Application-Aware Deadlock-Free Oblivious Routing

by

Michel A. Kinsy

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARCHIVES

June 2009

Author .....
Department of Electrical Engineering and Computer Science
May 11, 2009

Certified by .....
Srinivas Devadas
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students
Department of Electrical Engineering and Computer Science

Application-Aware Deadlock-Free Oblivious Routing

by

Michel A. Kinsy

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Systems that can be integrated on a single silicon die have become larger and increasingly complex, and wire designs as communication mechanisms for these systems on chip (SoC) have shown to be a limiting factor in their performance.

As an approach to remove the limitation of communication and to overcome wire delays, interconnection networks or Network-on-Chip (NoC) architectures have emerged. NoC architectures enable faster data communication between components and are more scalable.

In designing NoC systems, there are three key issues; the topology, which directly depends on packaging technology and manufacturing costs, dictates the throughput and latency bounds of the network; the flit control protocol, which establishes how the network resources are allocated to packets exchanged between components; and finally, the routing algorithm, which aims at optimizing network performance for some topology and flow control protocol by selecting appropriate paths for those packets.

Since the routing algorithm sits on top of the other layers of design, it is critical that routing is done in a matter that makes good usage of the resources of the network. Two main approaches to routing, oblivious and adaptive, have been followed in creating routing algorithms for these systems. Each approach has its pros and cons; oblivious routing, as opposite to adaptive routing, uses no network state information in determining routes at the cost of lower performance on certain applications, but has been widely used because of its simpler hardware requirements.

This thesis examines oblivious routing schemes for NoC architectures. It introduces various non-minimal, oblivious routing algorithms that globally allocate network bandwidth for a given application when estimated bandwidths for data transfers are provided, while ensuring deadlock freedom with no significant additional hardware.

The work presents and evaluates these oblivious routing algorithms which attempt to minimize the maximum channel load (MCL) across all network links in an effort to maximize application throughput. Simulation results from popular synthetic benchmarks and concrete applications, such as an H.264 decoder, show that it is possible to achieve better performance than traditional deterministic and oblivious routing schemes.

Thesis Supervisor: Srinivas Devadas
Title: Professor of Electrical Engineering and Computer Science

Acknowledgements

First, I would like to express my deep and sincere gratitude to Professor Srinivas Devadas for his extraordinary guidance, inexhaustible energy, and remarkable patience throughout this research. His support, encouragement, sound advice, and good teaching have been a true source of inspiration for me and I would have been lost without them.

I would like to thank Professor Edward Suh for his detailed and constructive comments, and for his important support throughout this work.

I am indebted to my many student colleagues for providing a stimulating and supportive environment in which to learn and conduct this research. I am especially grateful to Myong Hyon Cho for his participation in many aspects of this work, to Chih-chi Cheng for his help on the H.264 application, to Marten van Dijk for his valuable input and to Keun Sup Shim and Tina Wen for their help in generating simulation results.

I want to express my heartfelt gratitude to Myron King for his friendship in and out of the lab, to Charles O'Donnell for all his help on the formatting of the text, Mieszko Lis for his help with editing, and to Nirav Dave for his mentorship. I also thank Joel Emer, David Wentzlaff, Michael Pellauer, Bill Thies, and Vijay Ganesh for helpful comments on this research.

I am grateful to the Keller and Storace families for providing me a loving environment and to Dr. Sarma Vrudhula for his support and guidance.

Lastly, and most importantly, I wish to thank my parents and sister for their unconditional and ever lasting love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

System-on-chip (SoC) designs typically contain a large number of heterogeneous digital components, such as processors, dedicated hardware engines, and memories, integrated onto a single chip. Traditionally, buses have been used in establishing communications between these different components, but because of the increasing complexity of these designs and the lack of scalability of wired connections between components, network-on-chip (NoC) architectures have been introduced as an effective data communication infrastructure [2, 44].

NoC architectures are characterized by an on-chip packet-switched micro-network of interconnects; they allow the digital components or resources in an SoC to communicate by sending packets to one another over that network. These architectures have resources or processing elements (PEs) attached to switches or routers, forming nodes that are linked together. Figure 1-1 illustrates such an architecture.

An NoC architecture is defined by its topology (the physical organization of nodes in the network), its flow control mechanism (which establishes the data formating, the switching protocol and the buffer allocation), and its routing algorithm (which determines the path selected by a packet to reach its destination under a given application). Figure 1-2 shows a two-dimensional 3 x 3 mesh network topology.

It has been shown that the overall performance of these systems is dominated not by the individual computation power of its resources, but by their communication limits in terms of bandwidth, speed and concurrency [23, 25, 30], in other words, the router architecture and the routing algorithm that are employed.

Because of the importance of the routing algorithm in affecting the overall performance, intensive research efforts have been put in developing algorithms that deliver high performance; unfortunately the vast majority of these algorithms have never been implemented because of the complexity of the router hardware that they require. Therefore, still relevant today is the challenge of designing routing algorithms which achieve good network load balancing, relatively short paths for packets, and simple router architecture.

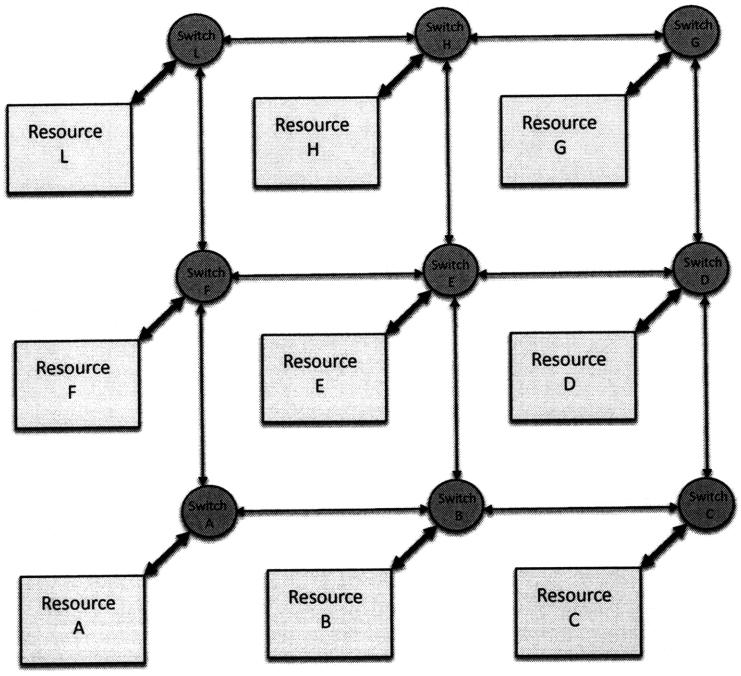


Figure 1-1: Resources and Switches Representation of 3×3 Mesh Network

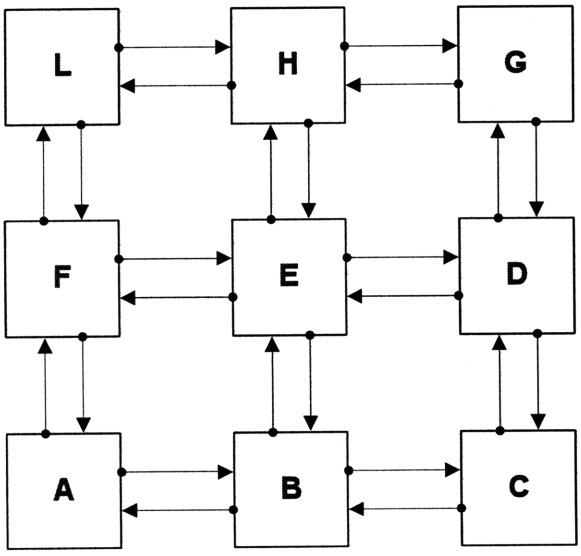


Figure 1-2: Node Representation of 3×3 Mesh Network

1.1 Topology of Networks-on-chip

The topology of an NoC is defined in terms of the theoretical shape formed by its nodes. Over the years, there have been a number of proposed topologies, but very few of them have ever been implemented in a real SoC. In today's Systems-on-chip, the most popular network topologies encountered are of the *orthogonal* shape; this means that the nodes are arranged in an orthogonal n-dimensional space, where each link between two nodes represents a displacement in a single dimension [15]. Figure 1-3 shows several of these topologies.

The popularity of these structures can be directly attributed to the simplicity of the router hardware needed to efficiently support them, because each node in the network can be assigned coordinates in the n-dimensional space and the router can use this underlying property of the topology to make forwarding decision with no overhead. To illustrate this work, the two-dimensional mesh topology has been adopted as shown in Figure 1-1, but the routing techniques presented are effectively topology independent.

1.2 Message Flow Control in Networks-on-chip

Flow control in NOCs architectures defines the unit of data in a given network, and deals with the mechanism for synchronizing the transmission and reception of these data units. Various techniques are used in allocating resources, namely channel bandwidth and buffers, needed by the data traversing the network.

With *circuit switching*, the network is first probed and physical channels are reserved from source to destination before packets are injected into the network. On the other hand in packet switching, when using *store-and-forward* or *virtual cut-through*, network resources are allocated on a per-hop basis. In the case of store-and-forward, a packet is forwarded to its next hop only when all its subparts are received, where cut-through starts the forwarding

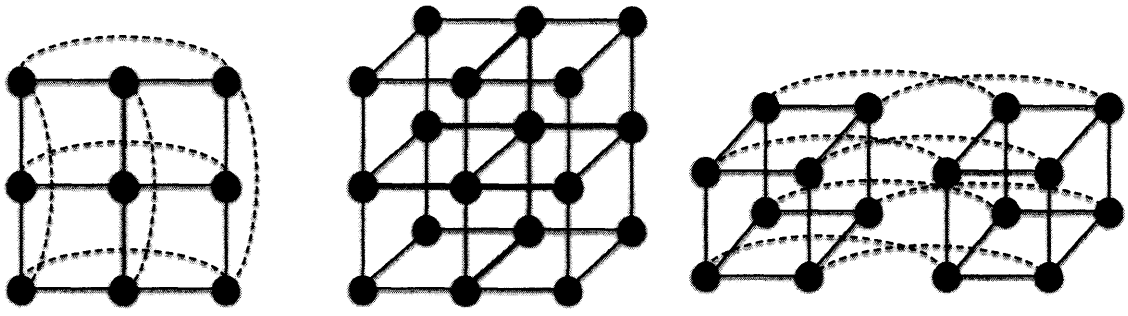


Figure 1-3: Three Examples of Orthogonal Network Topology.

as soon as the packet's header is received. Another flow control mechanism frequently used is *wormhole*, where packets are further divided into flits, and resource allocation is done on a flit basis rather than on a packet basis. Intel's 80-core processor [56] uses a two-dimentional mesh network that routes packets using wormhole routing.

Each of these approaches has its advantages and disadvantages; in this work we adopt the wormhole flow control technique, because of its efficient use of buffer space. The first flit of a packet is called the header flit. It holds the routing information about the packet, and sets up virtual channels for subsequent flits in the packet.

1.3 Network Resource Interface

Another important aspect of NoC design is the interface through which the network resources are integrated into the system. This aspect of the design deals with the conversion of data traffic, such as bus transactions, coming from the resources into packets or flits that can be routed inside the Network-on-chip, and the reconstruction of packets or flits into data traffic at the opposite side when exiting the NoC. Figure 1-4 (a) shows a typical approach used by designers when dealing with the processing element interface to the network.

Since the focus of this work is on routing techniques, the resource interface consideration has been put into the resource side of the overall system design [Figure 1-4 (b)].

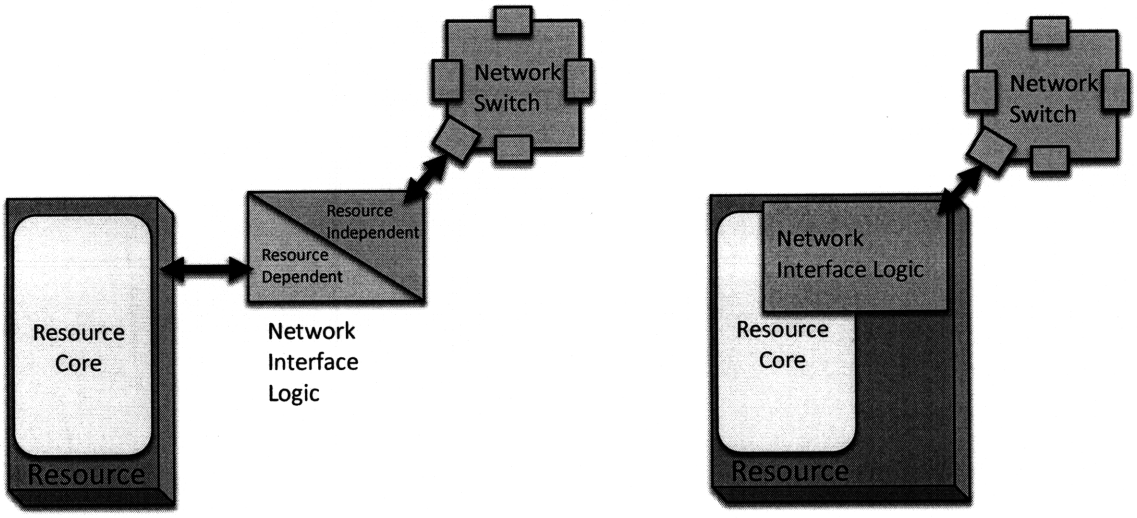


Figure 1-4: Two Resource Interface Approaches. (a) External to the resource. (b) Internal to the resource.

1.4 Routing in Networks-on-chip

Routing algorithms for NoC architectures can be generally classified into oblivious and adaptive [3]. With oblivious routing, which includes deterministic routing algorithms as a subset, the path followed by a packet is statically determined. This allows each node in the network to make its routing decisions independently from the others. Due to this distributed aspect, oblivious routing, such as dimension order routing [1], enables simple and fast router designs, and is widely adopted in today's on-chip interconnection networks. But the drawback with oblivious algorithms is their poor performance when dealing with applications which contain certain communication patterns, e.g., bursty data transfers, because generally no application or network state information is used in computing routes.

An adaptive routing algorithm, on the other hand, adjusts to the state of the network, using this state information, for example network congestion, in making routing decisions. With its dynamic load balancing, adaptive routing should theoretically outperform oblivious routing. However, adaptive routing schemes typically face the difficult challenge of balancing adaptiveness with router complexity. To achieve the best performance through adaptivity, a router ideally needs global knowledge of the current network status. However, due to router speed and complexity, dynamically obtaining a global and instantaneous view of the network is often impractical. As a result, adaptive routing in practice relies primarily on local knowledge, which limits its effectiveness.

These routing algorithms can be further classified as *minimal* and *non-minimal*. Minimal routing schemes only select paths that are the shortest possible routes between source and destination pairs. In Figure 1-2, for example, a packet traveling from node L to node B has three minimal path choices (L, H, E, B); (L, F, A, B) and (L, F, E, B). Although for applications for which latency is critical and should be minimized, *minimal* routing is desirable, it may exhibit poor load balancing over a range of applications.

Non-minimal routing sacrifices locality for better load balancing. Since it is less constrained in its path selection, it generally leads to lower network congestion and overall better network throughput [32]. For a given flow, latency may increase.

1.5 Contributions

In this work, a bandwidth-sensitive oblivious routing (BSOR) scheme that statically determines routes considering an application's communication characteristics is developed and evaluated. The main challenge with any oblivious routing is a fair and an effective trade-off between load balancing and communication latency minimization. Here the proposed oblivious routing scheme uses a function corresponding to minimizing the maximum channel load (MCL) across all network links, while providing a mechanism for controlling the average path length. The core premise of this approach is to efficiently balance network

load while having shortest possible path lengths and keeping the router architecture simple and practical. To that end, the algorithm exploits knowledge of estimated bandwidths for all or a subset of data transfers in a given application, and focuses on optimizing satisfaction of bandwidth demand and latency of individual data transfers. This scheme is livelock and deadlock free and produces routes that can be minimal or non-minimal in an effort to globally optimize application throughput.

This scheme will be particularly suitable for long-running applications with predictable communication patterns. For example, the approach is suitable for co-processing platforms such as reconfigurable hardware, where processing elements and their interconnection network can be configured much like an FPGA to speed up a computationally-intensive task such as video compression, processor simulation, or rendering. In reconfigurable computing, a computation is spatially partitioned into processing elements (PEs) and the network traffic pattern remains relatively static as each PE performs a fixed task. Evaluations on synthetic traffic with various patterns and applications such as H.264 decoding and processor performance modeling even when the network traffic varies at run-time due to data dependent behaviors show throughput improvements over traditional oblivious routing.

1.6 Organization

The rest of this thesis is structured as follows. Chapter 2 begins with a review of widely used oblivious routing algorithms followed by a short descriptive summary of previously proposed application-specific routing algorithms.

Chapter 3 formally presents our bandwidth-sensitive oblivious routing algorithm, and its synthesis flow for both small and large size problems to ensure scalability of the algorithm. Chapter 4 explores the router architecture needed to support the proposed bandwidth sensitive approach by taking a typical virtual-channel router architecture, and showing the modifications required.

Chapter 5 presents the illustrating applications followed by Chapter 6 where the experimental results are evaluated.

We conclude in Chapter 7 and explain briefly our future research efforts on this topic.

Chapter 2

Background and Related Work

Routing algorithms for on-chip networks have been a subject of research in both academia and industry for decades because they are aimed at solving a problem with many conflicting aspects [48], such as minimizing the router logic while efficiently balancing the network data traffic load. Oblivious routing algorithms generally lead to simpler hardware, and are very popular in systems being currently deployed, but for the most part they suffer from lack of good load balancing. Application-specific routing algorithms in their attempt to solve this problem, have been done largely using adaptive routing frameworks which essentially leads to larger and more complex hardware. For a recent survey on network-on-chip in general see [49].

