3.1 Three Dimensional Meshes

Let X, Y and Z denote the three dimensions of the 3D mesh. A "working" YZ plane is one with at least one operational node. A Z column is defined as all nodes with some fixed values of X and Y coordinates. A "working" Z column is one with at least one operational node. In 3D networks, a fault topology is defined as r-convex if it satisfies the following three requirements :

- any working YZ plane is reachable from any working node by either using only $+X$ and/or $\pm Y$ channels, or only $-X$ and/or $\pm Y$ channels,

- any working Z column is reachable from any working node in the network in the same YZ plane by either using only $+Y$ and/or $\pm Z$ channels, or only $-Y$ and/or $\pm Z$ channels,

  and

- within each working Z column any working node is reachable from any other working node in the same Z column, by either using only $+Z$ and $\pm X$ channels, or only $-Z$ and $\pm X$ channels.

If a node does not satisfy one or both of the first two conditions it is marked as unreachable and all its links are marked as failed. If a pair of nodes does not satisfy the third condition, one of these nodes is marked as unreachable and all its links are marked as failed. As before, the condition checks and the marking of unreachable nodes should be performed iteratively until no further change occurs to the fault topology.

Reconfiguration of virtual channels for a r-convex fault topology, is as follows. Consider a case where a working node has a single faulty X link (plus possibly faulty Y and Z links). Then one of the extra lanes in the Y direction which would allow the message to reach any Y column is reconfigured as an X_{sp} channel. Since the fault topology is r-convex such a direction has to exist. If possible, for performance reasons the X_{sp} channel could be configured in both Y directions. Consider a case where a working node has a faulty Y link (plus possibly faulty X and Z links). Then one of the extra lanes in the Z direction which would allow the message to reach any Z column is configured as the Y_{sp} channel. Since the fault topology is r-convex such a direction has to exist. If possible, for performance reasons the Y_{sp} channel could be configured in both Z directions. Consider a case where a working node has a faulty Z link (plus possibly faulty X and Y links). In that case an X direction has to exist such that a message can reach the first working node in the direction of the failed node. A special channel Z_{sp1} is reconfigured in the $X+$ or $X-$ direction as necessary. For every Z_{sp1} channel configured in the X direction a Z_{sp2} channels is also configured in the X direction in exactly similar manner as in 2D meshes. As before, for performance reasons special channels can be configured in both X directions. All the above configurations should be possible as the fault topology is r-convex.

With the above reconfiguration strategy, an X link can have two special Z channels, a Y link can have two special X channels, and a Z link can have two special Y channels. Thus the requirements are a minimum of three virtual channels on any link.

The routing strategy in a 3D mesh is nearly the same as that in a 2D mesh. The routing algorithm can be described informally as follows. When there are no faults in the network the X dimension is corrected first, the Y dimension is corrected next, and finally, the Z dimension is corrected. When a message faces a faulty X link, it takes an X_{sp} channel configured in the Y dimension to get past the fault. Similarly when a message faces a faulty Y link, it takes the Y_{sp} channel configured in the Z direction. The extra steps that are taken using the X_{sp} and Y_{sp} channels need not be recorrected. When a message faces a faulty Z link it takes the Z_{sp1} channel configured in the X direction and re-corrects this step by taking the earliest available Z_{sp2} channel. The above routing algorithm is described more formally by listing all possible switchings from the input channels to the output channels at any node in the network in Table 6.3.

THEOREM 4 : *The VCR approach extended for 3-D meshes is deadlock-free and livelock-free.*

Proof : The deadlock freedom can be shown by ensuring that at any arbitrary node in the network the routing algorithm routes packets along channels in strictly increasing order of channel numbers.

The channel numbering employed is a quintuplet $(x, y, z, s1, s2)$ where x is the most significant component and $s2$ is the least significant component. The numbering of regular channels in a 3D network is shown in Figure 6.10. The $s1$ and $s2$ component of regular channels is set to 0. Table 6.4 shows the numbering that special channels would have in the above figure depending on the direction in which they are configured. In Table 6.4 $x_{in(X)}$ refers to the x coordinate of the incoming X link (either $X+$ or $X-$) into the node. Similar notation is also followed for the Y and Z links. In Table 6.4, the Z_{sp1} channel is assumed to be configured in the $X+$ direction and the Z_{sp2} channels is assumed to be configured in the $X-$ direction. Note that this assumption does not change the generality of the numbering.

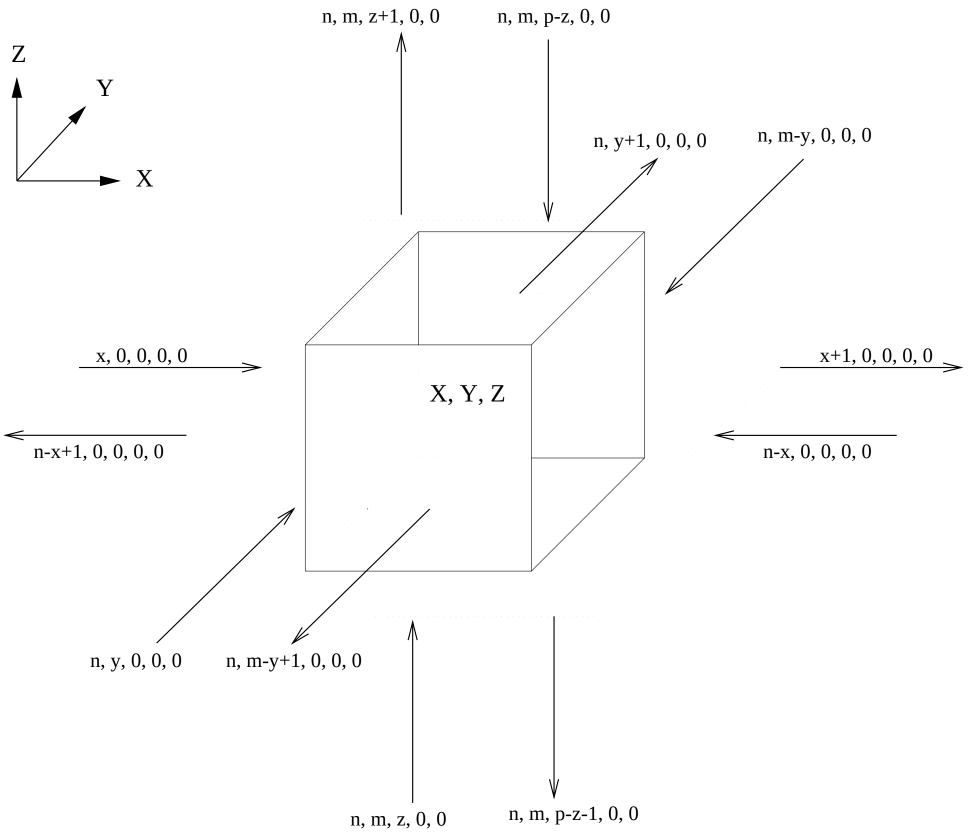


Figure 6.10: Channel numbering of regular channels at each node in a 3D mesh

For all the possible switchings listed in Table 6.3, the numbering in Figure 6.10 and Table 6.4 implies that channels are always traversed in increasing order. Thus, the algorithm is deadlock-free.

The proof of livelock freedom is a similar argument as in Theorem 3. The checking of the r-convexity, and the configuration of special channels would guarantee that a message can correct its X, Y and Z dimensions, thus ensuring that the VCR approach is livelock free.

□

Adaptive routing can also be added to the VCR approach in three dimensions by using the *-Channels algorithm. The *-Channels algorithm for 3D meshes requires

Table 6.3: Switching allowed by the routing algorithm at any node for a 3D mesh.

Input channels	Output channels							
	X	X_{sp}	Y	Y_{sp}	Z	Z_{sp1}	Z_{sp2}	Consumption
X	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
X_{sp}	$\surd$	$\surd$						
Y			$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
Y_{sp}			$\surd$	$\surd$				
Z					$\surd$	$\surd$	$\surd$	$\surd$
Z_{sp1}					$\surd$	$\surd$		
Z_{sp2}					$\surd$	$\surd$	$\surd$	$\surd$
Injection	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$		

Table 6.4: Numbering of special channels for the 3D mesh shown in Figure 6.10

Special channel	Direction Configured	Numbering
X_{sp}	$Y+$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, y+1, 0
X_{sp}	$Y-$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, m-y+1, 0
Y_{sp}	$Z+$	$x_{in(Y)}$, $y_{in(Y)}$, $z_{in(Y)}$, m, z+1
Y_{sp}	$Z-$	$x_{in(Y)}$, $y_{in(Y)}$, $z_{in(Y)}$, m, p-z-1
Z_{sp1}	$X+$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$+1, 0, z+1-m
Z_{sp2}	$X-$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$, n, m-y

one virtual channel per physical channel in the X dimension and two virtual channels per physical channel in the Y and Z dimensions. Of the two channels in the Y and Z dimension, one channel is the star channel, which is to be used only for dimension-order routing, and the other is the non-star channel, which is to be used for both dimension-order routing and non-dimension-order routing. Messages may not block waiting solely for a non-star star channel to become free. If the relevant star channel in the non-adaptive direction becomes idle, it must be taken. As before, for the sake of deadlock and livelock avoidance, adaptivity needs to be cancelled on all links on which special channels are defined.

In 3D meshes, more powerful fault tolerance is possible through variants of the VCR approach that permit some non-r-convex fault topologies, at a cost of use of additional virtual channels. The VCR approach discussed in this section configures X_{sp} channels only in the Y dimension (and not in the Z dimension) to overcome a faulty X link. Similarly, the Z_{sp1} and Z_{sp2} channels are configured only in the X dimension (and not in the Y dimension). Configuring these additional channels (in other dimensions) would increase the fault tolerance capacity of the network. Table 6.5 shows the proof of deadlock freedom by means of numbering for a network in which the X_{sp} channels are configured in both Y and Z dimensions and Z_{sp1} and Z_{sp2} channels are configured in both X and Y dimensions.

With the above strategy, an X link can have two special Z channels, a Y link can have two special X channels and two special Z channels, and a Z link can have two special X channels and two special Y channels. Thus, the requirements are a minimum of three channels on any X link and five channels on the Y and Z links.

Table 6.5: Numbering of special channels for the mesh shown in Figure 6.10

Special channel	Direction Configured	Numbering
X_{sp}	$Y+$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, y+1, 0
X_{sp}	$Y-$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, m-y+1, 0
X_{sp}	$Z+$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, m, z+1
X_{sp}	$Z-$	$x_{in(X)}$, $y_{in(X)}$, $z_{in(X)}$, m, p-z-1
Y_{sp}	$Z+$	$x_{in(Y)}$, $y_{in(Y)}$, $z_{in(Y)}$, m, z+1
Y_{sp}	$Z-$	$x_{in(Y)}$, $y_{in(Y)}$, $z_{in(Y)}$, m, p-z-1
Z_{sp1}	$X+$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$+1, x+1-n, -m
Z_{sp1}	$Y+$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$+1, 0, z+1-m
Z_{sp2}	$X-$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$, n-x, 0
Z_{sp2}	$Y-$	$x_{in(Z)}$, $y_{in(Z)}$, $z_{in(Z)}$, n, m-y

6.3.2 Higher Dimensional Mesh Networks

In a n-dimensional mesh let the dimensions be numbered from d_1 to d_n. A "working" $d_i - d_{i+1}$ plane (for all $1 \leq i < n$) is defined as a plane with at least one operational node. A d_n^{th} column is defined as all nodes with some fixed values for all coordinates other than the n^{th} coordinate. A "working" d_n^{th} column is one with at least one operational node. r-convexity in a n-dimensional mesh can be ensured by checking for the following conditions:

- any working $d_{i+1} - d_{i+2}$ plane (for all $1 \leq i < n-1$) is reachable from any working node by either using only $+d_i$ and/or $\pm d_{i+1}$ channels, or only $-d_i$ and/or $\pm d_{i+1}$ links,
- any working d_n^{th} column is reachable from any working node in the network, in the same $d_{n-1} - d_n$ plane, by either using only $+d_{n-1}$ and/or $\pm d_n$ channels or only $-d_{n-1}$ and/or $\pm d_n$ links,

  and

- within each working d_n^{th} column any working node is reachable from any other working node in the same d_n^{th} column, by either using only $+d_n$ and $\pm d_1$ links, or only $-d_n$ and $\pm d_1$ links.

If a node does not satisfy one or both of the first two conditions, it is marked as unreachable and all its links are marked as failed. If a pair of nodes does not satisfy the third condition, one of these nodes is marked as unreachable and all its links are marked as failed. As before, the condition checks and marking of unreachable nodes should be performed iteratively until no further change occurs to the fault topology.

The reconfiguration strategy is similar to that of the 3D mesh and is informally explained below. To handle faulty links in the d_i^{th} dimension $(1 \leq i \leq n-1)$ extra lanes in the d_{i+1}^{th} dimension are configured as special channels. For faults in the d_n^{th} dimension, special channels are configured in the first dimension. Since d_n^{th} dimension is the last dimension, two different types of special channels should be created (in the case of the 3D mesh they were Z_{sp1} and Z_{sp2}) so that the extra steps to avoid faults can be recorrected. Since faults are r-convex, the reconfiguration of all the above channels and the choice of these channels during routing would be possible. Since the first dimension can have two special channels of the last dimension and every other dimension can have two special channels of the previous dimension, the minimum requirement in terms of number of virtual channels still remains at three.

Deadlock and livelock freedom are not explicitly shown, as it follows directly from the extensions for the 3D mesh that similar numbering and proofs are also possible for the n-dimensional mesh. It is conceivable that more powerful fault tolerant strategies can be devised on similar lines as described earlier for 3D meshes.

6.3.3 Torus Networks

To perform deterministic routing in 2D torus networks, two virtual channels per physical channel are necessary. The virtual channel reconfiguration strategy discussed earlier for meshes can also be applied to torus networks. The fault topology that would be tolerated by such a strategy in a torus network would also be similar; the only difference being that there would be no boundary nodes which have to be handled differently. Thus the virtual channel requirements for the VCR approach applicable to 2D torus networks would be 4 channels (2 of which are configured as "special").

Torus networks offer a choice of two directions in which a message could correct its coordinates. Because of this choice some fault topologies that are not allowed in a 2D mesh network could be tolerated in a torus network. For example, the fault regions shown in Figure 6.2 can all be tolerated provided the router has a "smart" element present which would know that the message should try reaching some specific destinations by taking the alternate choice (which might use the wrap around links). The information necessary for these decisions could possibly be configured during the reconfiguration phase. Even under these conditions, some compound fault regions shown in Figure 6.11 cannot be tolerated.

The presence of wrap around links increases the possible fault topologies that torus networks can tolerate, but still, it may not justify the extra cost necessary for the "smart" element in the router. Thus, it is possibly desirable that torus networks could tolerate only the same set of fault topologies that could be tolerated in a mesh network (using the VCR approach).

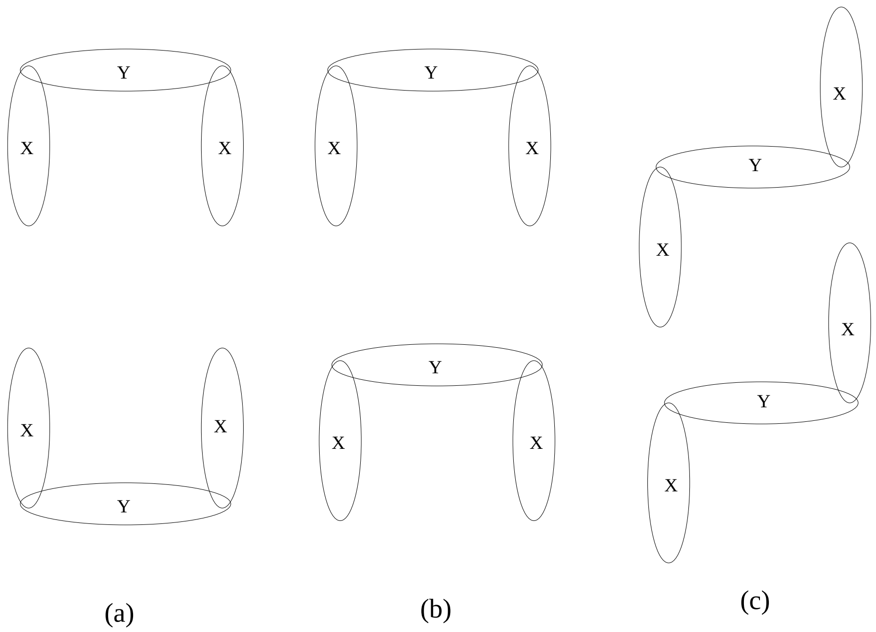


Figure 6.11: Compound fault regions that cannot be handled by the VCR approach in a torus network

6.4 Related Research

Previous approaches for fault tolerance in direct interconnection networks have been either hardware-redundancy based or based on the use of an adaptive routing algorithm. Hardware based approaches (e.g., [16]) add hardware redundancy (e.g., in the form of extra links) to the basic architecture and use this to isolate/tolerate faults. These techniques strive to provide good fault tolerance while minimising the cost of the redundant components.

Adaptive routing schemes can provide limited fault tolerance through exploitation of alternative paths to the desired destination. The Linder and Harden [47] scheme can tolerate a single faulty node in the network, at the cost of doubling the number of virtual channels in one dimension. These extra channels are used to route packets around faults, but are not used in the absence of faults. The planar adaptive scheme proposed by Chien and Kim [19] can tolerate multiple faults. However, the permissible topologies of fault regions are limited to rectangular shaped regions. The algorithm proposed by Glass and Ni [34] provides fault tolerance (to tolerate a single node or link fault), without adding any extra virtual channels. This algorithm is a modification of the turn model adaptive routing algorithm and can support misrouting. This algorithm has higher latencies than the corresponding non-fault tolerant algorithm, however, even in the absence of faults [34]. The dimension reversal algorithms proposed by Dally and Aoki [22] (discussed earlier in Chapter 3) also provide limited fault tolerance due to their adaptive routing capability. The main disadvantage of the Dally and Aoki algorithms is that the number of adaptive steps that a message can take is limited by a parameter, and therefore, messages that have lost their adaptivity cannot avoid a fault by taking an adaptive step.

To achieve powerful fault tolerance there should be some sort of special support in the interconnection network that could be used in the presence of faults. One such approach is the scheme called Fault Rings [17] which requires 4 additional virtual channels other than the channels needed for adaptive routing at every link in the network. The Fault Rings algorithm can tolerate an arbitrary number of rectangular faults in a mesh network. A rectangular fault is a set of faulty nodes and links that have a rectangular shaped topology. The basic idea of the Fault Rings algorithm is that a ring is constructed around the fault regions by using the fault free nodes and channels around the faulty region. The extra four channels remain unused under fault free conditions.

The MIT's Reliable Router [23] also provides special support for fault tolerance. The Reliable Router is a network switching element that can be used in 2D mesh interconnection networks. The Reliable Router can handle a single node or link fault anywhere in the network. The Reliable Router uses an adaptive algorithm similar to *-Channels. The routing scheme in Reliable Router tolerates single faults by having a separate fault tolerance network (with one special fault tolerance virtual channel on each link). Any message on encountering a fault uses the fault tolerance channel(s) to circumvent the fault. If the fault is in the X dimension, then messages encountering the fault use the fault tolerance channel to take a step off in the Y dimension and then reach their destination by using the normal channels in the network. If the fault is in the Y dimension, then messages on encountering the fault enter the fault tolerance network and use the Turn model partially adaptive algorithm to reach their destinations (using channels belonging to the fault tolerance network alone).

Table 6.6 compares the virtual channel requirements and fault tolerance capacity of different fault tolerance approaches. In contrast, the VCR approach proposed in this chapter needs just three virtual channels to handle "r-convex" fault topologies. Rectangular faults are special cases of r-convex faults, therefore, the VCR scheme is more powerful in its fault tolerance capacity than the other approaches.

6.5 Experimental Study

Two different experimental studies are described in this section. Section 6.5.1 presents the methodology that was followed in both these studies. The first study compares the VCR approach on a network employing adaptive routing against a scheme that is similar to the MIT Reliable Router. This scheme handles faults in a similar way as the MIT Reliable Router does, by having a dedicated network with turn model algorithm employed in it for the purposes of fault tolerance. In this study this scheme is referred to as "Dedicated Network with Turn model" (DNT)

Table 6.6: Comparison of the VCR approach with other approaches for the case of a 2D mesh network

Algorithm	Minimal Number of Virtual Channels	Guarantees on Fault Tolerance
LH	2	Single Node or Link fault
Planar Adaptive	6	Rectangular region
Turn Model	1	Single Node or Link fault
Dally and Aoki	$K^{\S}$	Faults limited by K
Fault Rings	Base Req. + 4 *	Multiple Rectangular regions
Reliable Router	3	Single Node or Link fault
VCR Approach	3	r-convex fault topologies

§ *K is the limit on the number of adaptive steps any message can take.*
* *Assuming a fully adaptive algorithm like the LH algorithm, the minimum number of virtual channels that would be needed is 6.*

approach. It is assumed that the DNT approach also disables adaptivity at necessary nodes so that deadlock possibilities (similar to the one illustrated in Figure 6.9) do not occur. Section 6.5.2 presents the results of the study comparing the VCR approach and the DNT approach. The second study studies the impact of faults on performance (using the VCR approach) and explores how some of the conclusions drawn in the previous chapters are affected by the presence of faults, in particular the relative performance of deterministic, adaptive, and source-directed routing. The results of this study are presented in Section 6.5.3.

6.5.1 Methodology

Experiments were performed using a flit-level network simulator. As in previous chapters, a 2D mesh network of size 12×12 was assumed.

Four different fault types were considered primarily: an X link fault, a Y link fault, a node fault, and a node with two faulty links (illustrated in Figure 6.12). Two different fault positions were considered for each fault type. The two fault positions that were experimented with for the X fault were at the center of the mesh diagonal and on the center node on the north edge of the network. Experiments with the Y fault involved placing it on the center node on the mesh diagonal and on center node on the east edge of the network. Experiments with the node fault and the two-link fault involved placing them on the center of the mesh diagonal and at the center node one column away from the east edge of the network. Experiments were also performed with a rectangular fault consisting of 4 node faults which was placed at the center of the network.

Experiments were performed with both the "open" workload and the "completion time" workload as used previously in Chapters 4 and 5. Both "matrix transpose" and "uniform traffic" patterns were considered with the open workload model. With the completion time workload model both the matrix transpose pattern and the general traffic pattern were considered. The number of virtual channels was varied in each study. The parameters chosen for simulation were the same as in the previous two chapters (unless specifically mentioned otherwise). For the VCR approach equi-partitioning was used to divide extra lanes in all experiments, and adaptivity was cancelled on all links on which special channels were configured (as discussed earlier in Section 6.2).

Two distinct studies were conducted. In the first study the performance of the VCR approach as implemented on a network with adaptive routing was compared against the DNT approach. The DNT approach uses a dedicated fault tolerance network (which has one special channel on every physical channel) in which the turn model algorithm is employed to get around faults. The DNT approach can handle only a single node or link fault. Note that the VCR approach also requires

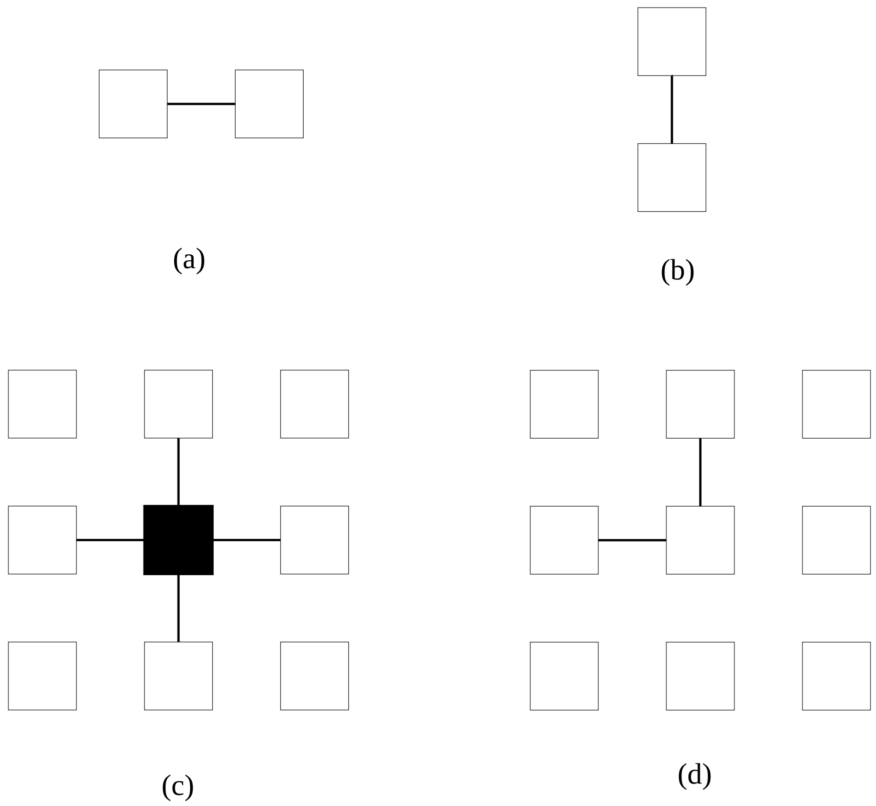


Figure 6.12: Four different fault types that were primarily considered

only a single extra channel (to be configured as special) on every physical channel for tolerating a single node or link fault. A new variation of the VCR approach was also considered where the use of channels is the same as in the VCR approach, the only difference being that there is one dedicated special channel present on all physical channels for the purposes of fault tolerance. This channel remains unused at all nodes expect for nodes adjacent to faults. The purpose of considering such a scheme was that it is similar to the DNT approach in that it has a dedicated channel on every physical channel for the purposes of fault tolerance. Comparisons with the DNT approach were made for the X fault, Y fault, and single node fault cases.

In the second study three different variations of the VCR approach were considered: the VCR approach on a deterministic network, the VCR approach on an adaptive network, and the VCR approach on a network with source-directed routing capability.

The conclusions from these studies are presented in the next two subsections. The complete set of simulation results (including the specific illustrative results presented in the next two subsections) are presented in Appendices A and B. In general, the inferences and conclusions drawn in the next two subsections are based on the entire set of results.

6.5.2 Comparisons between the DNT Approach and the VCR Approach

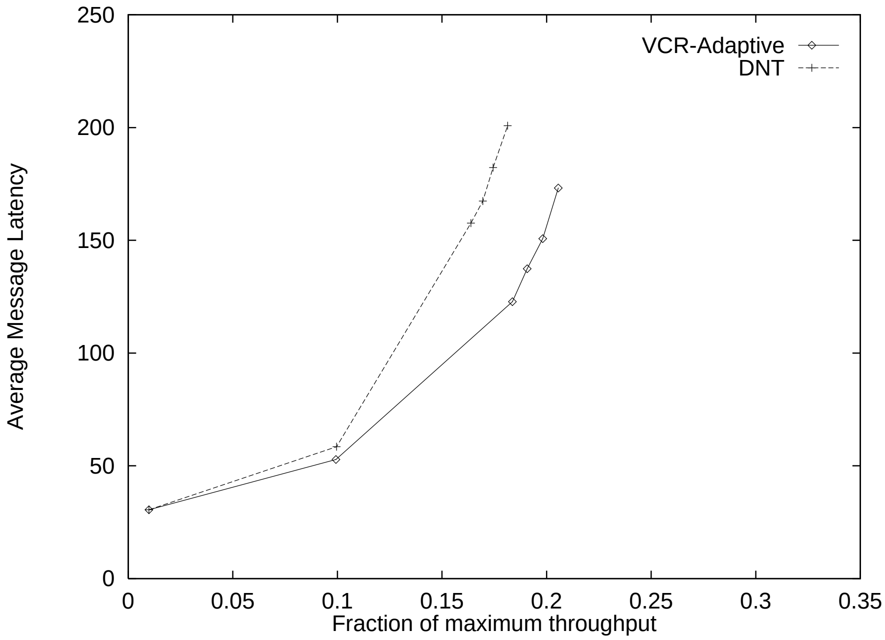


Figure 6.13: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single faulty X link at the side of the network

Figure 6.13 shows the results of a comparison between the DNT approach and the VCR scheme for an X fault, on a network employing adaptive routing, where the network is assumed to have 3 virtual channels for every physical channel, for the use of an X fault at the side of the network. Figure 6.14 shows the results for a Y fault with 3 virtual channels per physical channel.

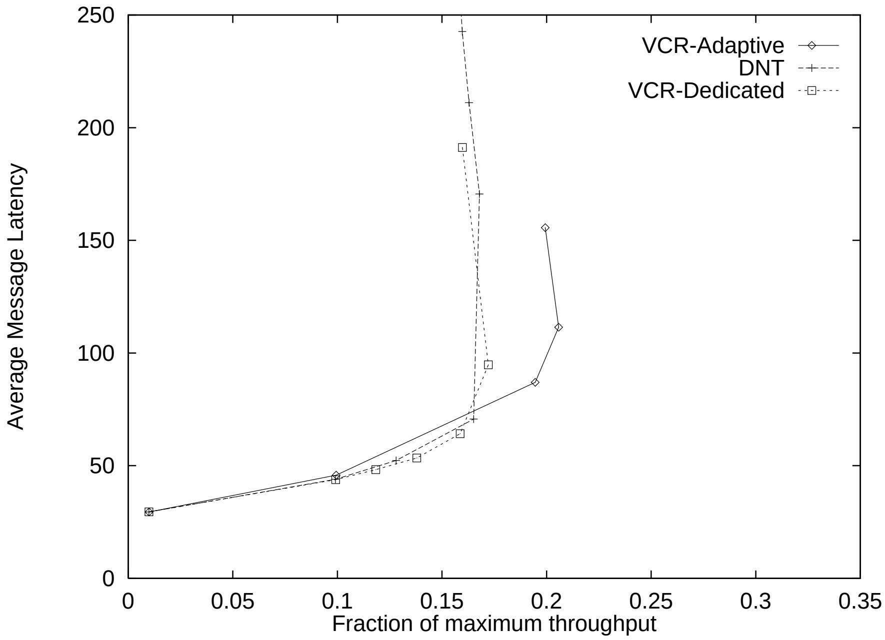


Figure 6.14: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single faulty Y link at the side of the network

In both these cases the performance of the VCR approach is better than the DNT approach because the VCR approach has a better utilisation of virtual channels at every node. Since the DNT approach has a dedicated fault tolerance network that can be used only by messages avoiding the fault, one channel remains unused at all the other nodes. Since the fault considered in Figure 6.13 is a single X fault the performance of the dedicated VCR approach would be the same as the DNT approach as both avoid the X faulty link in exactly same manner. In Figure 6.14 it can be seen that both the VCR-Dedicated and the DNT approach perform similarly.

These and the other results presented in Appendix A suggest that the VCR approach has better performance because it does not need a separate dedicated

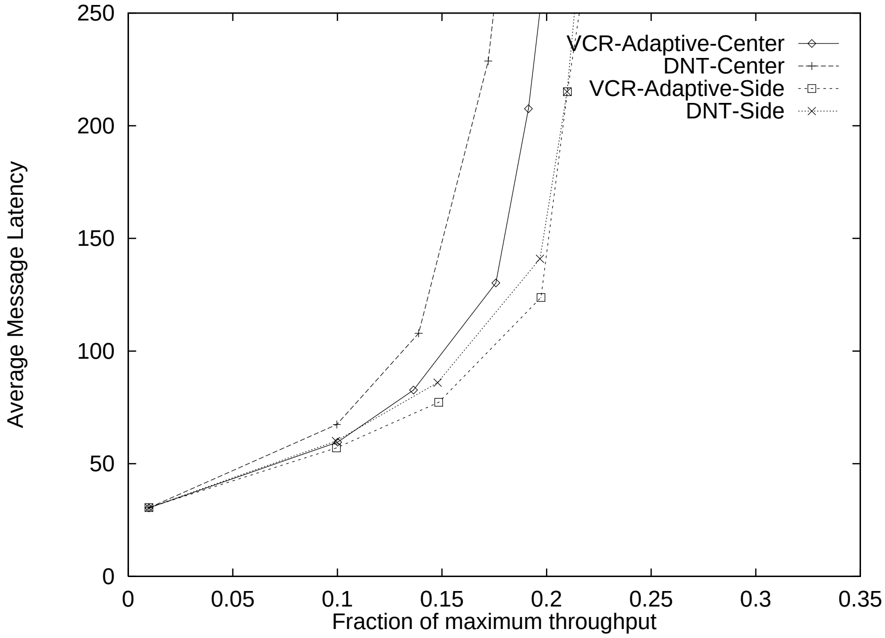


Figure 6.15: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty X link at the center or side of the network

virtual network. Usage of reconfigured channels around faults and choice of all allowed normal channels elsewhere proves to be quite helpful.

Fault type, fault location, type of traffic and number of virtual channels all affect the relative gains of the VCR approach over the DNT approach. Figure 6.15 compares these two approaches for an X fault when the fault is located at the center and at the side of the network. The gains of the VCR approach (over the DNT approach) are larger for a fault at the center than for a fault at the side.

The gains of the VCR approach over the DNT approach can vary by a large extent. For matrix transpose patterns, the gains are small, and for uniform traffic patterns the gains are large, as illustrated in Figures 6.16 and 6.17.

