DEFLECTION NETWORKS: PERFORMANCE EVALUATION, LIVELOCK PREVENTION, AND FORMAL SPECIFICATION

By

NOBUYUKI NEZU

Bachelor of Science
Gakushuin University
Tokyo, Japan
1991

Master of Science
Oklahoma City University
Oklahoma City, Oklahoma
1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 1999

DEFLECTION NETWORKS: PERFORMANCE EVALUATION, LIVELOCK PREVENTION, AND FORMAL SPECIFICATION

Thesis Approved:

H. Lu

Thesis Adviser

J Chandler

Sheldon Kay

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to thank Dr. H. Lu, who gave me valuable instruction and consistent support throughout this research. I would also like to thank Dr. John P. Chandler, Dr. K. M. George, and Dr. Sheldon Katz for their critical reviews and suggestions. I am grateful to Dr. G. E. Hedrick for chairing the defense meeting.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A proliferation of network users has led to the introduction of metropolitan area networks (MANs). MANs interconnect larger numbers of users and span longer distances than conventional local area networks (LANs). A recent generation of MANs is based on linear structures (bidirectional loops and buses). As in the case of LANs, these networks trade their throughputs for simple access strategies. The performance of linear structure networks degrades with increasing numbers of users.

As alternatives to linear structure networks, two-dimensional toroidal networks with deflection routing have been proposed for MANs. A toroidal (or torus) network is a grid network with boundary connections (Figure 1.1). The boundary connections eliminate edge effects and decrease inter-nodal distances. The symmetry (transitivity) and the global consistency (sense of direction) of toroidal networks support simple and locally implementable routing schemes. Toroidal networks provide multiple paths between a source and a destination and increase their throughputs with the number of nodes by decreasing the fraction of the network capacity needed to communicate between nodes. The reliability of networks is also increased [7, 53, 74].

Deflection routing, which supports communications in toroidal networks, elim-

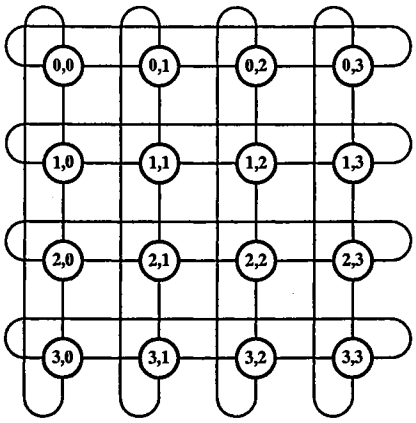

Figure 1.1: A toroidal network.

inates buffering resources. As a consequence, the networks do not suffer internal congestion. Packets can be accepted as long as there is room in the networks. This makes the behavior of the networks similar to that of LANs. The elimination of buffering resources also enables nodes to follow the link speed as close as possible. The deflection networks naturally provide high-speed adaptive datagram services [4, 7, 14].

Note that toroidal topologies and deflection routing are also applicable to (processor) interconnection networks for parallel processing computers [48, 74, 83].

1.1 Motivations

Toroidal deflection networks have potential as high-speed metropolitan area communication infrastructures. The performance of toroidal networks scales better than that of bus and loop networks. The elimination of buffers simplifies and accelerates the network operations. It is also important for the design and implementation of all-optical networks, which will be the basis for future communication systems.

Although the structures of deflection networks are simple, because of the inherent unpredictability of routes taken by packets, the performance analysis of deflection networks is difficult. The dynamic behavior of the networks may be best studied

through (discrete event) simulation [45, 78]. The quantitative results obtained by simulation can be used in assessing the feasibility of routing algorithms.

It seems unlikely that the topologies of realistic networks are perfectly symmetric. Hence, routing algorithms should be able to overcome topological irregularities. Naturally, deflection routing is capable of handling certain topological irregularities, yet the issue needs to be addressed.

One problem unique to deflection networks is *livelock* (the indefinite circulation of packets without reaching destinations), the counterpart of *deadlock* in store-forward networks. Livelock needs to be eliminated from deflection networks in order to ensure the eventual delivery of every injected packet.

As in the case of many proposed parallel and distributed systems, formal specifications for deflection networks have not appeared in the literature. For most systems, the existence of the interactions of the concurrent components of the systems and their environments is making the development of rigorous specifications (and the development of the systems themselves) a difficult task.

Formal methods are mathematical techniques for specifying and verifying complex systems. The use of formal methods eliminates inconsistencies, ambiguities, and incompletenesses that may remain undetected with informal specifications [24].

It is desirable that a specification language, besides being rigorous, should be compact and easy to learn for those who are new to formal specification (both the designers and the readers of the specifications). The logic of UNITY (Unbounded Nondeterministic Iterative Transformations) [17, 61, 62, 79], which has been designed for the specification and verification of concurrent systems, can be used as such a

language.

1.2 Objectives

The objectives of this research are:

1. to investigate the feasibility of using toroidal deflection networks as metropolitan area communication infrastructures and
2. to provide a logical framework for reasoning about routing and control functions of deflection networks.