2.1 Oblivious Routing Algorithms

Oblivious routing algorithms are broadly classified into two main categories, deterministic and non-deterministic routing schemes. With deterministic routing, the same route is always taken for any given pair of nodes. Non-deterministic routing uses randomness to archive better path diversity.

2.1.1 Deterministic Routing Algorithms

The two widely used deterministic routing algorithms are *dimension order routing* and *destination-tag routing.*

Dimension order routing (DOR) algorithms [7] are vastly popular and have many desirable properties, for example they generate deadlock-free routes in mesh or hypercube topologies [1, 45]. Either using XY-ordered or YX-ordered routing, each packet is routed along one dimension in its first phase followed by the other dimension. Figure 2-1 shows examples of these routing schemes. The strength of these algorithms is their high compatibility with orthogonal topologies and the low level of hardware complexity needed to

support their implementations. DOR algorithms, even with their lack of path diversity and network load balancing, which lead to poor network bandwidth utilization and low performance throughput, are frequently used by designers to avoid additional hardware overhead. They are in fact used in many commercial and research products: Intel Paragon, Cray T3D, MIT J-machine and Stanford DASH all use some version of DOR as their routing algorithm [15, 27].

Destination-tag routing was initially designed by Lawrie [6] to allow concurrent, collision-free access to various data banks of a primary memory system by an array of processors. It later became very useful for routing packets in butterfly networks, where the digits representing the destination address are used by hops to determine the proper output port to which to forward a given packet [27]. Many research prototype and commercial computers use this routing scheme; BBN Butterfly, IBM RP3, NYU Ultracomputer, and NEC Cenju-3 all use some version of destination-tag routing.

2.1.2 Non-Deterministic Routing Algorithms

Valiant [8] and ROMM [17] have generally served as the representatives of this class of routing algorithms.

Valiant *et al* propose two different versions of an algorithm which routes a packet from its source to its destination in multi-phase fashion [8]. The routing algorithm selects at random an intermediate node and in the first stage of the routing the packet is sent from the source to the intermediate node, then in the second stage from the intermediate node to its official destination. Figure 2-2 shows an example of this Valiant algorithm. Although an arbitrary routing algorithm can be used for each of the phases, generally a simple DOR algorithm is used. It is shown to provide a good network load balance but can also lead to considerably longer path lengths. Towles *et al* further refine the Valiant algorithm to eliminate loops in the routes, and reduce the average path length by 20% [33].

ROMM routing which stands for randomized, oblivious, multiphase, minimal routing, was design by Nesson and Johnsson as an alternative to Valiant. In an effort to retain locality in routing of packets in the network, the intermediate node random selection is confined to a minimal *quadrant* [17] as illustrated in Figure 2-2. This approach essentially translates into randomly selecting between the various minimal paths from the source to the destination.

In *Orthogonal one-turn* routing (O1TURN routing) [46], which can be described as a restricted version of ROMM routing where the intermediate node is one of four corners of the minimum rectangle, Seo *et al* show that simply balancing traffic between XY and YX routing can guarantee provable worst-case throughput while reserving the same degree of router complexity.

2.2 Application-Specific Routing Algorithms

In this section we briefly survey several proposed routing algorithms that use targeted application information to provide higher performance in NoC architectures; see [49] for a recent more detailed survey. Application-specific routing schemes have generally approached the load balancing problem in two ways; either by designing algorithms to optimize performance for some subset of applications with little or no change to the targeted router architecture or by proposing algorithms that require their own router architecture and/or topology.

Palesi *et al* [52, 59] provide a framework and algorithms for application-specific bandwidth-aware deadlock-free *adaptive* routing. Given the communication graph of an application, cycles are eliminated from the *channel dependency graph* (CDG) to minimize the impact

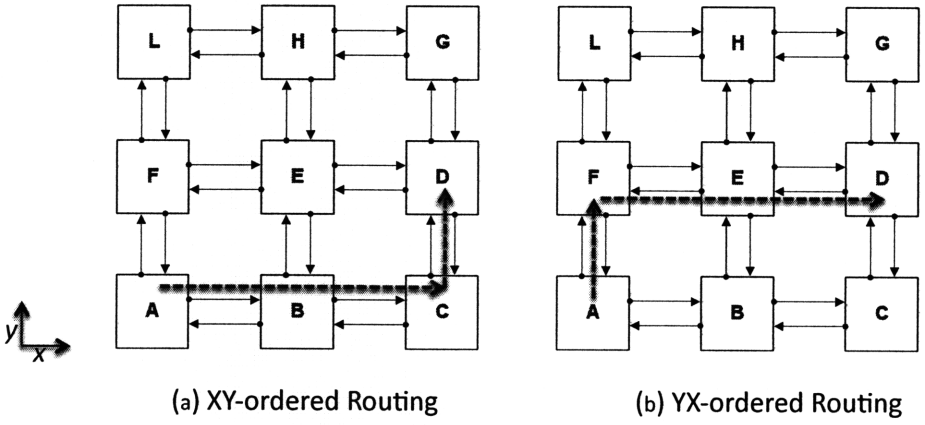


Figure 2-1: Example of Dimension Order Routing on 3 × 3 Mesh Network

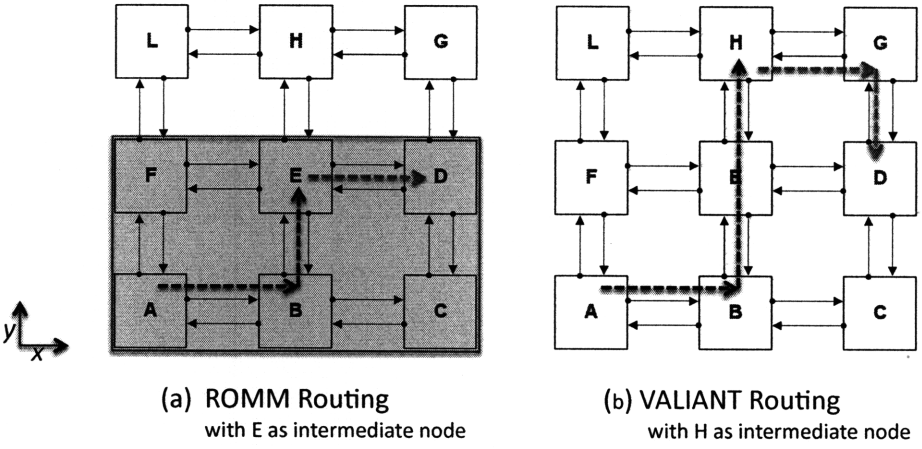


Figure 2-2: Example of ROMM and Valiant on 3 × 3 Mesh Network

on the average degree of adaptiveness. Bandwidth requirements are taken into account to spread traffic uniformly through the network. Their *Application Specific Routing Algorithms* (APSRAs) rely on the fact that the network router architecture is adaptive, and the designer's willingness to further increase the router area of the system.

Hansson *et al* [43] propose a unified approach to mapping and routing to minimize the network required to satisfy the constraints of the application. They show that overhead for this unified approach is only 20% higher than that of path selection alone in terms of run-time. The comparison is done against another approach [42] which generates network topology and mapping based on application specifications.

Many works on mapping of applications onto NoC architectures have also considered the routing problem during the NoC design phase (e.g., [31], [41], [50]).

Cho *et al* describe bandwidth-aware routing for diastolic arrays [57] and avoid deadlock by assuming that each flow has its own private virtual channel. Although this assumption simplifies deadlock avoidance in the routing algorithm, it may not be practical in cases where the number of flows traversing a router exceeds the number of allowed virtual channels.

Murali *et al* [40] present a tool for automatically selecting the best topology for a given application and producing a mapping of cores onto that topology. Although their SUNMAP algorithm supports different routing functions and takes into account bandwidth and area constraints, the direct impact of the routing functions on the overall performance of the network was not explored.

Srinivasan *et al* [47] using application specifications propose a slicing tree based floorplanner for the topology design. This technique does not insure deadlock-free routes which is crucial in packet-based NoC architectures.

Designing a network topology based of a application, though it has its advantages, presents the system designer with the choice of supporting only a small subset of applications that are part of the application class for which the topology is designed; or having different topologies for different classes of applications. The manufacturing cost of both of these choices, have made application-specific topologies highly unattractive in the larger domain of NoC architectures.

2.3 Buffer Space and Bandwidth Allocation

Several approaches to allocate buffer space and bandwidth in on-chip networks have been proposed. Most routing strategies group packets that need to be in-order at the receiver into a flow, and packets of a flow follow a single path (e.g., [12], [21]). Flows are then routed based on the routing algorithm adopted by the designer and this routing algorithm dictates the bandwidth and the buffer space allocation in the network.

2.3.1 Buffer Space Allocation

Buffers in NoC architectures generally occupy most of the physical area allocated to the router [29, 36]. So designers are forced to keep the number of buffers fairly small and make judicious use of that limited amount of buffering space. Dally's virtual channels [9] allocate buffer space for virtual channels in a way that is decoupled from bandwidth allocation. In his approach, each physical channel is associated with several small buffers, virtual channels, which compete with each other for the physical channel. This decoupling allows active flows to use network bandwidth more efficiently.

iWarp [64], is a system architecture for high-speed signal, image, and scientific computing, which implements virtual channels across single links. Its processing elements use these logical (virtual) channels to guarantee bandwidth to virtual circuits.

Hu and Marculescu, in their application-specific buffer allocation scheme, perform buffer allocation based on arrival rates [36]. More precisely, given the communication characteristics of an application and buffer space available in the network, their algorithm automatically assigns the buffer depth for each input channel, in different routers across the network.

Many other designs of virtual channel routers have been proposed. Nicopoulos *et al* have designed a dynamic virtual channel regulator called ViChaR, which dynamically allocates buffering resources depending on the network state [51]. Their approach aims at maximizing throughput by allocating virtual channels on demand. Mullin *et al* propose in [39] a router design for which the arbitration logic is removed from the the critical path in oder to improve the cycle-time. Bjerregaard and Sparso present two different non-blocking schemes in implementing virtual channels with minimal hardware overhead [35]. Kavaldjiev *et al* have proposed a 5-port virtual channel router architecture with simplified dynamic arbitration which allows fair and deterministic arbitration [38]. They claim that such an architecture reduces the area allocated to routing by 23% over an ASIC implementation and produces a speed improvement of 1.4X when compared to a conventional router.

Recently, express virtual channels have been proposed which skip routers along multiple-hop static paths to enhance performance in a dynamic routing scheme [28]. Support for multicast channels has also been proposed [58].

The partition of the router buffer space into virtual channels, either in a linked-list or disjoint forms, like those shown above, helps mitigate the *head-of-line blocking* issue that arises in NoC routing. Figure 2-3 shows how active flows can bypass blocked flows to use network bandwidth that would otherwise remain idle.

Our virtual channel router design is fairly standard and is described in Chapter 4. Our algorithms for static allocation assign flows to channels/lanes on a per-link basis, rather than assigning packets to a particular lane through the entire route.

2.3.2 Bandwidth Allocation

Bandwidth allocation to data flows in a given application is at the heart of the routing problem. An effective routing algorithm allocates bandwidth to flows in a way that balances the traffic loads across channels and provides application with a throughput close to the network ideal throughput. One approach is to formulate the bandwidth allocation problem as a linear programming problem.

Towles *et al* [34] give a multicommodity flow linear programming formulation for router algorithm design. When the linear program is optimized, deterministic algorithms that are worst case or average case optimal come out as solutions.

Racke uses concurrent multicommodity flow (CMCF) formulation to present his framework for solving on-line problems that aim to minimize the congestion in different networks [26]. Routing paths are selected according to the solution of the CMCF problem. His obliv-

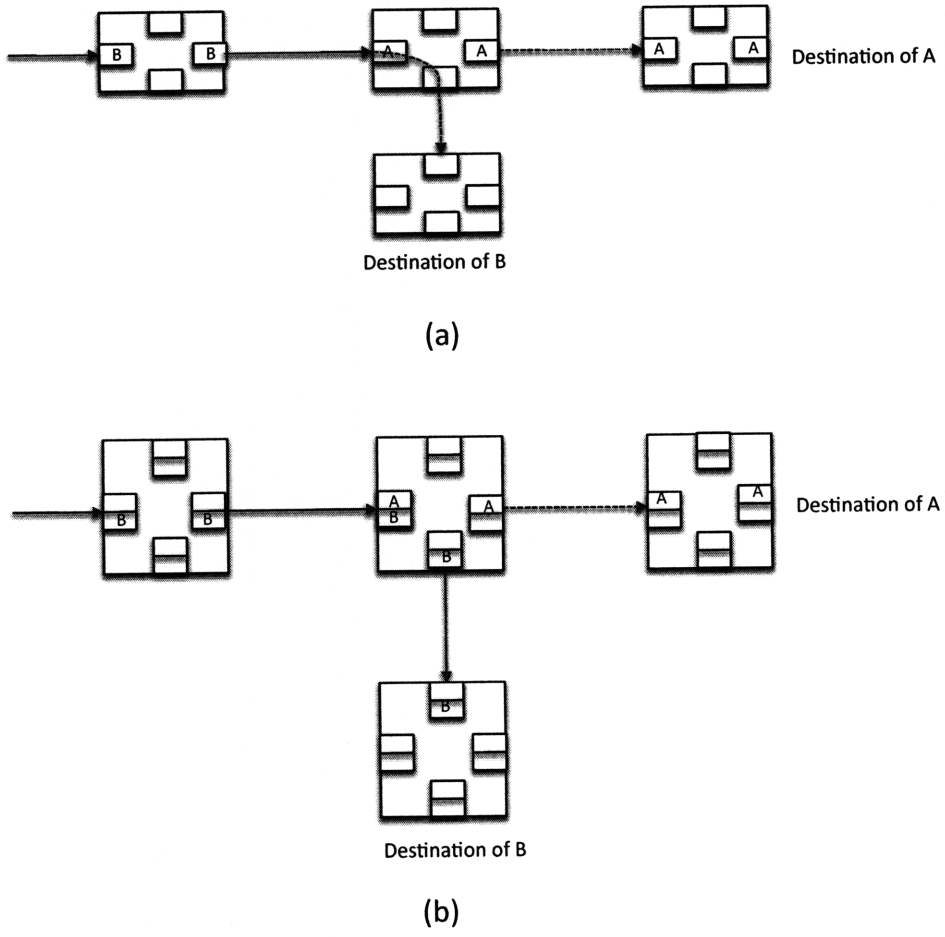


Figure 2-3: Virtual Channels: (a) packet B is blocked behind packet A. (b) Virtual Channels allow packet B to bypass blocked packet A.

ious path selection algorithm has a polylogarithmic competitive ratio in general networks.

Our goal is to find routes with maximal throughput for a *specific* application.

2.4 Adaptive Routing Algorithms

Classic adaptive routing schemes include the turn model routing methods [11] and odd even routing [22]. In [37] a hybrid scheme that switches between deterministic and adaptive modes depending on the application is presented, where local FIFO information is used to adapt routes. Duato (e.g., [4, 14]) gives necessary and sufficient conditions for adaptive routing in wormhole networks. Our algorithm is not adaptive; however, as described in chapter 3, we use the turn model to derive an acyclic channel dependence graph that drives our oblivious routing scheme. Our scheme can use cycle-breaking strategies other than using the turn model in the derivation of acyclic dependence graphs.

To summarize our framework is significantly different from previous work in its use of application specifications to efficiently balance network load while retaining an oblivious nature and an applicability to standard router architectures.

Chapter 3

Oblivious Routing with Bandwidth Sensitivity

The proposed *Bandwidth-Sensitive Oblivious Routing* (BSOR) algorithm exploits knowledge of estimated bandwidths for all or a subset of data transfers between modules for a given application in producing routes that globally optimize the application throughput. Although a two-dimensional mesh network topology is adopted in illustrating BSOR in this work, the algorithm is independent of both network topology and number of virtual channels per link. To present BSOR, the routing problem is formulated as a *multicommodity-flow* problem and the following standard definitions of flow networks and channel dependence graphs are used.

3.1 Definitions

Definition 1. *Given a flow graph $G(V, E)$, where an edge $(u, v) \in E$ has capacity $c(u, v)$. The capacities $c(u, v)$ are the available bandwidths on the edge. There is a set of k data transfers or flows $K = \{K_1, K_2, \ldots, K_k\}$. $K_i = (s_i, t_i, d_i)$, where s_i and t_i are the source and sink, respectively, for connection i, and d_i is the demand. We assume $s_i \neq t_i$. We may have multiple flows with the same source and destination. The flow variable i along edge (u, v) is $f_i(u, v)$. A route is a path p_i from s_i to t_i for a flow i. Edges along this path will have $f_i(u, v) > 0$, other edges will have $f_i(u, v) = 0$.*

If $f_i(u, v) > 0$, then route p_i will use both bandwidth and buffer space on the edge (u, v). The value of $f_i(u, v)$ indicates the amount of bandwidth allocated to flow i on the edge. In the case of single path flows $f_i(u, v)$ is equal to d_i. The buffer resource may be a packet buffer in the case of packet-buffer flow control, or a virtual channel in the case of flit-buffer flow control. We will assume flit-buffer flow control in this work, although our framework can be applied to other flow control schemes as well.