Also the gains of the VCR approach over the DNT approach are more for a Y fault than for an X link fault, as illustrated in Figures 6.17 and 6.18. Even for

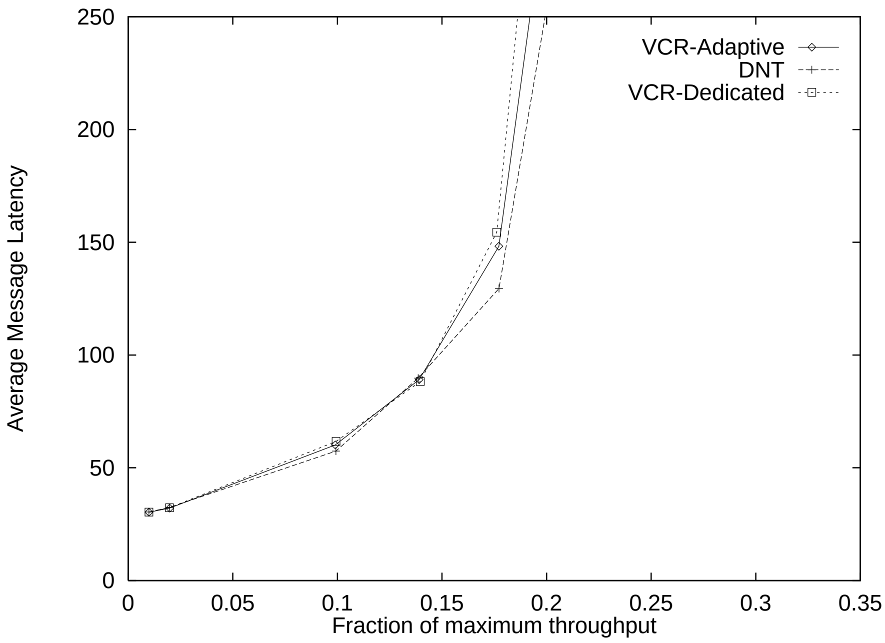


Figure 6.16: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty Y link at the center of the network

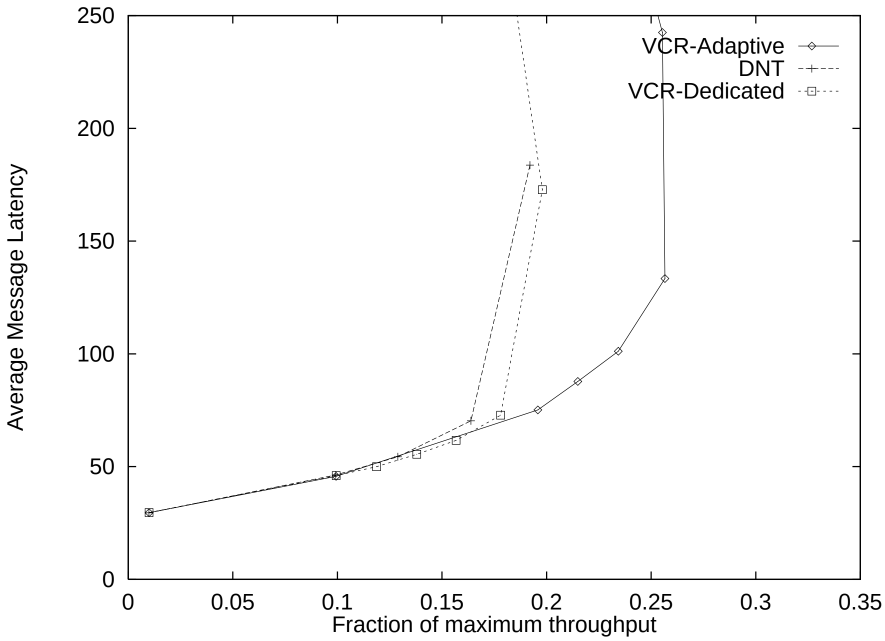


Figure 6.17: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty Y link at the center of the network

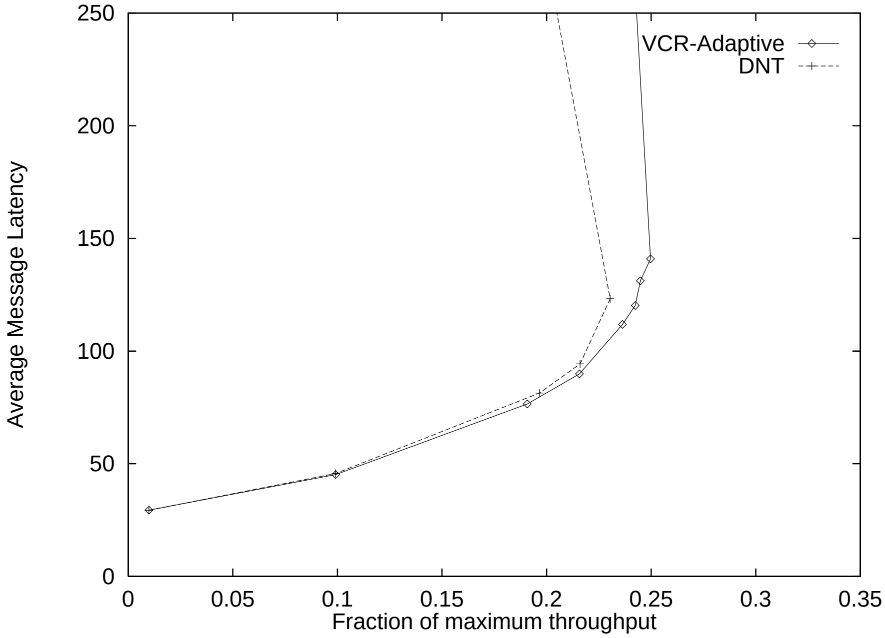


Figure 6.18: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty X link at the center of the network

node faults the VCR approach performs better than the DNT approach. This could be because for the Y fault and the node fault the DNT approach allocates only one special channel in a single direction (it cannot do that in both directions as that would cause a deadlock with the turn model algorithm) for traffic avoiding the fault. This places too much load on this single channel and thus affects the overall performance. Since the VCR approach allocates more channels based on equi-partitioning the gains are larger.

Under the completion time workload, both the DNT approach and the VCR approach perform poorly when compared against the completion time lower bound (Figures 6.19, 6.20 and 6.21). The lower bound still remains the same even in the presence of the simple faults, considered in these results, since analysis shows that these faults need not increase the maximum link multiplexing. In general, adaptive algorithms perform poorly on completion time metrics (as shown in Chapter 5),

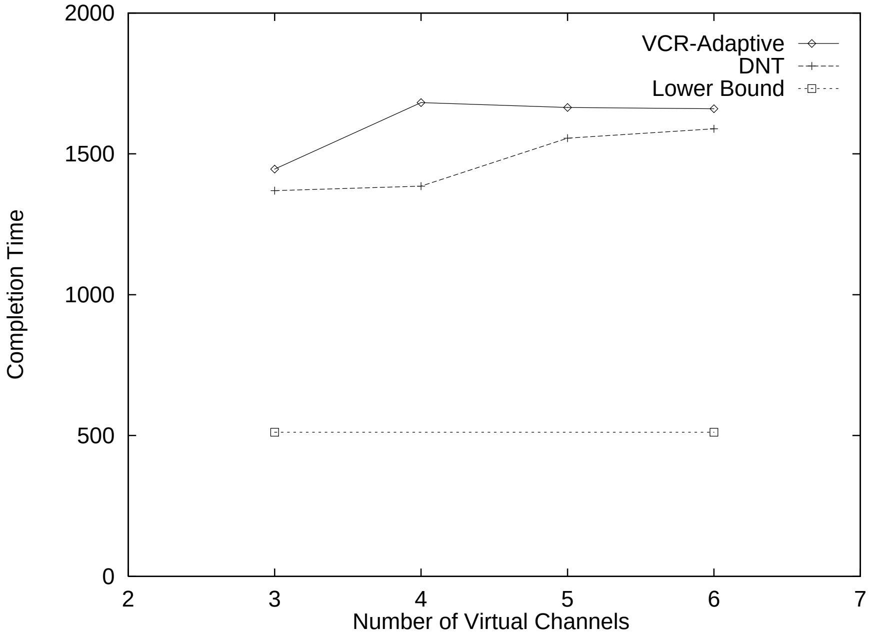


Figure 6.19: Completion times for matrix transpose traffic and a single faulty X link at the center of the network

and this problem becomes more pronounced with faults. Comparing Figures 6.19 and 6.20 it becomes clear that a fault at the center of the network inflates maximum latencies by a large extent. The VCR approach performs better in all cases where having an extra virtual channel per physical channel helps in improving performance. In Figure 6.19 the DNT approach performs slightly better than the VCR approach because here adding virtual channels does not help and in fact slightly hurts performance. Since the DNT approach effectively has one fewer channel (and thus one less multiplexing), it has slightly better performance.

In conclusion, it appears that the VCR approach outperforms the DNT approach, for experiments with an open workload model, in the three different fault topologies considered. The VCR approach does not have a separate fault tolerance network and, therefore, more channels are available for laning at any node, and this feature results in good improvement in performance over the DNT approach.

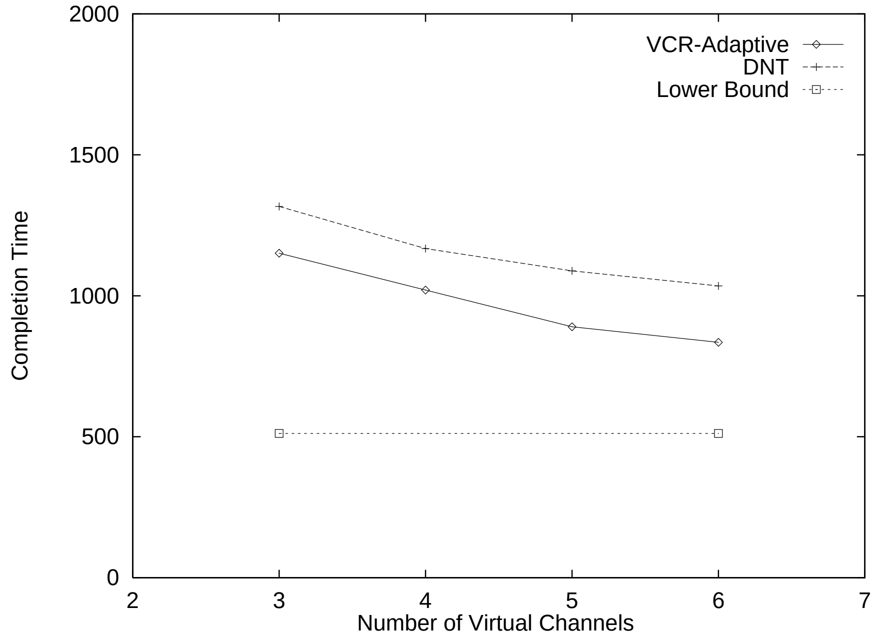


Figure 6.20: Completion times for matrix transpose traffic and a single faulty X link at the side of the network

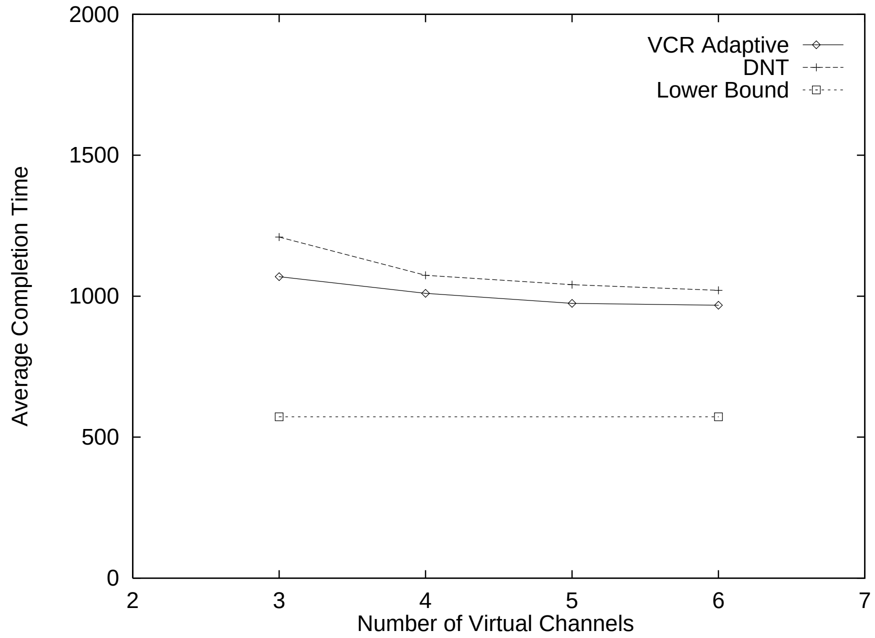


Figure 6.21: Average completion times for general traffic patterns and a single faulty X link at the center of the network

The VCR approach has an extra advantage of being able to configure more than the minimum number of required special channels (if necessary) around the faulty region for performance reasons. Such a scheme may not be attractive with the DNT approach as it would mean wasting more channels everywhere else in the network. On a completion time workload both the approaches perform poorly when compared against a completion time lower bound. If the VCR approach is tailored just to tolerate a single node or link fault (instead of the complete set of r-convex fault topologies) then the implementation complexity of this approach is quite likely to be comparable to that of the Reliable Router. The cost of reconfiguration is also likely to be justified as the performance increase appears to be significant.

6.5.3 Effect of Faults on Performance

The main motivation behind this study is to investigate how the conclusions of the previous chapters regarding the relative performance of various design options would be affected in the presence of faults in the network. Another goal is to investigate the absolute impact of faults on performance. The relative performance of VCR as applied to deterministic, adaptive and source-directed routing is also investigated.

The issue of improvements in performance of adaptive routing with increase in VCFC, as examined in Chapter 4 for a fault free network, was studied first. Figures 6.22 and 6.23 show the effect of faults on performance, for an open workload model, when the network has three and six virtual channels, respectively. The gains with VCFC remain quite similar (and do not seem to be affected by the presence of faults) to what was observed in Chapter 4. Similar results were observed for other fault types.

With the completion time workload, the addition of virtual channels did not always result in improved performance. In fact, with certain workloads adding extra virtual channels even hurt the performance (as illustrated earlier in Figure 6.19). This was because with more virtual channels per physical channel the potential number of messages that could reach the faulty link increases and this causes a drop in performance.

Thus one of the conclusions in this subsection is that faults do not affect the gains of combining adaptive routing and VCFC when the performance is dictated by average measures. On the other hand, when performance is dictated by maximum measures the gains of combining adaptive routing and VCFC are quite mixed.

The issue of relative performance of VCR applied with deterministic, adaptive and source-directed routing with an open workload model was studied next. It was observed that in the presence of faults adaptive routing (with sufficient virtual channels) manages to outperform deterministic routing by a larger margin than without faults, for certain traffic patterns as illustrated in Figures 6.24 and 6.25. This is because with adaptive routing, the back pressure of waiting messages makes other messages take alternate routes, which improves performance. The performance of adaptive algorithms is also as good as source-directed routing can provide with an open model workload (see Figure 6.26).

Thus, another conclusion of this subsection is that with an open workload model, where performance is dictated by average measures, faults have a larger impact on deterministic routing than on adaptive routing. Therefore, the use of adaptive routing becomes more attractive in the presence of faults. Also, when performance is dictated by average measures, adaptive routing performs close to source-directed routing even in the presence of faults.

Next, the issue of relative performance of VCR applied with deterministic, adap-

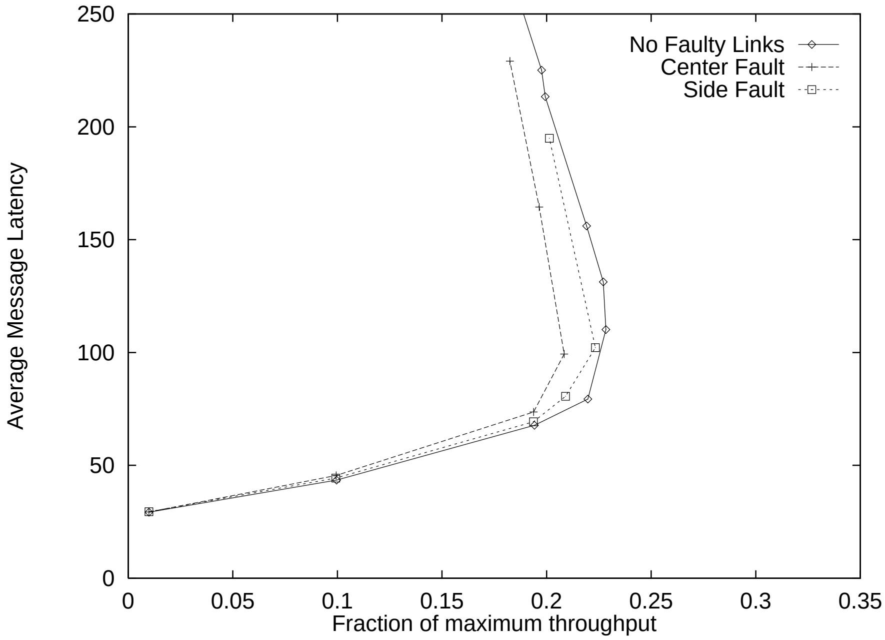


Figure 6.22: Effect of fault location on performance for a single faulty X link in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

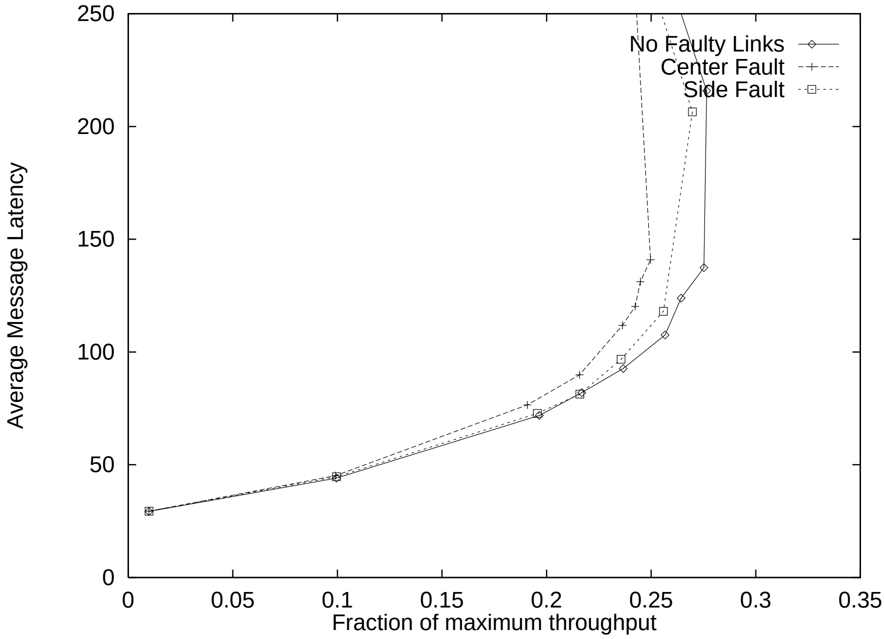


Figure 6.23: Effect of fault location on performance for a single faulty X link in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

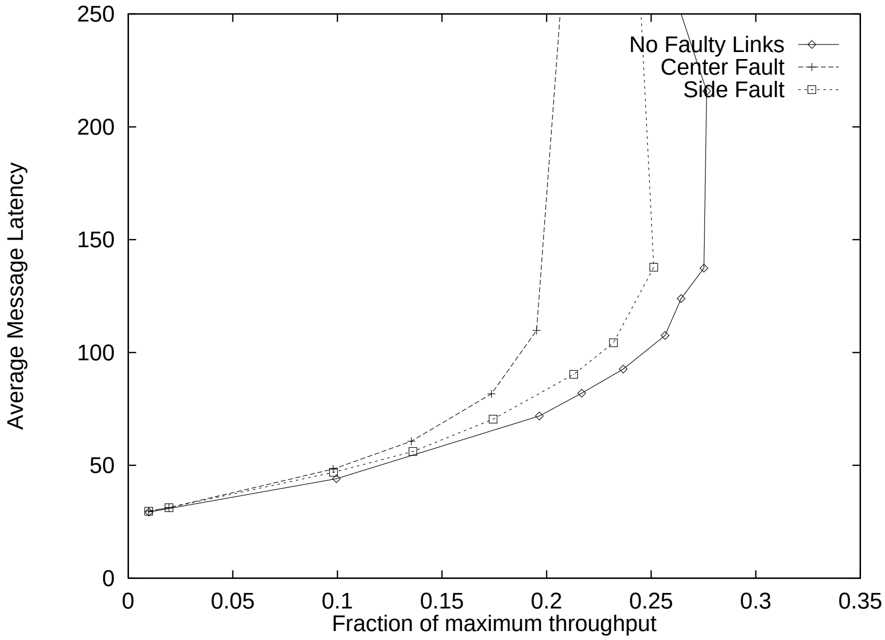


Figure 6.24: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

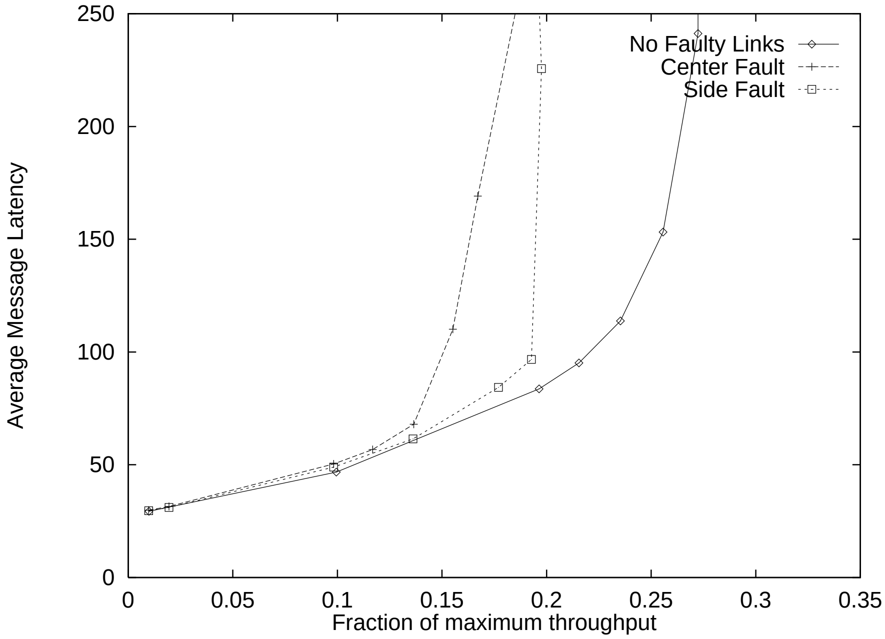


Figure 6.25: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

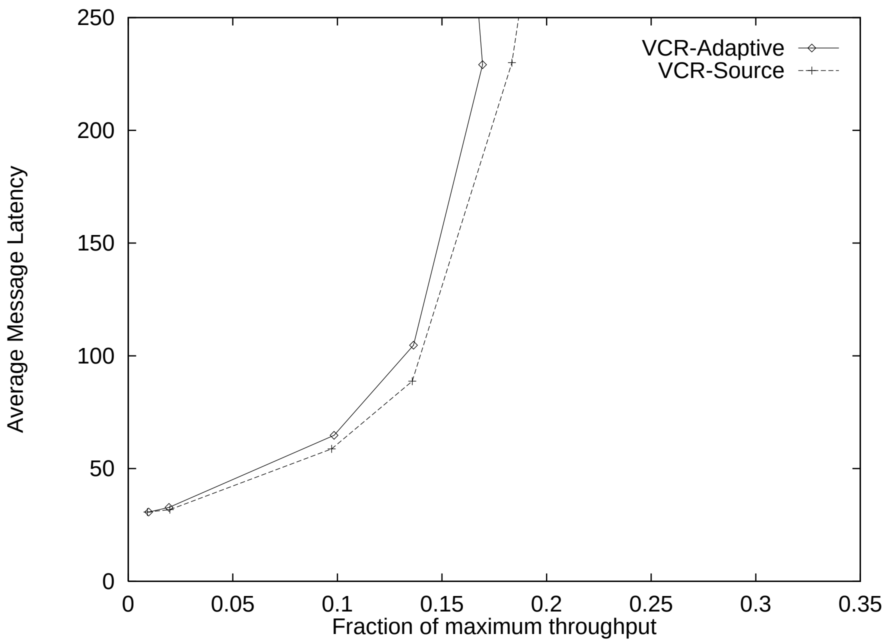


Figure 6.26: Comparison of VCR employed with adaptive routing against VCR employed with source-directed routing for a single node fault in a network for matrix transpose traffic

tive and source-directed routing with a completion time workload model was studied. With completion time workloads, somewhat different conclusions were drawn. Adaptive routing appears to perform very poorly in the presence of faults (see Figures 6.27 and 6.28). It performs poorly (in terms of maximum message latency) even when compared to deterministic routing. However, on the average latency measure for the same experiment, adaptive routing performed reasonably well (see Figure 6.29). This poor performance on the maximum message latency measure is because adaptive routing sends more messages into the center of the network and a fault there increases the multiplexing on the links surrounding the fault, thus inflating the maximum of message latencies (which determines the completion time). On the other hand in this figure, the VCR applied with source-directed routing does remarkably well and provides performance close to that for source-directed routing

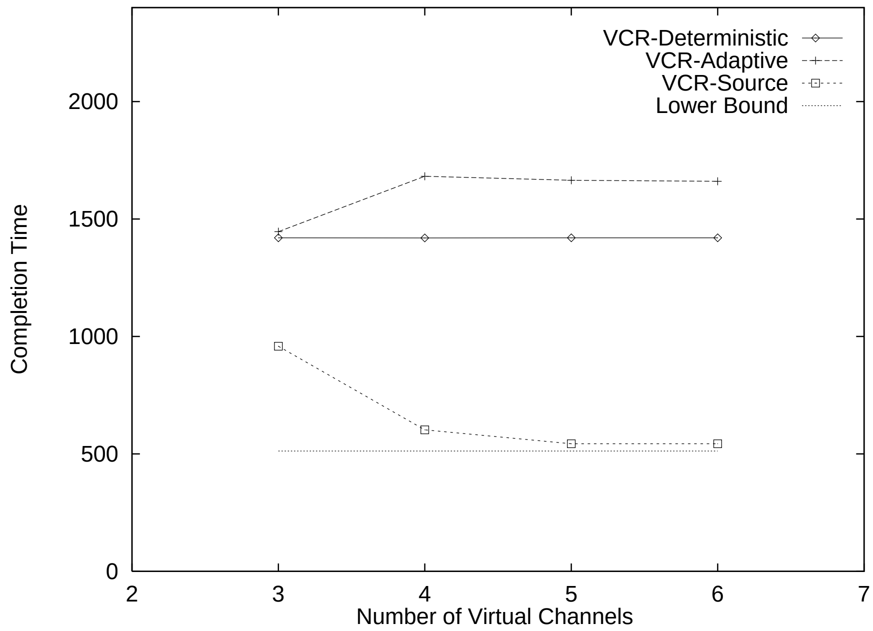


Figure 6.27: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the center of the network

on a non-faulty network.

With completion time workload, when the fault is moved from the center of the network to the side of the network, the performance of adaptive routing improves and becomes close to non-faulty performance (see Figure 6.30). Side faults happen to be very bad for deterministic routing.

Thus, when performance is dictated by the maximum of message latencies, with faults in the network, adaptive routing can potentially give very poor performance and can even perform worse than deterministic routing. Deterministic routing also suffers a drop in performance in the presence of faults (particularly side faults). However the impact of faults on the completion time measure to a very large extent depends on the location of the fault and the traffic pattern employed. Source-directed routing appears to be robust and does well with all fault cases. This leads to the conclusion that in the presence of faults, the global knowledge of the

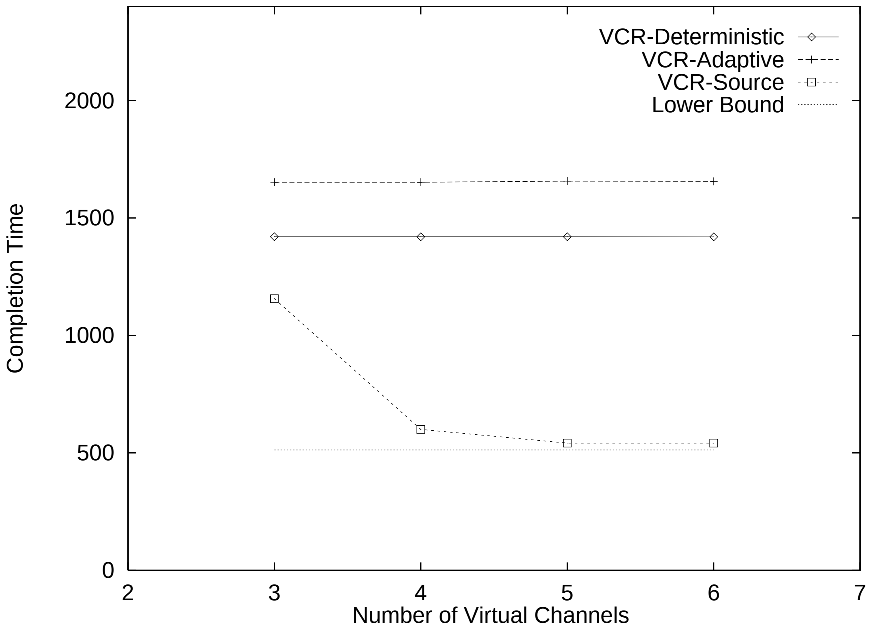


Figure 6.28: Completion times for routing algorithms for matrix transpose traffic and a single faulty Y link at the center of the network

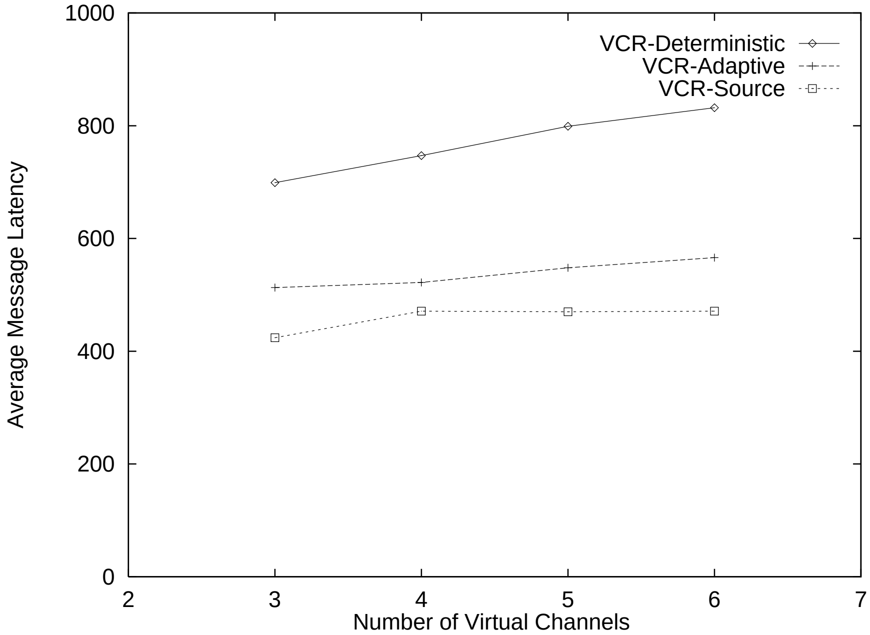


Figure 6.29: Average message latencies under completion time workloads for routing algorithms for matrix transpose traffic and a single faulty Y link at the center of the network

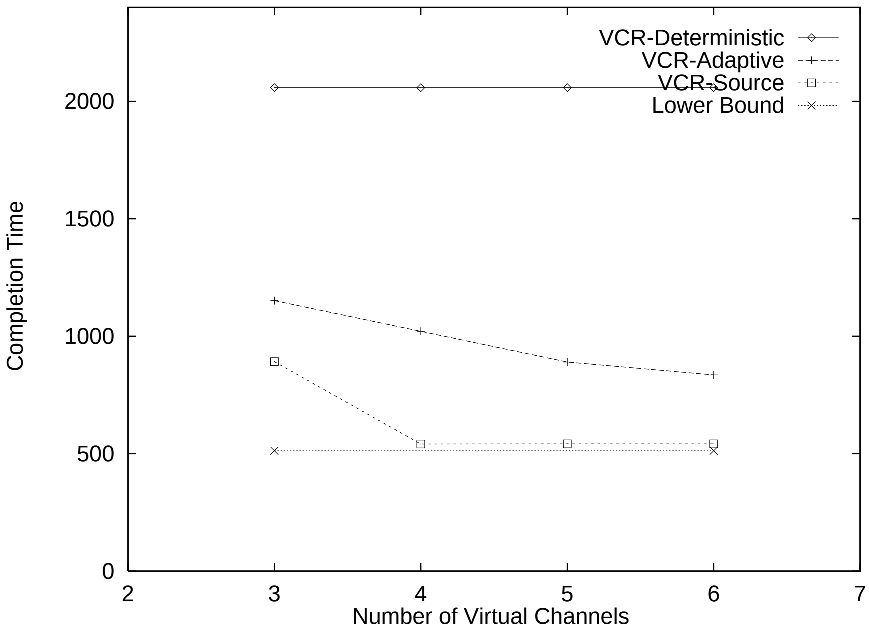


Figure 6.30: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the side of the network

fault pattern and the traffic pattern can be used very efficiently to avoid "routing mistakes".

Fault type, fault location, traffic pattern, and number of virtual channels all affect routing performance. Center faults usually cause a larger drop in performance for adaptive routing algorithms as adaptive algorithms generally tend to route messages into the center of the network (e.g., see Figure 6.24). Node faults cause a bigger drop in performance than link faults (e.g., compare Figures 6.23 and 6.24). Performance degrades considerably with increasing fault size as seen in Figure 6.31. All these results are presented in detail in Appendix B.

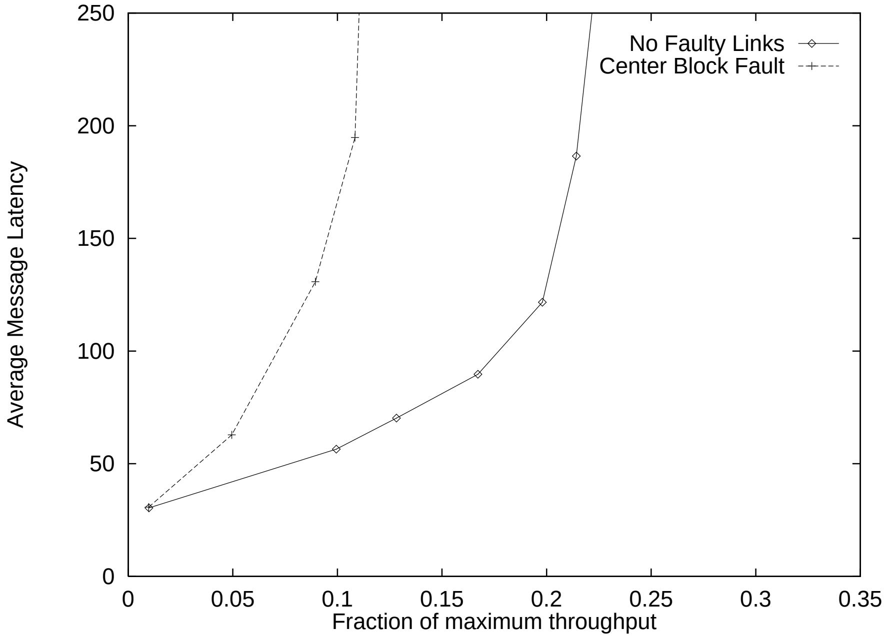


Figure 6.31: Effect of rectangular fault of four nodes at the center of the network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

6.6 Conclusions

The primary motivation in this chapter was to design a fault tolerance scheme that is simple and cost effective, and at the same time powerful enough to handle a large set of fault topologies. The Virtual Channel Reconfiguration technique was the fault tolerance approach developed for this purpose.

The VCR approach consists of three major steps. The initial step involves checking whether fault topologies obey the r-convexity requirement. If they do not obey the requirement, the non-r-convex topologies are converted into r-convex topologies by marking some nodes as unreachable, and the associated links as failed. Then channels at nodes adjacent to faults are reconfigured. These reconfigured channels are used by messages to avoid faults.

The VCR approach can tolerate multiple node and/or link faults in the network as long as they satisfy the r-convexity requirement. The routing in the VCR approach is general and does not require tagging messages. The requirement in terms of virtual channels is three virtual channels per physical channel. Thus the cost of implementing such a scheme is likely to be low. The scheme can be easily combined with adaptive routing and VCFC and can also be extended to work for higher dimensional meshes and tori. The virtual channel requirement remains constant at three even for higher dimensional meshes. The VCR approach does not affect performance in the absence of faults and in the presence of faults gives very reasonable performance when compared against the DNT approach. The fault tolerance capacity of the VCR approach is much more than any other approach in the current literature. Thus, in comparison with other approaches, the proposed VCR approach appears to be quite attractive.

The study of fault tolerance in this chapter has shown the following:

- The VCR approach outperforms the DNT approach; this is because the VCR approach does not have a separate fault tolerance network and, therefore, more channels are available for laning at any node, which results in good improvement in performance.

- Faults do not affect the gains of combining adaptive routing and VCFC (when employed with VCR) with an open workload model. On the other hand, with a completion time workload model the gains of combining adaptive routing and VCFC in the presence of faults (and employing VCR) are quite mixed.

- Adaptive routing (coupled with VCR) appears to provide better fault tolerance than deterministic routing and close to source-directed routing when performance is determined by average measures.