The research encompasses the analysis of throughput and delay characteristics, the development of routing and livelock prevention schemes, and the modeling and specification of deflection networks.

1.3 Organization of this Dissertation

The materials presented in this dissertation cover many key issues of deflection networks. The dissertation is organized as follows. In Chapter II, routing in wide area networks, local and metropolitan area networks, topologies of deflection networks, and formal specification methods are reviewed. Deflection routing is introduced. The characteristics of deflection networks are described. The materials of Chapter II provide background knowledge for the following chapters. The performance of deflection networks is studied in Chapter III. The bulk of the results in Chapter III are from [68], except the examination of deflection routing on irregular networks in Section 3.3 and random routing in Section 3.4. Section 3.3 demonstrates that a simple routing scheme can function on irregular networks, which may be incrementally constructed.

The examination of the random routing in Section 3.4 gives an interesting insight into the topological characteristics of networks. A livelock prevention mechanism, which is based on [69], is presented in Chapter IV. In Chapter V, a formal specification for deflection networks is developed. The UNITY computational model and logic are introduced. The operators of [61, 62], which are derived from the original work of UNITY [17], are re-defined by using the notion of the *strongest invariant* [80]. A network is modeled as a closed system with unbounded I/O queues. UNITY logic is used to formulate the properties of the network. The developed specification is mapped to a UNITY program (a network simulator in pseudocode). The basic idea of Chapter V is from [70]. Chapter VI evaluates and concludes this dissertation. A discussion of future research problems is also given in the final chapter.

CHAPTER II

LITERATURE REVIEW AND BACKGROUND

In this chapter, we review routing in general wide area networks, local and metropolitan area networks, deflection networks, and formal specification methods. This chapter provides background knowledge for the following chapters.

2.1 Routing in Wide Area Networks

Routing methods can be classified as centralized or distributed; static or adaptive; and link-state approach or distance-vector approach (The link-state approach replicates the entire network topology information at each node, whereas the distance-vector approach, at each node, stores only the distances to other nodes) [63, 71]. Computing the shortest (delay) paths in a distributed adaptive manner typically requires periodical routing information (e.g., the congestion of links/queues) exchange among nodes. In this section, we briefly review the routing methods used in the ARPANET [9, 30, 58, 81, 87], which was formed by the Advanced Research Project Agency (ARPA) of the U.S. government and has now grown into the worldwide Internet.

The first ARPANET routing algorithm was implemented in 1969. It is a distributed adaptive algorithm based on shortest path routing. Each packet is forwarded

to its destination along the path that minimizes the estimated delay (transit delay). The delay (cost) of a link is measured based on the number of packets waiting for transmission in the queue of the link. Each node exchanges its routing information (i.e., the estimated delays from the node to all destinations) with neighboring nodes periodically as short as every 2/3 seconds. The Bellman-Ford algorithm [8, 33] (distance-vector approach) is used to update the routing information (i.e., calculate the shortest delay paths). Because of the rapid routing information (link costs) updates, the algorithm had faced problems with message re-assembly (i.e., the packets of a message arrived at the destination from different paths out of the transmitted sequence) and looping. The rapid link cost changes (updates) caused routing path changes and the routing path changes again caused the link cost changes. To stabilize this situation, a large constant was added to each link cost. This, unfortunately, made the algorithm less sensitive to the congestion of links.

A decade later, the second algorithm, which is known as Shortest Path First (SPF), was implemented and replaced the first one. In the second algorithm, the update interval was changed to 10 seconds. Each link cost is calculated as the average packet delay (including processing, queueing, transmission, and propagation delays) on the link since the previous update (i.e., during the past 10 seconds). Each node broadcasts the costs of its outgoing links (by flooding) at least every 60 seconds. Based on the broadcasted link costs, each node individually calculates the shortest paths using Dijkstra's algorithm [31] (link-state approach). The algorithm improved the stability although later an increase in traffic load lowered the stability and necessitated a revision of the method for calculating link costs. In the 1987 implementation, the

range of link cost and the rate at which the cost can change were drastically reduced. As a result, the dynamic behavior of the algorithm improved substantially. The successor of the algorithm, which was named Open SPF (OSPF), was standardized by the Internet Engineering Task Force (IETF) in 1990. The standard are now supported by many router vendors.

2.2 Local and Metropolitan Area Networks

Generally, local and metropolitan area networks (LANs and MANs) have very simple topologies and do not require algorithmically sophisticated routing methods used in wide area networks. The Ethernet and the token ring are two popular local area networks [84]. The Ethernet is a bus network. The token ring is a single loop network. In both networks, only one node is allowed to transmit data at a time. The performance of the networks degrades linearly with the number of nodes. The Fiber Distributed Data Interface (FDDI) Network [77] and the Distributed Queue Dual Bus (DQDB) Network [64] use bidirectional loops and buses and have been proposed for metropolitan area networks. These networks simply double the throughput of earlier networks and can survive a single link failure.