Definition 2. *A channel dependence graph (CDG) $D(V', E')$ is derived from the flow net-*

work G as follows. Each vertex in V' is an edge in G. There is an edge from $v_1 \in V'$ to $v_2 \in V'$ if a packet can flow from the edge in G associated with v_1 into the edge associated with v_2, without traversing any other edges. That is, the edges are consecutive in G. We are disallowing 180-degree turns in routing and will later remove these edges.

Figure 3-1 shows the CDG associated with the 3×3 mesh network in Figure 1-2 . BC and CB are edges in opposite directions from B to C and C to B, respectively. They correspond to separate vertices in the CDG and are not connected when 180-degree turns are not allowed. Note that the CDG has cycles, for example, there are edges connecting DG to DH, DH to HE, HE to ED and ED to DG.

Definition 3. *The maximum channel load (MCL) U in a network is defined as*

$$U = \max_{(u,v)\in E} \sum_{i=1}^{k} f_i(u,v) \tag{3.1}$$

It denotes the channel with the highest load which is the bottleneck channel in the entire network and determines the saturation throughput of the system.

Definition 4. *For a given set of k data transfers or flows $K = \{K_1, K_2, \ldots, K_k\}$, let S_F be the selector function that chooses the path p_i taken by packets of flow i from s_i to t_i through the network.*

We define *load balancing* to be the degree to which resources in term of bandwidth and buffer space are uniformly utilized across the different links of the network; and latency as the required time or number of hops to router a packet from its source to the destination.

3.2 BSOR Framework

The BSOR algorithm uses the estimated bandwidths of an application and the targeted network topology and resource information to make advance plans to provide *load balancing*

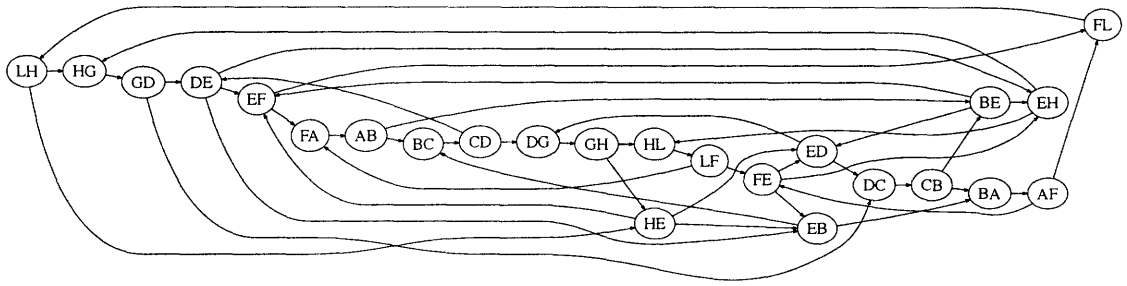


Figure 3-1: Channel Dependence Graph for the Mesh Network of Figure 1-2

during run-time of the application. It follows the framework outlined below; and many different bandwidth-sentive algorithms can be constructed based on the framework depending on the selector function S_F. For this work, we detail two instantiations of the framework, one using Mixed Integer Linear Programming (MILP) for small and medium size problems and the other using Dijkstra's weighted shortest path algorithm, for large size problems.

FRAMEWORK(Data Transfers K)

1. *Create an acyclic channel dependence graph (CDG) by deleting edges from D; call it D_A.*
2. *Transform D_A into a flow network G_A.*
3. *Choose routes p_i for each flow i in G_A, taking into account bandwidth availability using an S_F.*
4. *If desired, go to Step 1 to create a different acyclic CDG and repeat.*
5. *Select the best set of routes found.*

Offline Bandwidth-Sensitive Oblivious Routing Framework

This framework assumes that the underlying network has been made deadlock free. Deadlock in NoCs occurs when two or more flows are each waiting for the other to release link or buffer space in effort to finish routing a packet to its destination. Although there are techniques for recovering from a deadlock in these systems, performance can suffer a great deal and hardware complexity increases drastically. The usage of the acyclic channel dependence graph in step one ensures the deadlock freedom property of the routing algorithm.

Lemma 1. *A routing algorithm R is deadlock-free if and only if the set of routes it produces forms an acyclic channel dependence graph (CDG_A) [1].*

Dally and Aoki in [10] give the formal proof for *Lemma 1.*

3.3 Creating Acyclic Channel Dependence Graphs

According to *Lemma 1*, if packets follow routes that conform to an acyclic channel dependence graph, then deadlock will not occur. This is also a necessary condition provided false resource dependences do not exist [20].

Therefore, routing of packets is systematically restricted by breaking all the cycles in the CDG D associated with the network. There are many ways to remove these cycles; the *turn model* [11] provides a few systematic approaches. Figure 3-2 illustrates some of these approaches where the dotted segments represent the prohibited turns.[1]

For the 3×3 mesh network from Figure 1-2, the two acyclic CDGs derived from D

[1]Note that the turn model was developed to enable adaptive routing; here we are concerned with choosing routes in an offline fashion for oblivious routing.

using the *north-last* and *west-first* turn models shown in Figure 3-2 to break cycles are exhibited by Figure 3-3.

Cycles can also be broken in an *ad hoc* or random fashion as shown in Figure 3-4. Typically, a larger number of dependences need to be removed to obtain an acyclic CDG but after route selection under this type of CDG, we may obtain a better result (cf. Section

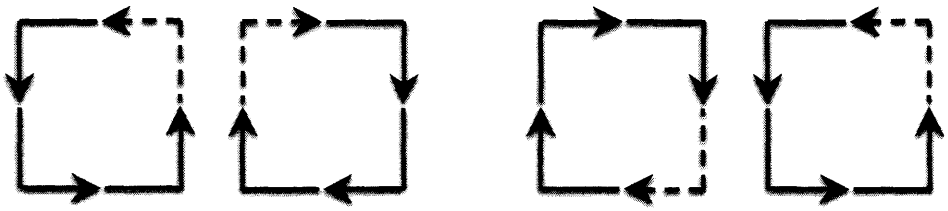


Figure 3-2: Two Turns Prohibited by the *turn model.* (a) North-Last turn. (b) West-First turn.

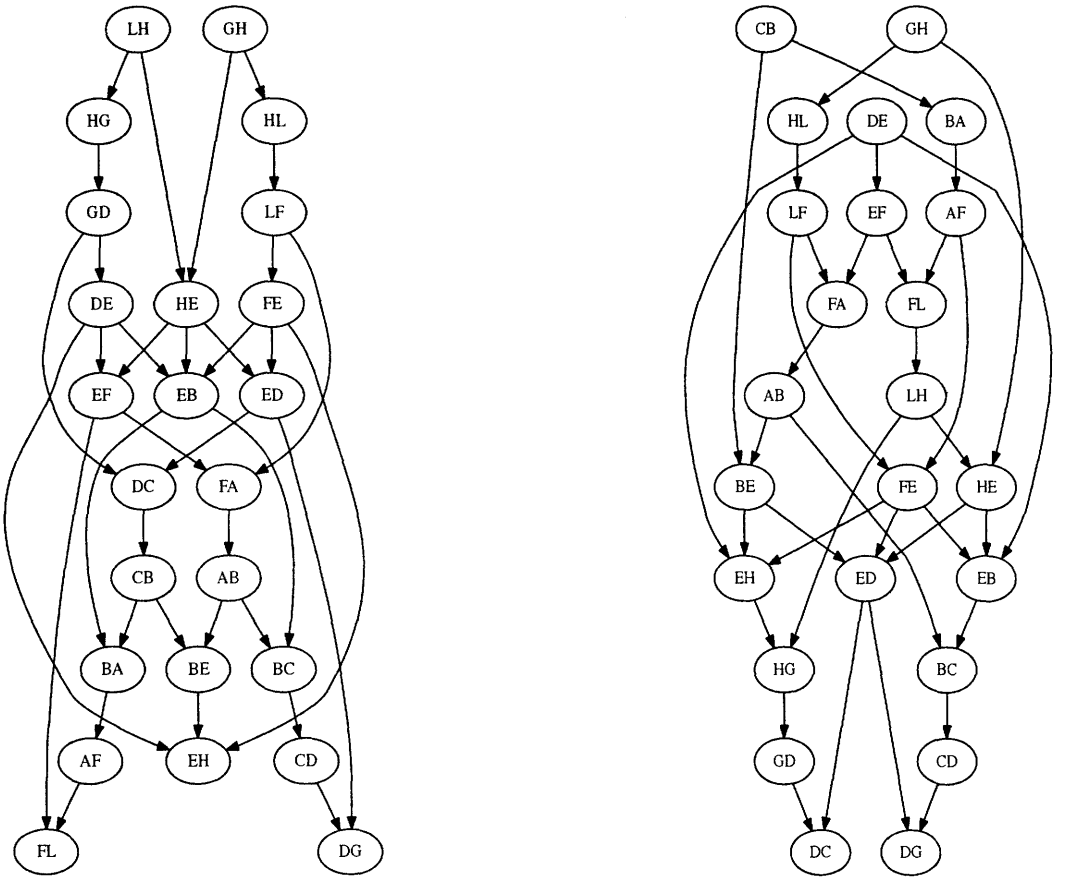


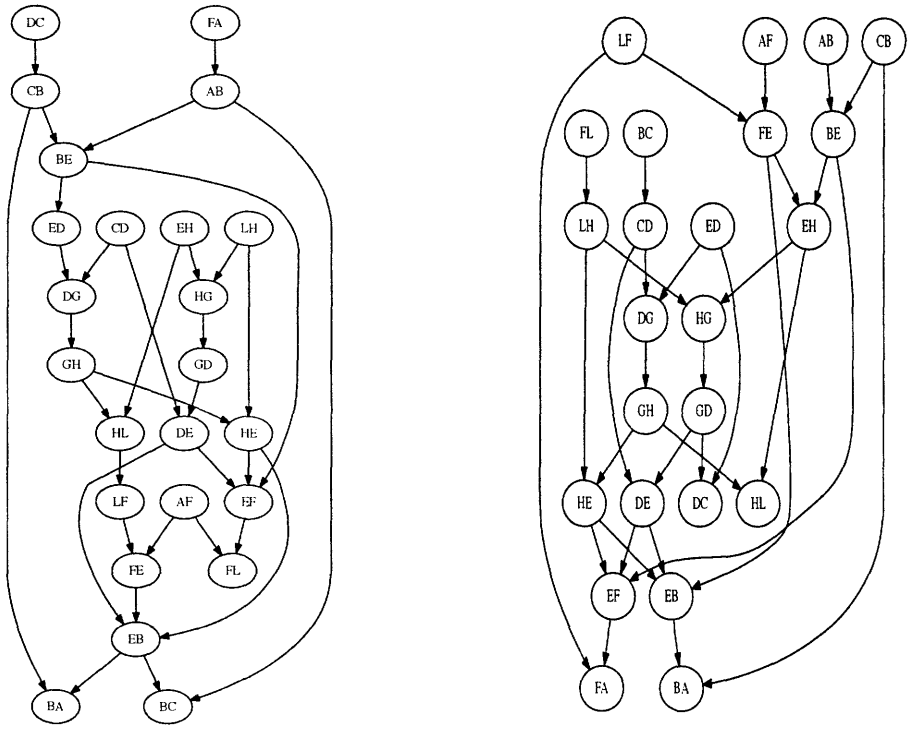
Figure 3-3: Acyclic CDG Based of North-Last and West-First Prohibited Turns.

6). In both of the cases presented in Figure 3-4, 12 edges needed to be removed from the original CDG with no 180-degree turns, as opposed to 8 in the turn model. We can use *any* acyclic CDG to drive our bandwidth-sensitive oblivious routing algorithm. Given that different CDG's may result in different qualities of routes, we can perform route selection under many different CDG's and select the best result. We next show how to derive a flow network from an acyclic CDG so the routes generated are guaranteed to be deadlock-free.

3.4 Deriving a Flow Graph from an Acyclic CDG

Given source and destination network nodes s_i and t_i respectively, for each flow i, we will derive a flow graph or network G_A from an acyclic CDG D_A. We will then run our route selection algorithm on G_A, to find the "best" routes for the flows. This will have the effect of running route selection on the given flow network G (corresponding to the interconnection network) but with the route conforming to D_A. If the routes for all flows conform to D_A, deadlock freedom is assured.

Note that G corresponds to the original on-chip network, whereas G_A corresponds to a flow network where the vertices are links of the original network, and the edges are



(a) (b)

Figure 3-4: (a) Acyclic CDG: 12 edges removed (b) Acyclic CDG: 12 edges removed

dependences. We next focus on the route selection step.

G_A is derived from D_A as follows. D_A is copied over to G_A. Add vertices to G_A corresponding to s_i and t_i, for each i. Add edges from s_i to all vertices in G_A that have s_i as the source node of the corresponding link. For example, if s_i is network node A in the 3×3 mesh network shown in Figure 1-2, then add edges from s_i to AB and AF. For each vertex in G_A that has t_i as the destination node of the corresponding link, add an edge from the vertex to t_i. For example, if t_i is network node I in the 3×3 mesh mesh shown in Figure 1-2, then add edges from FL to t_i and from HL to t_i.

Figure 3-5 shows a flow network derived from the acyclic CDG of Figure 3-4(a), given the source-destination pair A, L. The weights on the edges are assigned randomly for illustration; their generation and utility will be described at a later stage. Other source-destination pairs can be added to G_A in a similar fashion. Each link in G_A will have an initial capacity and its residual capacity will change through the course of route selection. Both the MILP and the Dijkstra-based algorithm we use assume weights/capacities on edges. Each edge in G_A^i has a capacity or residual capacity associated with the link vertex that it is incident on.

In this thesis we present two route selection schemes; an optimal route selector for some cost functions using mixed integer-linear programming. Solvers such as CPLEX are able to find optimal routes within a reasonable amount of time for moderate-sized examples, but not for large examples since the convergence time may be very long. To address this, we will also propose a heuristic algorithm for route selection based on Dijkstra's weighted shortest path algorithm [63].

3.5 Mixed Integer-Linear Programming Selector

Bandwidth allocation given the rate demands for each of the connections can be viewed as a multicommodity flow problem which can be optimally solved in polynomial time using linear programming (LP) [63]. The routes produced, using linear programming, are not by default deadlock free and can lead to splitting of flows. The mixed integer-linear programming (MILP) formulation below can produce an optimal result either minimizing maximum channel load, or maximizing throughput, therefore providing a way of selecting best routes for moderate-sized problems.

MILP: Find an assignment of flow in G_A, i.e., $\forall i,\ \forall (u,v) \in E\ \ f_i(u,v) \geq 0$, which satisfies the constraints:

Capacity constraints :

$$\forall (u,v) \in E\ \ \sum_{i=1}^{k} f_i(u,v)\ \leq\ c(u,v)$$

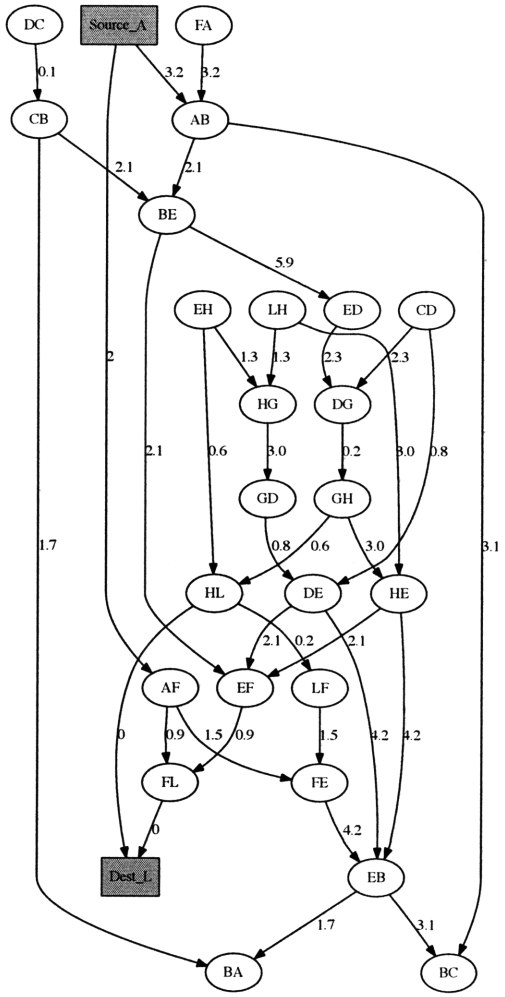


Figure 3-5: Flow network from acyclic CDG of Figure 3-4 with source-destination pair A, L and example weights.

Flow conservation :

$$\forall i,\ \forall u \neq s_i,\ t_i \quad \sum_{(w,u)\in E} f_i(w,u) = \sum_{(u,w)\in E} f_i(u,w)$$

$$\forall i \quad \sum_{(s_i,w)\in E} f_i(s_i,w) \ = \ \sum_{(w,t_i)\in E} f_i(w,t_i) \ = \ g_i$$

Unsplittable flow :

$$\forall i,\ \forall (u,v) \in E \quad f_i(u,v) \leq b_i(u,v) \cdot d_i$$

$$\forall i,\ \forall u \quad \sum_{(u,v)\in E} b_i(u,v) \leq 1$$

Hop Count :

$$\forall i \quad \sum_{(u,v)\in E} b_i(u,v) \leq hop_i$$

and minimizes the maximum channel load:

$$minimize \;\; U \;\; = \;\; \max_{(u,v)\in E} \sum_{i=1}^{k} f_i(u,v) \tag{3.2}$$

or maximizes the total throughput, given as

$$maximize \;\; S \;\; = \;\; \sum_{i=1}^{k} g_i \tag{3.3}$$

or maximizes the minimal fraction of the flow of each commodity to its demand:

$$maximize \;\; T \;\; = \;\; \min_{1\leq i\leq k} \frac{g_i}{d_i} \tag{3.4}$$

The variables $b_i(u,v)$ are Boolean variables, i.e., they can take on values of 0 or 1 only. They enforce the restriction that a flow i can only take a single path from source s_i to destination t_i. They also enforce path length restrictions. hop_i is a specified constant that can be set to be equal to the minimal path length between s_i and t_i. This will imply that only minimal paths will be considered. hop_i should be incremented by 2 or more to allow for non-minimal routing. The $f_i(u,v)$ variables can take on any positive value less than or equal to the demand d_i. Thus, we have a mixed integer-linear program, which if solved, finds the "best" set of routes, while ensuring unsplittable flows that conform to D_A and are therefore deadlock-free.