- With completion time workloads, where performance is dictated by the maximum of message latencies, adaptive routing can give very poor performance and can even perform worse than deterministic routing.

- With faults in the network, the performance to a large extent depends on the type of the fault, its location, the traffic pattern in the network, and the number of available virtual channels. However, in general, center faults appear to have a greater negative impact on performance than side faults. Also, the impact of an X link fault is small compared to other fault cases. Y link faults are generally as expensive as a node fault.

- Source-directed routing appears to be robust and does well with all fault cases. Thus source-directed routing becomes potentially more valuable as a performance improvement strategy in the presence of faults in the network.

Chapter 7

Conclusions

This chapter summarises the objectives of the thesis, the algorithms developed, the performance studies undertaken, and the conclusions arising from this research. The major contributions of this thesis are outlined and directions for future work are also identified.

7.1 Summary

This thesis was concerned with the issue of how routing can be done most effectively in wormhole networks. The following questions were addressed:

- How well does adaptive routing perform when compared to deterministic routing?
- Can adaptive routing be made to perform uniformly well for all traffic patterns? Would a combination of adaptive routing and VCFC be fruitful?
- How much does the purely local nature of decisions limit adaptive routing performance?

- Is there scope for further improvements? If so, what approach should be followed to achieve these improvements?
- How can single and multiple faults be handled effectively?
- How does the relative performances of deterministic, adaptive and source routing change in the presence of faults?

A flit-level network simulator was developed to aid in studying these questions. Experiments were performed with different traffic patterns, workload models and metrics.

The study reported in Chapter 4 varied the amount of VCFC, in the context of two different adaptive algorithms, and studied the effect on performance with an open workload model. It was shown that employing adaptive routing with a minimal channel topology (no extra virtual channels) can result in poor performance for uniform and some non-uniform traffic patterns. It was also shown that using VCFC alone with deterministic routing can severely degrade performance for certain traffic patterns. But when VCFC is combined with adaptivity in routing, these effects are mitigated and good performance results. Thus, the main conclusion of this study was that both adaptive routing and VCFC are indeed beneficial (and can provide a large improvement over deterministic routing) and their real benefit can be felt when they are employed together. This study also showed that only a limited number of virtual channels per physical channel are needed (at most six in this study) for good performance.

In Chapter 5, the issue of the room for improvement in the network performance provided by locally adaptive routing policies was considered. Initially, bounds were derived on maximum achievable performance in terms of throughput, latency, and completion time. Next, limits on realistically achievable performance were quantified by considering source-directed routing methods, under the favourable assumption of full predictability of the traffic pattern. A specific source-directed routing method was proposed, of interest in its own right, and its performance was studied in comparison to that of adaptive and deterministic routing in a set of simulation experiments on 2-dimensional meshes.

Both the theoretical limits and experimental results show that adaptive routing provides good performance when performance is dictated by average message latency. When performance is determined more by maximum message latencies (such as may be the case when application performance is limited by the time to fully complete a message exchange phase), however, it appears that adaptive routing techniques are less effective, and that there is a large scope for improvement. The source-directed routing method proposed in Chapter 5 appears to be a reasonable approach to achieving such improvements. The proposed method can interoperate with either deterministic or adaptive routing, so as to allow these latter techniques to be used as desired when message patterns are unpredictable. Further, it imposes minimal additional router cost beyond that required anyway for the deterministic or adaptive routing technique that is otherwise employed.

Another major thrust of the thesis was the development, in Chapter 6, of a cost-effective and powerful fault tolerance approach (called the VCR approach) capable of tolerating a large set of faults. The fault tolerance capacity of the VCR approach is much more than that of any other approach in the current literature. A performance study was also undertaken in Chapter 6 using the VCR approach to study the impact of faults on performance. It was shown that the VCR approach achieves better performance by not having a dedicated sub-network for fault tolerance. This study also showed that adaptive routing (coupled with VCR) can provide better fault tolerance than deterministic routing when performance is governed by average measures. On the other hand, with completion time workloads, where performance is dictated by the maximum of message latencies, adaptive routing can give very poor performance and can even perform worse than deterministic routing. Source routing appears to be robust and does well with all fault cases. Thus source routing becomes potentially more valuable as a performance improvement strategy in the presence of faults in the network. In general, with faults in the network the performance to a large extent depends on the type of the fault, its location, the traffic pattern in the network and the number of available virtual channels.

7.2 Major Contributions

The following are the contributions of the thesis.

- Development of a taxonomy for various approaches to adaptive routing in wormhole networks.
- Demonstration of the benefits of combining VCFC with adaptive routing and the study of the interaction between these two performance improvement strategies.
- Derivation of theoretical bounds for routing performance and quantification of adaptive routing performance against these bounds, and against more realistically achievable limits.
- Development of a new cost-effective source-directed routing algorithm.

- Illustration of benefits and limitations of deterministic, adaptive and source-directed routing.
- Development of a novel approach to fault tolerance based on virtual channel reconfiguration that is very cost effective, yet powerful enough to handle a large set of fault topologies.
- Illustration of how faults impact performance and how they affect some of the previous conclusions.

7.3 Directions for Future Research

One particular topic for future research is to understand the impact of network dimension on the results presented in this thesis. With higher dimensions the number of alternate paths that become available would increase and this can potentially increase the gains of adaptive routing.

This thesis demonstrated the benefits that could be achieved with source-directed routing. These benefits can be achieved only if proper mechanisms for compiler/run-time support for source-directed routing are implemented. Software-level implementation issues concerning source-directed routing is another topic which needs further study.

This thesis was concerned primarily with point-to-point message communication. Collective communications like broadcast, multicast, and barrier synchronisation are becoming increasingly common in wormhole networks. Support for collective communication has emerged as an area of active research [49]. In this domain, source-directed routing might be very useful in delivering messages efficiently to multiple nodes. Tailoring a source-directed routing scheme to work in a multicast/broadcast environment would be an another interesting area for future work. Performance studies comparing different multicast/broadcast approaches would also be of great interest to the research community.

Prevention has been the traditional approach to dealing with deadlock in wormhole networks. Recently some new deadlock recovery schemes [6, 19] have been proposed which promise good improvements in performance. Some limitations of these schemes are that they could possibly have a high hardware cost [19] or they can make the router less fault tolerant [6]. Such schemes would eliminate the load imbalances caused due to deadlock prevention [2] and can, therefore, potentially improve the performances of adaptive algorithms. A thorough investigation of the benefits/limitations of these new approaches, and how they would affect some of the conclusions of this thesis, is yet another potentially fruitful area of research.

Bibliography

[1] B. Abali and C. Aykanat, "Routing Algorithms for IBM SP1", *Proceedings of 1st International Workshop on Parallel Computer Routing and Communication*, Seattle, Washington, May 1994, pp. 161-175.

[2] V.S. Adve and M.K. Vernon, "Performance Analysis of Mesh Interconnection Networks with Deterministic Routing", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 3, March 1994, pp. 225-246.

[3] A. Agarwal, "Limits of Interconnection Network Performance", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, October 1991, pp. 398-412.

[4] A. Agarwal, B.H. Lim and J. Kubiatowicz, "April: A Processor Architecture for Multiprocessing", *Proceedings of the 17th International Symposium of Computer Architecture*, Seattle, Washington, May 1990, pp. 104-114.

[5] J.D. Allen, P.T. Gaughan, D.E. Schimmel and S. Yalamanchili, "Ariadne - An Adaptive Router for Fault-tolerant Multicomputers", *Proceedings of the 21st International Symposium on Computer Architecture*, Chicago, Illinois, April 1994, pp. 278-288.

[6] K.V. Anjan and T.M. Pinkston, "An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA", *Proceedings of the 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pp. 201-210.

[7] K. Aoyama and A.A. Chien, The Cost of Adaptivity and Virtual Lanes in a Wormhole Router, Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1993.

[8] H.G. Badr and S. Podar, "An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh-Connected Topologies", *IEEE Transactions on Computers*, Vol. 38, No. 10, October 1989, pp. 1362-1371.

[9] S. Balakrishnan and D.K. Panda, "Impact of Multiple Consumption Channels on Wormhole Routed k-ary n-cube Networks", *Proceedings of the 1993 International Parallel Processing Symposium*, Newport Beach, California, April 1993, pp. 163-167.

[10] P.E. Berman, L. Gravano, G.D. Pifarre and J.L.C. Sanz, "Adaptive Deadlock- and Livelock-Free Routing with All Minimal Paths in Torus Networks", *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1992.

[11] S. Bokhari, Communication Overhead on the Intel Paragon, IBM SP2 and Meiko CS-2, ICASE Interim Report No. 28, Institute for Computer Applications in Science and Engineering, Hampton, Virginia, September 1995.

Case for Chaotic Adaptive Routing, Technical Report CSE-94-02-04, Department of Computer Science, University of Washington, February 1994.

[12] K. Bolding, M.L. Fulgham and L. Snyder, The Case for Chaotic Adaptive Routing, Technical Report CSE-94-02-04, Department of Computer Science, University of Washington, February 1994.

[13] R.V. Boppana and S. Chalasani, "A Comparison of Adaptive Wormhole Routing Algorithms", *Proceedings of the 20th International Symposium on Computer Architecture*, San Diego, California, May 1993, pp. 351-360.

[14] S. Borkar *et al.*, "Supporting Systolic and Memory Communication in iWarp", *Proceedings of the 17th International Symposium on Computer Architecture*, Seattle, Washington, May 1990, pp. 70-81.

[15] Y.M. Boura and C.R. Das, "A Class of Partially Adaptive Routing Algorithms for n-Dimensional Meshes", *Proceedings of the 1993 International Conference on Parallel Processing*, St. Charles, Illinois, Vol. III, August 1993, pp. 175-182.

[16] J. Bruck, R. Cypher and C. Ho, "Fault-Tolerant Meshes and Hypercubes with Minimal Numbers of Spares", *IEEE Transactions on Computers*, September 1993, Vol. 42, No. 10, pp. 1089-1104.

[17] S. Chalasani and R.V. Boppana, "Fault-Tolerant Wormhole Routing in Tori", *Proceedings of 8th ACM International Conference on Supercomputing*, Manchester, England, July 1994, pp. 146-155.

[18] A.A. Chien and J.H. Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors", *Proceedings of the 19th International Symposium on Computer Architecture*, Gold Coast, Australia, May 1992, pp. 278-287.

[19] A.A. Chien and M. Konstantinidou, "Workloads and Performance Metrics for Evaluating Parallel Interconnects", *IEEE TCCA Newsletter*, Fall 1994.

[20] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks", *IEEE Transactions on Computers*, Vol. 39, No. 6, June 1990, pp. 775-785.

[21] W.J. Dally, "Virtual-Channel Flow Control", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, March 1992, pp. 194-205.

[22] W.J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 4, April 1993, pp. 466-475.

[23] W.J. Dally, L.R. Dennison, D. Harris, K. Kan and T. Xanthopoulos, "The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers", *Proceedings of the 1st International Workshop on Parallel Computer Routing and Communication*, Seattle, Washington, May 1994, pp. 241-255.

[24] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", *IEEE Transactions on Computers*, Vol. 36, No. 5, May 1987, pp. 547-553.

[25] B.V. Dao, J. Duato and S. Yalamanchili, "Configurable Flow Control Mechanisms for Fault-Tolerant Routing", *Proceedings of the 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pp. 220-229.

[26] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 12, December 1993, pp. 1320-1331.

[27] C. Fang and T. Szymanski, "An Analysis of Deflection Routing in Multidimensional Regular Mesh Networks", *Proceedings of IEEE INFOCOM '91*, Miami, Florida, April 1991, pp. 859-868.

[28] E.W. Felten, The Case for Application-Specific Communication Protocols, Technical Report 92-03-11, Department of Computer Science, University of Washington, March 1992.

[29] P.T. Gaughan and S. Yalamanchili, "Adaptive Routing for Hypercube Interconnection Networks", *Computer*, Vol. 26, No. 5, May 1993, pp. 12-23.

[30] P.T. Gaughan and S. Yalamanchili, "A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 5, May 1995, pp. 482-497.

[31] C.J. Glass and L.M. Ni, Adaptive Routing in Mesh-Connected Networks, Technical Report MSU-CPS-ACS-45, Department of Computer Science, Michigan State University, October 1991.

[32] C.J. Glass and L.M. Ni, Maximally Fully Adaptive Routing in 2D Meshes, Technical Report MSU-CPS-ACS-51, Department of Computer Science, Michigan State University, January 1992.

[33] C.J. Glass and L.M. Ni, "The Turn Model of Adaptive Routing", *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992, pp. 278-287.

[34] C.J. Glass and L.M. Ni, "Fault-Tolerant Wormhole Routing in Meshes", *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, Toulouse, France, June 1993, pp. 240-249.

[35] I.S. Gopal, "Prevention of Store-and-Forward Deadlock in Computer Networks", *IEEE Transaction on Communication*, Vol. 12, No. 12, December 1985, pp. 1258-1264.

[36] Intel Corporation, *Paragon XP/S Product Overview*, 1991.

[37] C.R. Jesshope, P.R. Miller and J.T. Yantchev, "High Performance Communications in Processor Networks", *Proceedings of the 16th International Symposium on Computer Architecture*, May 1989, pp. 150-157.

[38] P. Kermani and L. Kleinrock, "Virtual Cut-through: A New Computer Communication Switching Technique", *Computer Networks* , Vol. 3, October 1979, pp. 267-286.

[39] J.H. Kim and A.A. Chien, "Evaluation of Wormhole Routed Networks Under Hybrid Traffic Loads", *Proceedings of the 1993 Hawaii International Conference on System Sciences*, Koloa, Hawii, January 1993, pp. 43-52.

[40] J.H. Kim, Z. Liu and A.A. Chien, "Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing", *Proceedings of the 21st International Symposium on Computer Architecture*, Chicago, Illinois, April 1994, pp. 289-300.

[41] S. Konstantinidou and L. Snyder, "Chaos router: Architecture and Performance", *Proceedings of the 18th International Symposium of Computer Architecture*, Toronto, Canada, April 1991, pp. 212-221.

[42] V. Kumar, A. Graham, A. Gupta and G. Karypis, *Introduction to Parallel Computing*, Benjamin Cummings, 1994.

[43] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik, *Quantitative System Performance : Computer Systems Analysis Using Queuing Network Models*, Prentice-Hall Inc., 1984.

[44] D. Lenoski, J. Laudon, K. Gharacharloo, W.D. Weber, A. Gupta, J. Hennessy, M. Horowitz and M. Lam, "The Stanford DASH multiprocessor", *Computer*, Vol. 25, No. 3, March 1992, pp. 63-79.

[45] J. Li and M. Chen, "Compiling Communication-Efficient Programs for Massively Parallel Machines", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 3, July 1991, pp. 361-375.

[46] X. Lin, P.K. McKinley and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 8, August 1994, pp. 793-804.

[47] D.H. Linder and J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes", *IEEE Transactions on Computers*, Vol. 40, No. 1, January 1991, pp. 2-12.

[48] V.M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M.A. Mohamed, B. Nitzerg, J.A. Telle and X. Zhong, "OREGAMI: Tools for Mapping Parallel Computations to Parallel Architectures", *International Journal of Parallel Programming*, Vol. 20, No. 3, March 1991, pp. 237-269.

[49] P.K. McKinley, Y. Tsai and D.F. Robinson, A Survey of Collective Communication in Wormhole-Routed Massively Parallel Computers, Technical Report MSU-CPS-94-35, Department of Computer Science, University of Michigan, June 1994.

[50] T. Nesson and L. Johnsson, "ROMM Routing: A Class of Efficient Minimal Routing Algorithms", *Proceedings of the 1st International Workshop on Parallel Computer Routing and Communication*, Seattle, Washington, May 1994, pp. 185-199.

[51] J.Y. Ngai and C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks", *Proceedings of the 1st Symposium on Parallel Algorithms and Architectures*, April 1989, pp. 1-9.

[52] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks", *Computer*, Vol. 26, No. 2, February 1993, pp. 62-76.

[53] L.M. Ni and D.K. Panda, A Report on the ICPP '94 Panel on Sea of Interconnection Networks: What's your Choice?, Technical Report, Department of Computer Science, Michigan State University, 1994.

[54] D.K. Panda, S. Singal and P. Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme", *Proceedings of the 1st International Workshop on Parallel Computer Routing and Communication*, Seattle, Washington, May 1994, pp. 131-146.

[55] M.J. Pertel, A Critique of Adaptive Routing, Technical Report CS-TR-92-06, Department of Computer Science, California Institute of Technology, June 1992.

[56] G.F. Pfister and V.A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks", *IEEE Transactions on Computers*, Vol. 34, No. 10, October 1985, pp. 943-948.

[57] G.D. Pifarre, L. Gravano, S.A. Felperin and J.L.C. Sanz, "Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes and Other Networks: Algorithms and Simulations", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 3, March 1994, pp. 247-263.

[58] S. Ramany and D. Eager, "The Interaction Between Virtual Channel Flow Control and Adaptive Routing in Wormhole Networks", *Proceedings of the 8th ACM International Conference on Supercomputing*, Manchester, England, July 1994, pp. 136-145.

[59] J. Saltz, H. Berryman and J. Wu, "Multiprocessors and Runtime Compilation", ICASE Report No. 90-59, September 1990.

[60] L. Schwiebert and D.N. Jayasimha, "Optimal Fully Adaptive Wormhole Routing for Meshes", *Proceedings of Supercomputing 1993*, Portland, Oregon, November 1993, pp. 782-791.

[61] C.L. Seitz, "The Cosmic Cube", *Communications of the ACM*, Vol. 28, No. 1, January 1985, pp. 22-33.

[62] C. Su and K.G. Shin, "Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel", *Proceedings of the 1993 International Conference on Parallel Processing*, St. Charles, Illinois, Vol. I, August 1993, pp. 227-231.

[63] H. Sullivan and T.R. Bashkow, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine", *Proceedings of the 4th International Symposium on Computer Architecture*, March 1977, pp. 105-124.

[64] A.S. Tanenbaum, *Computer Networks*, Prentice-Hall Inc., 1988.

[65] X. Zhong and V.M. Lo, "An Efficient Heuristic for Application-Specific Routing on Mesh Connected Multicomputers", *Proceedings of the 1992 International Conference on Parallel Processing*, St. Charles, Illinois, Vol. III, August 1992, pp. 216-219.

Appendix A

Comparisons of the DNT approach and the VCR approach

In this appendix the results of the comparison between the VCR approach and the DNT approach are presented. The performances of these two schemes are judged when the network has either a single X link fault, or a single Y link fault, or a node fault. The comparisons are done on both average and maximum measures. Wherever appropriate, the VCR approach with dedicated channel is also considered in the comparisons (as explained in Chapter 6).

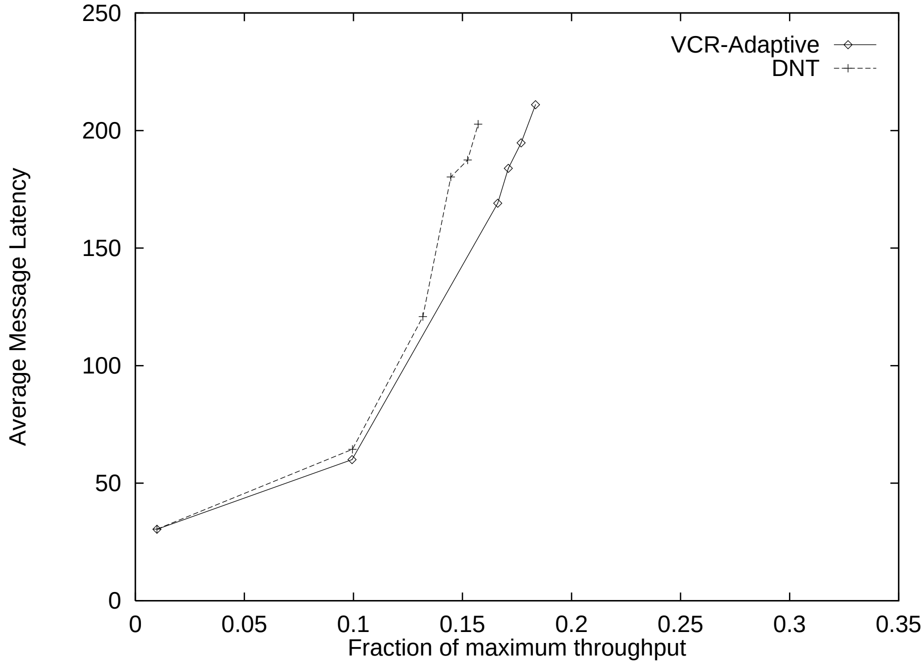


Figure A.1: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single faulty X link at the center of the network

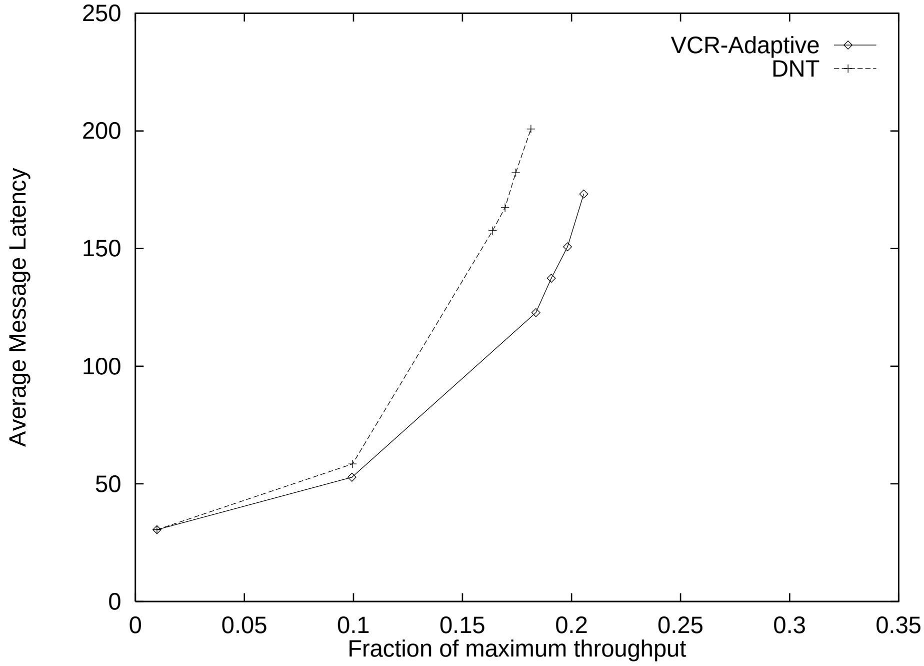


Figure A.2: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single faulty X link at the side of the network

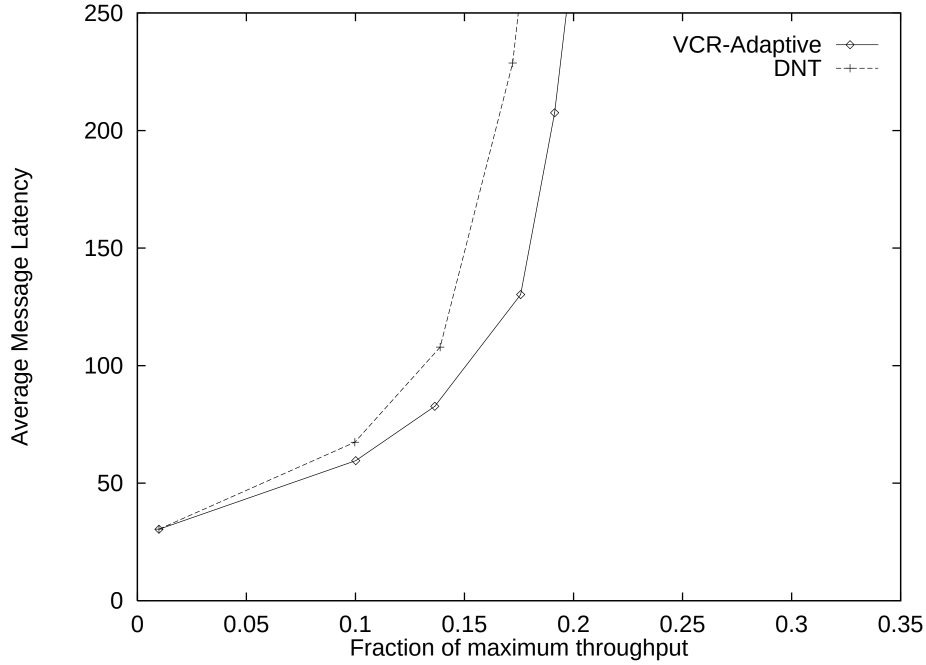


Figure A.3: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty X link at the center of the network

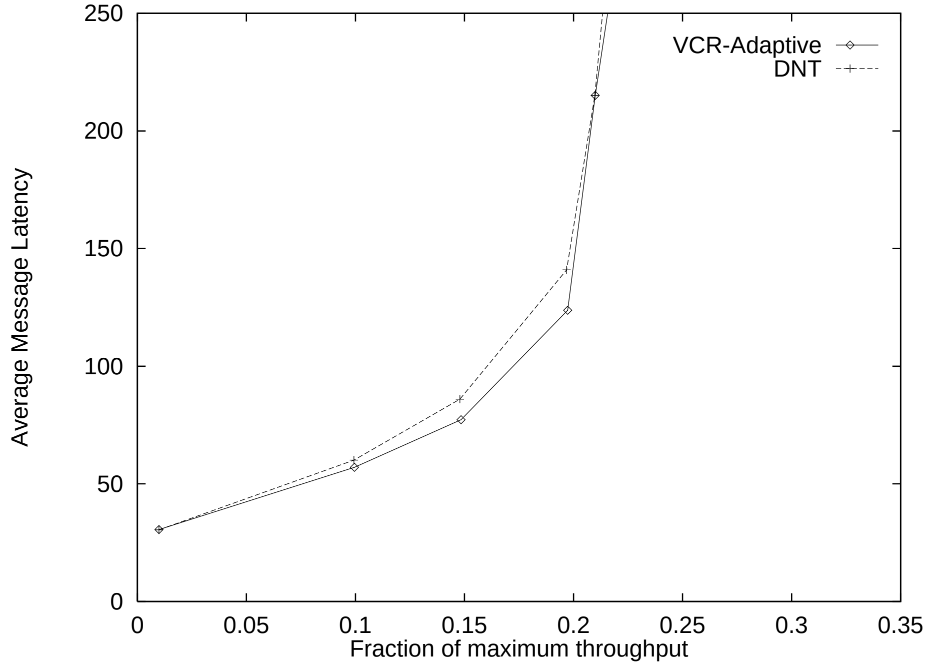


Figure A.4: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty X link at the side of the network

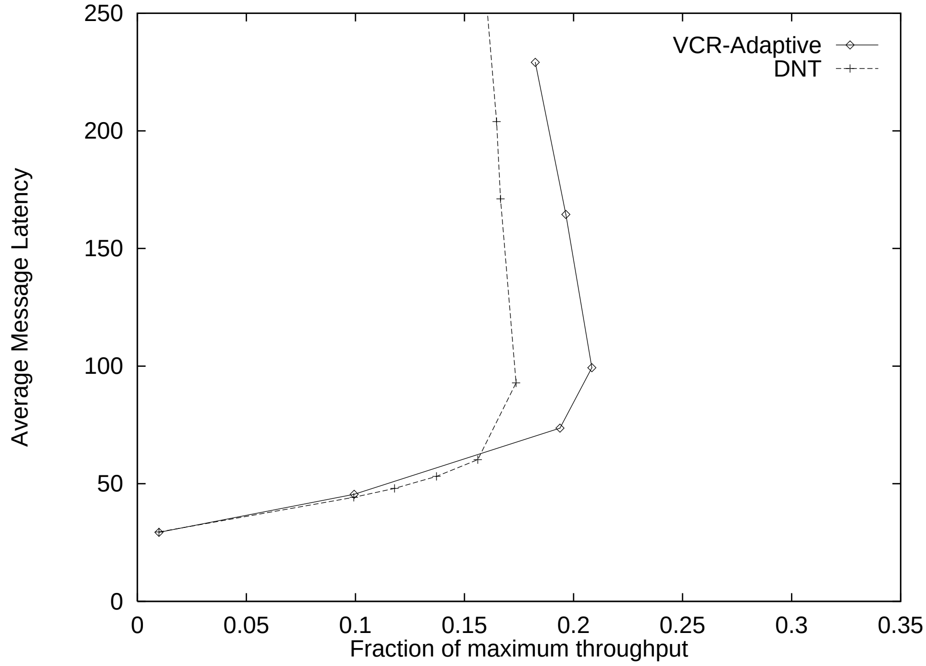


Figure A.5: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single faulty X link at the center of the network

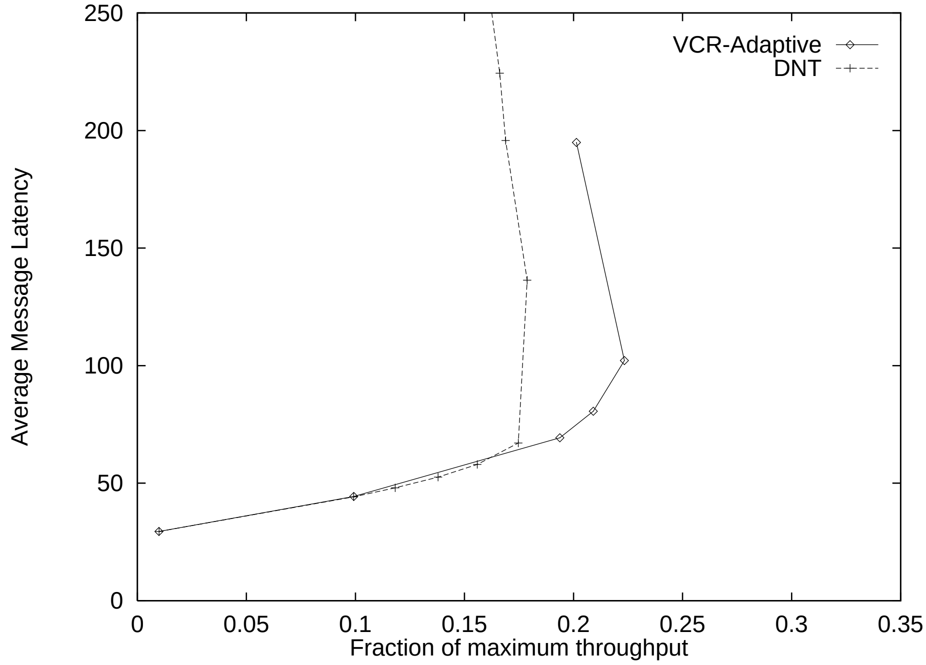


Figure A.6: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single faulty X link at the side of the network

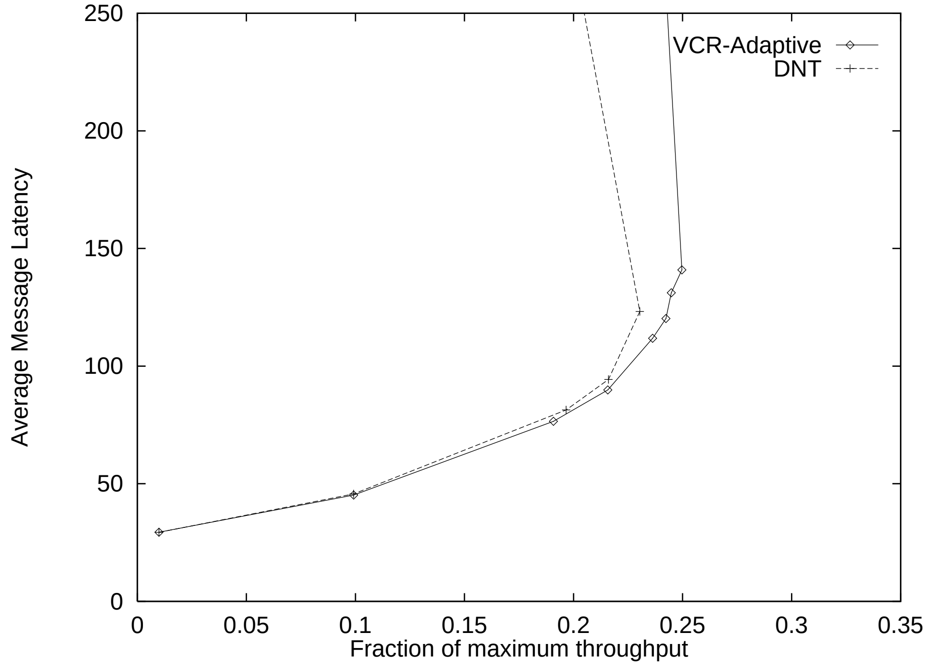


Figure A.7: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty X link at the center of the network

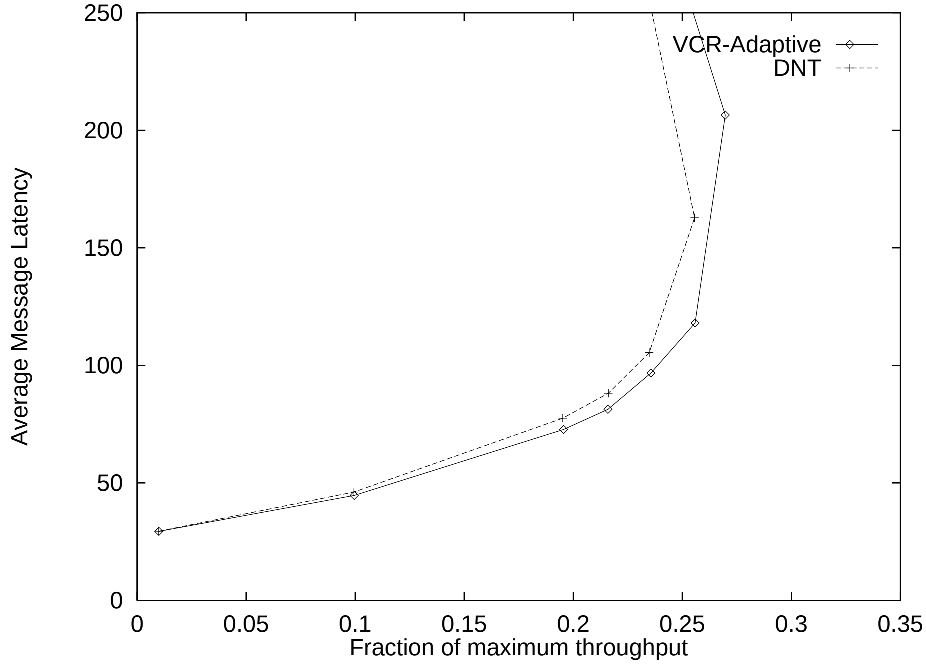


Figure A.8: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty X link at the side of the network