3.6 Dijkstra Weighted Shortest Path Selector

Since the unsplittable flow problem is NP-hard even for single sources [19], MILP which is an approximation algorithm may not converge quickly enough to explore a good range of cycle breaking schemes and weight functions for large size problems. Therefore we present a heuristic for dealing with such cases, it consists of running Dijkstra on a weighted version of G_A, deriving weights from the residual capacities of each link/vertex. Consider a link e in the original network G (e.g., AB) which is a vertex in G_A. This link has a capacity $c(e)$. We have a variable for each link e, called $\tilde{c}(e)$, which is the current residual capacity of link e. Initially, it is equal to the capacity $c(e)$. If a flow i is routed through this link e, we will subtract the demand d_i from the residual capacity $\tilde{c}(e)$.

We have experimented with various metrics, and have selected the reciprocal of link residual capacity which is similar to the CSPF metric described by Walkowiak [54]. The

weight function we use is $w(e) = \frac{1}{\tilde{c}(e)-d_i+M}$. $\tilde{c}(e)$ is the current residual capacity that is decremented by d_i if the flow goes through e. $\tilde{c}(e)$ may become negative when demands are higher than link bandwidths. M is a constant comparable to the maximum link bandwidth, large enough to ensure that the weights $w(e)$ remain positive.

Dijkstra assumes weights on edges in G_A; however, the links are vertices in G_A. The weight of an edge in G_A is merely the weight of the link/vertex that the edge is incident on. For example, the edge from s_i to AB will be assigned the weight of link/vertex AB. An edge from AB to BC will be assigned the weight associated with link/vertex BC. The edges incident on t_i are always assigned a weight of 0. Figure 3-5 shows a flow network derived from the acyclic CDG of Figure 3-4(a), assuming the source is network node A and the destination is network node L, with weights assigned to edges. We run Dijkstra on the weighted G_A to find a minimum weight path from A to L, or in general from an s_i to a t_i. Then, the weights are updated, and a new source-destination pair is selected to be routed. This continues until all the flows have been routed.

This Dijkstra weighted shortest path based heuristic can be run on thousands of nodes within seconds. The underlying Dijkstra algorithm has polynomial-time complexity of $O(flows * (E + VlogV))$.

This strategy results in a bandwidth allocation that tries to distribute traffic uniformly through the network, minimizing the maximum channel load. The length of the paths are minimized in a secondary fashion, since the weight of a path is the sum of the weights of the links. Increasing M gives more weight to minimizing the number of hops in each path, therefore providing a mechanism, like in the MILP based approach, to generate only minimal length routes for some applications where latency in terms of number of hops should be minimized.

3.7 Multiple Virtual Channels

Channel dependence graph representations of networks are expandable to networks with multiple *virtual channels* (VCs) per physical channel. Having multiple VCs per link does affect the deadlock properties of the network. Since resources in the network are no longer physical links but buffer spaces.

If there are z virtual channels per link in the network , then we expand the CDG D to include z vertices for each link, with each vertex corresponding to a virtual channel. There are edges between vertices if the corresponding links can be consecutively traversed by a packet. Since a packet can switch virtual channels, we will have z^2 edges between the sets of vertices that correspond to consecutive links.

The CDG D for the 2×2 sub-mesh (with nodes F, E, A and B in the 3×3 mesh of Figure 1-2) for $z = 2$ is shown in Figure 3-6 (a).

As before, we can break cycles in D by removing edges using a strategy based on a turn model, or using *ad hoc* or random strategies. Figure 3-6(b) shows an acyclic CDG that was derived using a turn model.

We have additional flexibility with multiple virtual channels; all turns are allowed provided the route switches virtual channels as shown in the acyclic CDG D_A of Figure 3-6(c). Either (or both) of these acyclic CDGs can be used to generate the flow network G_A by adding source and destination nodes as before. When a path p_i is selected in G_A, it implies a static allocation of virtual channels along the route. A best set of routes, i.e., the set with smaller maximum channel load, can be chosen across different acyclic CDGs. Therefore, static allocation of virtual channels gives additional flexibility in route selection. Shim *et al* show in [53] that this type of static allocation can match or exceed the performance of dynamic allocation schemes.

To evenly distribute flows across virtual channels, we modify the weight function slightly to include the number of flows that use a virtual channel. The weights of edges in G_A^i incident on AB0 or AB1 will be different if AB0 has been assigned to a flow, and AB1 has not.

Another approach, when we have a network with multiple VCs per physical channel, is to represent the network as multiple virtual networks. Each virtual network contains one or more virtual channels from the original network and it is represented by its own CDG (V_{CDG}). Cycles in a V_{CDG} are eliminated using either a *turn* or *ad hoc model* independently of other V_{CDG}s. Given source and destination network nodes s_i and t_i respectively, for each flow i, we will connect s_i and t_i to all the acyclic V_{CDG}s. Figure 3-7 shows a flow network derived from the 3×3 mesh network in Figure 1-2 with two virtual channels per link. Each virtual network has one virtual channel and the two virtual networks are shown

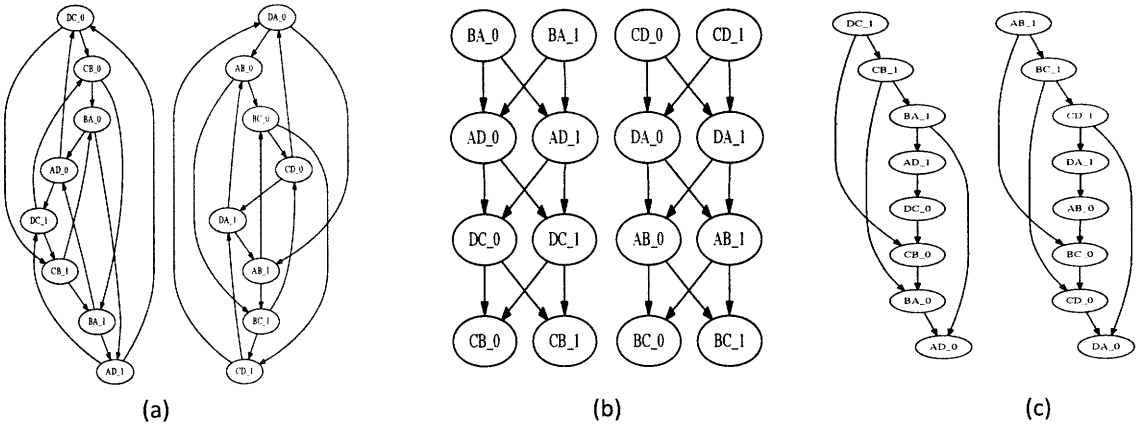


Figure 3-6: (a) CDG for 2×2 sub-mesh FEAB with 2 virtual channels (b) Acyclic CDG using the turn model. (c) Ad hoc Acyclic CDG.

in Figures 3-3(a) and 3-4(a). For illustration purposes, we have the source-destination pairs $s_1 = G,\ d_1 = L$.

Our routing scheme, by exploring virtual channel allocation based on application static information, helps prevent performance degradation associated with a single flow consuming multiple virtual channels and blocking other flows, essentially eliminating the resource coupling created by the addition of buffers to the router architecture.

Both minimal and non-minimal paths can be selected in a bandwidth-sensitive manner

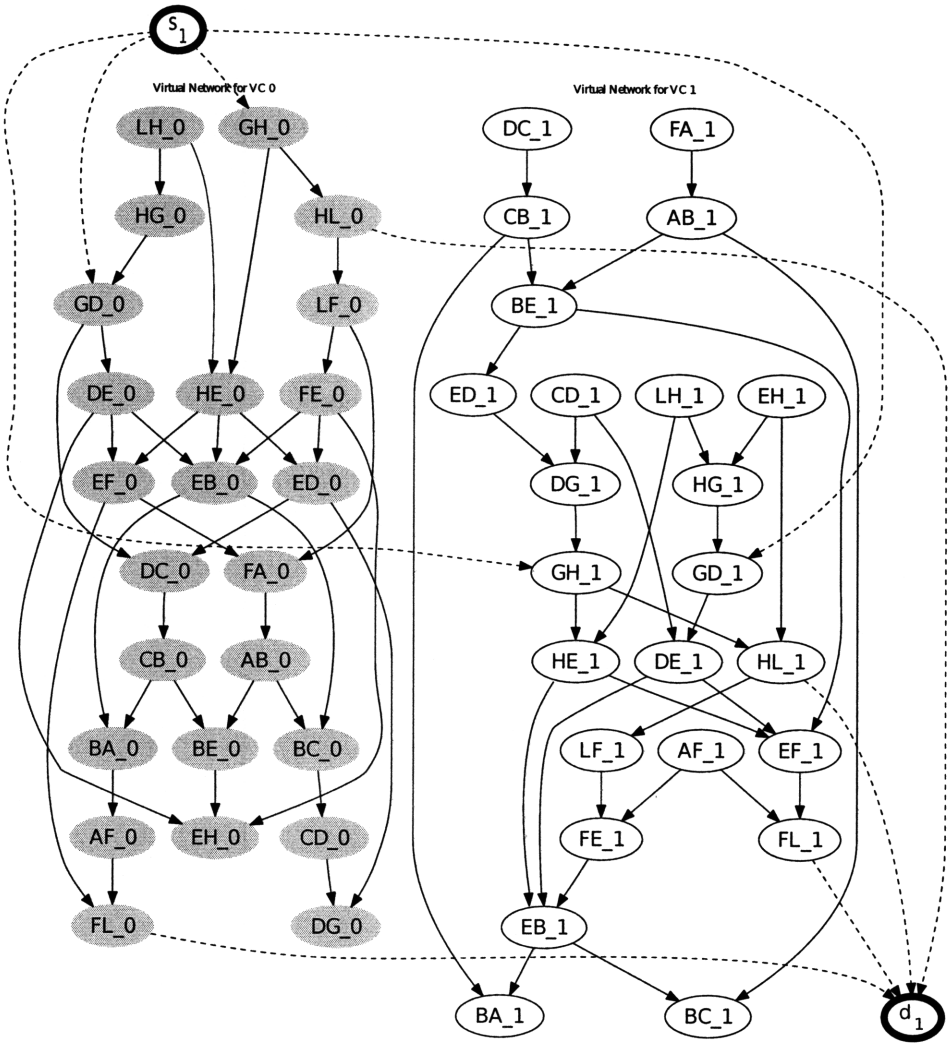


Figure 3-7: Acyclic Virtual Networks for Multiple Virtual Channels

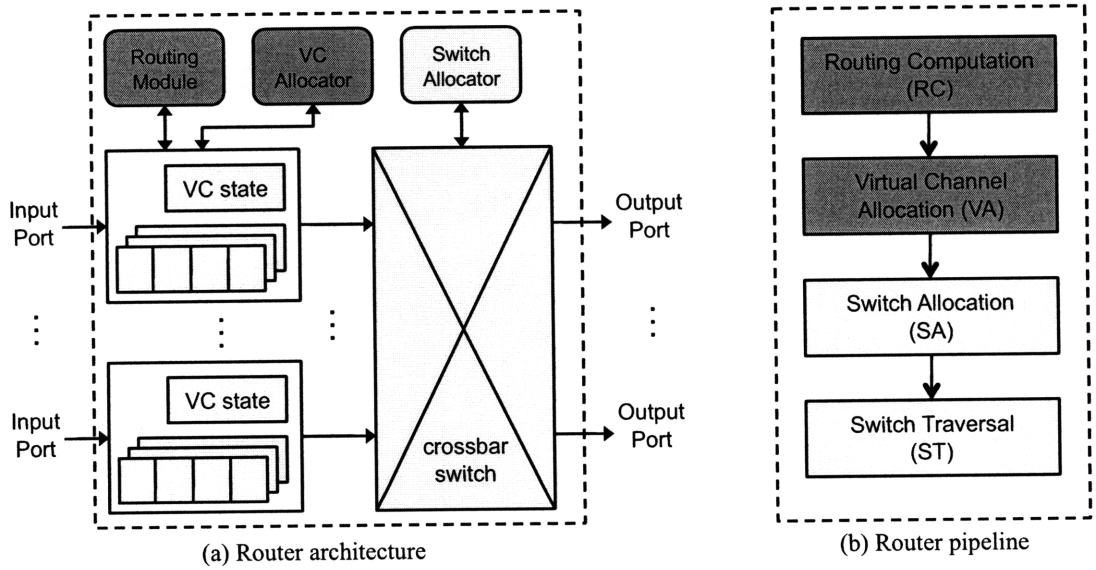
in our framework, while ensuring that deadlock does not occur. Hardware restrictions such as limiting the number of flows through a link can be enforced when searching for a new route. Many different acyclic CDGs and cost functions can be used in an effort to obtain the best performance as determined by a simulator or router hardware. Packet routes can be determined in different orders. Finally, other route selectors can be plugged into the framework rather than using MILP or Dijkstra as long as required deadlock-avoidance checks on the set of routes are made.

Chapter 4

Router Architecture

This chapter discusses the impact of our oblivious routing technique on the router architecture, and compares the modified architecture with standard routers for other oblivious routing algorithms. The following discussion assumes a typical virtual-channel router on a two-dimentional mesh network as a baseline. However, as previously noted the proposed routing technique is largely independent of network topology and flow control mechanisms. Therefore, the same approach to routing can be applied to other network topologies and either packet-buffer or flit-buffer flow control.

4.1 Typical Virtual Channel Router



(a) Router architecture

(b) Router pipeline

Figure 4-1: Typical virtual-channel router architecture. The dark blue indicates that the modules and pipeline stages may be modified for our approach.

Routing algorithm	Routing mechanics	VC allocation
DOR, ROMM, etc.	Algorithmic: fixed logic	Dynamic
BSOR / No Cycle (NC)	Table-based: source or node-table routing	Dynamic or Static

Table 4.1: Router architecture designs for routing algorithms.

Figure 4-1 illustrates a typical virtual-channel router architecture and its operation [24, 27, 39]. As shown in the figure, the datapath of the router consists of buffers and a switch. The input buffers store flits while they are waiting to be forwarded to the next hop. There are often multiple input buffers for each physical channel so that flits can flow as if there are multiple "virtual" channels. When a flit is ready to move, the switch connects an input buffer to an appropriate output channel. To control the datapath, the router also contains three major control modules: a router, a virtual-channel (VC) allocator, and a switch allocator. These control modules determine the next hop, the next virtual channel, and when a switch is available for each packet/flit.

The routing operation takes four steps, namely routing (RC), virtual-channel allocation (VA), switch allocation (SA), and switch traversal (ST), which often represent four pipeline stages in modern virtual-channel routers. When a head flit (the first flit of a packet) arrives at an input channel, the router stores the flit in the buffer for the allocated virtual channel and determines the next hop for the packet (RC stage). Given the next hop, the router then allocates a virtual channel in the next hop (VA stage). Finally, the flit competes for a switch (SA stage) if the next hop can accept the flit, and moves to the output port (ST stage).

For existing oblivious routing algorithms such as Dimension Ordered Routing (DOR) [1], ROMM [16], Valiant [8], and o1turn [46], the next hop of a packet can be easily computed at each router node based on the packet's destination. Moreover, these algorithms are fixed and commonly used for all types of applications and traffic patterns. As a result, traditional oblivious routing algorithms are implemented as dedicated logic in the RC stage of each router. For these routing algorithms, the RC stage is quite simple and the router frequency is typically dominated by the VA stage [24].

4.2 Router Architecture for Bandwidth-Sensitive Oblivious Routing (BSOR)

The router architecture for the proposed oblivious routing scheme is almost identical to the typical virtual-channel router architecture. The router uses the exact datapath that is described above. The only change in our routing architecture is in its routing module, which is summarized in Table 4.1.

For simple oblivious routing algorithms such as DOR, the baseline architecture implements the algorithm with fixed logic and dynamically allocates virtual channels to a packet. To support our routing scheme with any algorithm variant, our routing module needs table-based routing so that routes can be configured for each application. This single change is sufficient because our routing algorithm ensures that there is no cyclic dependence in routes (MILP-NC).

We next discuss the details of this modification. The routing algorithm is described in Chapter 3.

4.2.1 Programmable Routing

Our routing technique determines the routes for each flow based on an application's bandwidth requirements as well as its source and destination nodes. Additionally, to maximize the throughput, our routing algorithm can utilize any path from the source to the destination; routes may be either minimal or non-minimal. Therefore, the router must be *programmable* so that the routes for each flow can be configured depending on the application, and be *flexible* enough to support arbitrary routing paths.

In order to provide programmability and the flexibility, our router uses *table-based* routing where the path between a pair of nodes is stored in a routing table. Unlike cases where a simple routing algorithm is hardwired with fixed logic (algorithmic routing), the routing table can be simply re-programmed with new routes before an execution of a new application in order to update the routing. The table-based approach also allows our routing algorithm to select almost any path from a source to a destination as long as the route can fit into the table.

Table-based routing can be realized in two different ways: source routing and node-table routing, and our routing technique can also be implemented in both styles. In the *source routing* approach, each node has a routing table that contains a route from itself to each destination node in the network. The routes are pre-computed by our routing algorithm and programmed into the tables before the execution of an application. When sending a packet, the node prepends this routing information to the packet. Routers along the path can determine the output port simply by looking up the routing flits. Figure 4-2 (a) illustrates source routing where a packet is routed through node A, B, and C. The route corresponds to East, North, and North, which is reflected in the routing flits in the packet.

The source routing approach simplifies the router design because the routing stage (RC stage) in the router pipeline now only needs to read the output port from the flit without any computation. Effectively, source routing eliminates the routing step in the router and can potentially reduce the number of pipeline stages. In fact, thanks to its speed and simplicity, source routing has been widely used in many router designs including the IBM SP1 [13] and SP2 [12] and the Avici TSR [21]. On the other hand, source routing results in

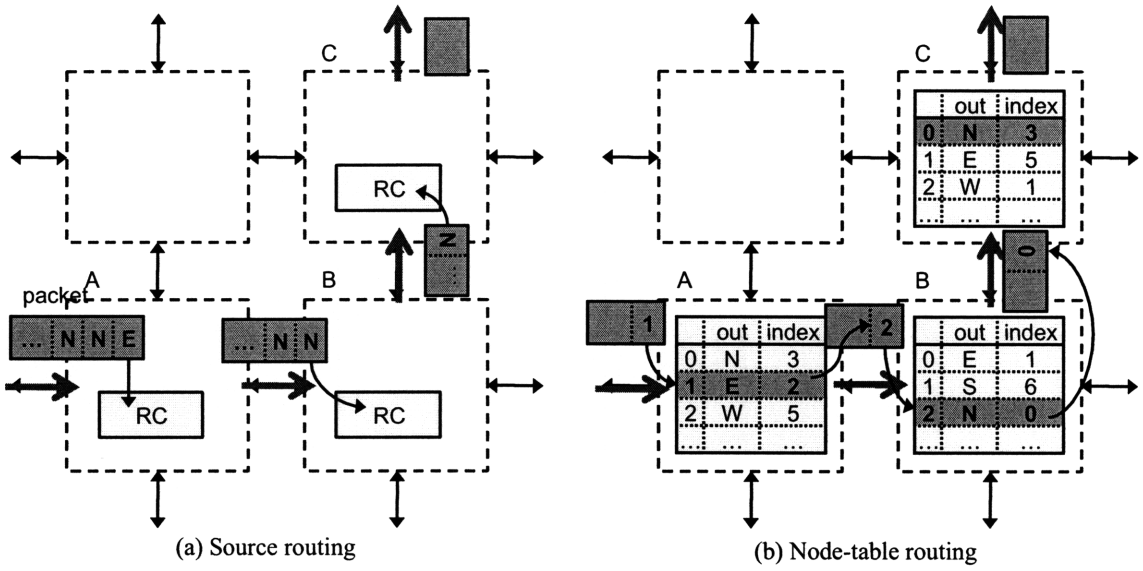


Figure 4-2: The table-based routing architecture. (a) Source routing. (b) Node-table routing.

longer packets containing routing flits as compared to the case where the route is computed for each hop.

Instead of carrying the entire route with every packet, the nodes along the path can be programmed with routing information for relevant flows. In this *node-table* routing approach, the routing module of a node contains a routing table that has the output port for each flow that is routed through the node. To determine which table entry corresponds to each packet, the packet carries an index field for the current node and the routing table provides the new index for the next hop. To set up the route, our routing algorithm computes a route for each flow and configures the routing tables accordingly. Upon receiving a packet, a router reads its routing table to determine the proper output port and forwards the packet with the new index field from the table.

Figure 4-2 (b) shows an example of the node-table routing when a packet is routed through the same path with the source routing example. As shown in the figure, the incoming packet to node A contains the table index of 1. To route this packet to B (East), the entry (1) in A's routing table is set as (East, 2), indicating that the packet should be routed to East with the new index of 2. In the same way, the packet looks up the second entry in node B for routing.

The router architecture for node-table routing essentially replaces the fixed logic in the RC stage of the baseline router with a table look-up. While the table look-up can take longer than evaluating the routing logic for simple deterministic routing such as DOR, it will not change how fast a router can operate because the router's clock frequency is most often determined by other routing stages or external factors such as a processor's frequency.

A previous study shows that the latency of a pipelined virtual-channel router is dominated by virtual channel allocation, which takes 15-20 FO4 [24]. Even if we conservatively assume that each routing table has 256 entries (256 flows), the table only takes a couple of KB; an entry needs 2 bits to represent the output port in a 2-D mesh and 8 bits for the next table index (256 entries). Therefore, a routing table will be easily accessible within a single cycle without impacting the clock frequency.

In practice, both table-based routing techniques place a restriction on the maximum number of flows that can be supported depending on the size of a routing table. In source routing, flows with an identical source-destination pair will have to share the same route unless the routing table has multiple entries for each destination. Similarly, in node-table routing, the size of each routing table limits the number of flows that can be routed through a node. Our routing algorithm can include restrictions enforced by the router hardware.

We have described two routing module designs, namely source routing and node-table routing, that can support bandwidth-sensitive oblivious routing. Both routing methods are widely known and have been implemented in multiple routers [12, 13, 18, 21]. In other words, the proposed routing approach can be realized with standard routing hardware without new specialized mechanisms. Also, our routing approach will not have noticeable impact on the latency or the organization of the router pipeline. The only overhead of our routing technique compared to other oblivious routing schemes is the addition of routing flits when source routing is chosen.

4.2.2 Static Virtual-Channel Allocation

Our bandwidth sensitive routing framework supports multiple virtual channels. It allows virtual channels to be statically allocated by the routing algorithm. The static allocation of virtual channels allows the algorithm to choose more diverse routes compared to the dynamic allocation case by enabling finer deadlock analysis, and also simplifies the VC allocation stage of a router, which is the highest latency step [24]. On the other hand, static allocation may result in worse utilization of available virtual channels because it does not consider dynamic behavior.

Chapter 5

Applications

To test the efficiency our oblivious deadlock-free routing algorithm, we explore both synthetic benchmarks and flows from real applications. The synthetic benchmarks are useful for evaluating specific communication patterns, while real applications allow us to test for more general and less well-behaved traffic patterns.

5.1 Synthetic Benchmarks

The term *synthetic* simply implies a level of abstraction where, based on well known computation tasks graphs, a systematic approach for deriving flows, in this case source and destination pairs, has been established for a given topology.

5.1.1 Bit-Complement

The Bit-complement traffic pattern arises when performing operations such as vector reversals or distributed matrix multiplications. Faster matrix multiplication provides more efficient algorithms for many standard linear algebra problems. Matrix multiplication in distributed fashion are particularly important, since known methods for performing such multiplication in a non-distributed manner require $O(n^e)$ run-time where $e > 2$. Bit-complement has a very symmetric communication behavior and the source and destination pair is simply given by: $d_i = \neg s_i$.

Where s_i denotes the i^{th} bit of the source address and d_i denotes the i^{th} bit of the destination address.

5.1.2 Transpose

The transpose traffic pattern is directly related to matrix transpose and corner-turn operations. Corner-turn operations, for example, are very useful in preserving a certain amount of data locality when dealing with signal and image processing applications which operate

on multi-dimensional data. The ideal computational platform for these types of applications has been System-on-chips, since they can be easily distributed across the multiple processing elements to exploit data parallelism.

The source and destination pair in transpose is generated by: $d_i = s_{i+b/2 \mod b}$.

The bit length of an address is $b = \log_2 N$, where N is the number of nodes in the network.

5.1.3 Shuffle

The shuffle benchmark is derived from sorting algorithms, or Fast Fourier Transform (FFT) based applications. There have been substantial efforts invested into optimizing sorting algorithms using a large number of processing elements.

Source and destination pair in shuffle is generated by: $d_i = s_{i-1 \mod b}$.

The fact that all these synthetic benchmarks are geared toward scientific computations is primarily due to historical research efforts. These benchmarks are selected to validate our scheme primarily because they are considered to be conventional testing traffic patterns in NoC architectures.

5.2 Applications

5.2.1 H.264 Decoder

H.264 is widely used for video compression. Figure 5-1 shows a specification of the H.264 decoder, which receives a compressed bitstream and subsequently entropy decodes the data elements into a set of quantized coefficients.

The entropy decoder module in the H.264 decoder performs context-adaptive variable length decoding (CAVLD) that uses 20 different code tables. Each image block from the input stream requires access to different code tables; the number of table lookups varies significantly across inputs. Because the table lookup and the resulting computations take up the majority of time in entropy decoding, we can assume that the latency of the entropy decoder module is proportional to the number of table lookups for each input (image block). In the inter-prediction module, the latency is dominated by the number of pixels it reads from reference frames, which depends on the input block's offset from the reference block (motion vector). Therefore, the latency of the inter-prediction module is again highly dependent on the input block and can be different for each input. Table 5.1 shows the profiling results of both modules for the input stream 'toys and calendar', which illustrates the large difference between the worst-case latency and the average-case latency.

H2.64 is a particularly interesting application, because it has very high data communications between the modules and it is also latency sensitive.

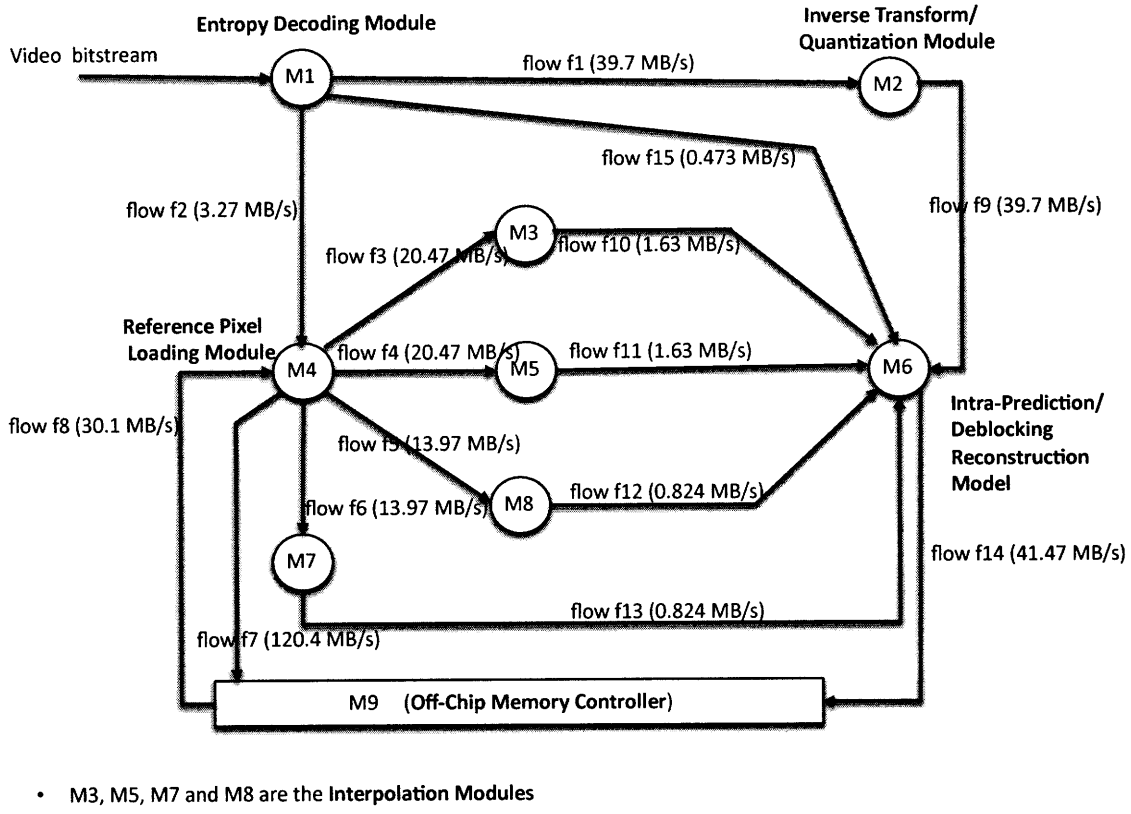


Figure 5-1: High-level Data flow description of H.264 decoder.

5.2.2 Performance Modeling

Performance modeling is a structured and repeatable approach to estimate the performance of a hardware design so that architects can evaluate various alternatives at an early stage of the design. Fast performance modeling is important because it enables the system designer to explore more design choices for more complex designs. Traditionally, performance modeling has been done purely in software. However, to further speed up performance modeling, recent work such as HAsim [60, 61] and FAST [55] uses FPGAs to implement performance modeling in hardware.

The goal of performance modeling is simply to obtain timing information of a target system, not to faithfully emulate the target system cycle by cycle. Therefore, a performance model may take multiple substrate clocks (FPGA or array processor clocks) to perform a single-cycle operation on the target machine. For example, an associative cache look-up can be implemented with a single-ported SRAM in multiple cycles by checking one cache line in each cycle as long as the model counts one model cycle for all these look-ups. The latency of such a cache module varies dramatically depending on the input: one cycle if there is a hit in the first line to check, or many cycles to check every line in the set if an

Table 5.1: H.264 profiling results for a standard input stream.

Entropy Decoder		Inter-prediction	
#lookups	Occurrence	Data read (bytes)	Occurrence
0~5	43.5	0~239	0.01
6~11	38.6	240~399	9.3
12~17	14.4	400~559	19.6
18~23	3.0	560~719	67.5
24~	0.4	720~	0.4
Average	7.56 lookups	Average	589.3 bytes
Maximum	32 lookups	Maximum	954 bytes

Table 5.2: Estimates Data Rates for the IEEE 802.11a/g Wireless LAN Transmitter.

Flow	Source	Destination	Bandwidth	Flow	Source	Destination	Bandwidth
f1	M4	M1	0.7	f11	M15	-	36
f2	M1	M2	36.2	f12	M7	M11	18
f3	M2	M5	36.2	f13	M7	M10	18
f4	M3	M5	48	f14	M7	M9	18
f5	M13	M6	36.8	f15	M7	M8	18
f6	M5	M6	38.9	f16	M8	M12	9
f7	M6	M7	37	f17	M9	M12	9
f8	M12	M13	36.7	f18	M10	M12	9
f9	M13	M14	58.72	f19	M11	M12	9
f10	M14	M15	36.8	Data bits	-	M1	18.1

access eventually incurs a cache miss. Therefore, the performance modeling application has a significant difference between the average-case latency and the worst-case latency; which makes it a great candidate for testing our routing algorithm.

Figure 5-2 shows the three stages pipeline of a microprocessor where the register file, instruction cache and data cache are presented are independent modules.

5.2.3 IEEE 802.11a/g Wireless LAN Transmitter

SoCs platforms have generally attracted digital signal processing (DPS) applications, because such applications can be easily partitioned and mapped onto a distributed computational environment where task-level parallelism can be exploited. For this reason, we also decompose the IEEE 802.11a/g Wi-Fi baseband transmitter block diagram into several functional modules, based on the IEEE standard guidelines [65].

Figure 5-3 shows the structural decomposition of the transmitter for *Orthogonal Frequency Division Multiplexing* (OFDM). It performs forward error correction (FEC) coding, interleaving, symbol mapping, inverse fast fourier transform (IFFT), and guard interval (GI)

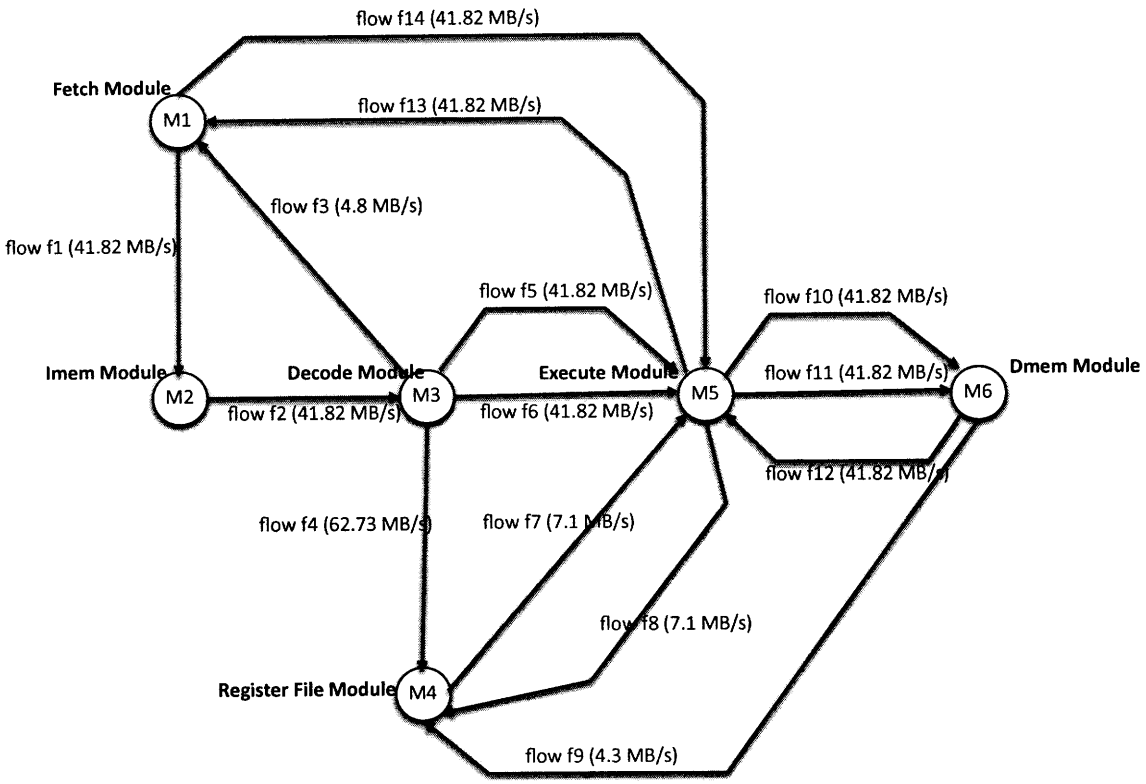


Figure 5-2: Processor Performance Modeling Data Flow.

insertion. The IFFT module, being the most computation intensive, is partitioned over four different processing elements, or nodes. In the profiled example there are 10 OFDM symbols per frame, and the convolutional code rate is 1/2 with 64 32-bit fixed-width complex numbers. Table 5.2 has the rough estimates of data traffics between the modules. Bandwidth is measured in MBits per second.