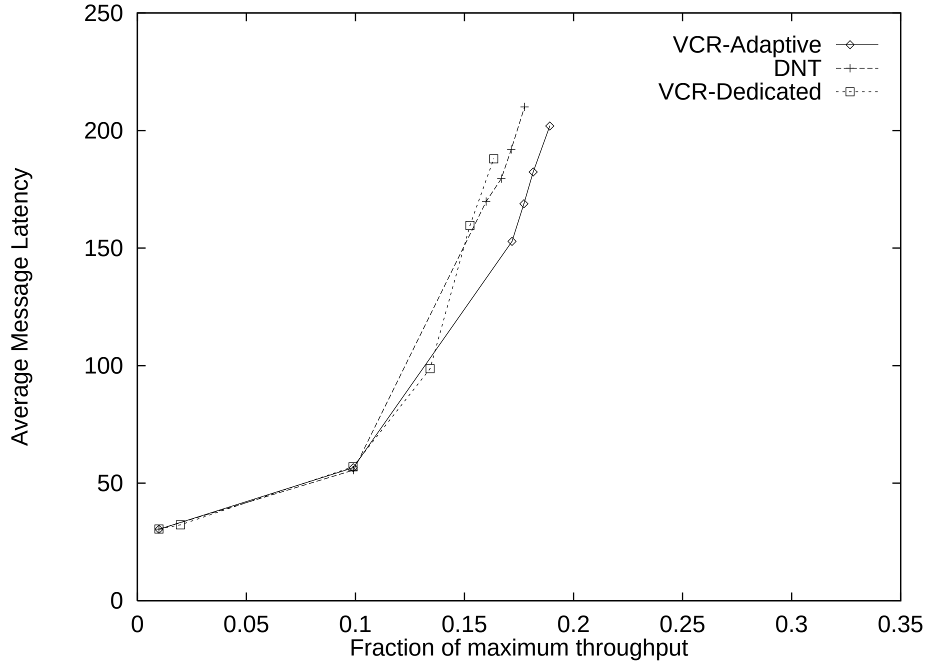


Figure A.9: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single faulty Y link at the center of the network

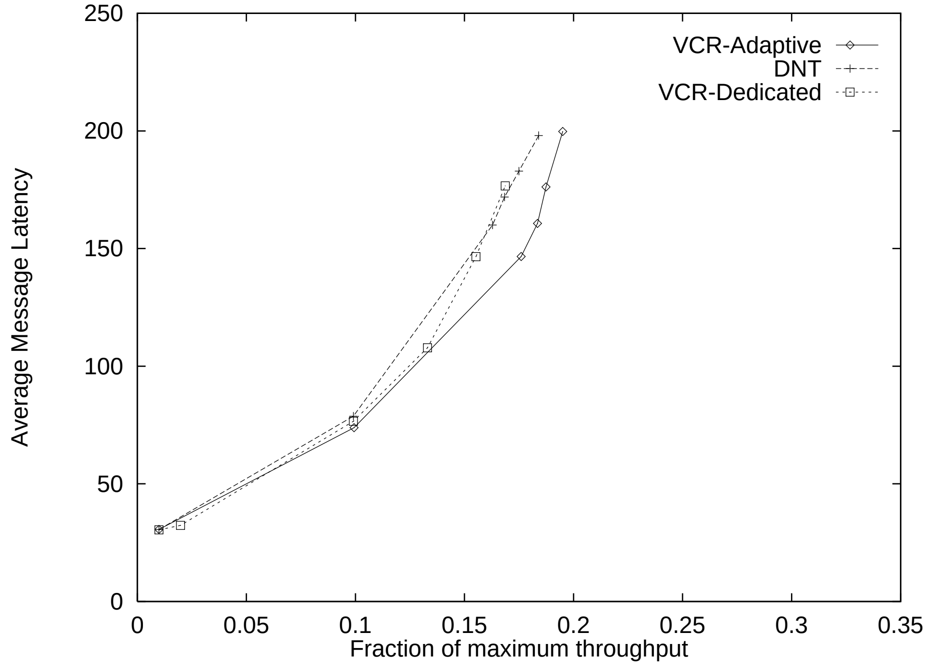


Figure A.10: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single faulty Y link at the side of the network

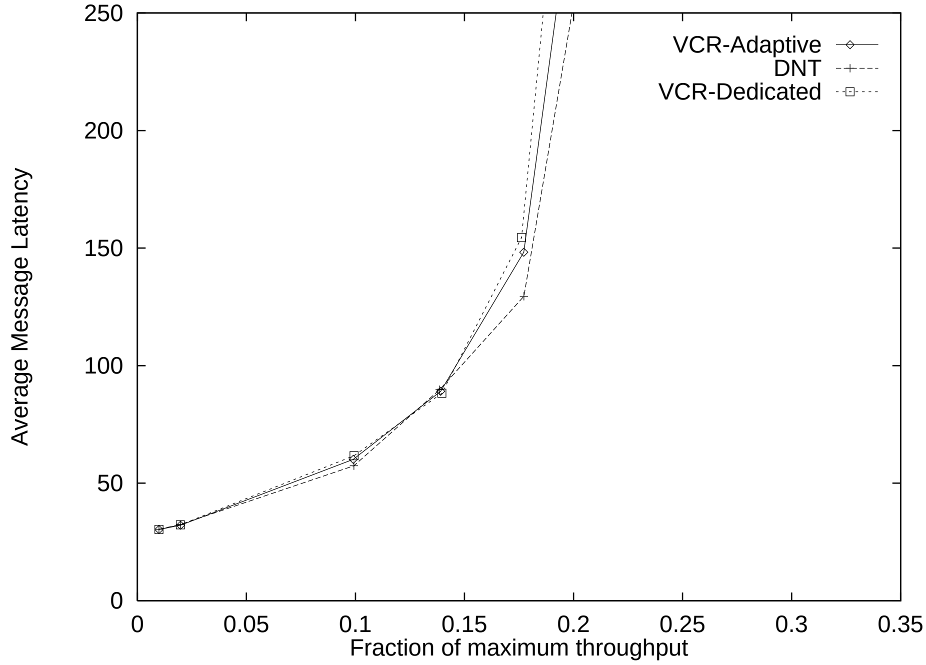


Figure A.11: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty Y link at the center of the network

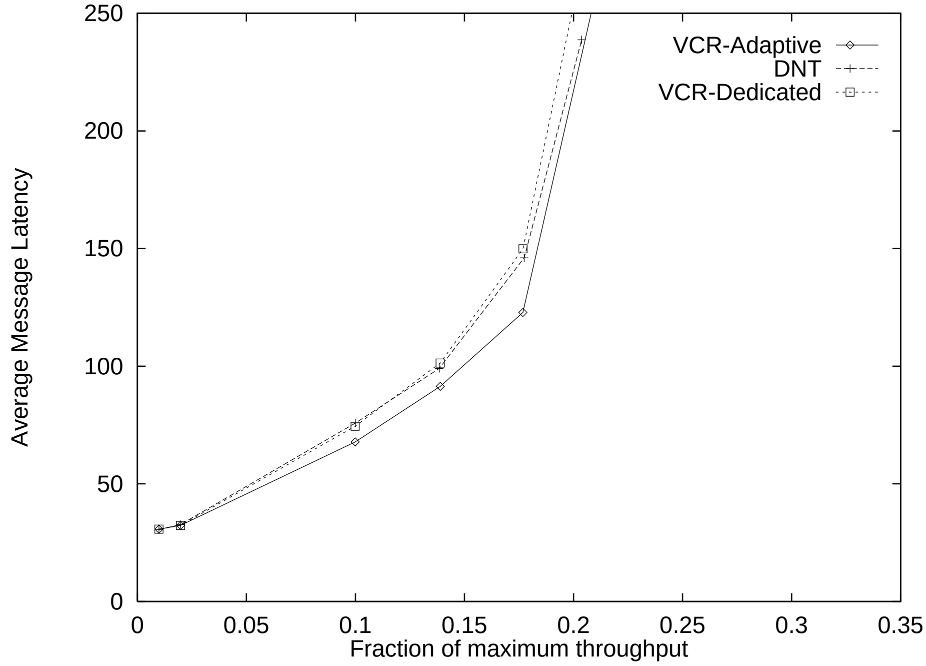


Figure A.12: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single faulty Y link at the side of the network

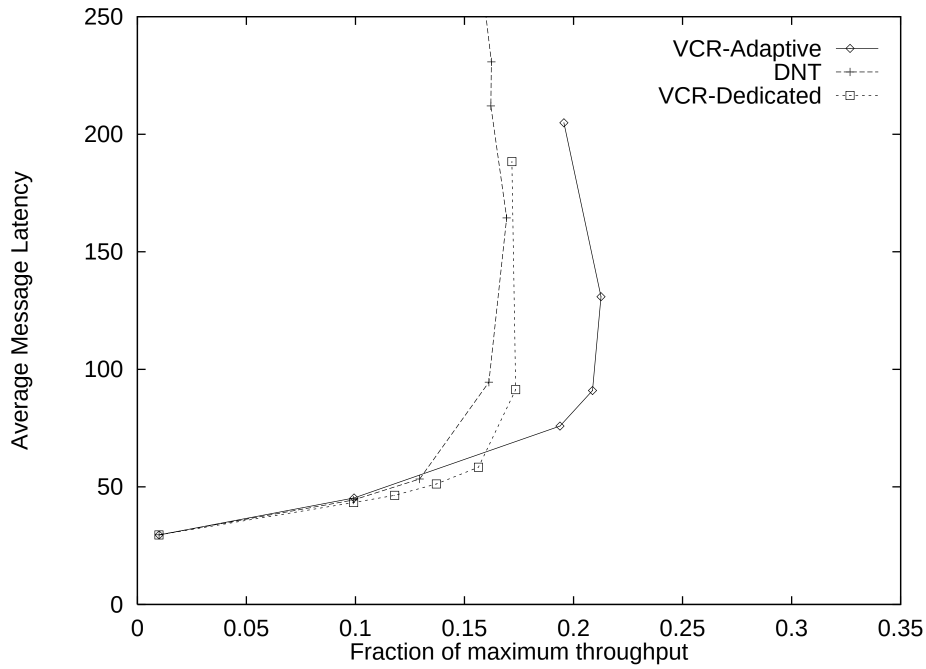


Figure A.13: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single faulty Y link at the center of the network

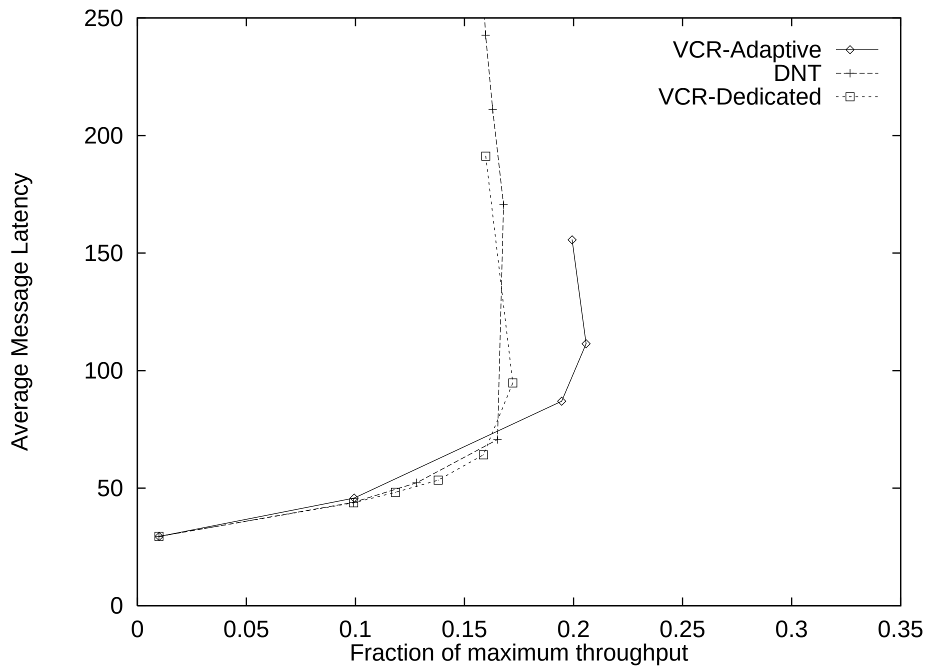


Figure A.14: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single faulty Y link at the side of the network

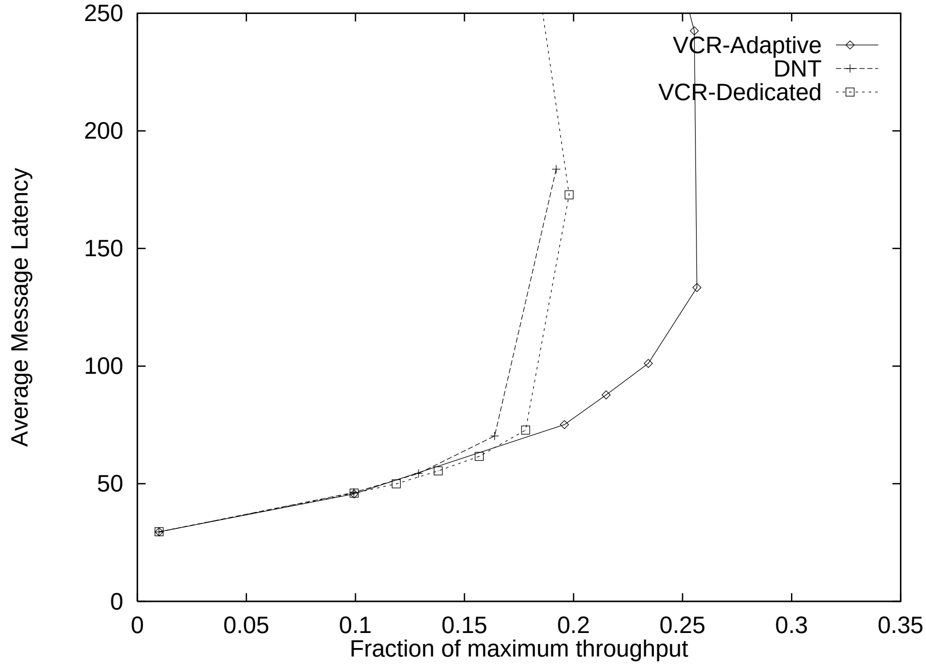


Figure A.15: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty Y link at the center of the network

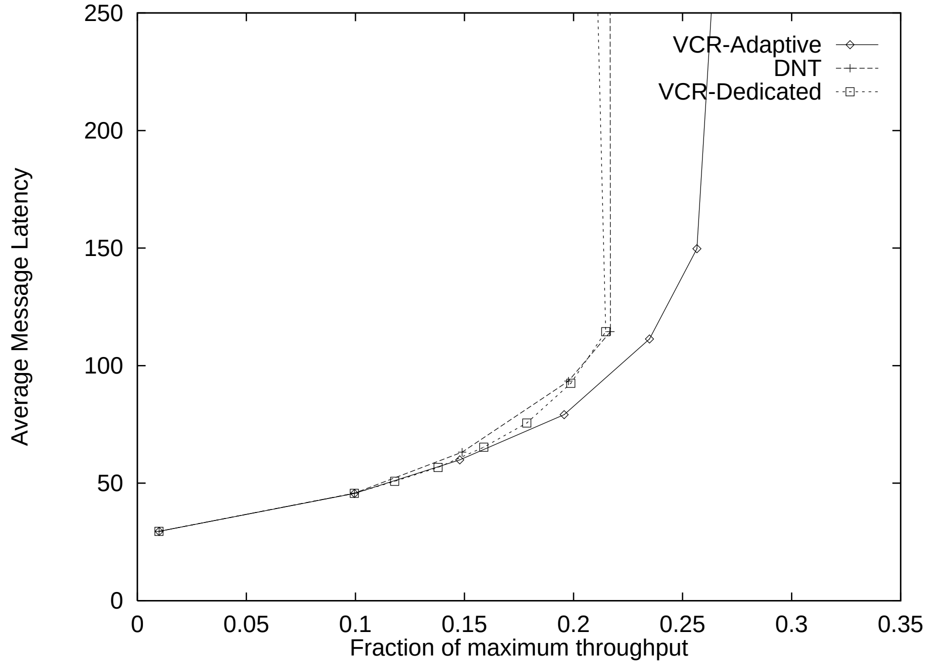


Figure A.16: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single faulty Y link at the side of the network

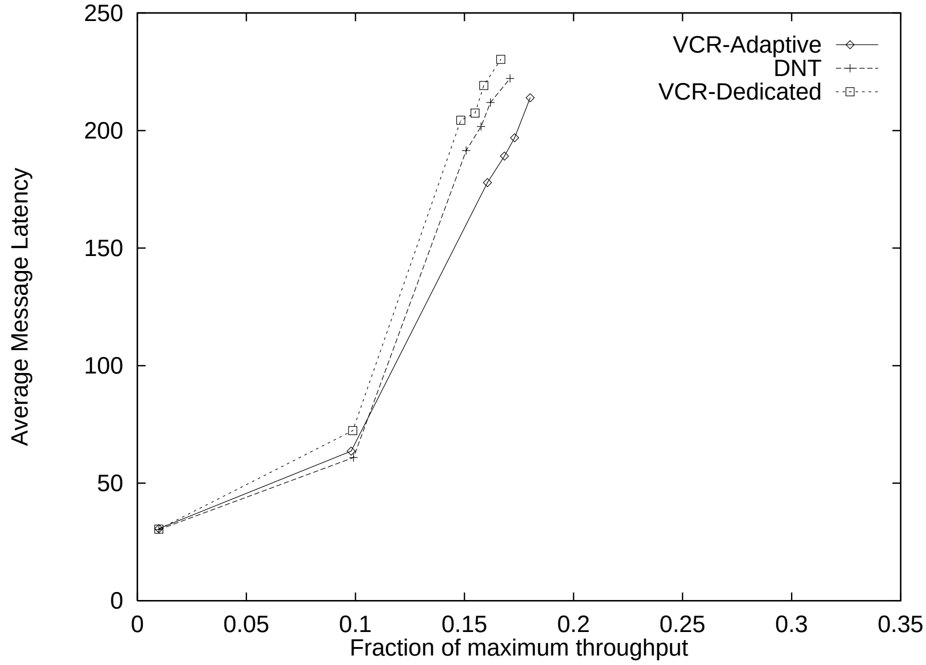


Figure A.17: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single node fault at the center of the network

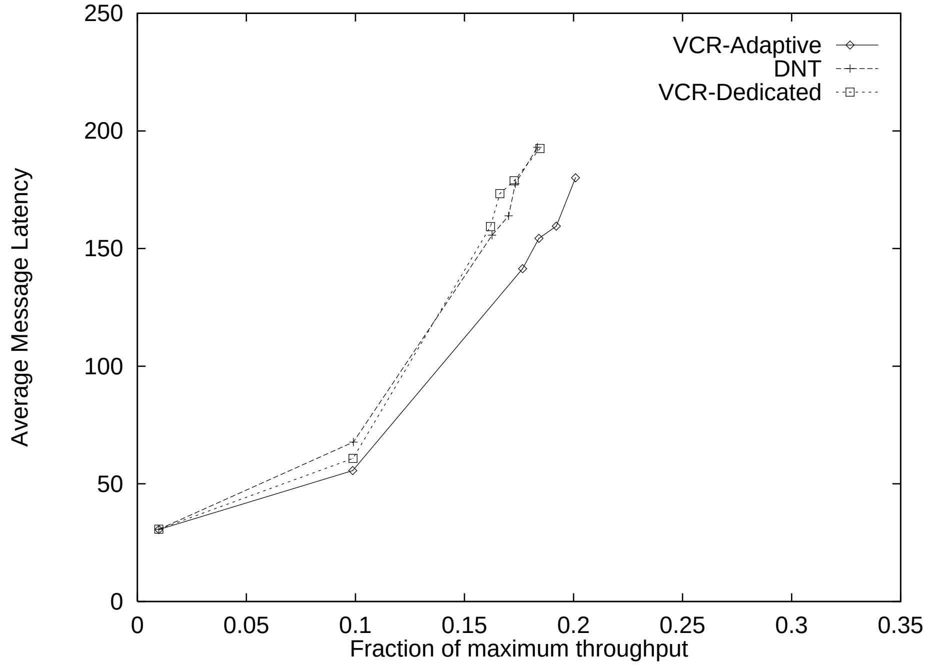


Figure A.18: Comparison of routing algorithms for matrix transpose traffic with 3 virtual channels per physical channel and a single node fault at the side of the network

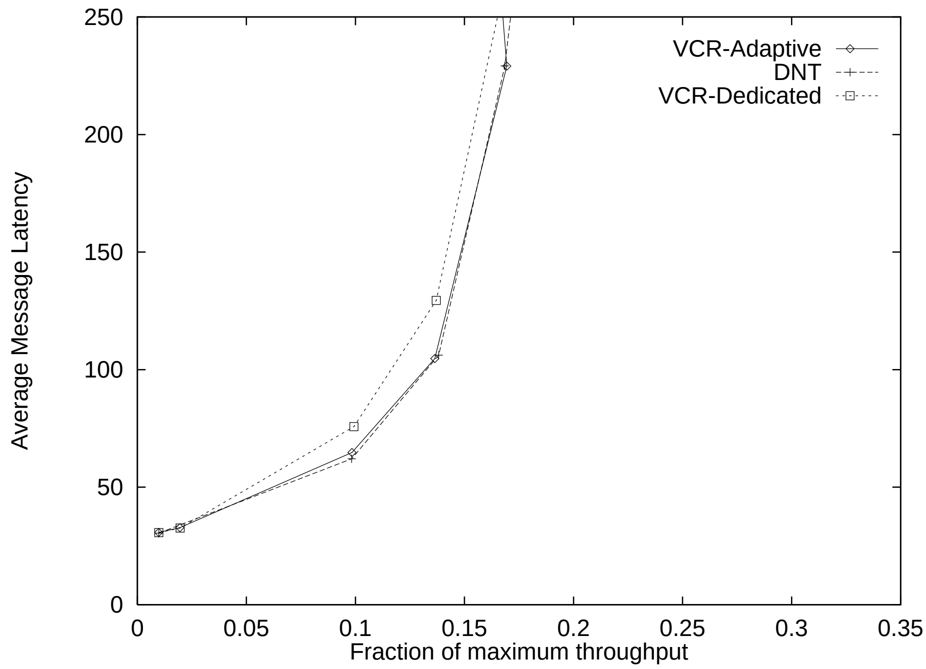


Figure A.19: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single node fault at the center of the network

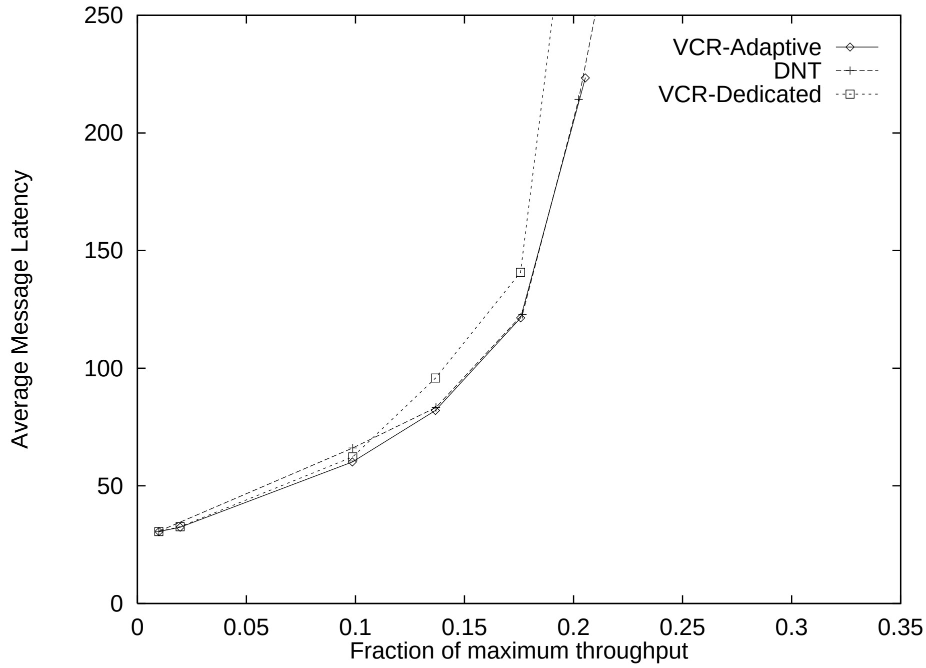


Figure A.20: Comparison of routing algorithms for matrix transpose traffic with 6 virtual channels per physical channel and a single node fault at the side of the network

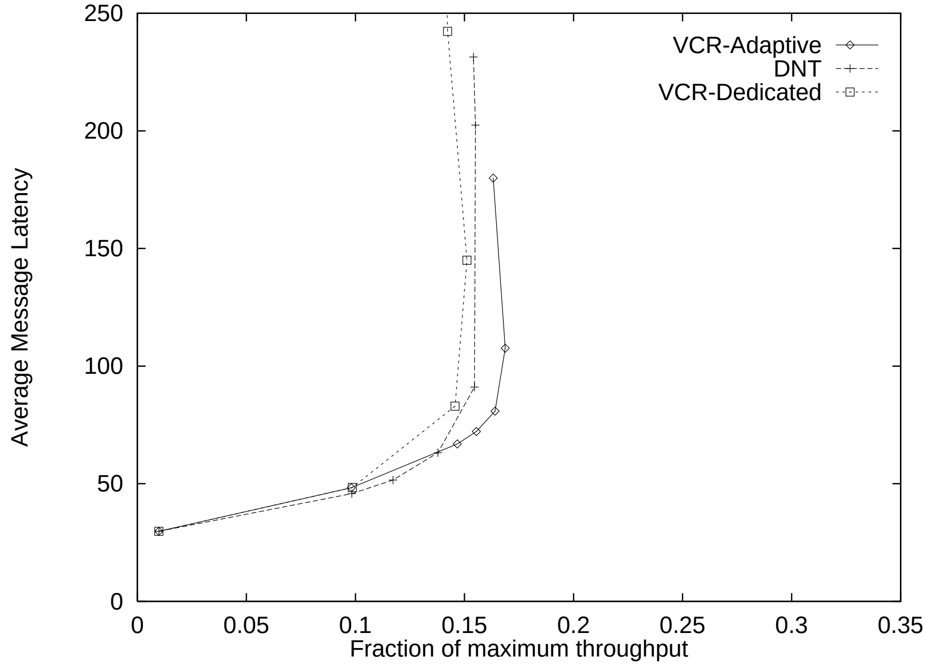


Figure A.21: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single node fault at the center of the network

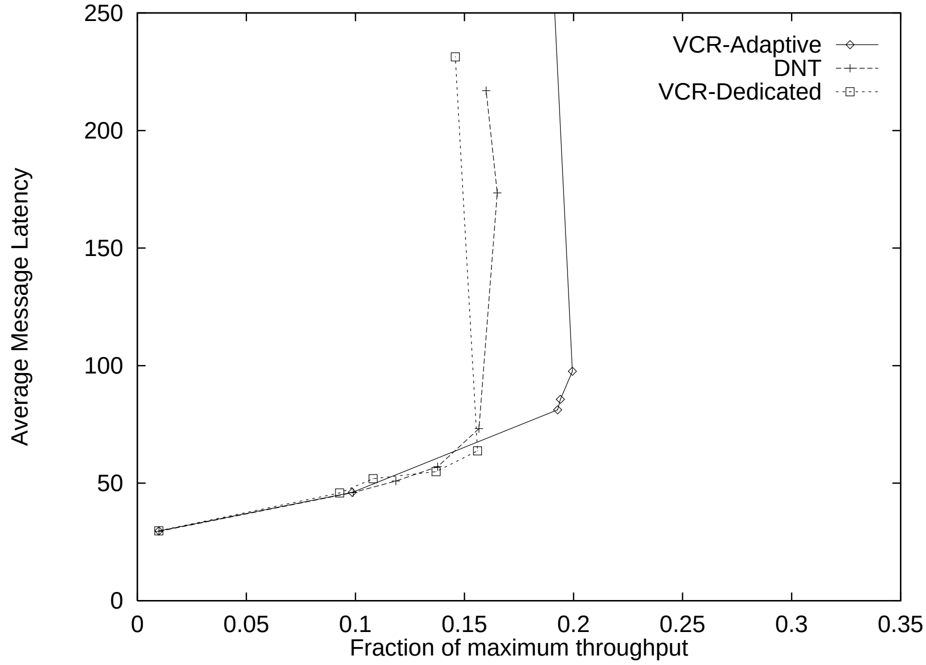


Figure A.22: Comparison of routing algorithms for uniform traffic with 3 virtual channels per physical channel and a single node fault at the side of the network

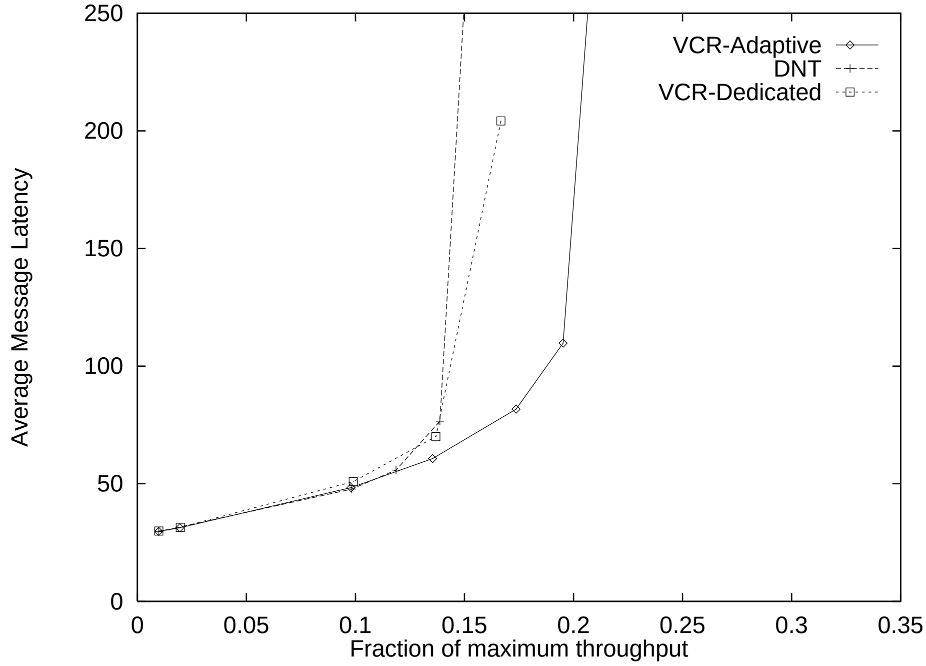


Figure A.23: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single node fault at the center of the network

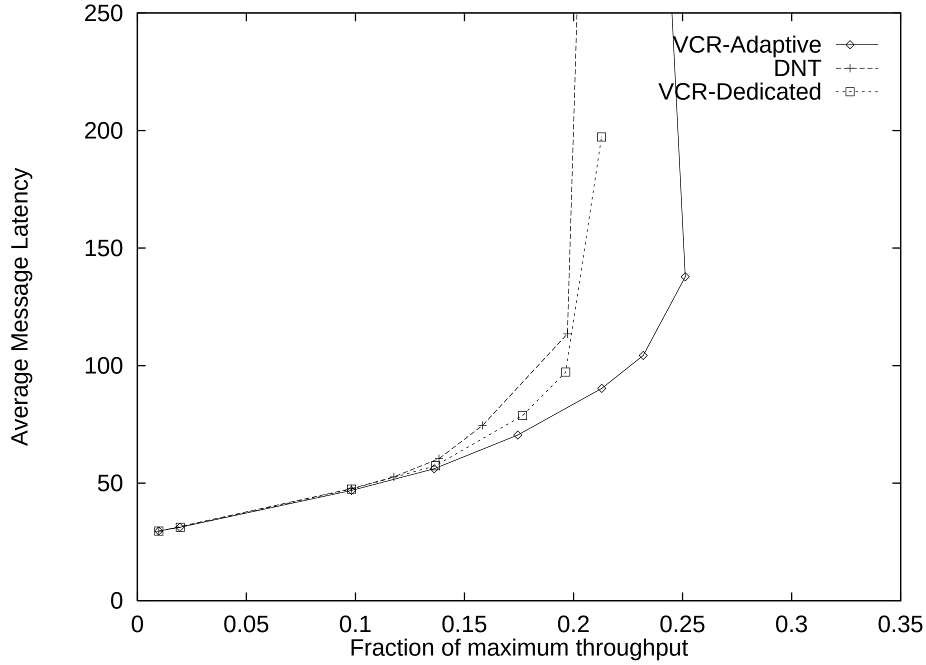


Figure A.24: Comparison of routing algorithms for uniform traffic with 6 virtual channels per physical channel and a single node fault at the side of the network

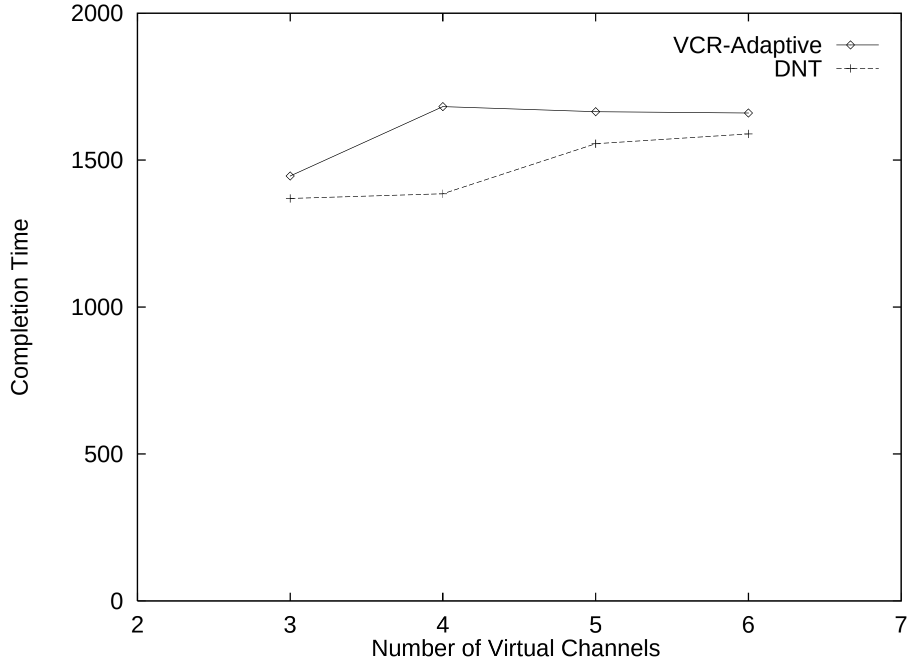


Figure A.25: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the center of the network

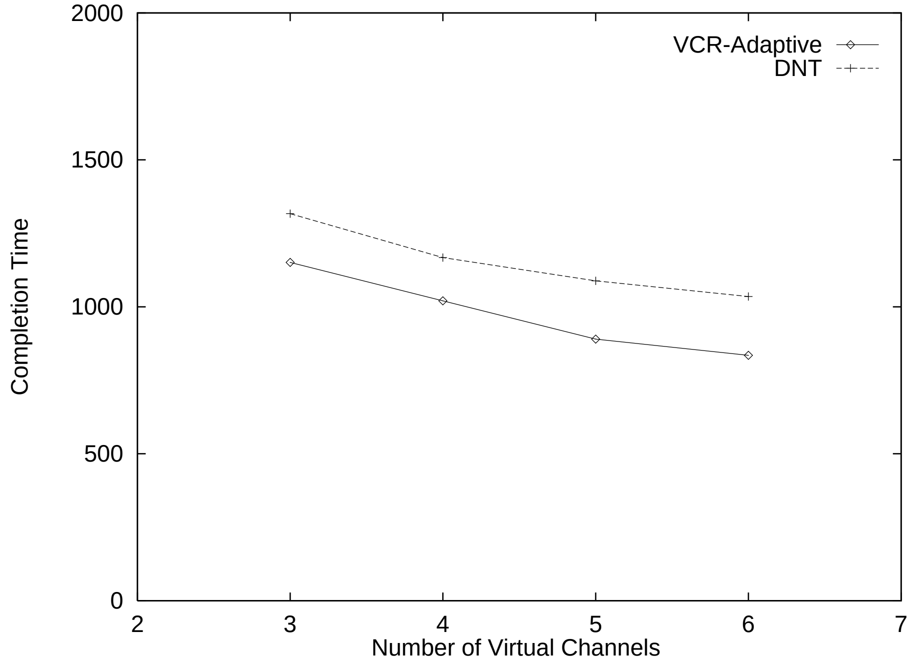


Figure A.26: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the side of the network