5.3 Bandwidth Variations

Applications rarely maintain the same input and output data rates for their internal submodules from start to finish. For example, in the H.264 application, data rates between modules vary depending on the frame sequences that are being decoded. The scene of an after-hours office lobby, with little or no change from one frame to the other, has different data rates compared to a scene of a moving camera recording the movements of people at a busy shopping mall. Therefore, to best model the run-time data variations seen in applications, we vary the bandwidth demands for the H.264 and the transpose benchmark by 10%, 25% and 50% in some of the experiments. Using the same routes generated from the initial data rate estimates, we increment or decrement these data rates by some random amount within the percent range. We use a two-stage Markov modulated process to decide

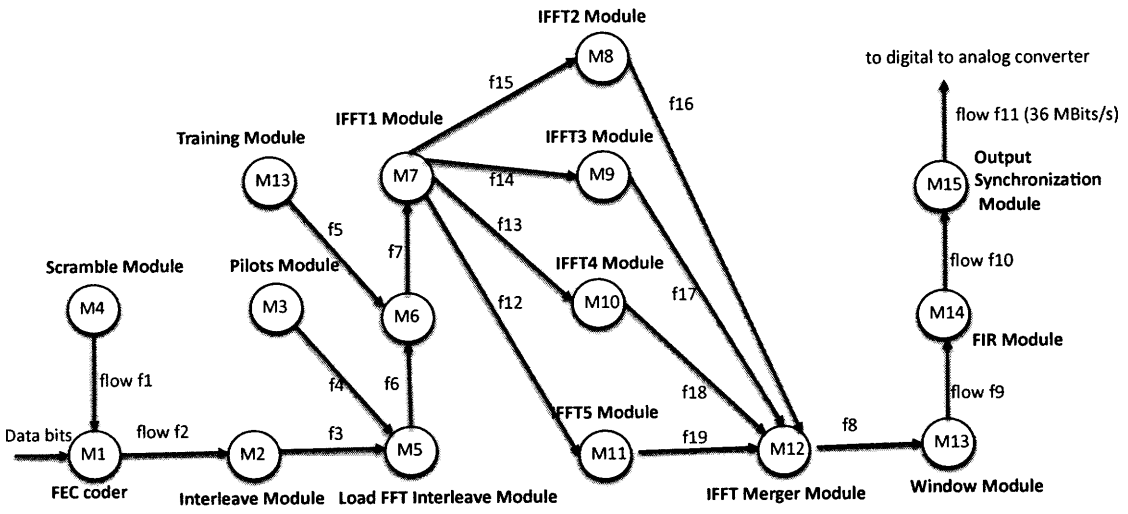


Figure 5-3: Wireless LAN Transmitter Data Flow.

when to increment or decrement, and each rate is kept constant for a random number of cycles. Figure 5-4 shows, as an example, the injection rates of node 52 for transpose during a 25% bandwidth variation test run.

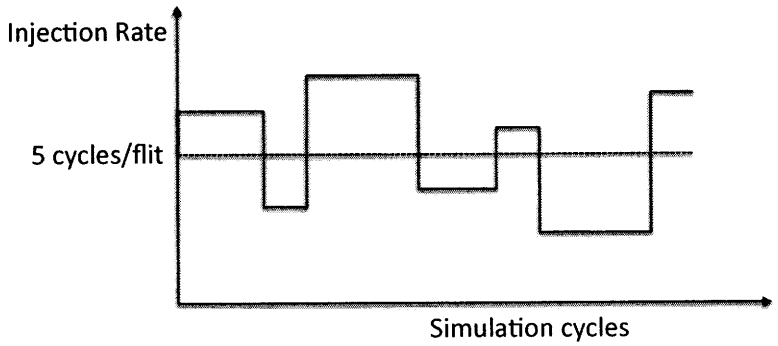


Figure 5-4: Transpose Node 52 Injection Rates when modeling burstiness

Exploring bandwidth variations helps verify the strength of our routing framework when run-time bandwidth demands differ from estimated data rates by some factor.

These various workloads impose a wide range of data communication patterns on the network, which will test the performance, load balancing and latency reduction of our oblivious deadlock-free scheme.

Chapter 6

Performance Evaluation

This section discusses settings for the testing environment where the different workloads, presented in Chapter 5, are simulated. It presents the results from these simulations and the performance analysis of our routing algorithm compared to other oblivious routing algorithms.

6.1 Simulation Methodology

As discussed in the previous chapter, we use a set of standard synthetic traffic patterns, namely, transpose, bit-complement, and shuffle, and a set of real application data communication patterns, H.264 decoder, performance modeling, and 802.11a/g wireless transmitter, in our experiments. The synthetic patterns provide basic comparisons between our routing scheme and other oblivious algorithms, as they are widely used to evaluate routing algorithms. For these synthetic benchmarks, flows have the same average bandwidth demands in all the test cases; except when testing the effect of bandwidth variations on the transpose benchmark. In the case of H.264, performance modeling, and 802.11 transmitter applications, flow bandwidths are derived from profiling results. In effort to obtain a fair representation of the general traffic patterns in these applications, multiple profiling results are taken into account. For the H.264 decoder, several video streams are run and profiled in deriving the flow demands. In these applications, bandwidth varies from flow to flow; performance modeling, for example, has flow demands ranging from 4.3 Mbytes/second to 41.82 Mbytes/second. The H.264 decoder application has flow rates from 0.824Mbytes/second up to 120.4 Mbytes/second.

A cycle-accurate network simulator is used to estimate the throughput and the average latency of flows for each application under the various routing algorithms. The simulator models the router microarchitecture described in Chapter 4. As discussed in router architecture chapter, our routing scheme only requires minor changes in the router microarchitecture. Therefore, we assume an identical clock frequency and pipeline stages for all

Example	North-Last	West-First	Negative-First	Ad Hoc 1	Ad Hoc 2
transpose	175	175	75	175	75
bit-complement	100	100	150	100	150
shuffle	75	100	75	100	100
H.264	140.87	184.94	120.4	174.07	140.87
perf. modeling	62.73	83.65	62.73	95.04	83.65
transmitter	7.34	7.34	9.46	10.52	9.0

Table 6.1: Finding the routes with the minimum MCL (in MB/second) by exploring different acyclic CDGs using $BSOR_{MILP}$.

routing algorithms. We use an 8×8 two-dimentional mesh network with 1, 2, 4 or 8 virtual channels per port. The simulator is configured to have a per-hop latency of 1 cycle, and the flit buffer size per VC of 16 flits.

For each simulation, the network is warmed up for 20,000 cycles before being simulated for 100,000 cycles to collect statistics. Increasing the simulation time beyond 100,000 cycles did not change the results.

The link bandwidth from resource to switch is set to be 4 times the link bandwidth between adjacent switches in the network. For synthetic benchmarks, this setting has no impact on the results. For the other applications, where multiple flows can leave a source and multiple flows can have the same sink node, this allows for more accurate measurement of the switch network performance.

In these experiments, we define the *latency* of a packet as the total amount of cycles spent in the network, from the injection of its header flit into the network at the source, to the reception of its tail flit at its destinations. The latency of a flow is simply the average latency of its packets. Our *throughput* is defined as the amount of information (packets) delivered per simulation cycle on average when in steady state; we call it *average delivery rate* .

6.2 Simulation Results

Using our MILP and *Dijkstra*'s weighted shortest path based BSOR, termed $BSOR_{MILP}$ and $BSOR_{Dijkstra}$, we find the minimum MCL over flow networks G_A's corresponding to 15 different acyclic CDGs D_A's; 12 of these correspond to the D_A's derived from D using the turn model, and 3 correspond to removing edges from D in an *ad hoc* manner. Tables 6.1 and 6.2 summarize some of the results for $BSOR_{MILP}$ and $BSOR_{Dijkstra}$, respectively. The routes corresponding to the minimum MCL are chosen and simulated. Note that in our experiments describing $BSOR_{MILP}$ and $BSOR_{Dijkstra}$, we only consider a single set of routes for each benchmark.

Example	North-Last	West-First	Negative-First	Ad Hoc 1	Ad Hoc 2
transpose	200	200	75	250	75
bit-complement	150	100	150	200	150
shuffle	100	100	75	100	100
H.264	238.44	240.8	188.06	268.74	242.85
perf. modeling	104.55	83.65	83.65	146.38	83.65
transmitter	9.1	10.5	9.1	10.52	10.6

Table 6.2: Finding the routes with the minimum MCL (in MB/second) by exploring different acyclic CDGs using $BSOR_{Dijkstra}$.

Traffic	XY	YX	ROMM	Valiant	$BSOR_{MILP}$	$BSOR_{Dijkstra}$
transpose	175	175	150	175	75	75
bit-complement	100	100	300	200	100	100
shuffle	100	100	100	175	75	75
H.264	253.97	364.73	283.56	254.31	120.4	188.06
perf. modeling	95.04	146.38	104.55	132.57	62.73	83.65
transmitter	10.52	10.6	9.46	22.36	7.34	9.1

Table 6.3: Comparison of Maximum Channel Load (MCL) in MB/second presented by various routing algorithms.

For problems of small or medium sizes, such as the benchmarks considered in this work, MILP converges and produces solutions fairly quickly. MILP solutions, when available, always have MCLs that are equal or smaller than MCLs produced under *Dijkstra*'s weighted shortest path for the same average hop counts.

The minimum MCL results from other oblivious routing algorithms, namely, XY-ordered, YX-ordered, ROMM, and Valiant, are presented in Table 6.3. These algorithms do not take into account the bandwidth requirements. MCLs are determined across all links in the mesh network after all routings are done. For ROMM and Valiant, in the experiments, routing is done by selecting intermediate nodes per-flow basis rather than per-packet basis.

$BSOR_{MILP}$ and $BSOR_{Dijkstra}$ have the lowest MCLs; they also have longer average path lengths than the DOR algorithms and ROMM. It is also worth noting that shorter path length does not necessarily lead to a shorter network latency. Among these algorithms, Valiant produces the longest routes; when the network communication is not sparse, having longer paths creates extra congestion which leads to a higher MCL. Sparsity, in this context, means that for a given topology and a small number of flows, large parts of the network remain unused.

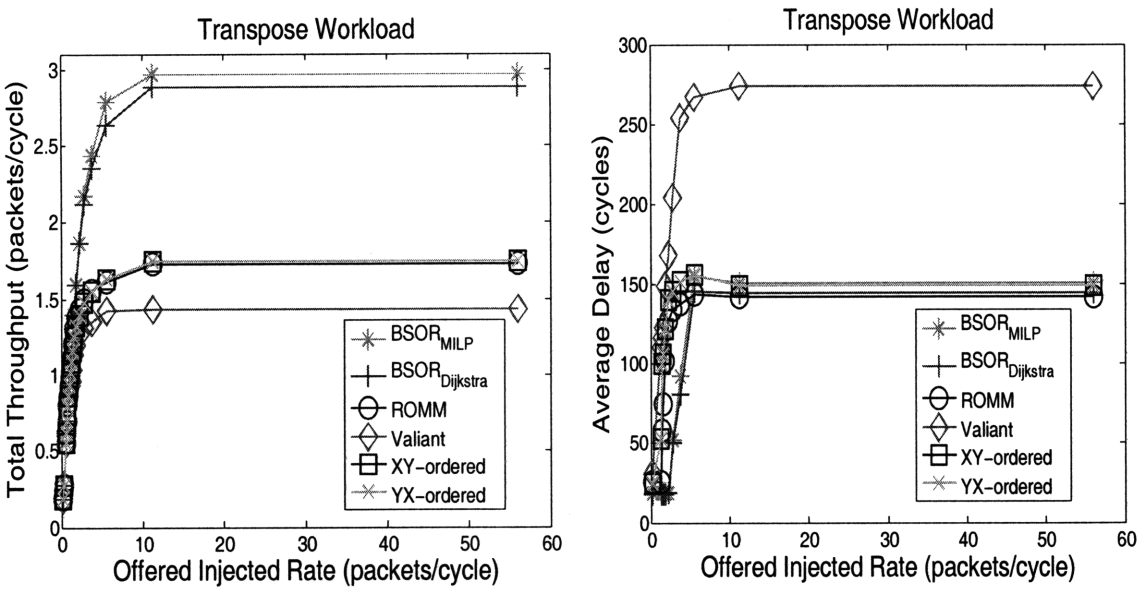


Figure 6-1: Network Throughput and Average Latency graphs for Transpose Benchmark

6.2.1 Transpose Performance Comparisons

Figure 6-1 shows the throughput and the average latency under the transpose synthetic traffic pattern. Total throughput, in packets per cycle, for each routing algorithm is plotted as a function of the injected rate also in packets per cycle. Latencies (average number of cycles per packet) are plotted as a function of the injected rate. In this simulation test case, the number of virtual channels is set to to 2, to guarantee deadlock freedom to the ROMM and Valiant algorithms. Our BSOR scheme, for the transpose traffic pattern, produces routes that archive a network throughput of approximately 70% greater than other routing algorithms, at a comparable average packet latency.

6.2.2 Bit-Complement Performance Comparisons

Figure 6-2 presents the throughput and the average latency under the bit complement synthetic traffic pattern with 2 virtual channels. In this graph, XY-ordered, YX-ordered and $BSOR_{MILP}$ all have the same data points due to the symmetric nature of the flows in the bit-complement application. $BSOR_{Dijkstra}$, ROMM, and Valiant curves exhibit some instability under this application. A routing algorithm is *stable* if its throughput remains constant even as the traffic load is increased beyond the network saturation point. In general, the throughput curves in this test case, in cross-reference with Table 6.3, convey that minimizing MCL generally leads to higher network throughput, when head-of-line blocking and routing instability are not prevalent factors.

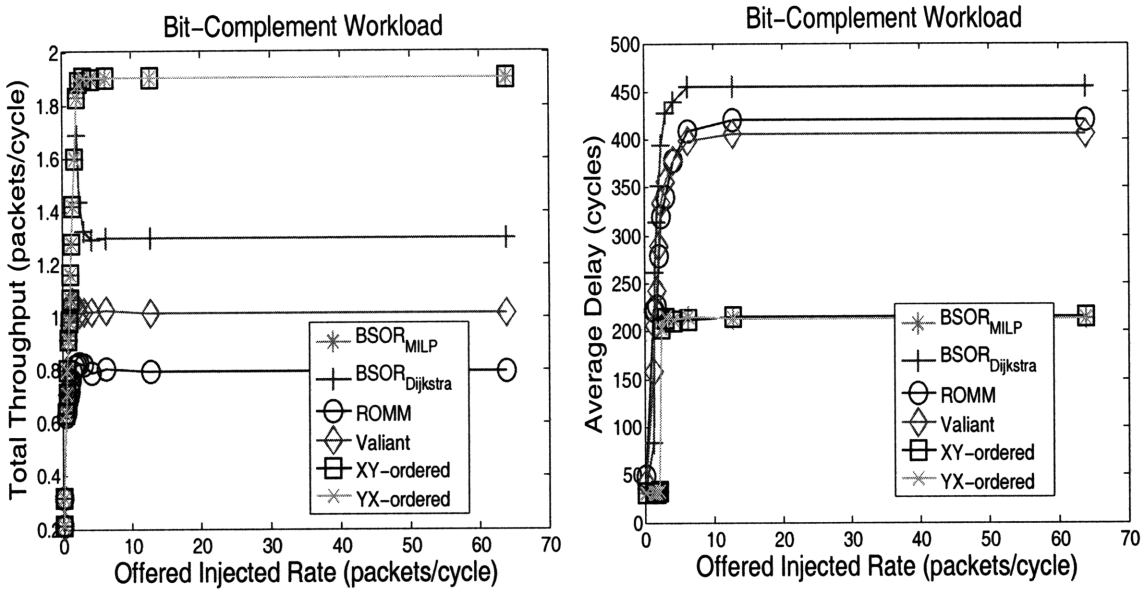


Figure 6-2: Network Throughput and Average Latency graphs for Bit-Complement Benchmark

6.2.3 Shuffle Performance Comparisons

Figure 6-3 shows the load-throughput and the latency graphs for the shuffle synthetic traffic pattern. In this test case, $BSOR_{Dijkstra}$ outperforms $BSOR_{MILP}$ at high injection rates, although they both have the same MCL. $BSOR_{MILP}$, in its attempt to minimize path lengths, losses some of its load balancing effects; its routes are shorter than the routes generated under $BSOR_{Dijkstra}$. But, longer routes do not systematically lead to better load balancing, in fact they may lead to longer packet latencies, which can cause performance degradation if not well leverage, Valiant is a perfect example.