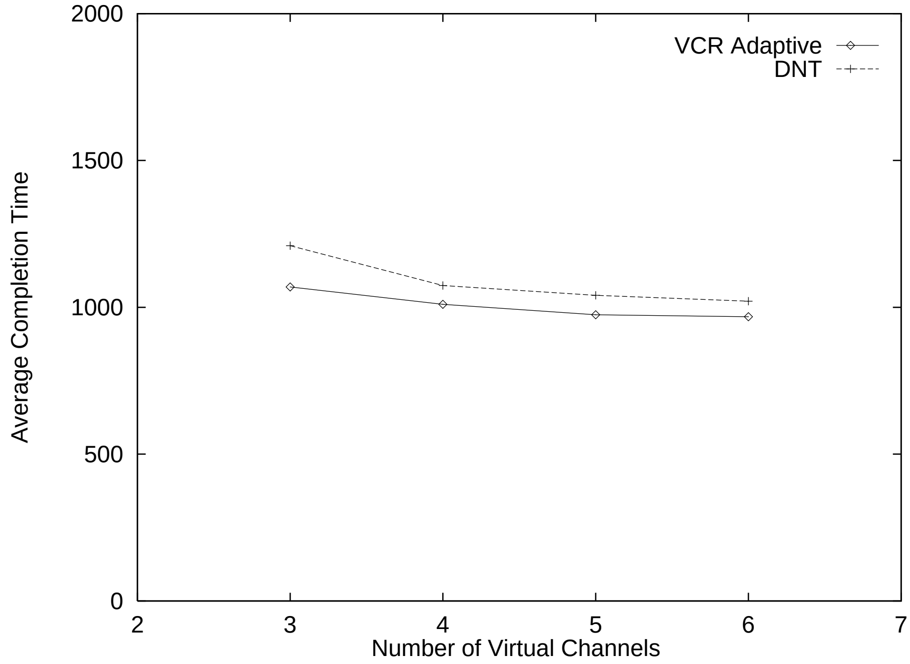


Figure A.27: Completion times for routing algorithms for general traffic patterns and a single faulty X link at the center of the network

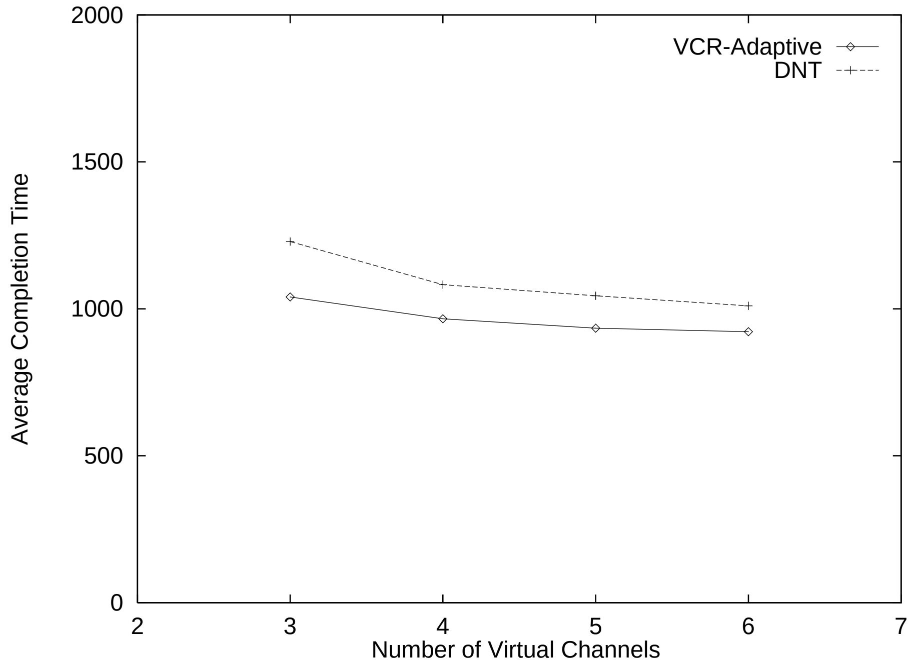


Figure A.28: Completion times for routing algorithms for general traffic patterns and a single faulty X link at the side of the network

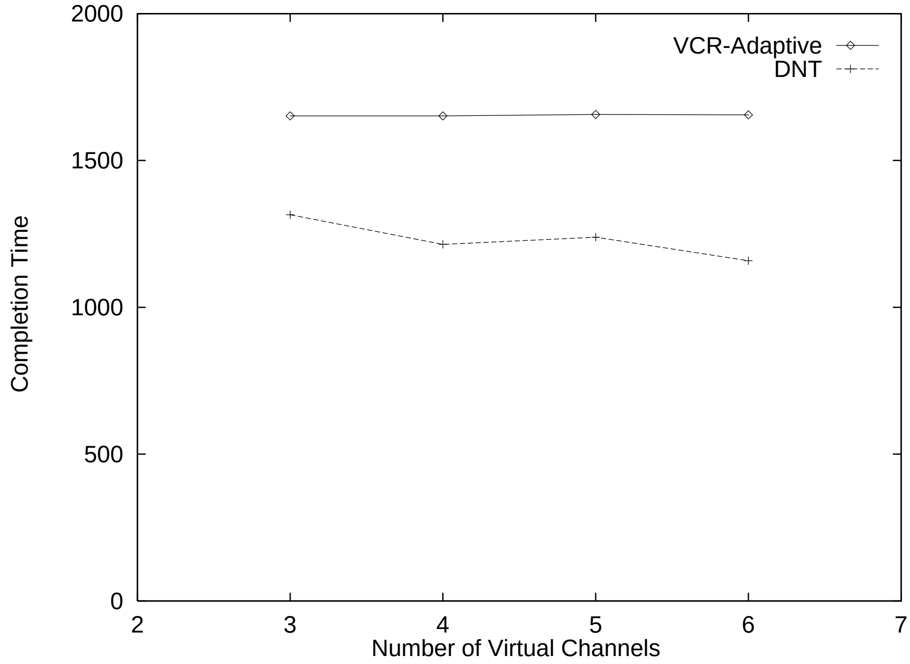


Figure A.29: Completion times for routing algorithms for matrix transpose traffic and a single faulty Y link at the center of the network

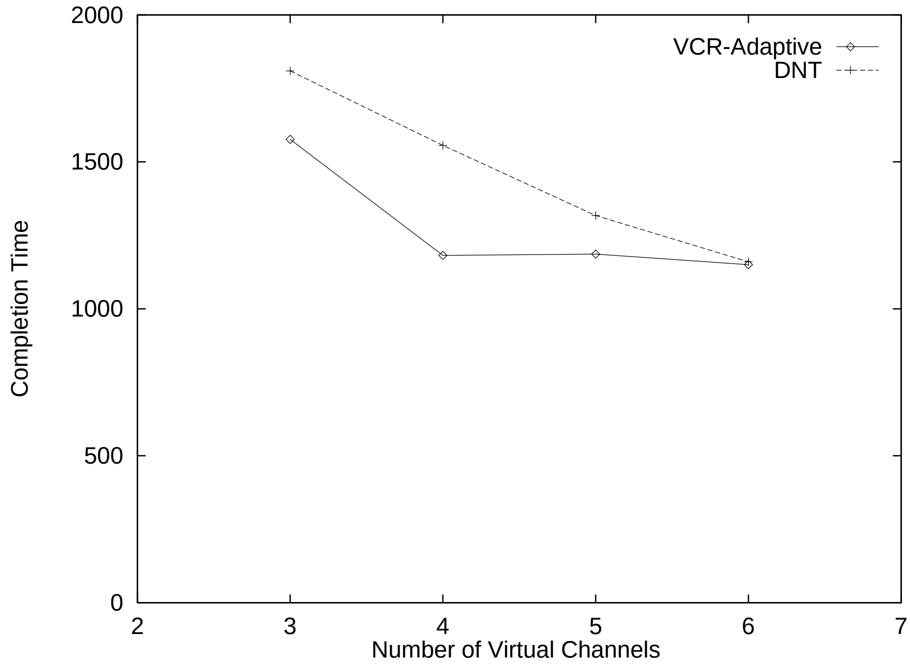


Figure A.30: Completion times for routing algorithms for matrix transpose traffic and a single faulty Y link at the side of the network

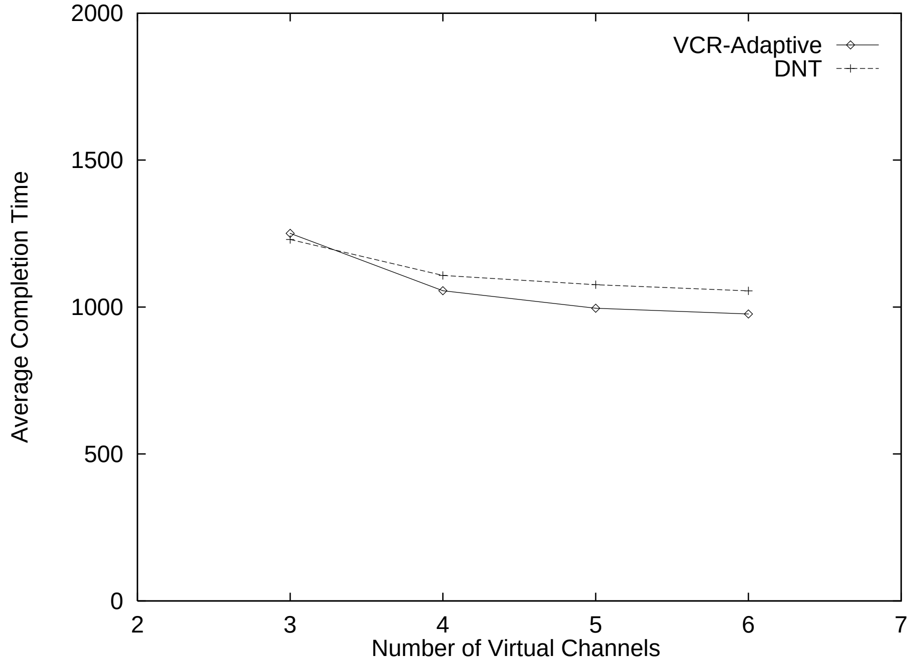


Figure A.31: Completion times for routing algorithms for general traffic patterns and a single faulty Y link at the center of the network

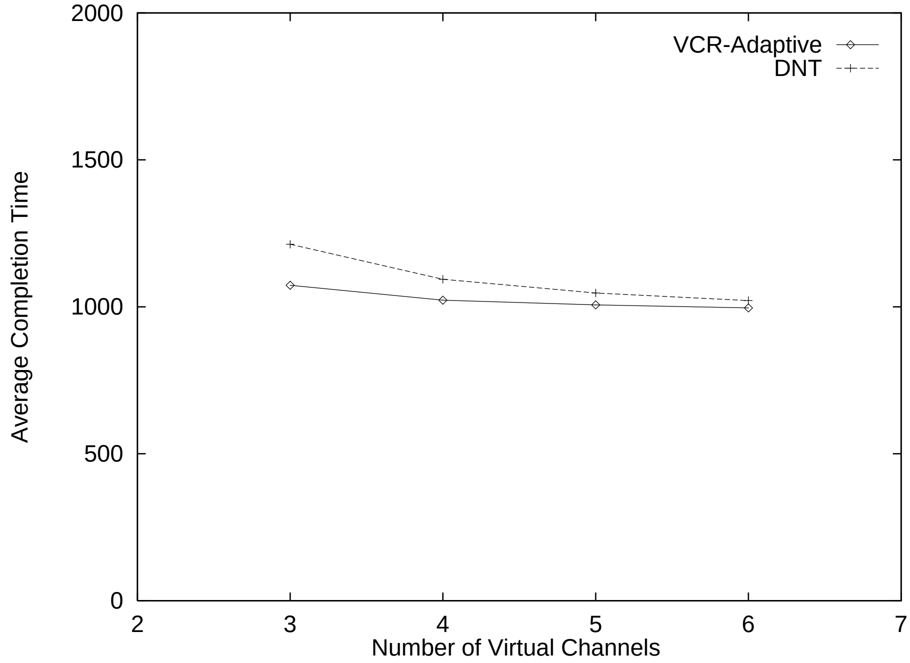


Figure A.32: Completion times for routing algorithms for general traffic patterns and a single faulty Y link at the side of the network

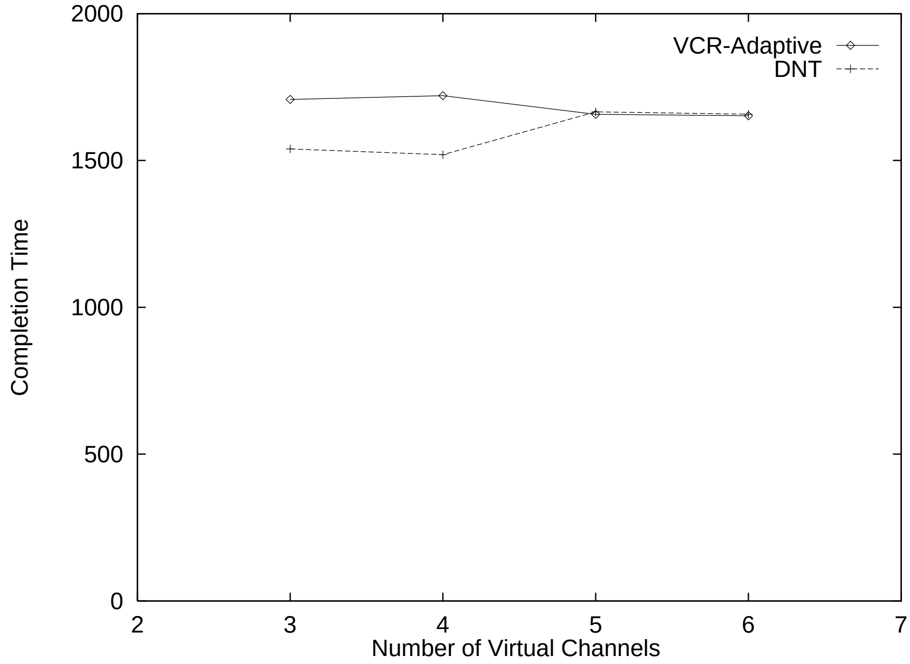


Figure A.33: Completion times for routing algorithms for matrix transpose traffic and a single node fault at the center of the network

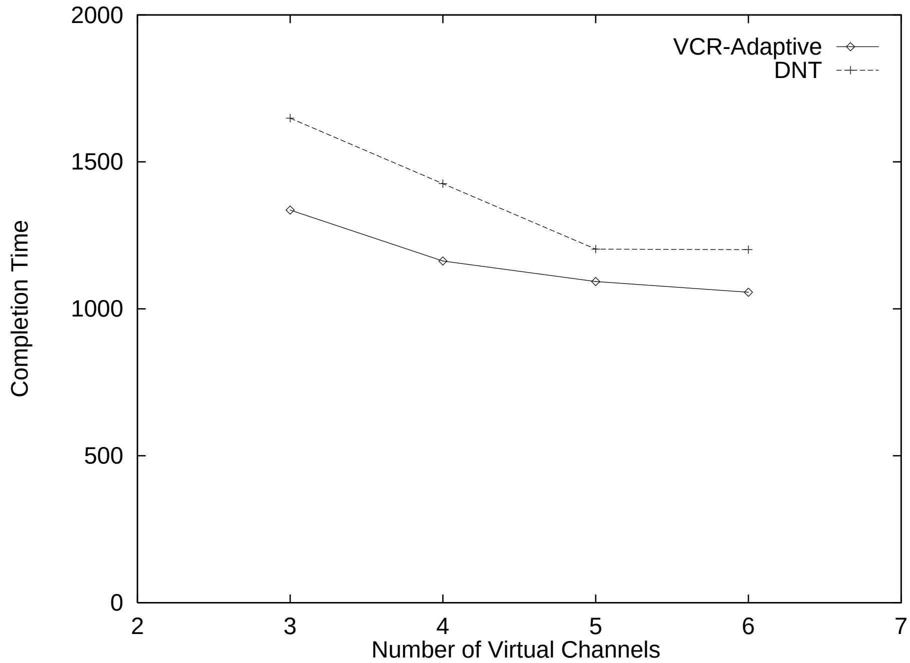


Figure A.34: Completion times for routing algorithms for matrix transpose traffic and a single node fault at the side of the network

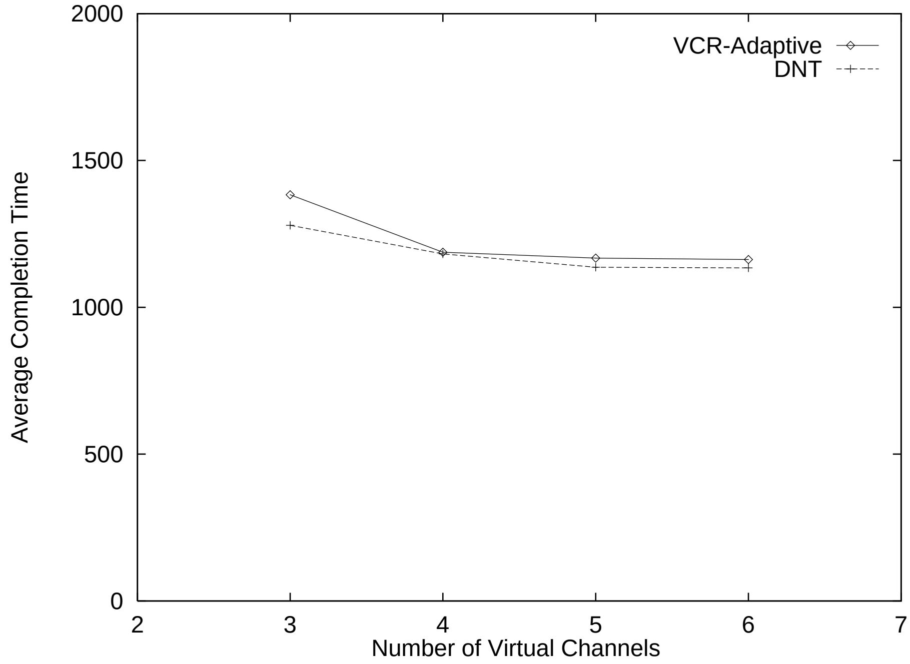


Figure A.35: Completion times for routing algorithms for general traffic patterns and a single node fault at the center of the network

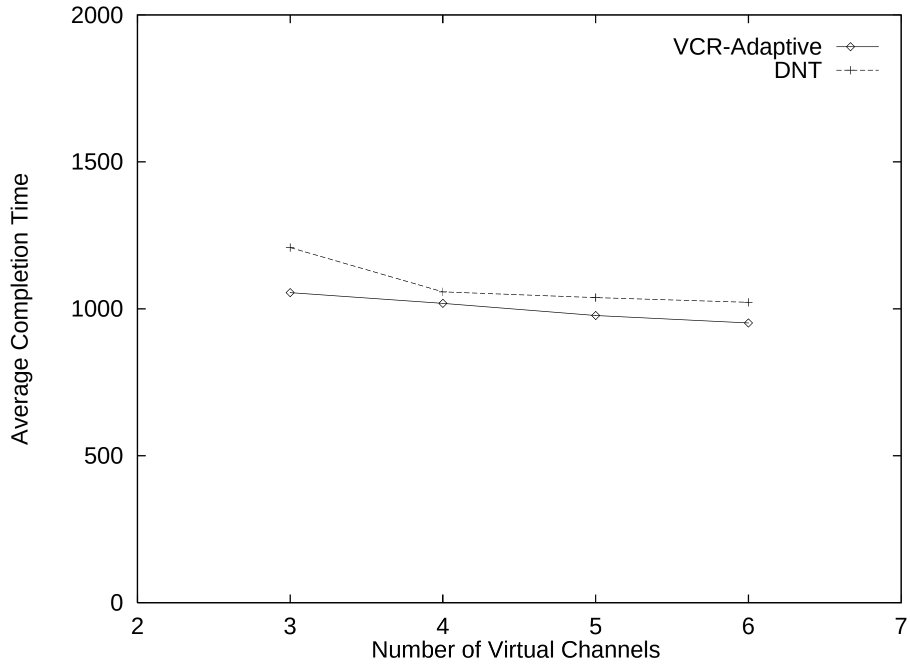


Figure A.36: Completion times for routing algorithms for general traffic patterns and a single node fault at the side of the network

Appendix B

Effect of Faults on Performance

The results of the study which dealt with the effect of faults on performance (when the VCR approach is employed) are presented in this appendix. This study deals with five different fault models, and considers both average and maximum performance measures.

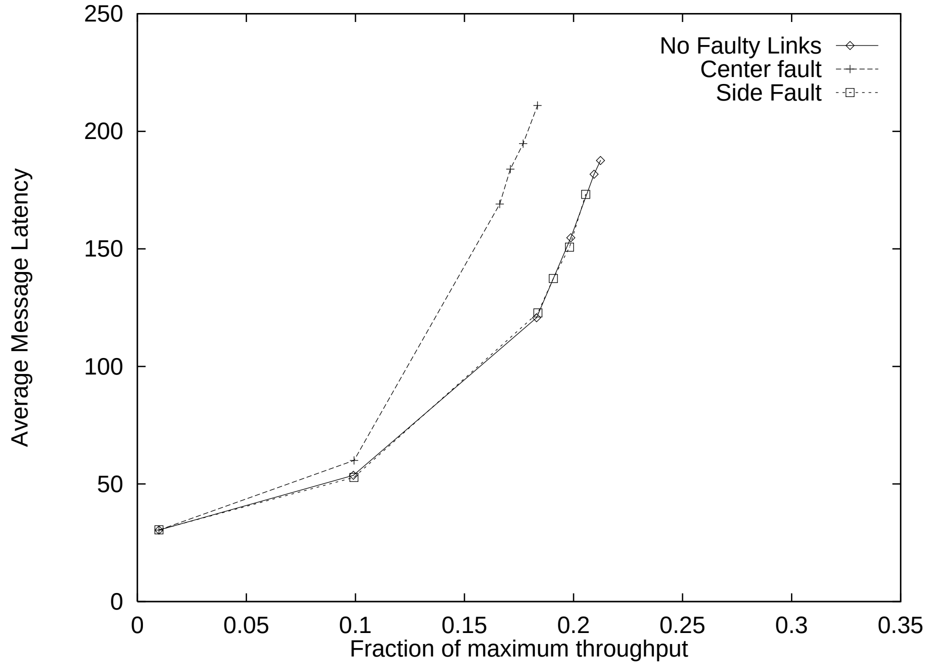


Figure B.1: Effect of fault location on performance for a single faulty X link in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

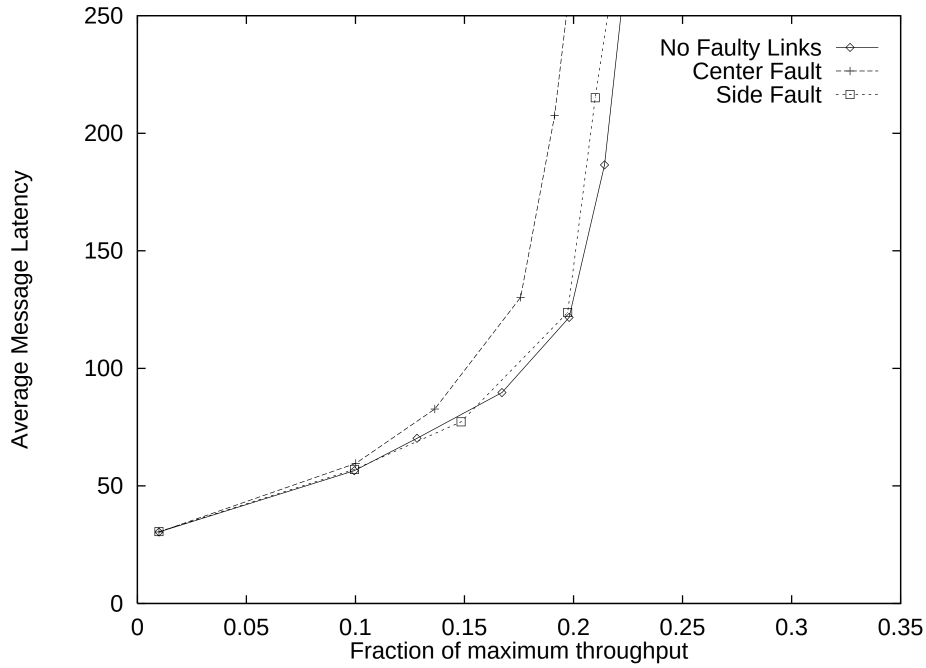


Figure B.2: Effect of fault location on performance for a single faulty X link in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

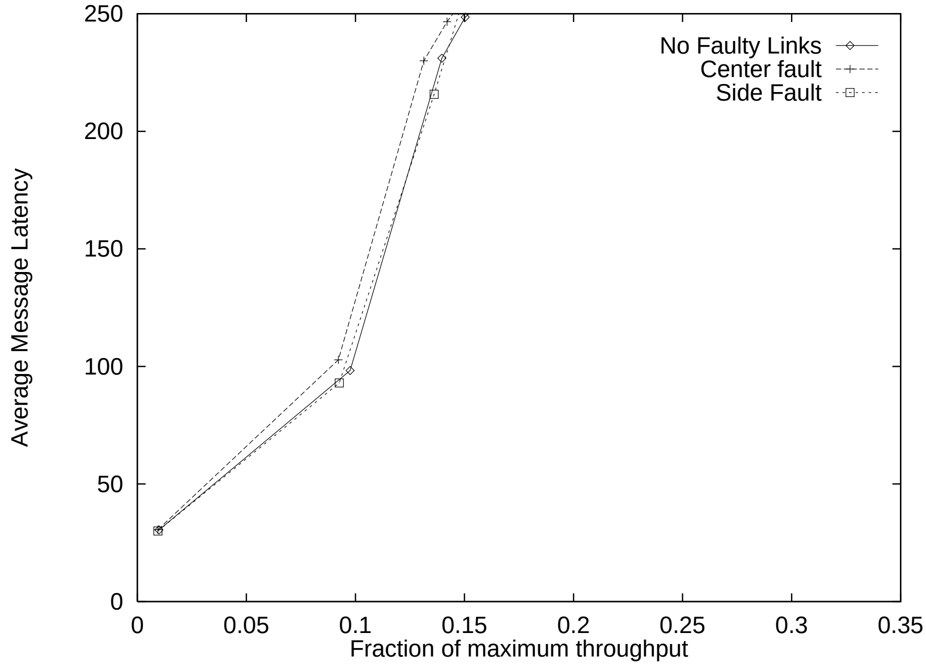


Figure B.3: Effect of fault location on performance for a single faulty X link in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

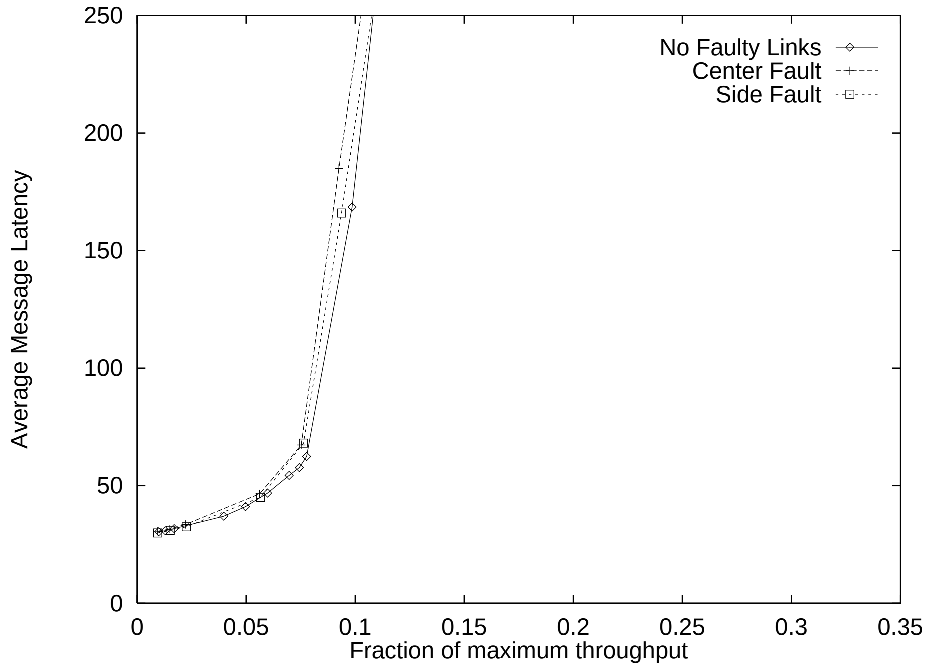


Figure B.4: Effect of fault location on performance for a single faulty X link in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

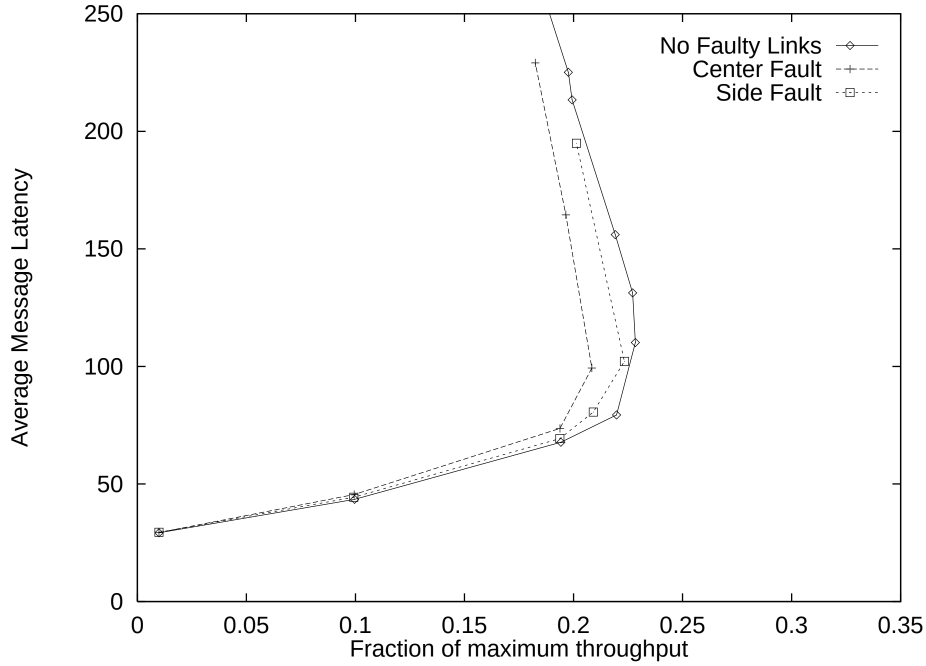


Figure B.5: Effect of fault location on performance for a single faulty X link in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

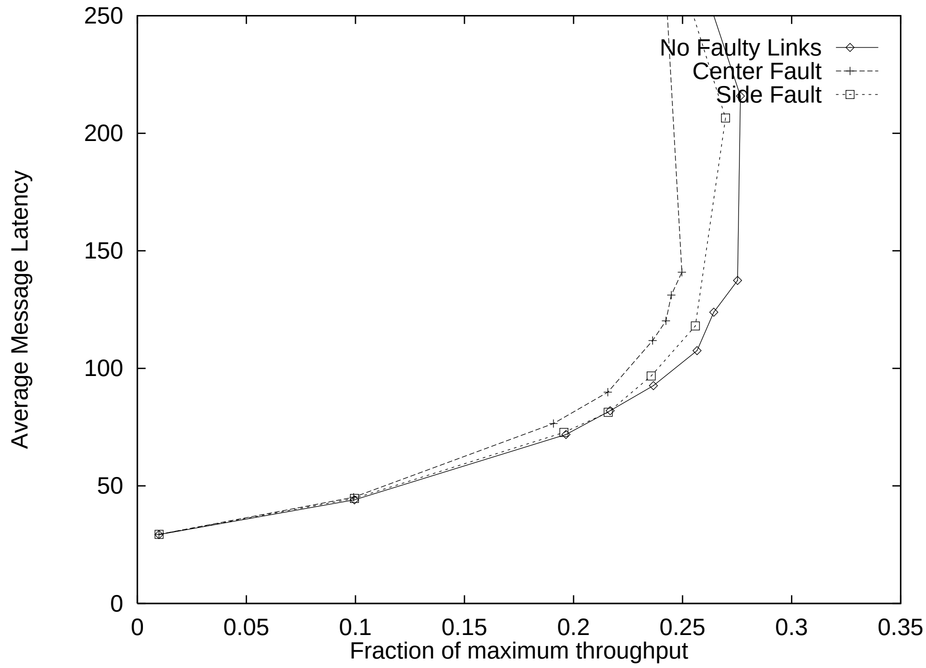


Figure B.6: Effect of fault location on performance for a single faulty X link in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

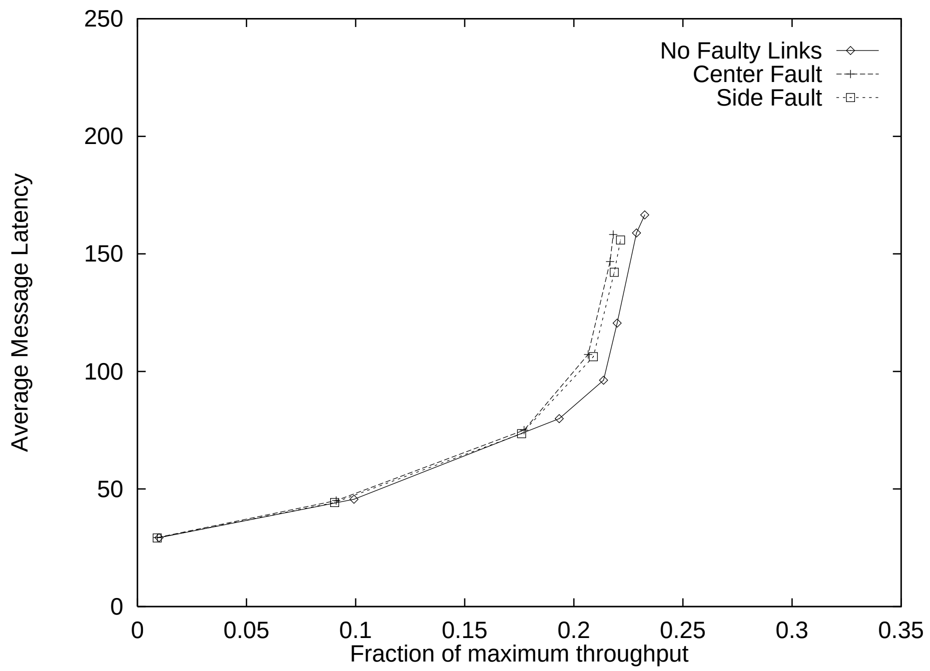


Figure B.7: Effect of fault location on performance for a single faulty X link in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

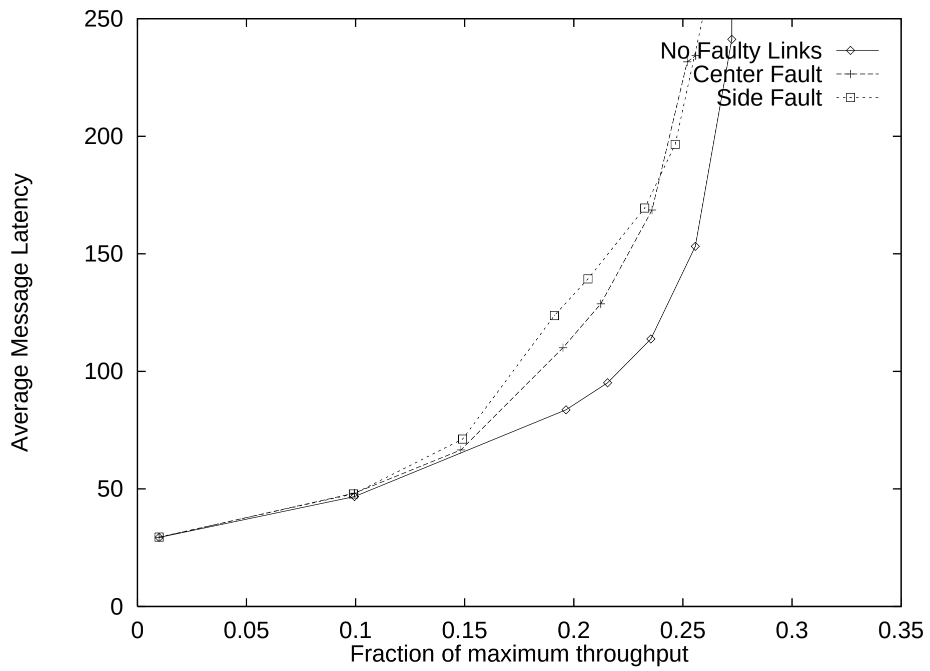


Figure B.8: Effect of fault location on performance for a single faulty X link in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

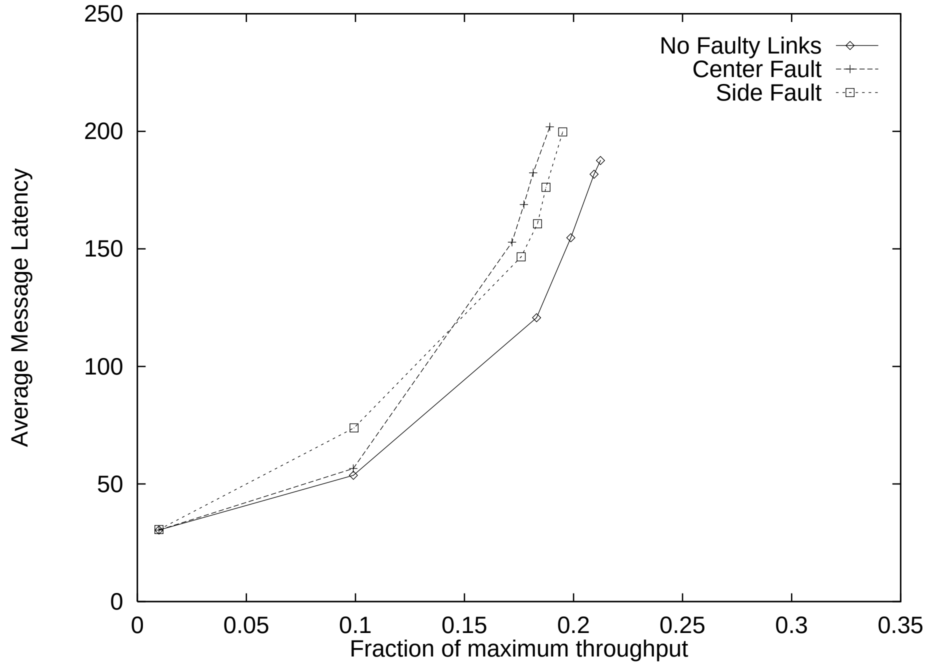


Figure B.9: Effect of fault location on performance for a single faulty Y link in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

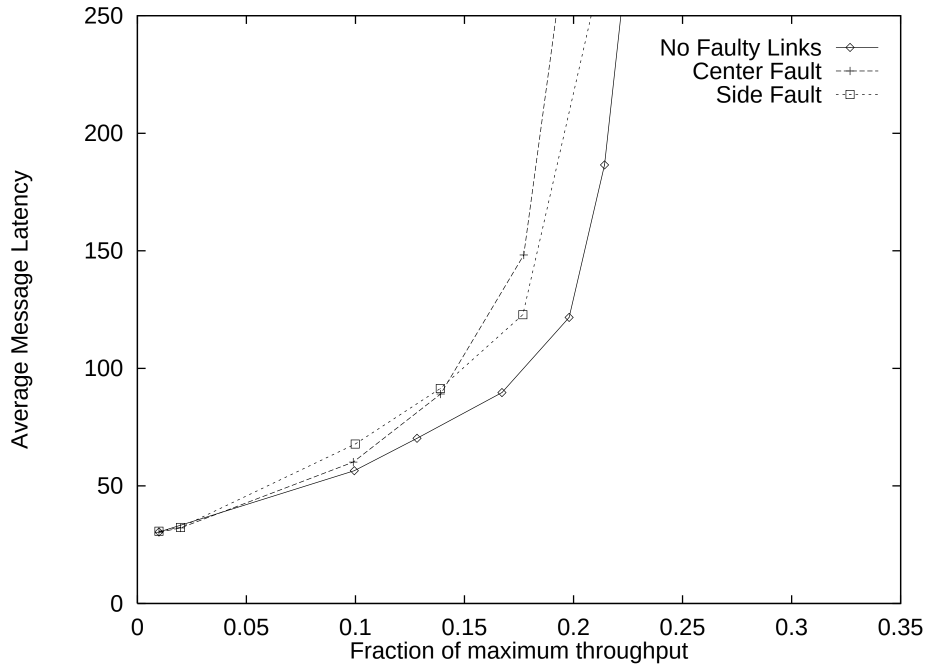


Figure B.10: Effect of fault location on performance for a single faulty Y link in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

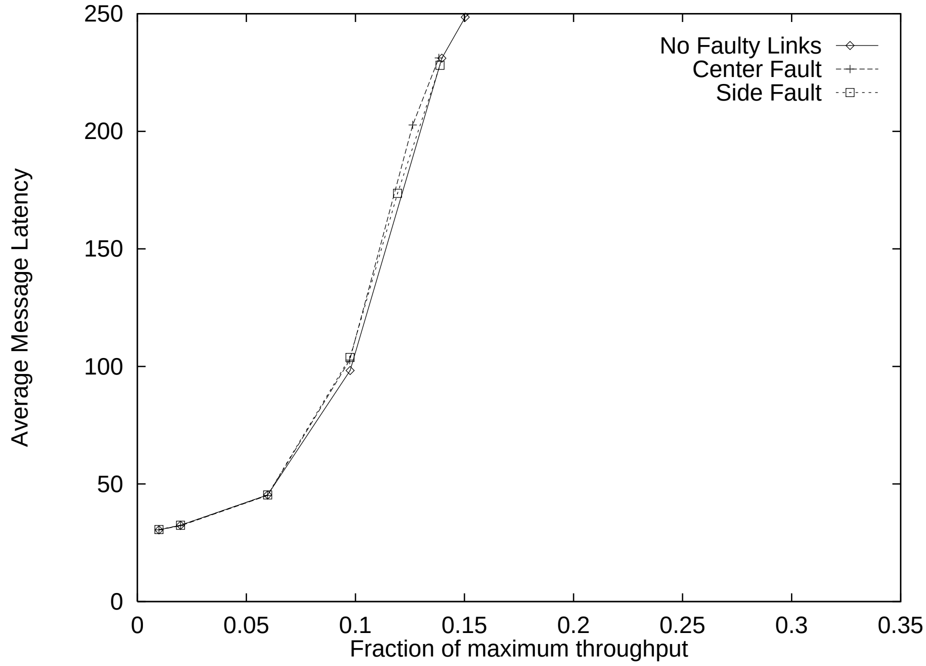


Figure B.11: Effect of fault location on performance for a single faulty Y link in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

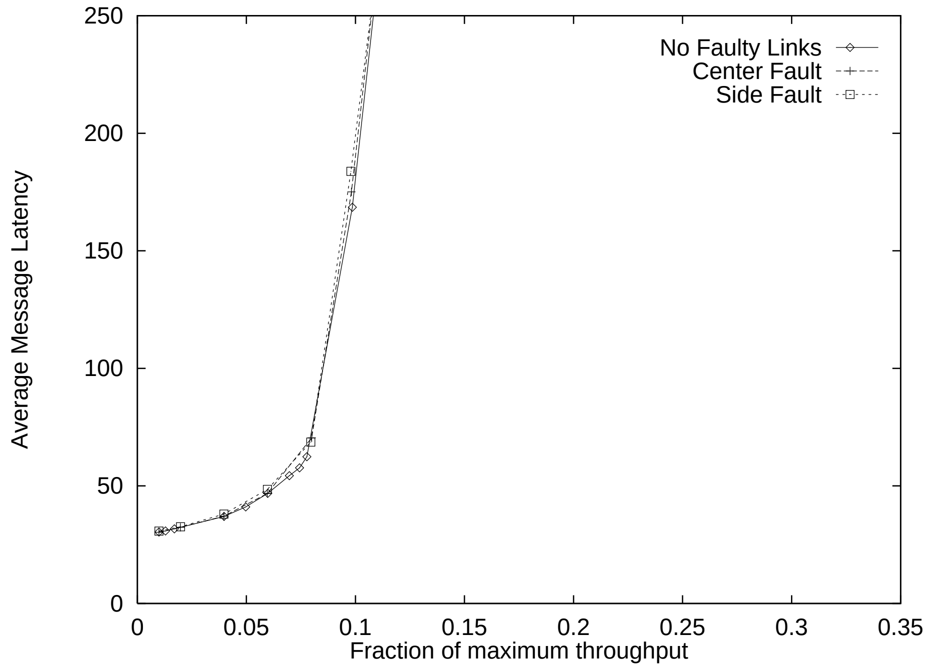


Figure B.12: Effect of fault location on performance for a single faulty Y link in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

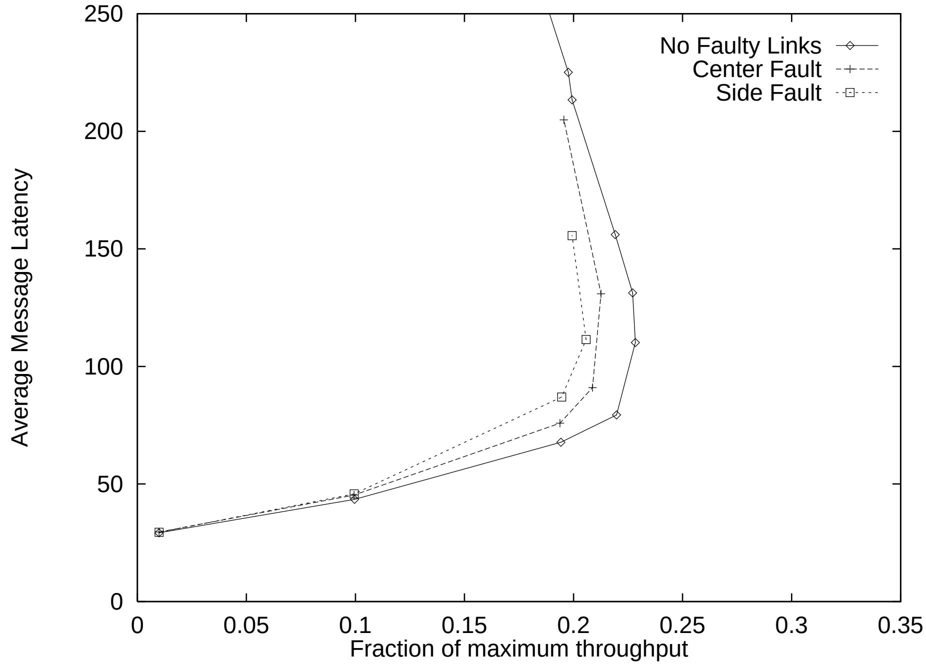


Figure B.13: Effect of fault location on performance for a single faulty Y link in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

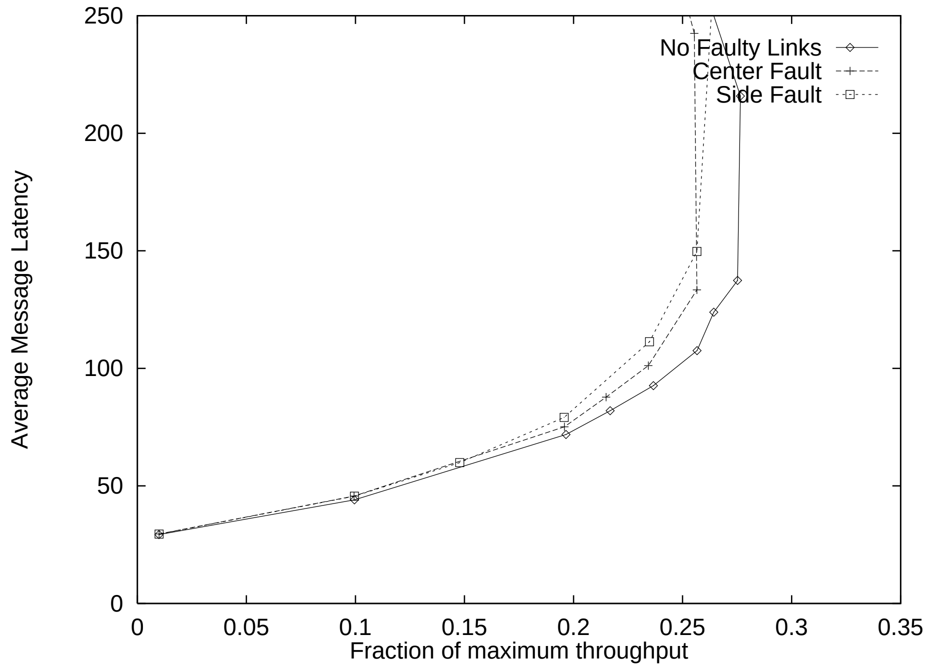


Figure B.14: Effect of fault location on performance for a single faulty Y link in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

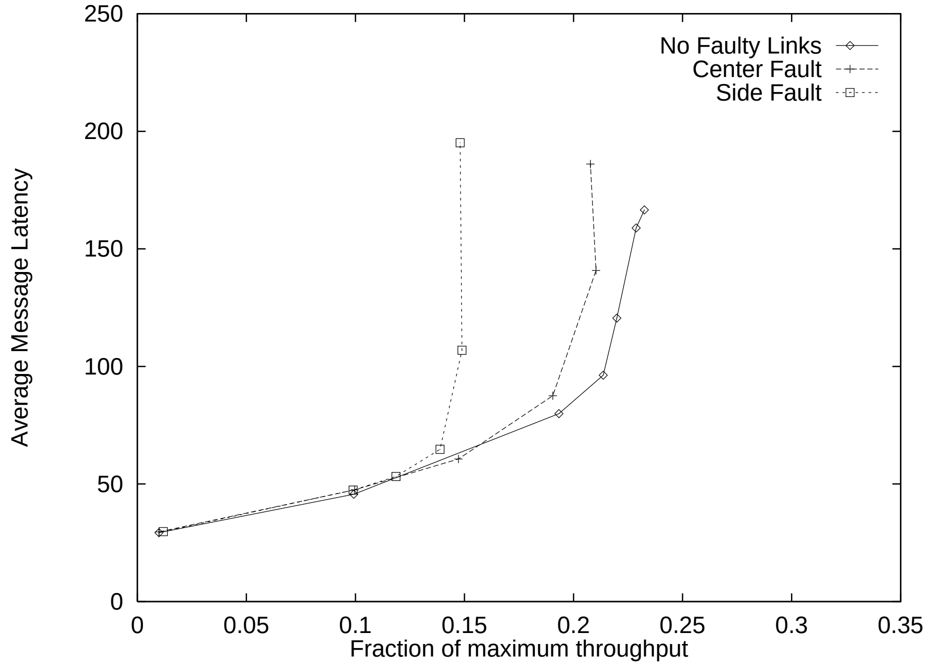


Figure B.15: Effect of fault location on performance for a single faulty Y link in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

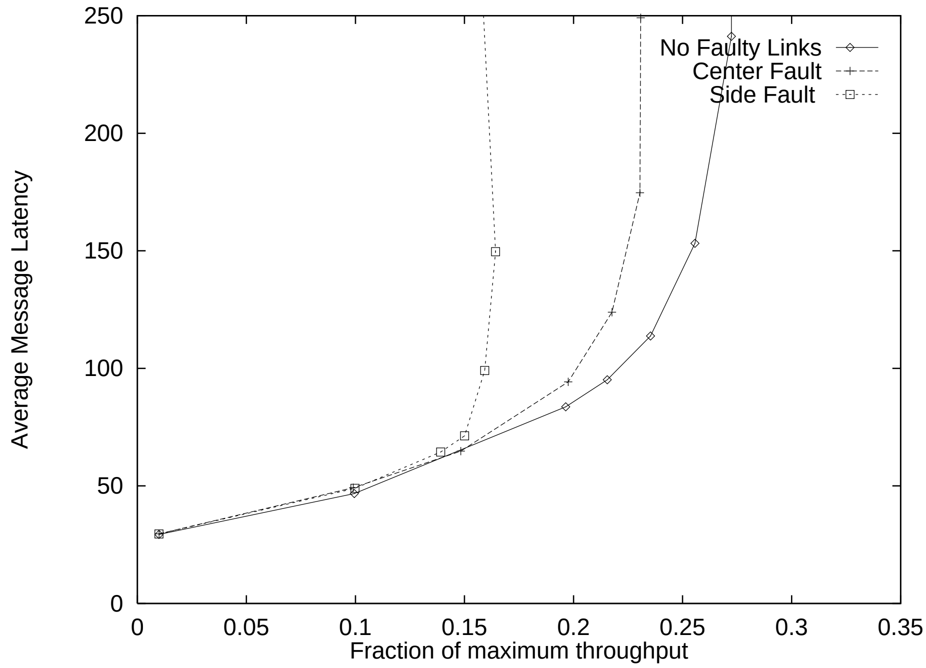


Figure B.16: Effect of fault location on performance for a single faulty Y link in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

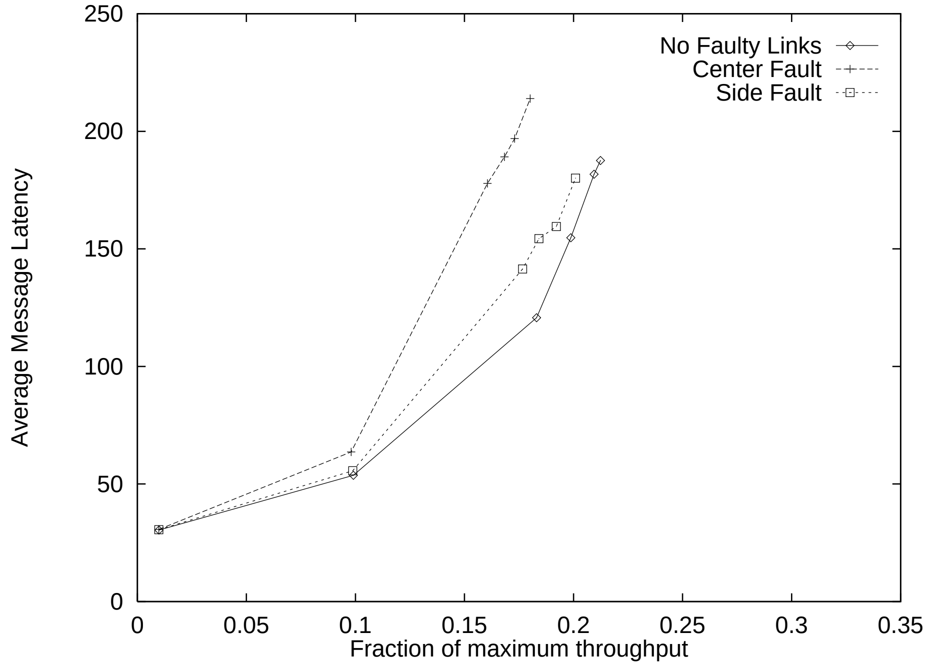


Figure B.17: Effect of fault location on performance for a single node fault in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

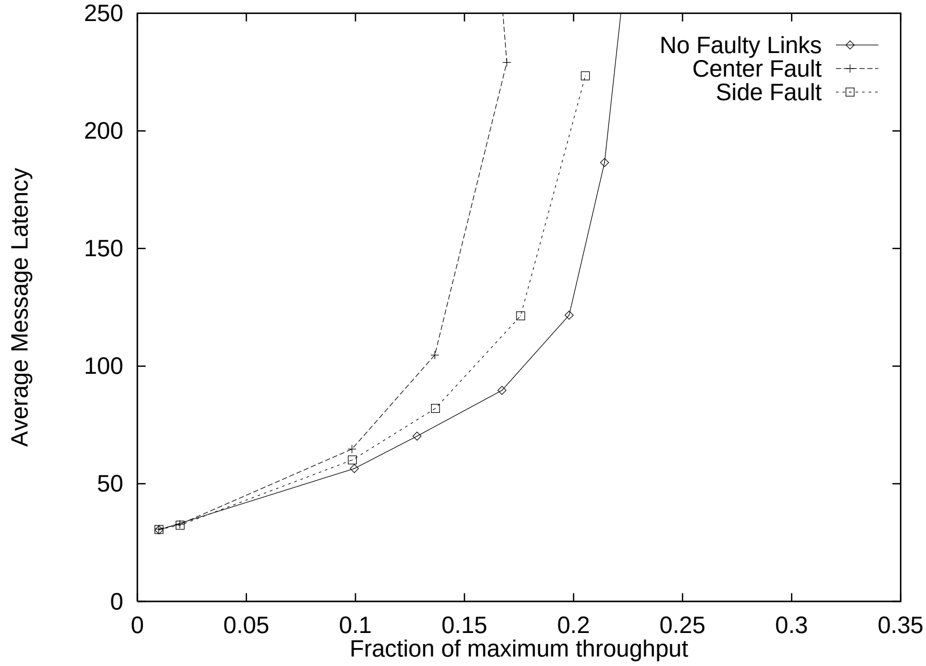


Figure B.18: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

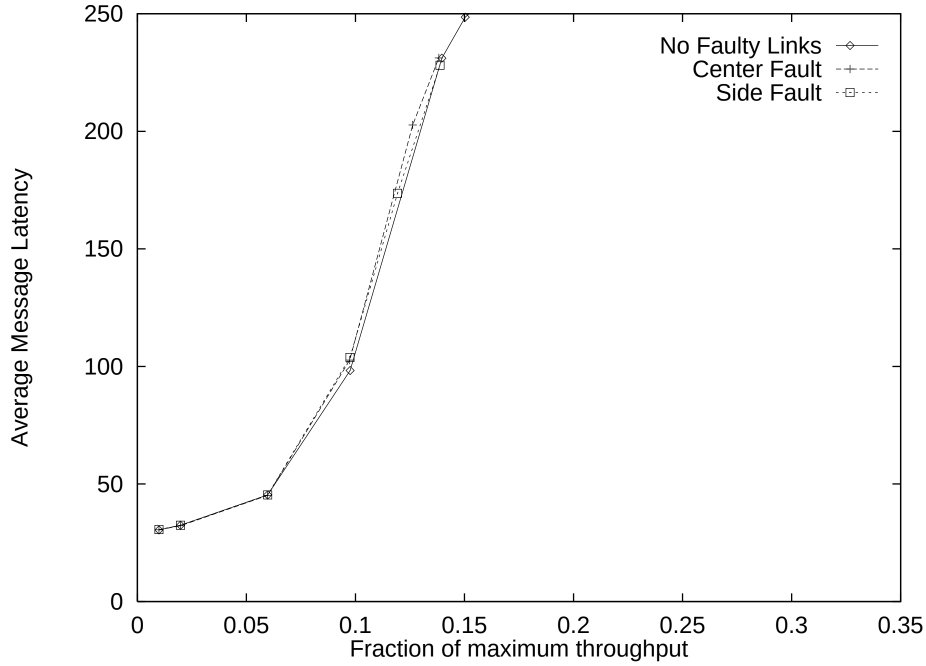


Figure B.19: Effect of fault location on performance for a single node fault in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

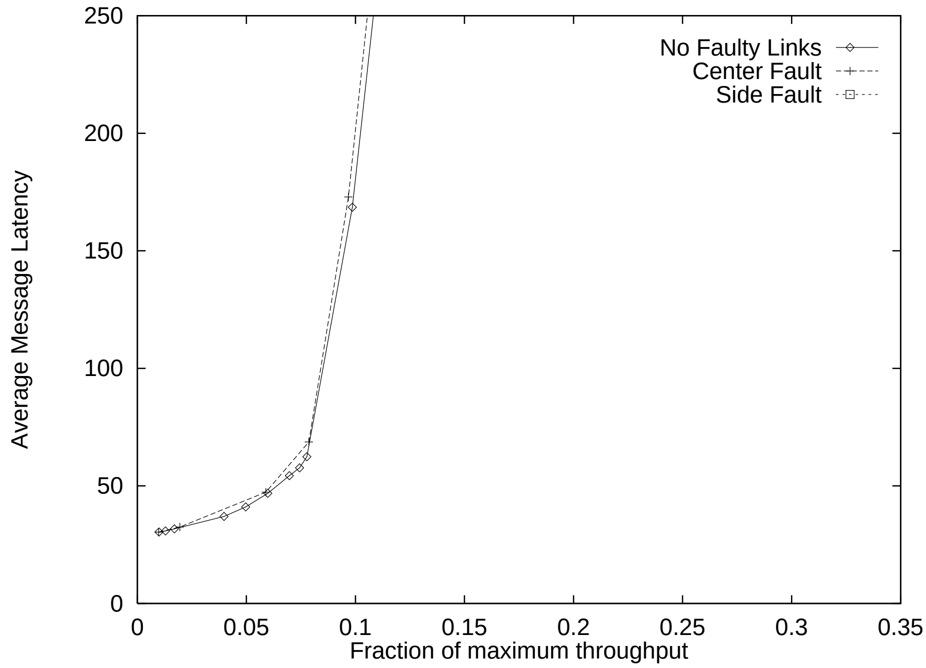


Figure B.20: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

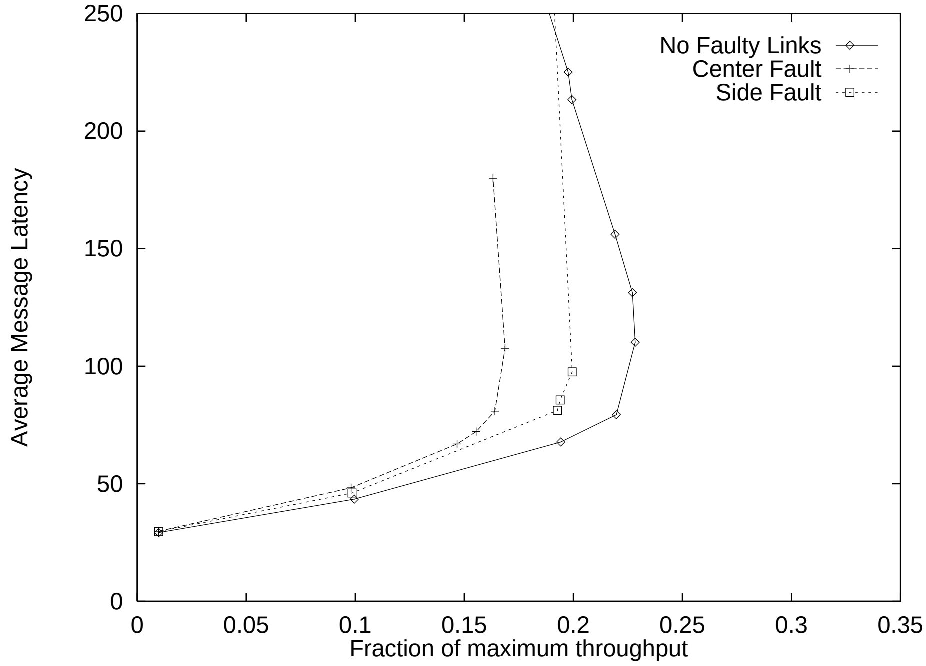


Figure B.21: Effect of fault location on performance for a single node fault in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

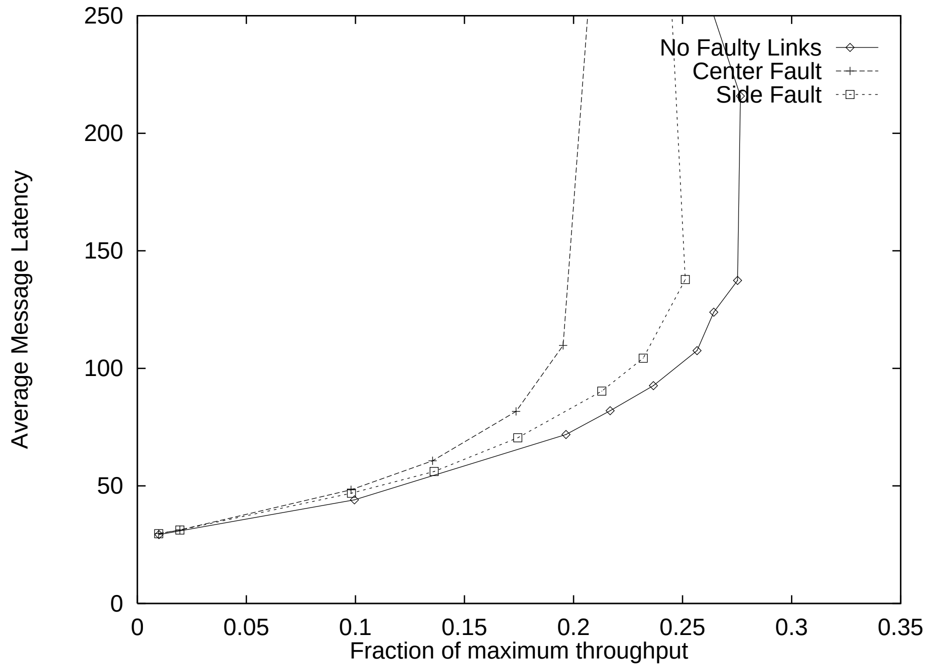


Figure B.22: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

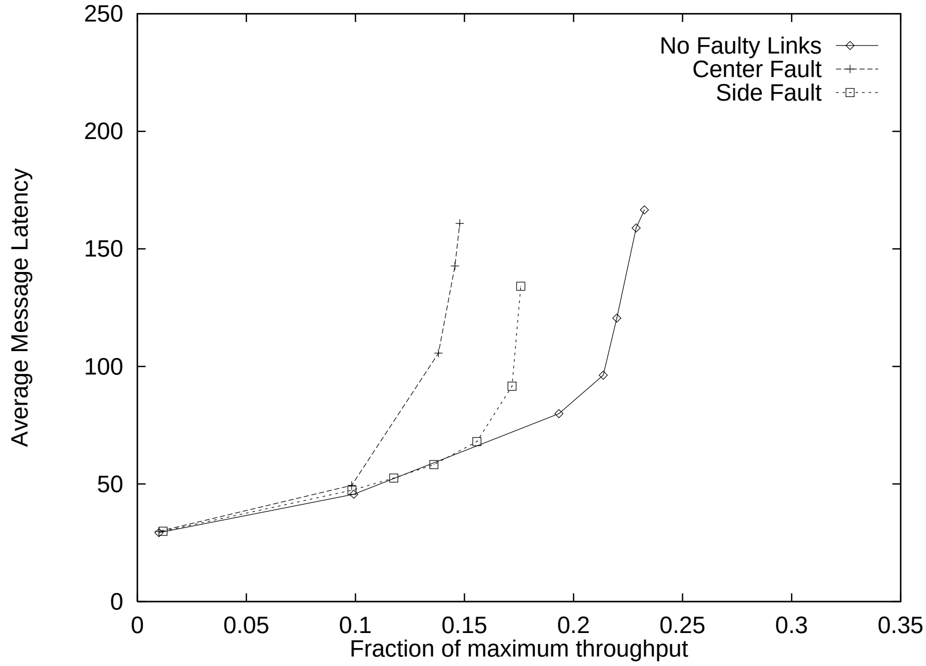


Figure B.23: Effect of fault location on performance for a single node fault in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

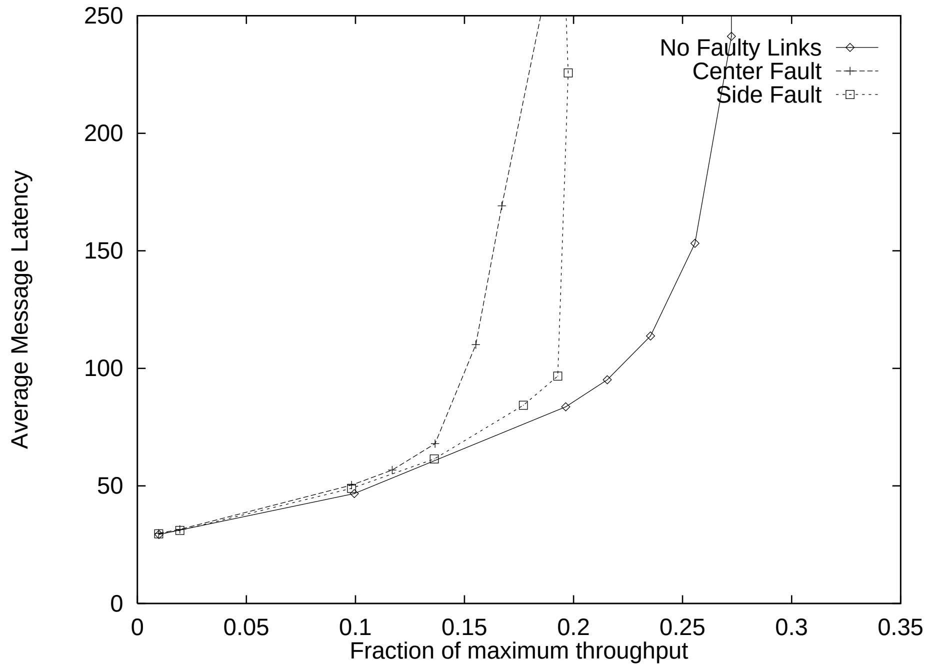


Figure B.24: Effect of fault location on performance for a single node fault in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