6.2.4 H.264 Decoder Performance Comparisons

Figure 6-4 shows the throughput and latency curves for H.264 decoder traffic patterns for the various routing algorithms. This application is particularly interesting because it is highly throughput and latency sensitive. BSOR algorithms, by minimizing MCL, effectively lower the network congestion and the average latency per packets. But DOR algorithms outperform the BSOR schemes at very high injection rates because congestion hot spots are more isolated. In fact when all 15 acyclic CDGs are run through the simulation, the CDGs with good balance between MCL minimization and locality outperform the DOR algorithm. Locality describes the degree to which the path assigned to a flow goes outside the *minimum quadrant* formed by the source and destination pair.

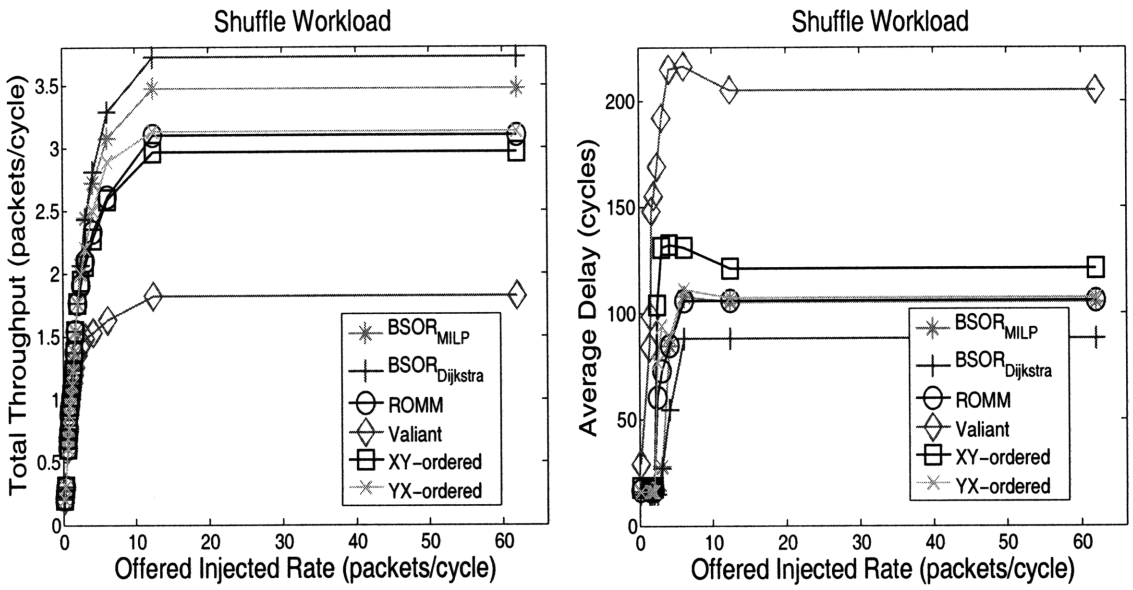


Figure 6-3: Network Throughput and Average Latency graphs for Shuffle Benchmark

6.2.5 Performance Modeling Performance Comparisons

Figure 6-5 illustrates the performance of the network, in terms of throughput and latency, under the performance modeling traffic pattern for the different algorithms. Unlike the H.264 decoder, the performance modeling application is less latency sensitive. The performance gap between BSOR and the other oblivious routing algorithms is most noticeable at higher injection rates. BSOR average latency is similar to the average latency recorded for the DOR algorithms, but with higher network throughput because of BSOR's ability to load balance in a more intelligent way than the random load balancing of ROMM and Valiant. $BSOR_{MILP}$ produces routes that achieve a network throughput approximately 33% greater than other routing algorithms, at a comparable average packet latency.

6.2.6 802.11a/g Transmitter Performance Comparisons

Figure 6-6 shows the throughput and latency graphs for IEEE 802.11a/g Wi-Fi baseband transmitter traffic patterns. The general trends are the same as seen in previous communication patterns. At low traffic loads where there is enough bandwidth for all flows, latency is more relevant. Our BSOR algorithms can balance more efficiently between the need of taking longer paths, when bandwidth constraints require, and short paths due to latency.

6.2.7 Multiple Virtual Channels Performance

In this stage of testing, we run each application with 1, 2, 4 and 8 virtual channels. For the 1 virtual channel case, only DOR algorithms are compared against our BSOR algorithms, because of the deadlock conditions in ROMM and Valiant. Here, we only present the 2,

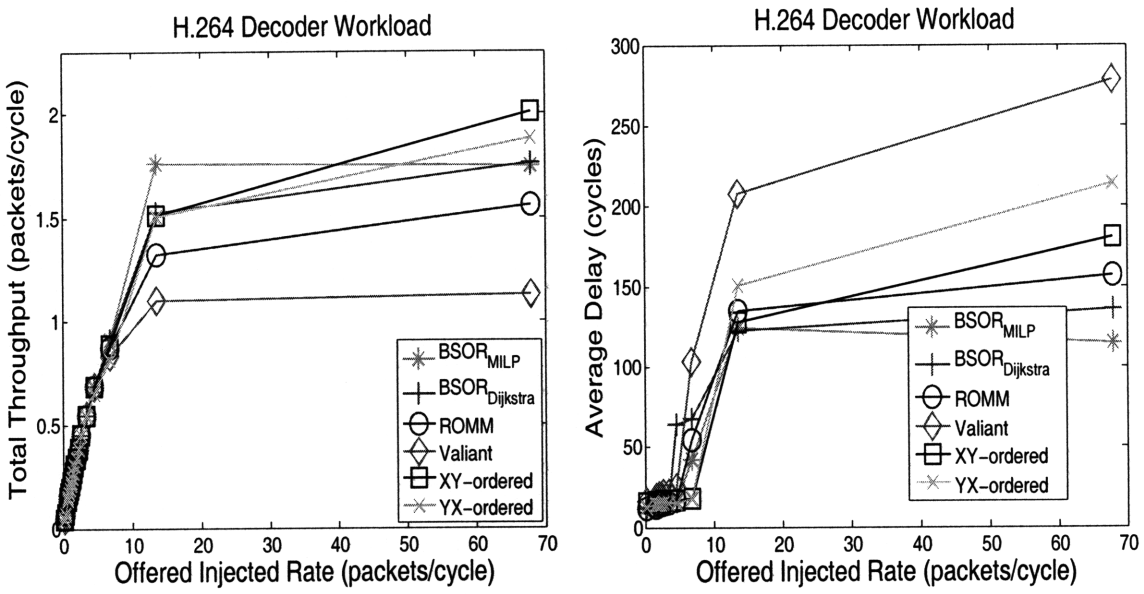


Figure 6-4: Network Throughput and Average Latency graphs for H.264 Decoder Benchmark

4, and 8 virtual channels for the transpose and the H.264 decoder benchmarks. The same general behavior is shown in other applications.

Figure 6-7 illustrates the effect of the number of virtual channels on the network performance under synthetic and real applications. The simulation results show that increasing the number of virtual channels from two to four improves performance, in terms of throughput, by almost 40%. In this case having additional virtual channels per link helps mitigate head-of-line blocking and allows greater bandwidth utilization. In contrast, increasing the number of virtual channels from four to eight does not have the same impact on the throughput; in these applications, link bandwidth seems to be the limiting factor. Both the MILP and Dijkstra-based routes continue to outperform the other oblivious routing schemes.

6.2.8 Bandwidth Variation Performance Comparisons

Figure 6-8 shows the effect of 10% bandwidth variations on transpose and H.264. In the transpose case, this variation has little or no effect on the network throughout, all the routing algorithms maintain relatively the same output. With the H.264 decoder, the variation helps BSOR outperform other routing algorithms. BSOR, by minimizing MCL, leaves more bandwidth available which can be appropriated during high demand periods.

Figure 6-9 presents the effect on the performance when the bandwidth variations are within 25% of the estimated data rates. Overall, the trends remain the same as in the 10% bandwidth variation case. BSOR algorithms show the least performance degradation in presence of run-time bandwidth variations at low injection rates.

Not surprisingly, 50% bandwidth variations, Figure 6-10, have the most effect on the

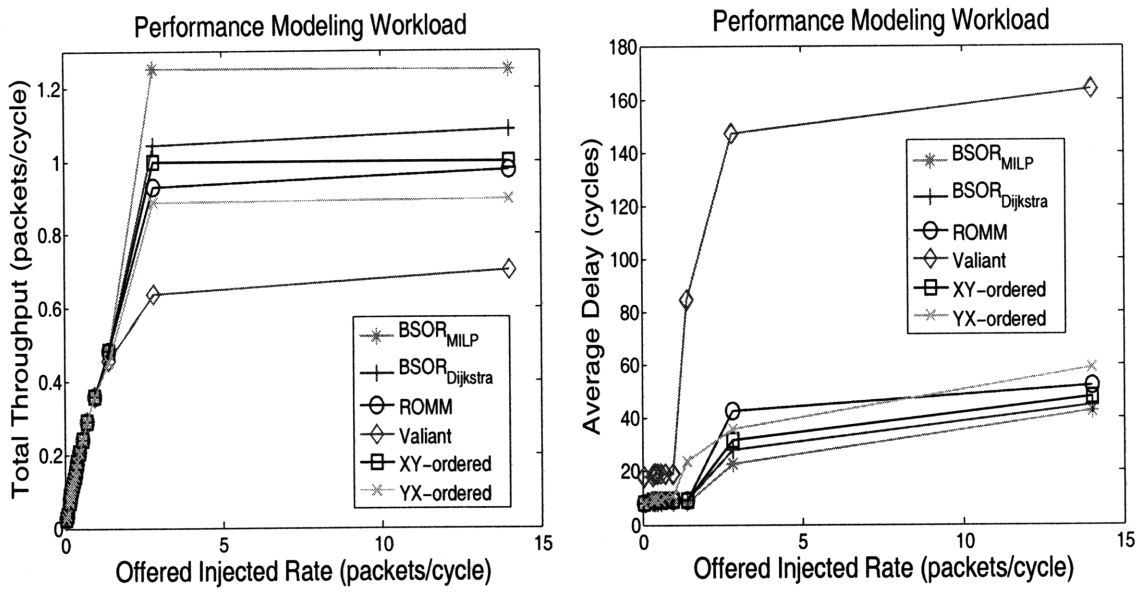


Figure 6-5: Network Throughput and Average Latency graphs for Performance Modeling Benchmark

performance of these benchmarks under the different routing algorithms. However, with transpose our routing algorithms absorb these variations and preserve their higher throughputs over other routing algorithms. In H.264, *minimum* routing algorithms (XY-ordered, YX-ordered and ROMM) outperform the non-minimal schemes. Therefore, in the presence of considerable data rate estimation inaccuracy, the effectiveness of our routing algorithm can no longer be guaranteed, unless routing is done targeting an alternative objective value, such as minimizing the number of flows sharing a link.

6.3 Discussion

Our bandwidth-sensitive scheme is able to achieve better performance than DOR, ROMM and Valiant for benchmarks where it is able to find a lower MCL. For benchmarks with the same MCL (bit-complement), it produces routes of similar quality to DOR.

Maximum channel load is a convenient cost function to optimize, but it may not reflect average channel load, or the complexity of the routes. It is possible that there are many links with load very close to MCL in one routing scheme versus another, which can cause performance to suffer more in one scheme than the other. Typically, the routes produced by our algorithm have more turns than DOR routes. Further, care should be taken to not ignore latency. Non-minimal routes are at an inherent disadvantage when the network is congested, even if they correspond to a lower MCL. For these reasons, improvements in MCL do not translate to equivalent improvements in throughput. Tuning the cost function to better estimate network performance is a subject of ongoing experimentation.

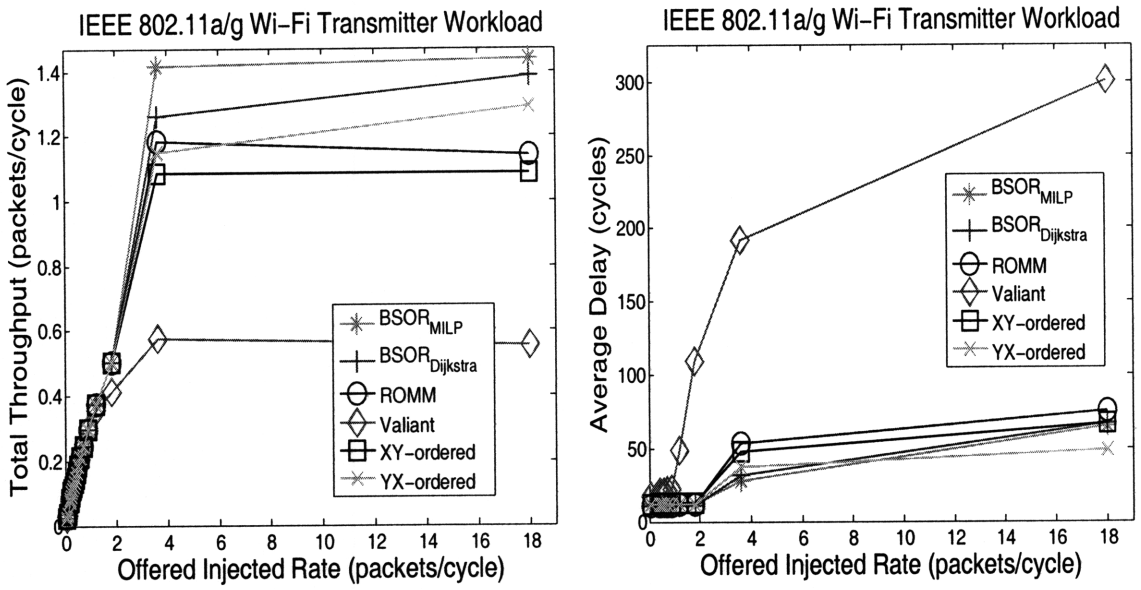


Figure 6-6: Network Throughput and Average Latency graphs for Transmitter Benchmark

6.4 Summary of Results

These results provide evidence that our oblivious deadlock-free algorithm, in its attempt to globally minimize the MCL, does the proper load balancing needed to improve performance. With light network traffic loads, all the algorithms seem to perform about the same, with the exception of Valiant that can suffer from lack of locality. With multiple virtual channels, as expected, head-of-line blocking is mitigated to a certain degree. This is reflected in the performance gain observed when moving from two to four virtual channels. BSOR has a higher path diversity than ROMM, because it is not confined to the *minimal quadrant*; while it avoids the pitfalls encountered with Valiant, where paths can become excessively long, which ultimately leads to longer latency and poor performance.

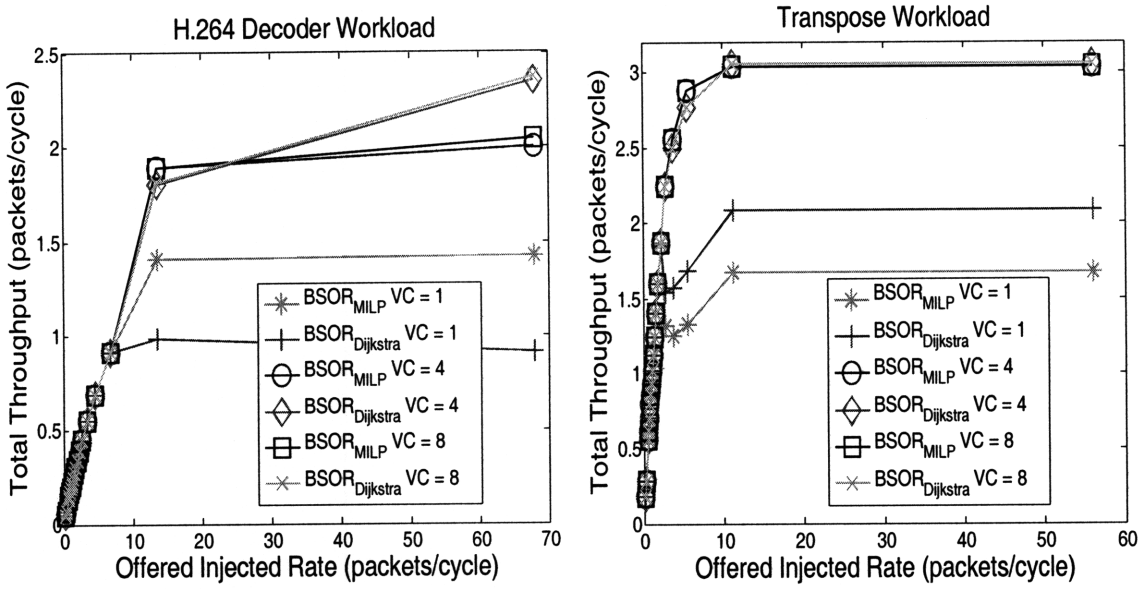


Figure 6-7: Varying the number of VCs for transpose and H.264 Decoder. Results for other examples show the same trend.

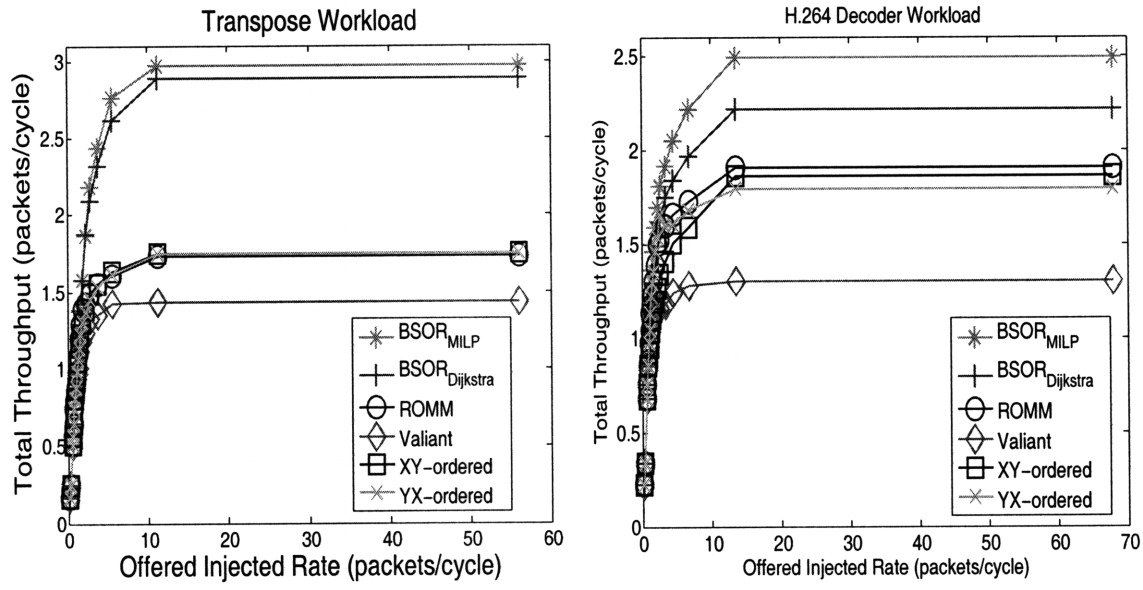


Figure 6-8: The performance of various algorithms with 10% bandwidth variations . (a) Transpose (b) H.264 .

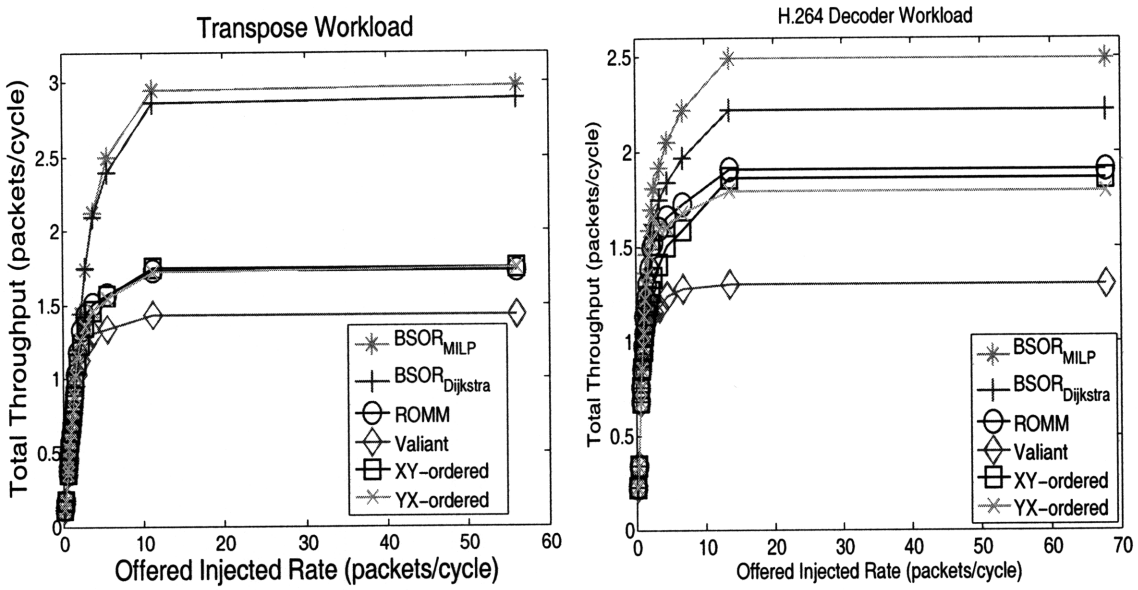


Figure 6-9: The performance of various algorithms with 25% bandwidth variations . (a) Transpose (b) H.264 .

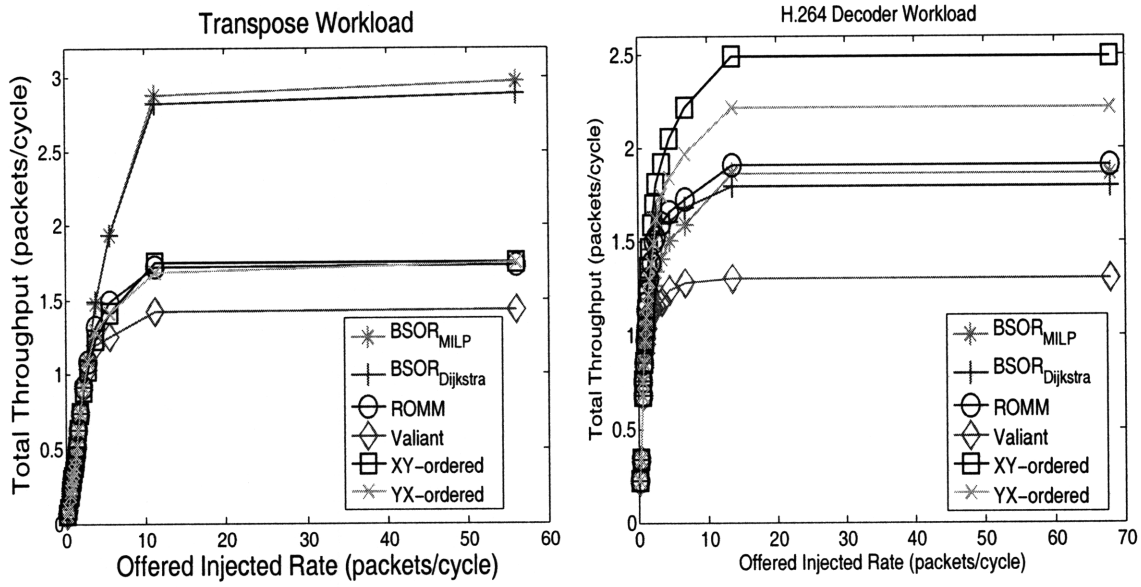


Figure 6-10: The performance of various algorithms with 50% bandwidth variations . (a) Transpose (b) H.264 .

Chapter 7

Conclusions

In conclusion, we present a brief summary of this work, review a number of important research topics that are not covered in this thesis, and part with some final thoughts on the problem of routing data in network-on-chip.

7.1 Summary

The topology of a network on-chip bounds the performance of the network. The routing algorithm used in the network plays a significant role towards achieving these performance bounds. Central to this work is the idea of better load balancing network channels, using application communication characteristics, leading to improved throughput while maintaining a simple router implementation.

We have proposed an offline strategy to compute routes, based on knowledge of the application's data transfers, to arrive at a bandwidth-sensitive oblivious routing scheme that does not require significant modification to standard routers. We have shown that applying estimates of the bandwidth demands of an application's data transfers to routing decisions can help improve application performance through better load balancing.

Chapter 3 presents our bandwidth-sensitive oblivious routing (BSOR) algorithm, and its optimization for small and medium size problems using mixed integer-linear programming (MILP) formulation. Using the Dijkstra's weighted shortest path algorithm, as a route selection scheme for large problem sizes, demonstrates the scalability of the algorithm. Chapter 4 briefly explores the table-based router architecture needed to support the proposed bandwidth sensitive approach by taking a typical virtual channel router architecture, and showing the modifications required. In chapter 5, illustrating applications are described followed by the experimental results.

7.2 Limitations

The primary feature of our approach is also its limitation; we need knowledge of the application communication characteristics. This does not necessarily have to be bandwidth demands, although we have focused on bandwidth in this thesis. It could be knowledge of data transfers whose latency is critical to performance. These transfers can be forced to have minimal routes. Alternately, we can simply minimize the maximum number of flows sharing a link without knowing bandwidths. For scenarios such as reconfigurable computing or coprocessing, we believe that an offline compilation and network configuration methodology is viable, and is in fact the norm, for example, in FPGAs.

In this work, fault-tolerance issues, such as packet loss, link failures or node failures, are not considered. In our assumptions and simulation environment, there is no packet loss. Also, buffer utilization by the various algorithms are not investigated. The expansion of the proposed algorithm, to multicast communication routing, is not considered.

7.3 Future Work

There are many avenues for future work. Different instantiations of the framework we have proposed may result in better routes for benchmarks. For example, a more systematic way of exploring acyclic CDGs may lead to improved results. Given the speed of our Dijkstra-based algorithm, a very large space of acyclic CDGs can be explored, and the framework can be applied to larger problems. We note that the ILP solver can be used as a heuristic approach by limiting the number of iterations for large examples. We are conducting a comparative study of static virtual channel allocation versus dynamic allocation. We aim to find methods to statically allocate virtual channels which can lead to reductions in router complexity without loss of performance.

To handle bursty flows, we have proposed bandwidth-adaptive networks that contain adaptive bidirectional links and can improve the performance of conventional oblivious routing methods [62]. Ongoing work includes evaluating BSOR and BSORM on a bandwidth-adaptive network. We note that most routers only distinguish between source-destination pairs, not individual flows or data transfers. Distinguishing between flows with the same source-destination pair can improve network performance through better (static) load balancing.

7.4 Final Comments

NoC architectures are still growing and design methodologies are being perfected. With the current intensive push toward more parallelized computation, the need for good and relatively simple hardware for routing algorithms is even more pressing. Our hope, in

presenting this work, is to renew interest in oblivious routing algorithms in networks on chip, since they remain the most widely used in today's systems.

Bibliography

[1] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Computers*, 36(5):547–553, 1987.

[2] William J. Dally and Brian Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of the 38th Design Automation Conference (DAC)*, June 2001.

[3] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[4] José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, 1993.

[5] A. Ivanov and G. De Micheli. The Network-on-Chip Paradigm in Practice and Research. *Design & Test of Computers*, 22(5):399–403, 2005.

[6] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, 24(12):1145–1155, 1975.

[7] Herbert Sullivan and T R Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. *SIGARCH Comput. Archit. News*, 5(7):105–117, 1977.

[8] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, New York, NY, USA, 1981. ACM.

[9] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 03(2):194–205, 1992.

[10] W.J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, 1993.

[11] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, 1994.

[12] Craig B. Stunkel and Peter H. Hochschild. SP2 high-performance switch architecture. In *Proceedings of the Symposium on Hot Interconnects*, pages 115–121, August 1994.

[13] Craig B. Stunkel, Dennis G. Shea, Don G. Grice, Peter H. Hochschild, and Michael Tsao. The SP1 high-performance switch. In *Proceedings of the Scalable High Performance Computing Conference*, pages 150–157, May 1994.

[14] José Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 6(10):1055–1067, 1995.

[15] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[16] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA '95*, pages 275–287, 1995.

[17] Ted Nesson, Ted Nesson, S. Lennart Johnsson, and S. Lennart Johnsson. routing on mesh and torus networks. In *in Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 275–287. ACM Press, 1995.

[18] Mike Galles. Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip. In *Proceedings of the Symposium on Hot Interconnects*, pages 141–146, August 1996.

[19] Jon Michael Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, Massachusetts Institute of Technology, 1996. Supervisor-Michel X. Goemans.

[20] Loren Schwiebert. Deadlock-free oblivious wormhole routing with cyclic dependencies. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 149–158, 1997.

[21] William J. Dally, P. P. Carvey, and L. R. Dennison. The Avici terabit switch/router. In *Proceedings of the Symposium on Hot Interconnects*, pages 41–50, August 1998.

[22] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[23] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. pages 29–35, 2001.

[24] Li-Shiuan Peh and William J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proc. International Symposium on High-Performance Computer Architecture (HPCA)*, pages 255–266, January 2001.

[25] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[26] Harald Rcke. Minimizing congestion in general networks. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:43, 2002.

[27] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2003.

[28] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Toward ideal on-chip communication using express virtual channels. *IEEE Micro*, 28(1):80–90, 2008.

[29] Girish Varatkar and Radu Marculescu. Traffic analysis for on-chip networks design of multimedia applications. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 795–800, New York, NY, USA, 2002. ACM.

[30] Cesar A. Zeferino, Márcio E. Kreutz, Luigi Carro, and Altamiro A. Susin. A study on communication issues for systems-on-chip. In *SBCCI '02: Proceedings of the 15th symposium on Integrated circuits and systems design*, page 121, Washington, DC, USA, 2002. IEEE Computer Society.

[31] J. Hu and R. Marculescu. Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures. In *Proc. Design, Automation and Test in Europe Conference*, 2003.

[32] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. Goal: a load-balanced adaptive routing algorithm for torus networks. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 194–205, New York, NY, USA, 2003. ACM.

[33] Brian Towles, William J. Dally, and Stephen Boyd. Throughput-centric routing algorithm design. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, New York, NY, USA, 2003. ACM.

[34] Brian Towles, William J. Dally, and Stephen Boyd. Throughput-centric routing algorithm design. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, 2003.

[35] Tobias Bjerregaard and Jens Sparsø. Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Proceedings of the IEEE Norchip Conference (NORCHIP 2004)*. IEEE, 2004.

[36] Jingcao Hu and Radu Marculescu. Application Specific Buffer Space Allocation for NetworksonChip Router Design. In *Proc. IEEE/ACM Intl. Conf. on Computer Aided Design*, San Jose, CA, November 2004.

[37] Jingcao Hu and Radu Marculescu. DyAD smart routing for networks-on-chip. In *Design Automation Conference*, June 2004.

[38] N. K. Kavaldjiev, G. J. M. Smit, and P. G. Jansen. A virtual channel router for on-chip networks. In *IEEE Int. SOC Conf., Santa Clara, California*, pages 289–293. IEEE Computer Society Press, September 2004.

[39] Robert D. Mullins, Andrew F. West, and Simon W. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. of the 31st Annual Intl. Symp. on Computer Architecture (ISCA)*, pages 188–197, 2004.

[40] Srinivasan Murali and Giovanni De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 914–919, New York, NY, USA, 2004. ACM.

[41] Srinivasan Murali and Giovanni De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 914–919, 2004.

[42] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago Gonzalez Pestana, Andrei Radulescu, and Edwin Rijpkema. A design flow for application-specific networks

on chip with guaranteed performance to accelerate soc design and verification. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1182–1187, Washington, DC, USA, 2005. IEEE Computer Society.

[43] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80, New York, NY, USA, 2005. ACM.

[44] A. Ivanov and G. De Micheli. The Network-on-Chip Paradigm in Practice and Research. *Design & Test of Computers*, 22(5):399–403, 2005.

[45] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proc. of the 32nd Annual International Symposium on Computer Architecture (ISCA)*, pages 432–443, 2005.

[46] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005)*, pages 432–443, 2005.

[47] K. Srinivasan, K. S. Chatha, and G. Konjevod. An automated technique for topology and route generation of application specific on-chip interconnection networks. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 231–237, Washington, DC, USA, 2005. IEEE Computer Society.

[48] James Balfour and William J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, June 2006.

[49] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 2006.

[50] Zvika Guz, Isask'har Walter, Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Efficient link capacity and qos design for network-on-chip. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 9–14, 2006.

[51] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, Narayanan Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *Proc. of the 39th Annual Intl. Symp. on Microarchitecture (MICRO)*, 2006.

[52] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In *Proc. Intl. Conf. on Hardware-Software Codesign and System Synthesis*, Seoul, Korea, October 2006.

[53] K. S. Shim, M. H. Cho, M. Kinsy, T. Wen, G. E. Suh, and S. Devadas. A Comparison of Static and Dynamic Virtual Channel Allocation in Oblivious Routing. In *Proceedings of the 3^{rd} ACM/IEEE International Symposium on Networks-on-Chip*, May 2009.

[54] Krzysztof Walkowiak. New algorithms for the unsplittable flow problem. In *ICCSA (2)*, volume 3981 of *Lecture Notes in Computer Science*, pages 1101–1110, 2006.

[55] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William H. Reinhart, D. Eric Johnson, and Zheng Xu. FAST Methodology for High-Speed SoC/Computer Simulation. In *International Conference on Computer-Aided Design*, November 2007.

[56] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27(5):51–61, Sept/Oct 2007.

[57] M. H. Cho, C-C. Cheng, M. Kinsy, G. E. Suh, and S. Devadas. Diastolic Arrays: Throughput-Driven Reconfigurable Computing. In *Proceedings of the Int'l Conference on Computer-Aided Design*, November 2008.

[58] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *ISCA '08: Proceedings of the 35th annual international symposium on Computer architecture*, 2008.

[59] Maurizio Palesi, Giuseppe Longo, Salvatore Signorino, Rickard Holsmark, Shashi Kumar, and Vincenzo Catania. Design of bandwidth aware and congestion avoiding efficient routing algorithms for networks-on-chip platforms. *Proc. of the ACM/IEEE Int. Symp. on Networks-on-Chip (NOCS)*, pages 97–106, 2008.

[60] M. Pellauer, M. Vijayaraghavan, M. Adler, J. Emer, and Arvind. Quick Performance Models Quickly: Timing-Directed Simulation on FPGAs. In *International Symposium on Performance Analysis of Systems and Software (ISPASS 2008)*, April 2008.

[61] Michael Pellauer, Muralidaran Vijayaraghavan, Michael Adler, Arvind, and Joel Emer. A-ports: an efficient abstraction for cycle-accurate performance models on fpgas. In *Proceedings of FPGA '08*, pages 87–96, 2008.

[62] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas. Oblivious routing in on-chip bandwidth-adaptive networks. Technical Report CSAIL-TR-2009-011 (http://hdl.handle.net/1721.1/44958), Massachusetts Institute of Technology, March 2009.

[63] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 2001.

[64] Thomas Gross and David R. O'Hallaron. *iWarp: anatomy of a parallel computing system.* MIT Press, Cambridge, MA, USA, 1998.

[65] *IEEE. IEEE standard 802.11a supplement. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specications, 1999.*