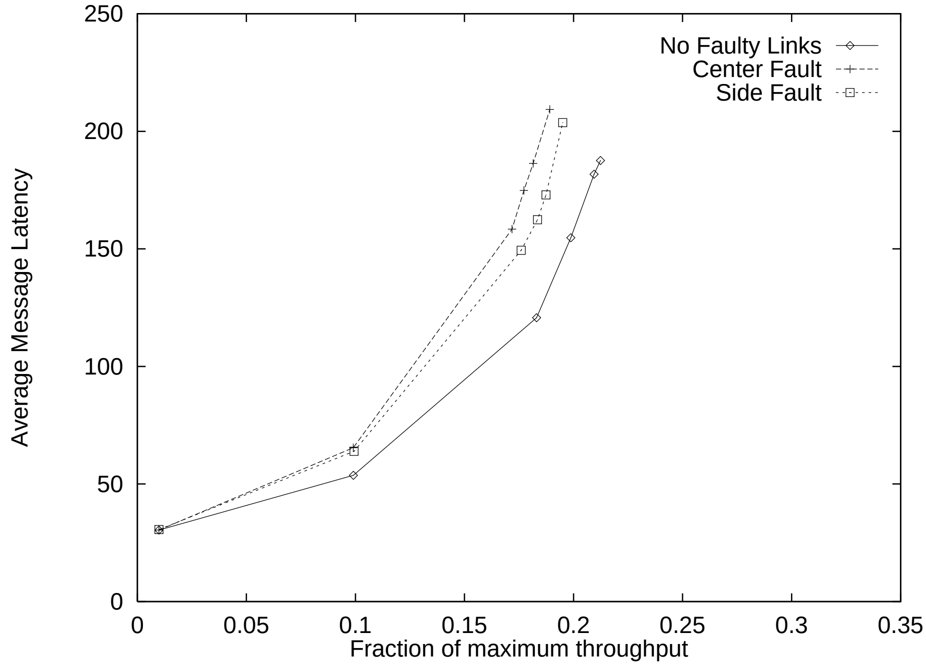


Figure B.25: Effect of fault location on performance for a node with two link faults in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

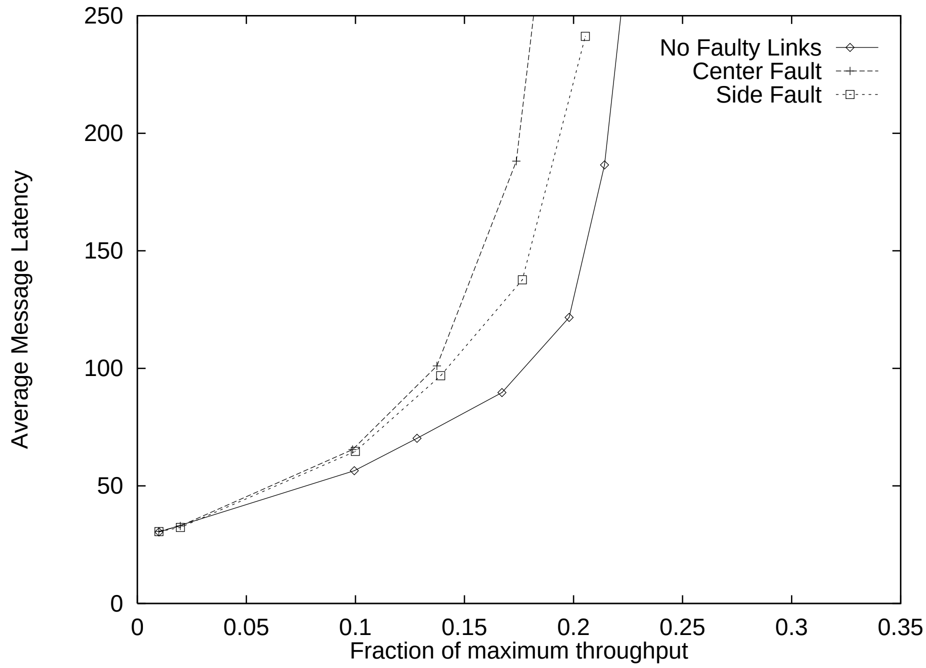


Figure B.26: Effect of fault location on performance for a node with two link faults in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

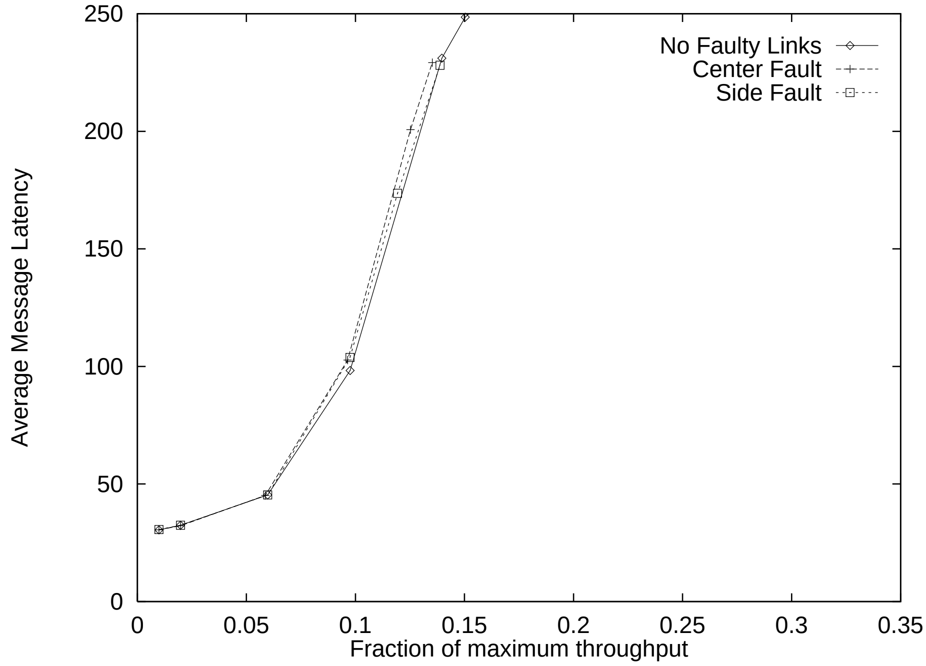


Figure B.27: Effect of fault location on performance for a node with two link faults in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

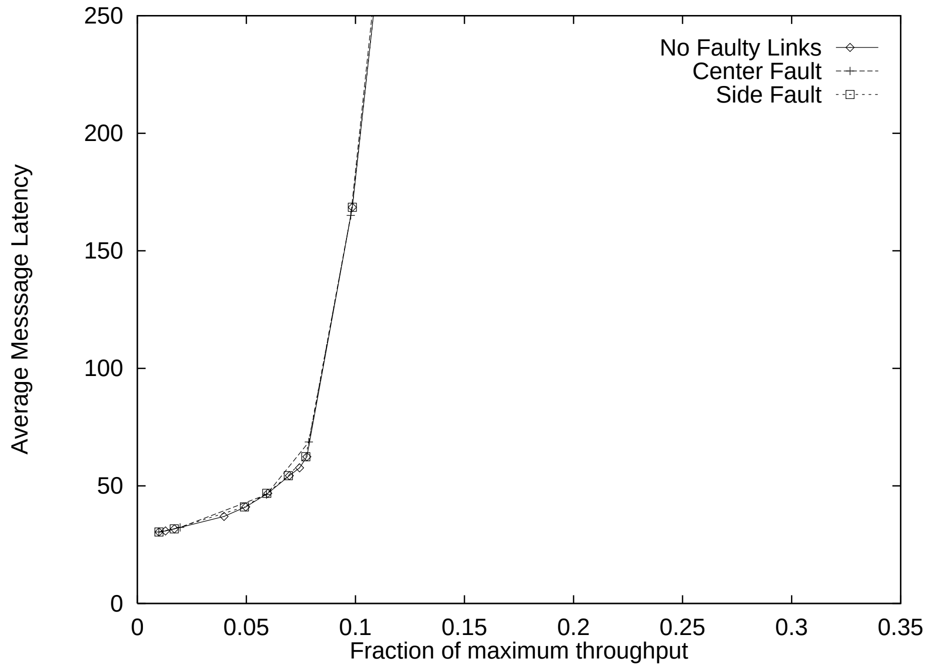


Figure B.28: Effect of fault location on performance for a node with two link faults in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

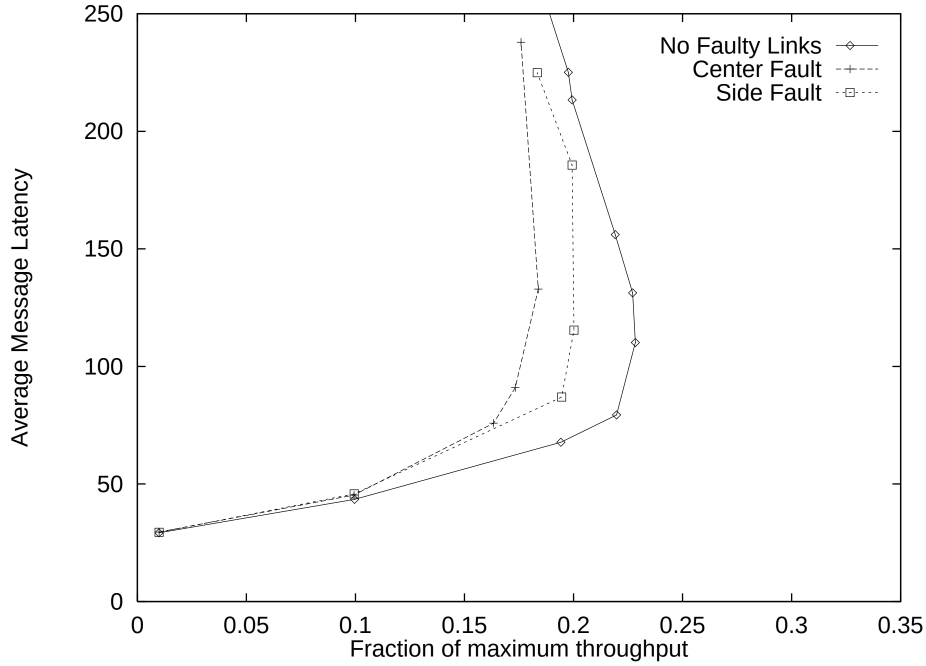


Figure B.29: Effect of fault location on performance for a node with two link faults in a network with 3 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

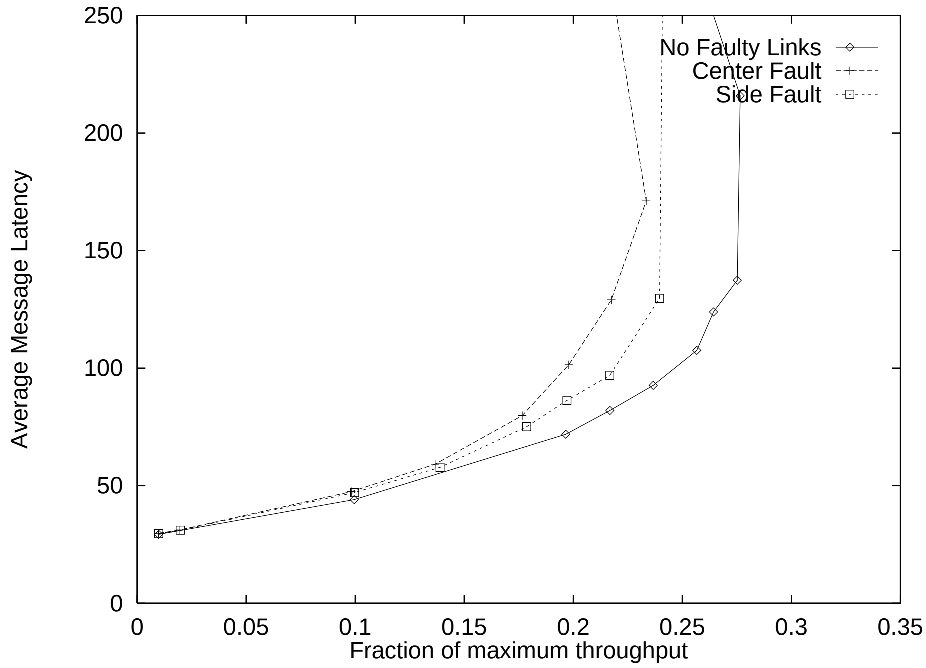


Figure B.30: Effect of fault location on performance for a node with two link faults in a network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

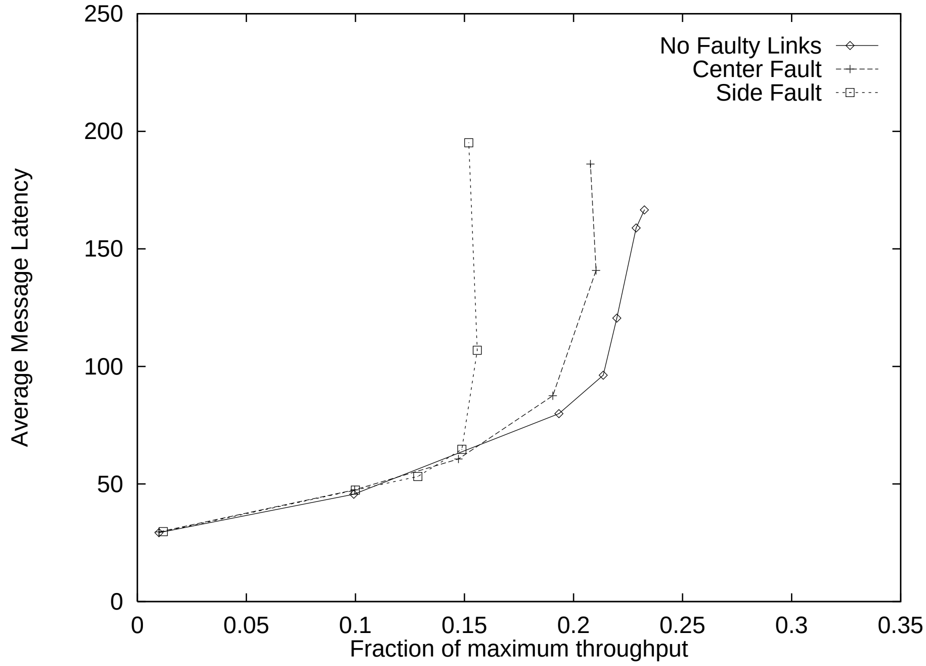


Figure B.31: Effect of fault location on performance for a node with two link faults in a network with 3 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

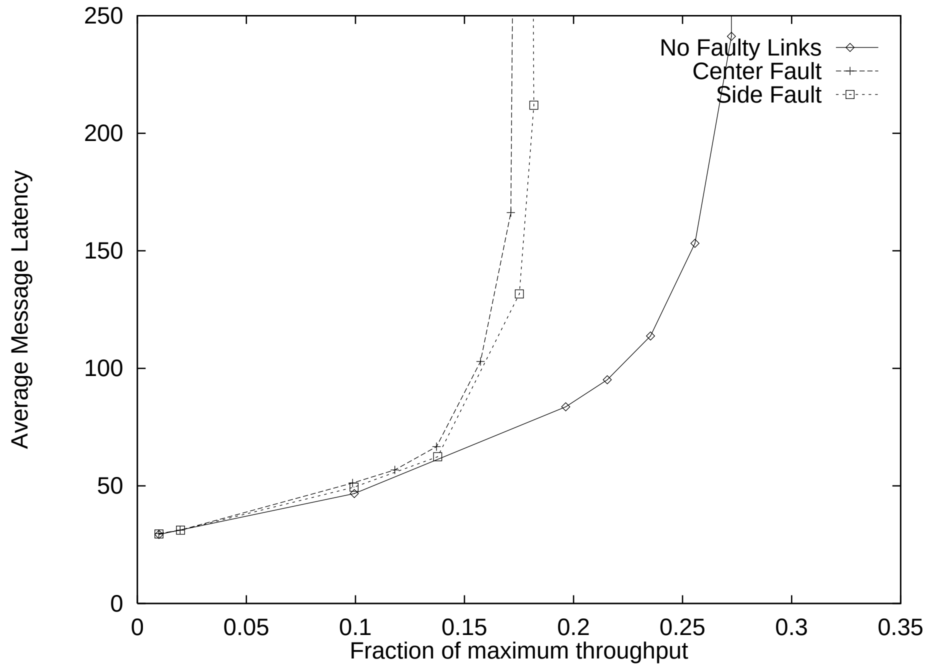


Figure B.32: Effect of fault location on performance for a node with two link faults in a network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

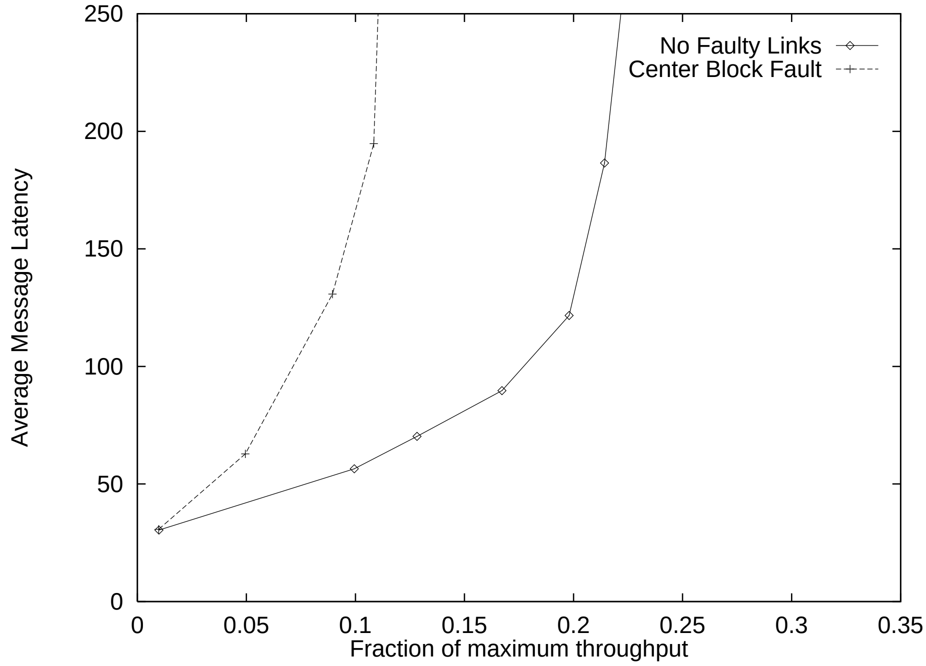


Figure B.33: Effect of block fault of four nodes at the center of the network with 6 virtual channels per physical channel and employing VCR with adaptive routing for matrix transpose traffic

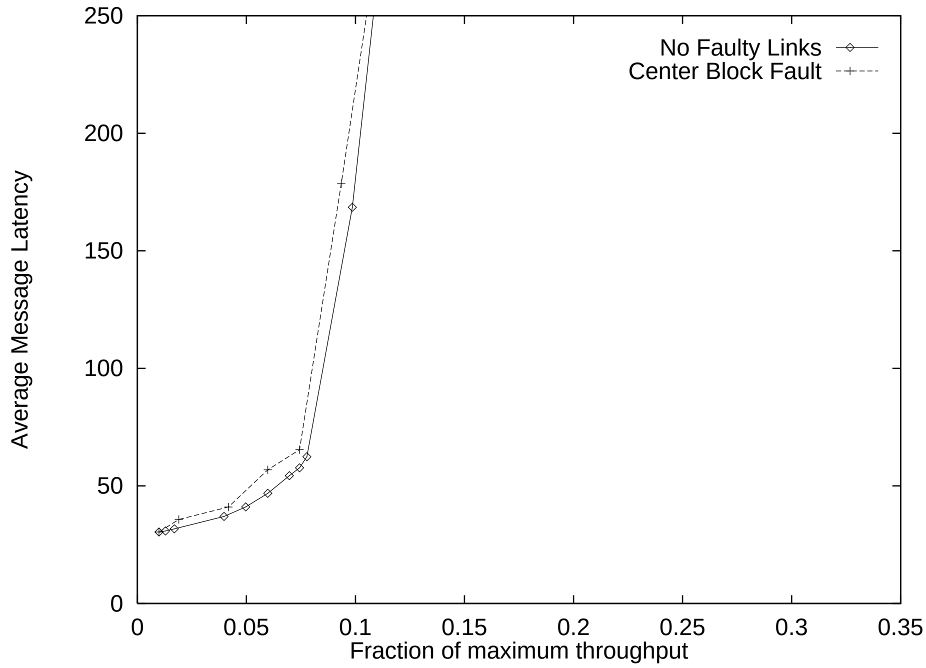


Figure B.34: Effect of block fault of four nodes at the center of the network with 6 virtual channels per physical channel and employing VCR with deterministic routing for matrix transpose traffic

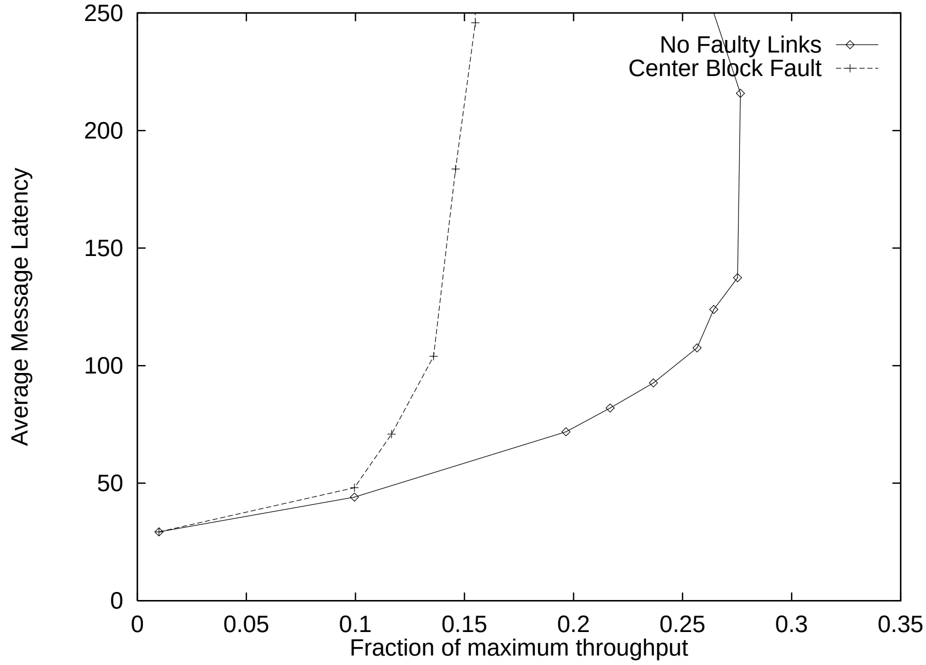


Figure B.35: Effect of block fault of four nodes at the center of the network with 6 virtual channels per physical channel and employing VCR with adaptive routing for uniform traffic

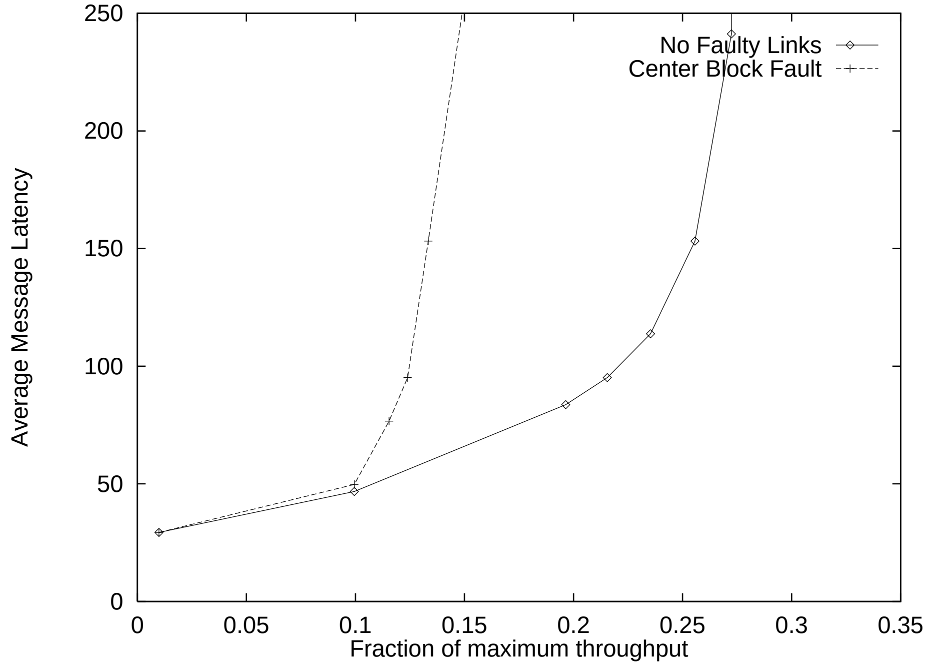


Figure B.36: Effect of block fault of four nodes at the center of the network with 6 virtual channels per physical channel and employing VCR with deterministic routing for uniform traffic

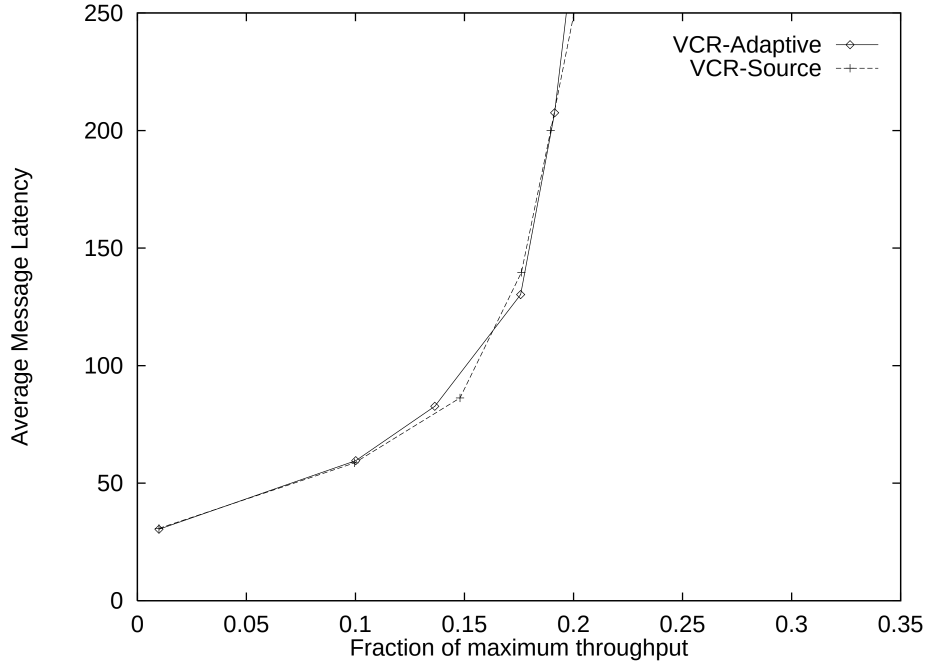


Figure B.37: Comparison of VCR with adaptive routing against source-directed routing for a single faulty X link at the center of the network for matrix transpose traffic

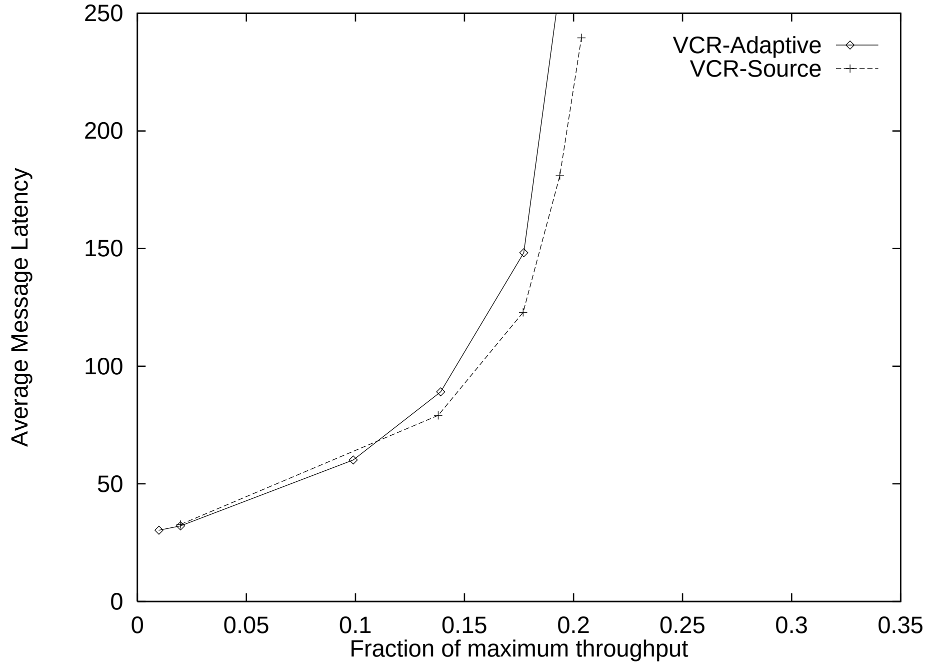


Figure B.38: Comparison of VCR with adaptive routing against source-directed routing for a single faulty Y link at the center of the network for matrix transpose traffic

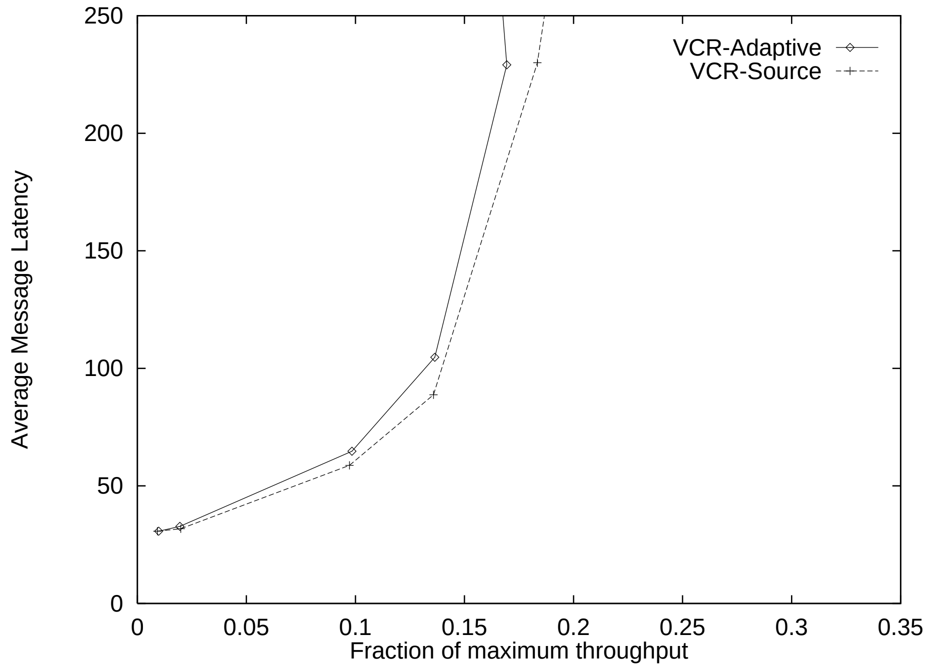


Figure B.39: Comparison of VCR with adaptive routing against source-directed routing for a single node fault at the center of the network for matrix transpose traffic

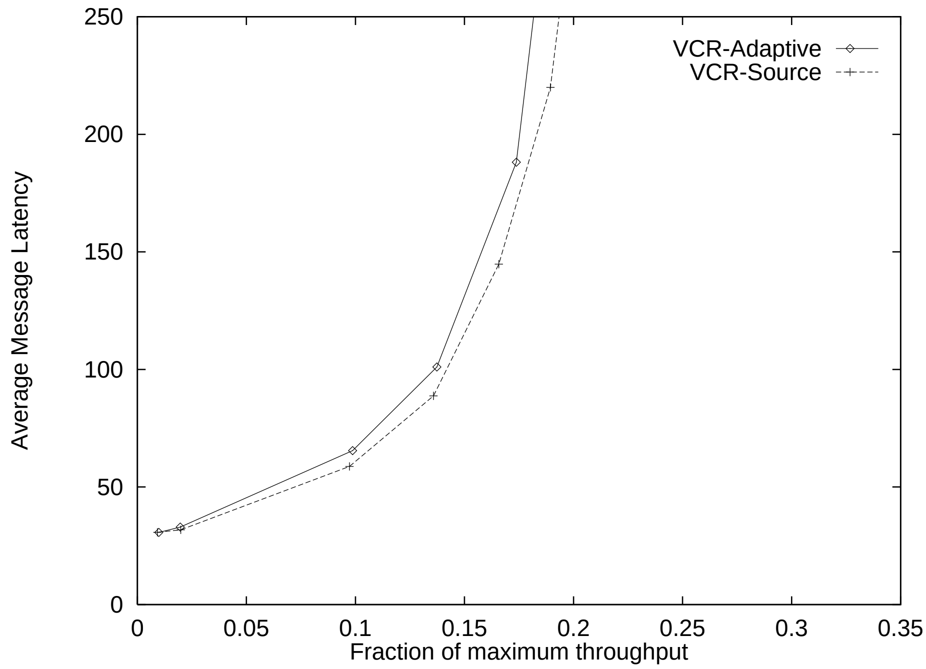


Figure B.40: Comparison of VCR with adaptive routing against source-directed routing for a single node with two link faults at the center of the network for matrix transpose traffic

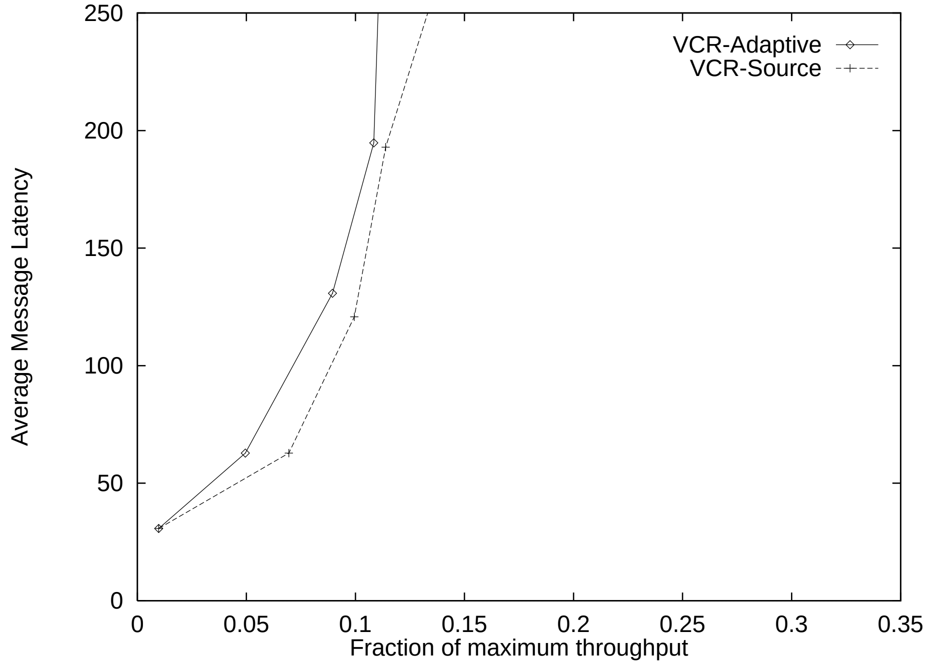


Figure B.41: Comparison of VCR with adaptive routing against source-directed routing for a block fault of four nodes at the center of the network for matrix transpose traffic

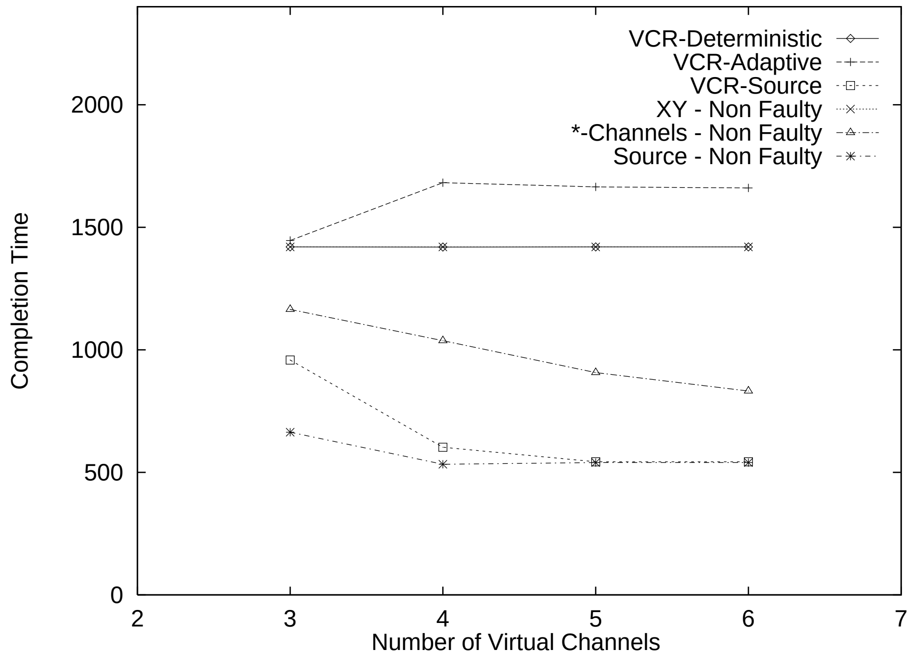


Figure B.42: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the center of the network

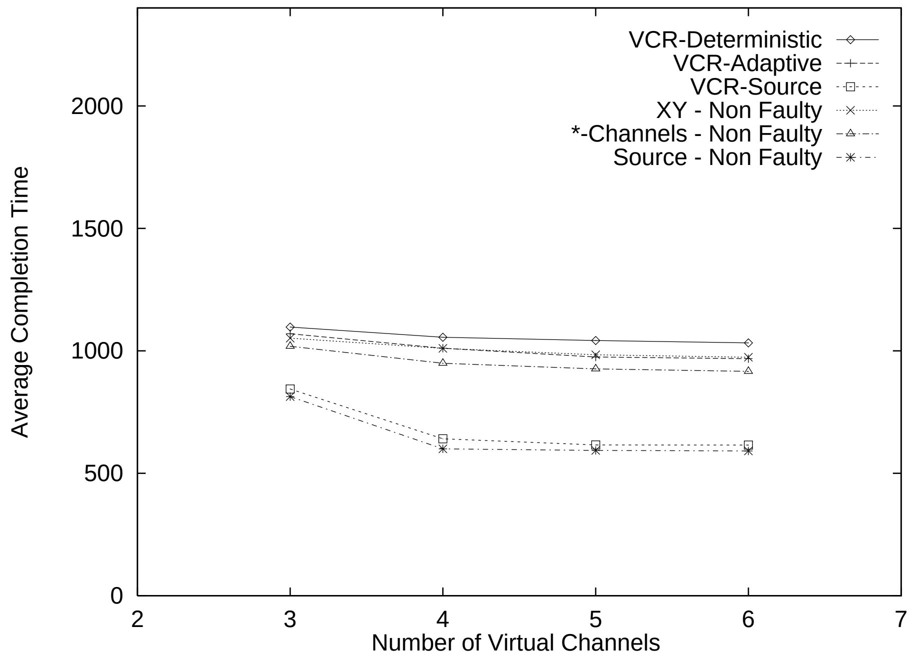


Figure B.43: Completion times for routing algorithms for general traffic patterns and a single faulty X link at the center of the network

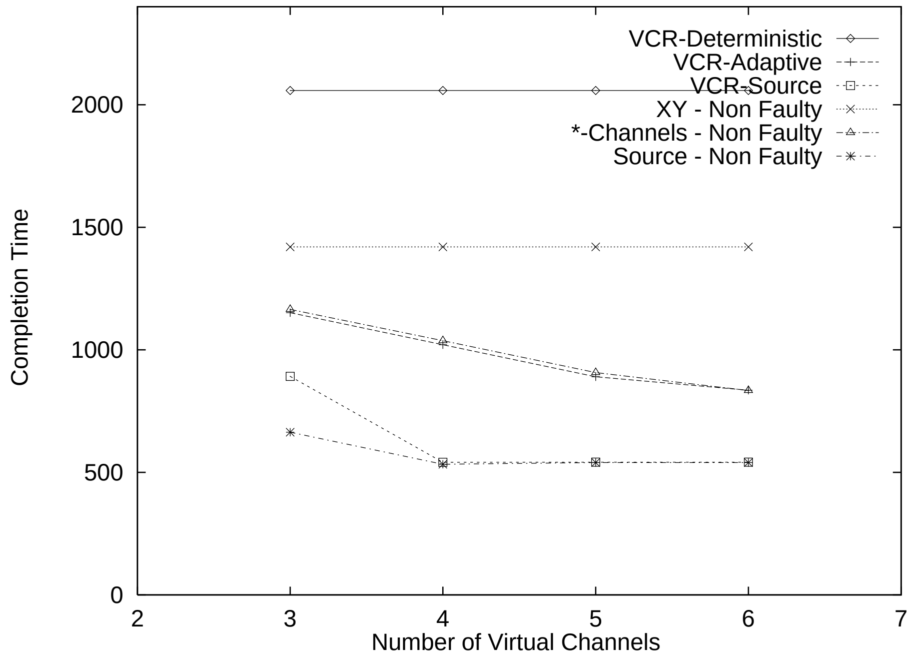


Figure B.44: Completion times for routing algorithms for matrix transpose traffic and a single faulty X link at the side of the network

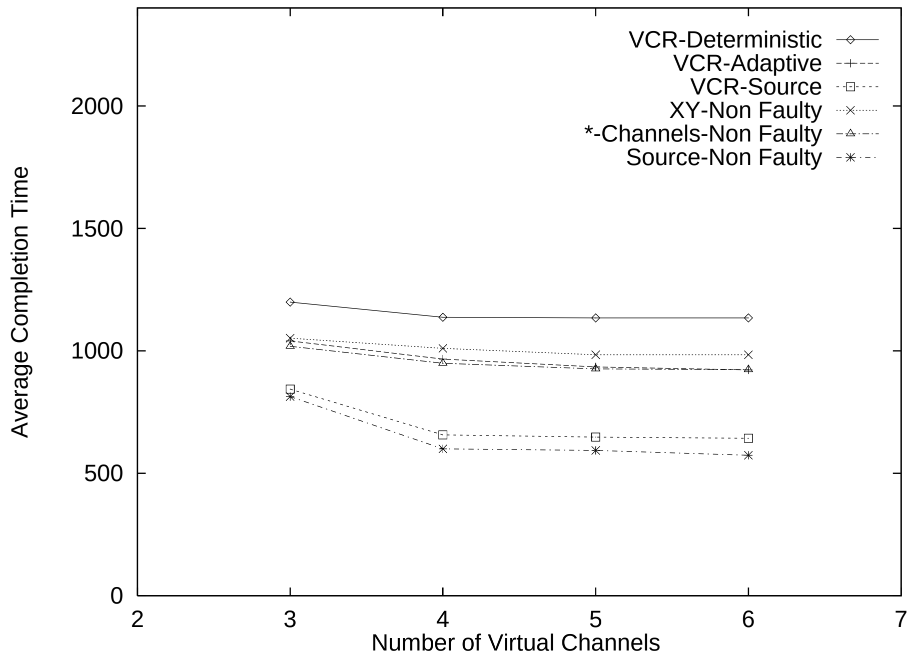


Figure B.45: Completion times for routing algorithms for general traffic patterns and a single faulty X link at the side of the network

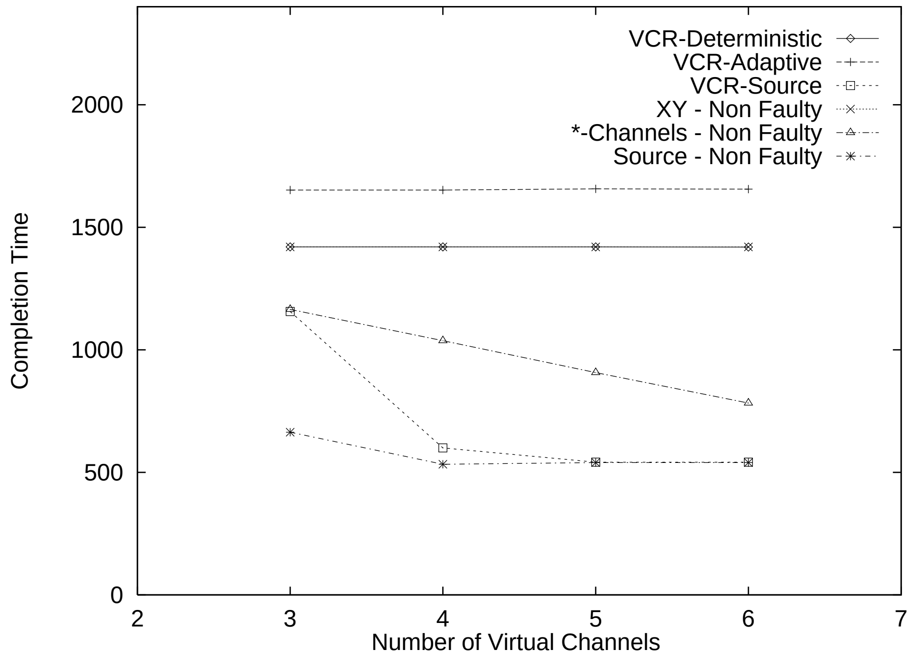


Figure B.46: Completion times for routing algorithms for matrix transpose traffic and a single faulty Y link at the center of the network

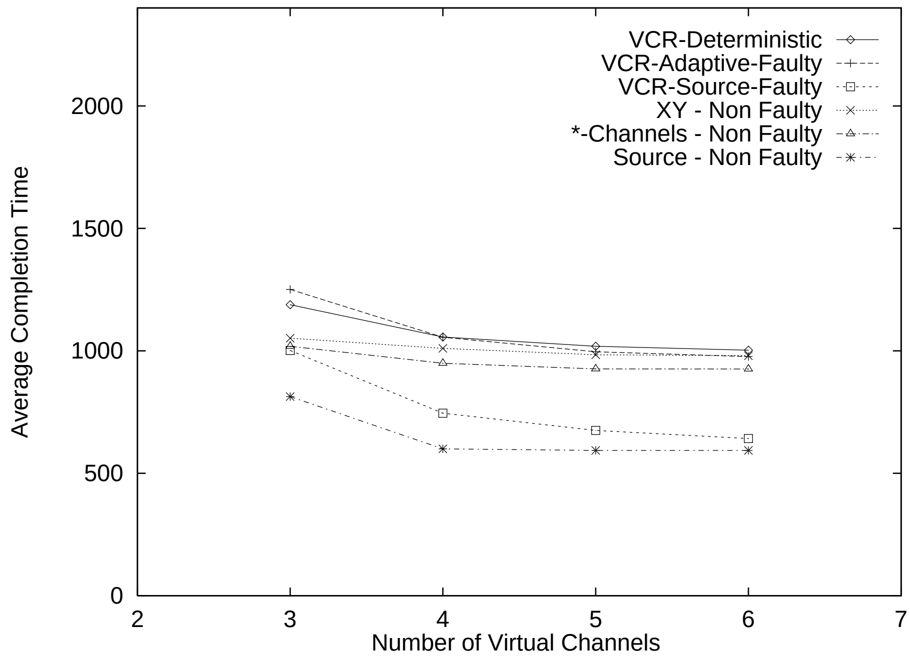


Figure B.47: Completion times for routing algorithms for general traffic patterns and a single faulty Y link at the center of the network

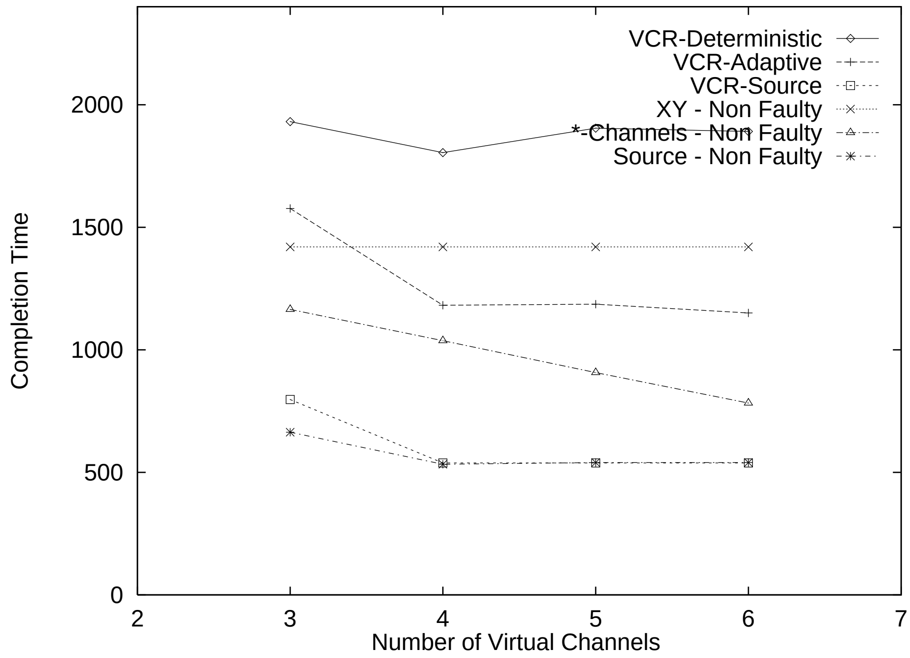


Figure B.48: Completion times for routing algorithms for matrix transpose traffic and a single faulty Y link at the side of the network

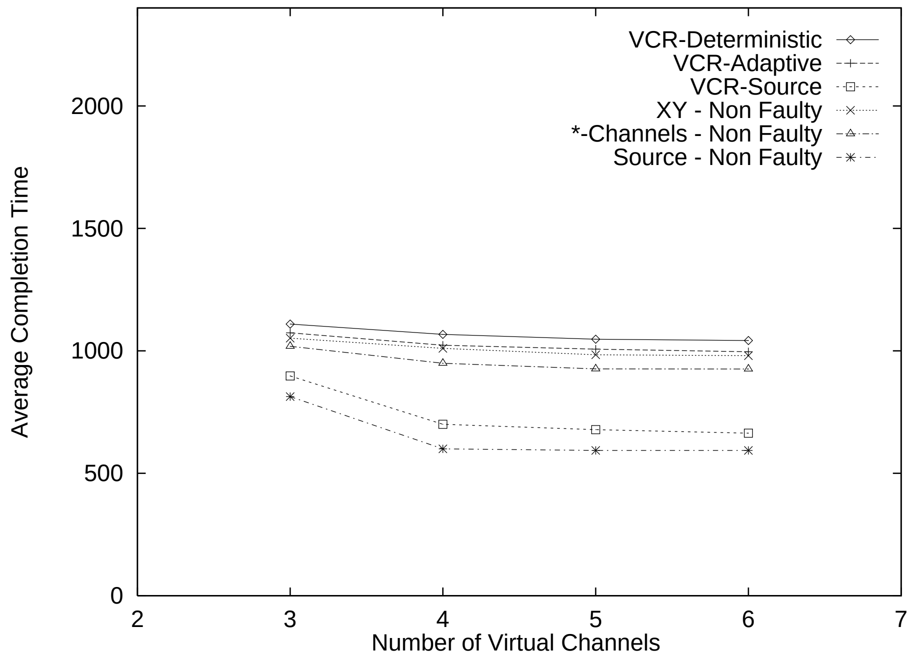


Figure B.49: Completion times for routing algorithms for general traffic patterns and a single faulty Y link at the side of the network

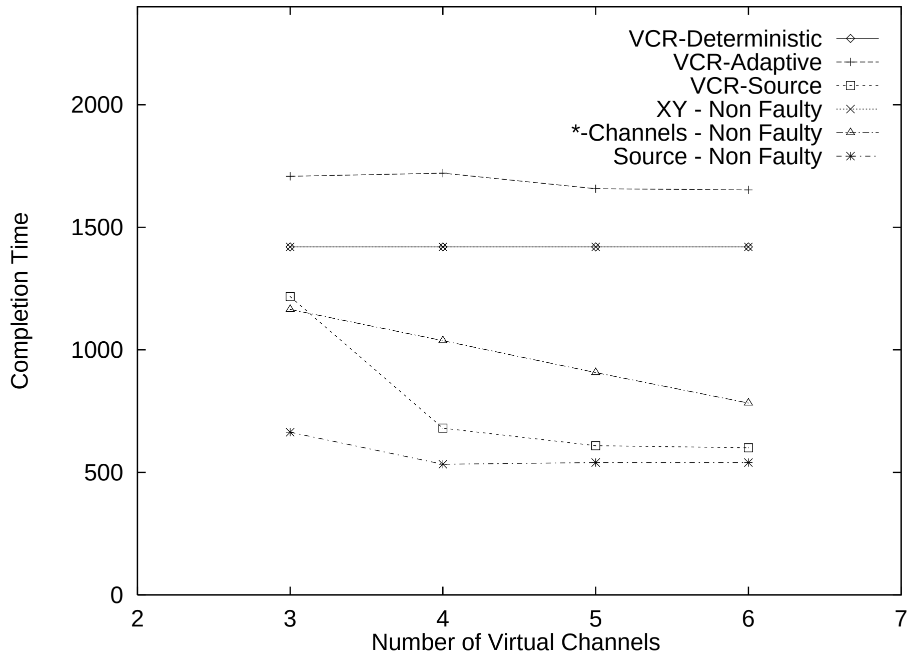


Figure B.50: Completion times for routing algorithms for matrix transpose traffic and a single node fault at the center of the network

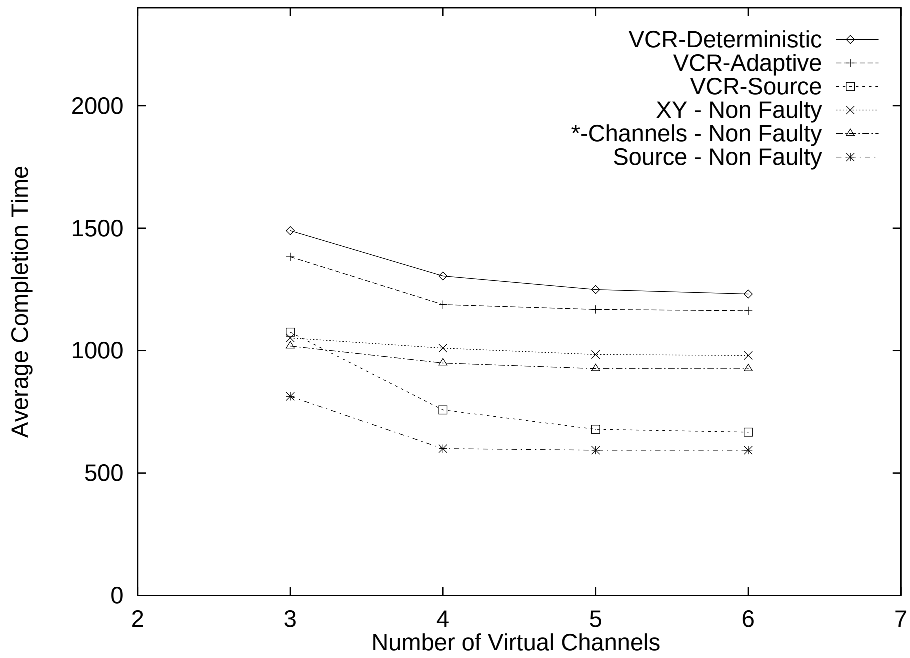


Figure B.51: Completion times for routing algorithms for general traffic patterns and a single node fault at the center of the network

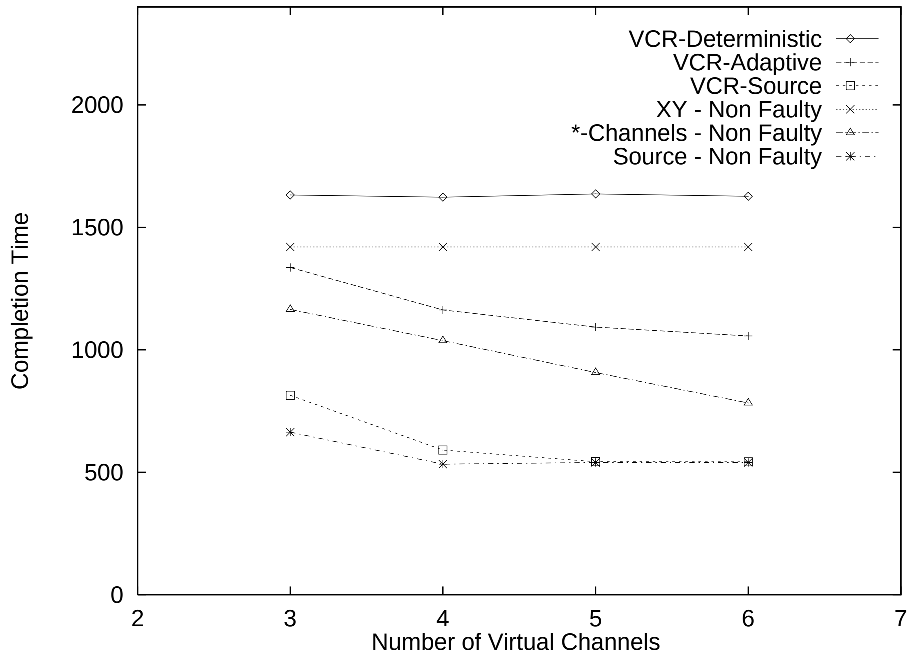


Figure B.52: Completion times for routing algorithms for matrix transpose traffic and a single node fault at the side of the network

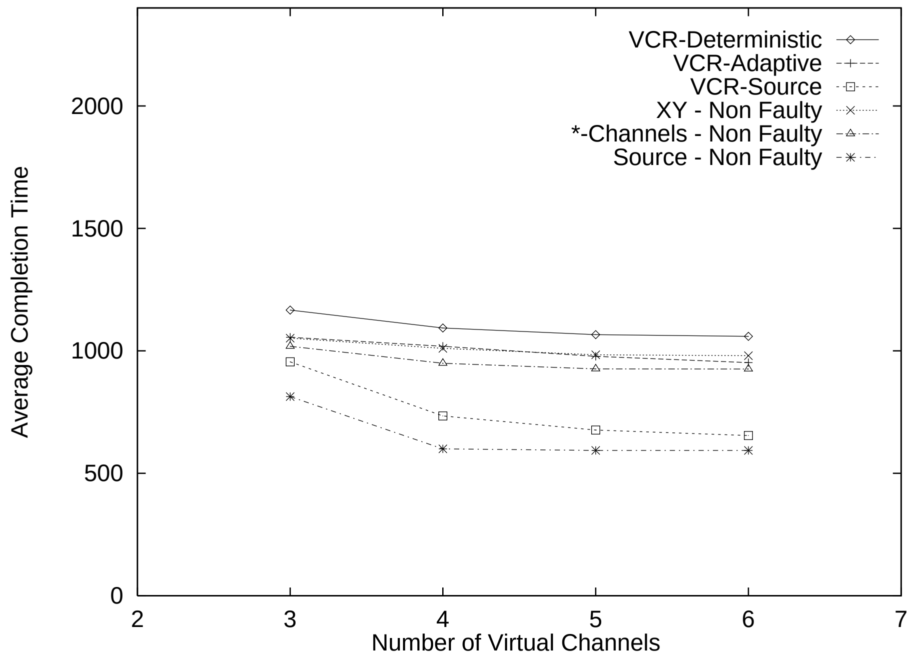


Figure B.53: Completion times for routing algorithms for general traffic patterns and a single node fault at the side of the network

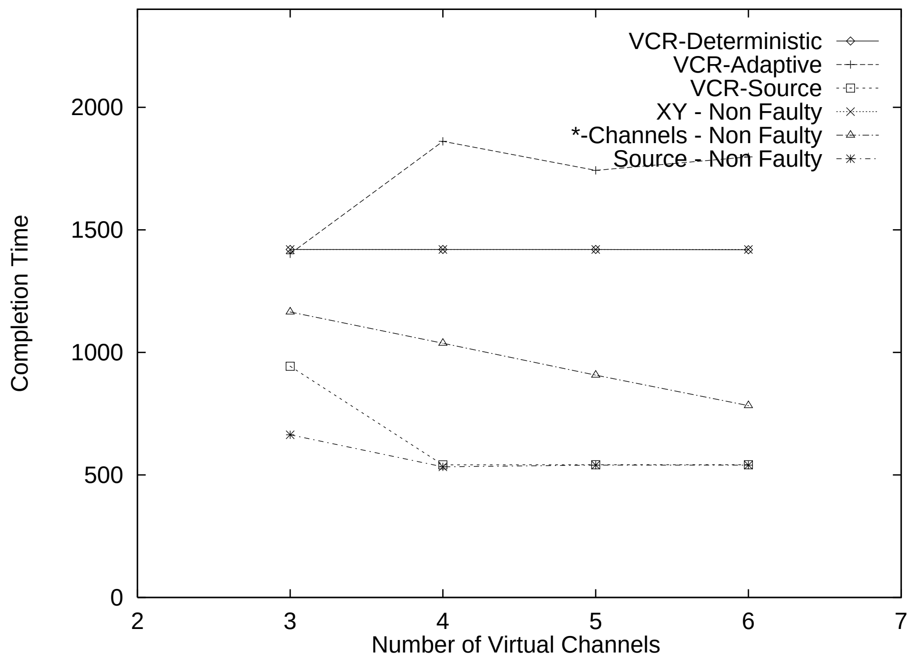


Figure B.54: Completion times for routing algorithms for matrix transpose traffic and a node with two link faults at the center of the network

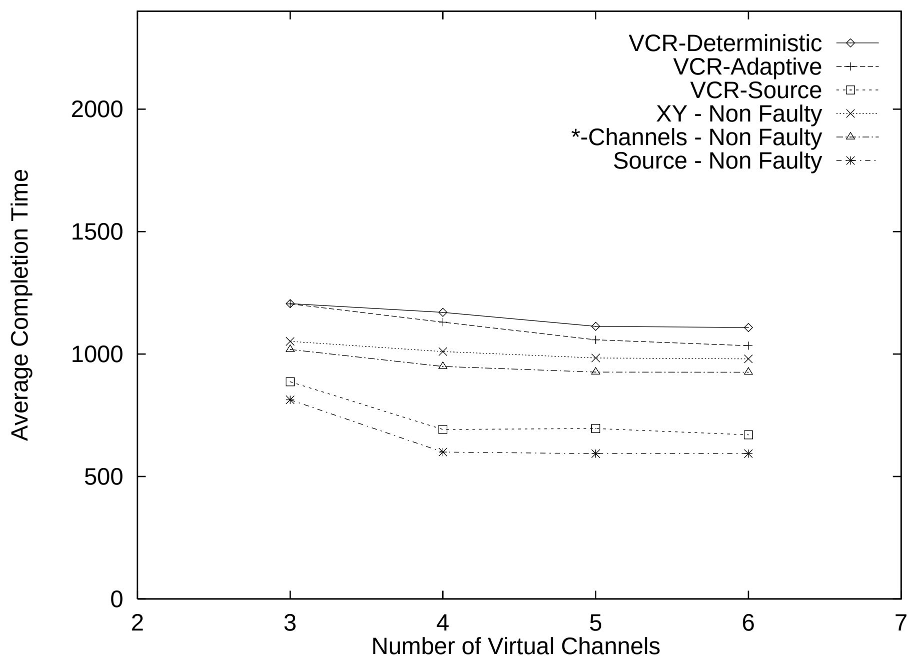


Figure B.55: Completion times for routing algorithms for general traffic patterns and a node with two link faults at the center of the network

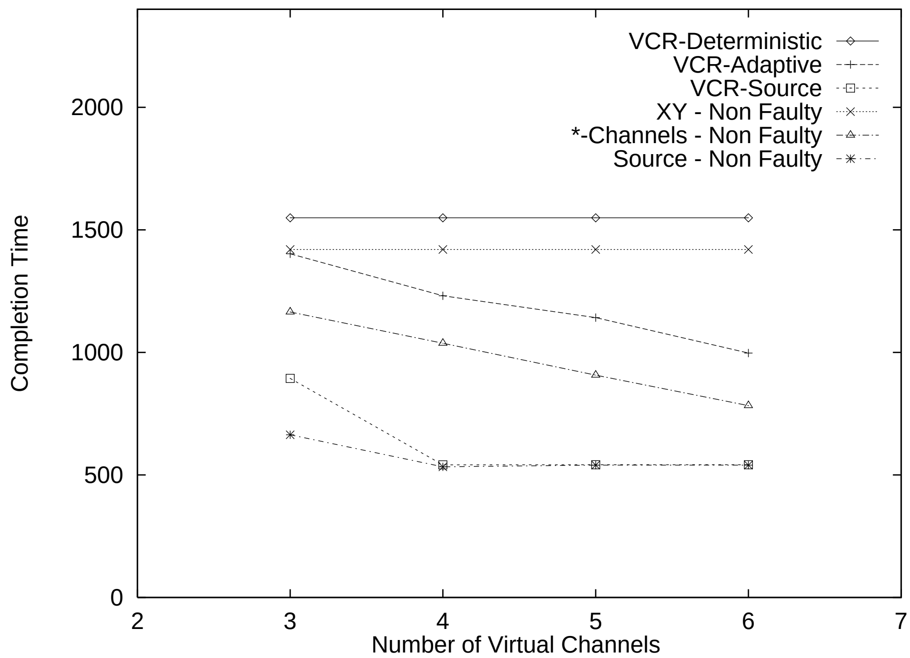


Figure B.56: Completion times for routing algorithms for matrix transpose traffic and a node with two link faults at the side of the network

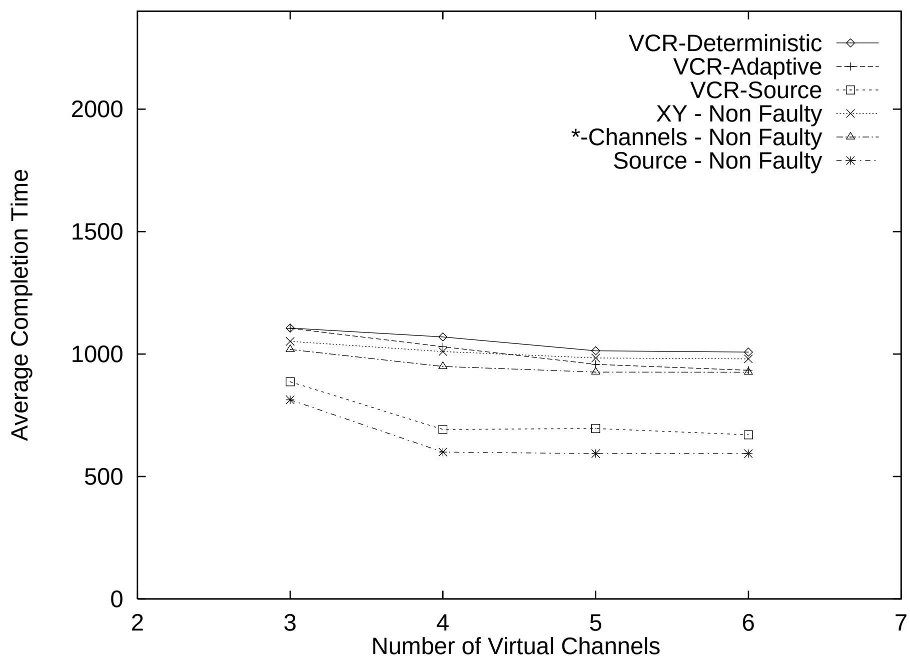


Figure B.57: Completion times for routing algorithms for general traffic patterns and a node with two link faults at the side of the network

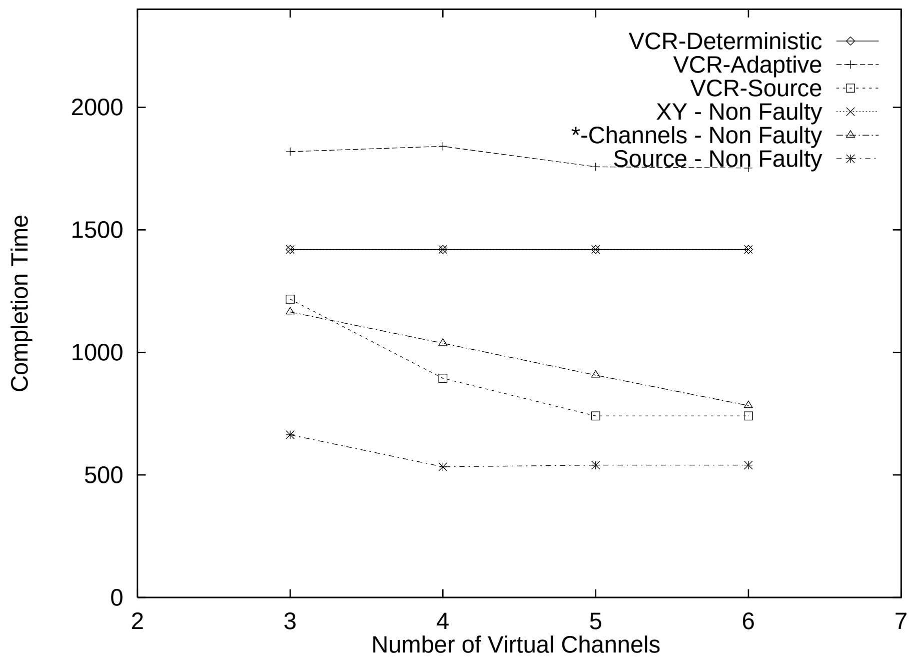


Figure B.58: Completion times for routing algorithms for matrix transpose traffic and a block fault of four nodes at the center of the network

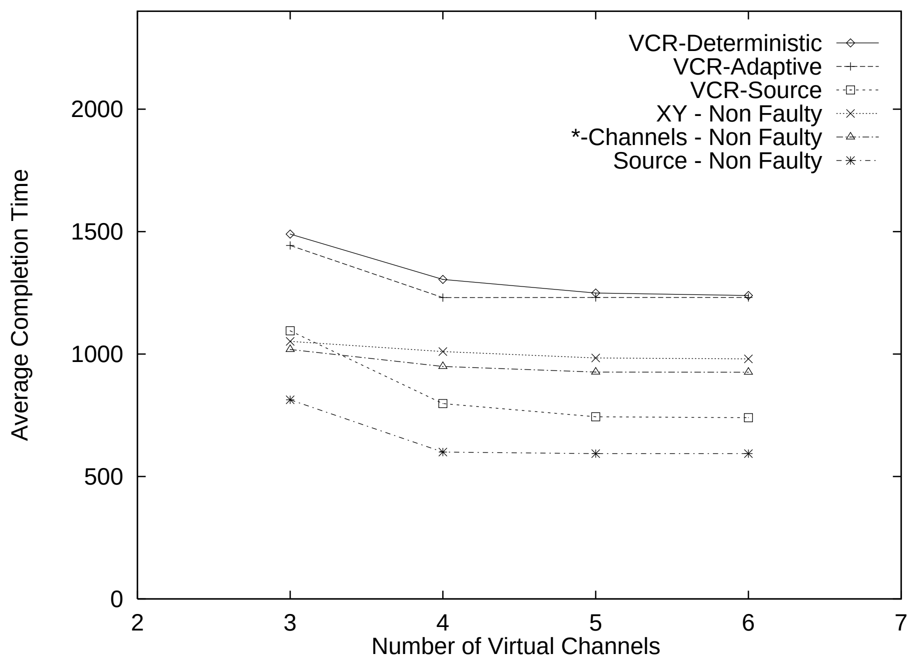


Figure B.59: Completion times for routing algorithms for general traffic patterns and a block fault of four nodes at the center of the network