2.3 Deflection Networks

In this section, we examine the characteristics, topologies, possible failure recovery mechanisms, and expandability of deflection networks.

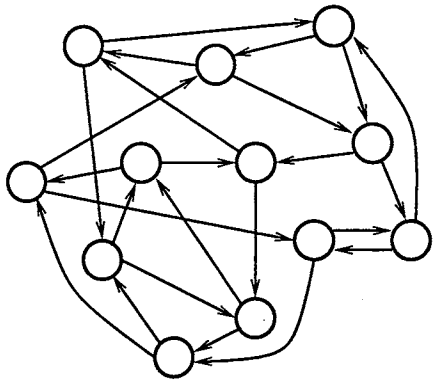

Figure 2.1: A two-connected network.

2.3.1 Characteristics

Deflection routing, which is similar to hot-potato routing [6], is a distributed adaptive routing method that can be applied to the networks in which the in-degree and the out-degree (the number of incoming links and the number of outgoing links) of each node are equal. An example of such a network is shown in Figure 2.1. The network is two-connected (i.e., every node has two incoming links and two outgoing links). Note that deflection networks are not necessarily regular. Different nodes in a network may have different connectivities. The networks operate in a slotted mode with fixed length packets. The switching and transmission processes take place on a time slotted basis. Generally, all links are one slot in length (i.e., every packet travels one link per slot). (Links may be multiple slots long in some networks.) A packet (from an input source) enters a network if an empty slot is available. When two or more simultaneously arriving packets contend for the same output link, a local contention resolution rule is used to select the packet that gets the use of the link. The packets that lost a contention are deflected (misrouted). Because the in-degree and the out-degree of a node are equal, it is always possible to assign each one of simultaneously

arriving packets to a distinct output link.

For each time slot, a node of a deflection network

1. extracts a packet addressed to the node,
2. injects a source packet if the node obtains an empty slot (i.e., a slot has been received empty or a packet has been extracted), and
3. selects a switching configuration.

The switching and transmission of packets follow the above steps.

The buffering resources required by the standard store-forward method are eliminated in deflection networks. Hence, there is no internal congestion. The behavior of the networks is stable. Packets are accepted as long as nodes recognize empty slots. Deflection routing is naturally adaptive to hot spots that may arise under certain fixed routing schemes. No routing information exchanges among the nodes are required. It is also noted that deadlock due to buffer overflow will not occur in deflection networks.

One problem uniquely associated with deflection networks is *livelock* (the indefinite circulation of packets without reaching destinations). Although livelock may be rare in practice, as deadlock needs to be prevented for store-forward networks, livelock needs to be prevented for deflection networks. This issue will be addressed in a later chapter.

2.3.2 Topologies

The topology of a deflection network plays an important role in developing routing algorithms that require no routing tables and accelerate nodal processing. In reviewing

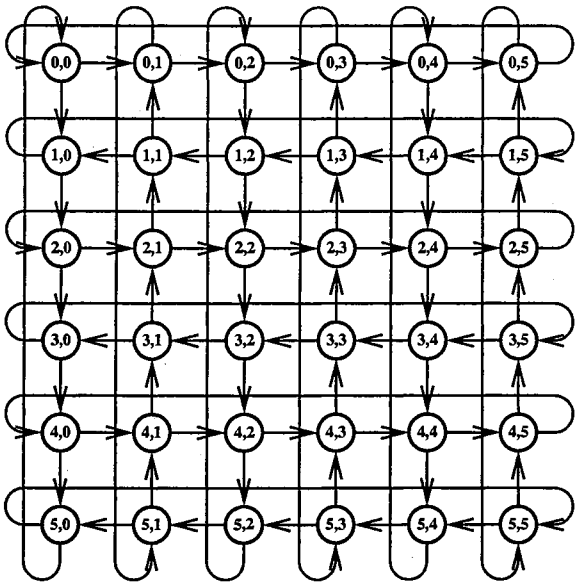

Figure 2.2: A Manhattan Street Network.

the previous works, we classify deflection networks by topology.

The most widely studied deflection network is the Manhattan Street Network (MSN) [52, 53] (Figure 2.2). The MSN is a toroidal network. Its topology resembles the one-way streets and avenues in Manhattan. The alternation of row and column link directions keeps inter-nodal distances small, but also restricts the number of rows and columns to even. The network has two incoming links and two outgoing links at each node and hence has the same degree of connectivity (the same number of transmitters and receivers) as the bidirectional loop network.

Deflection routing strategies for the MSN have been studied extensively in the literature [35, 54, 75]. The relative superiority (higher throughput and reliability) of the MSN over linear topology networks (the FDDI Network and the DQDB Network) are reported in [15, 42, 57]. In [19], buffered deflection routing in the MSN is studied; it is reported that the use of a few buffers (to hold some transit packets) provides a significant improvement in performance. (Note that no buffers, except some delay

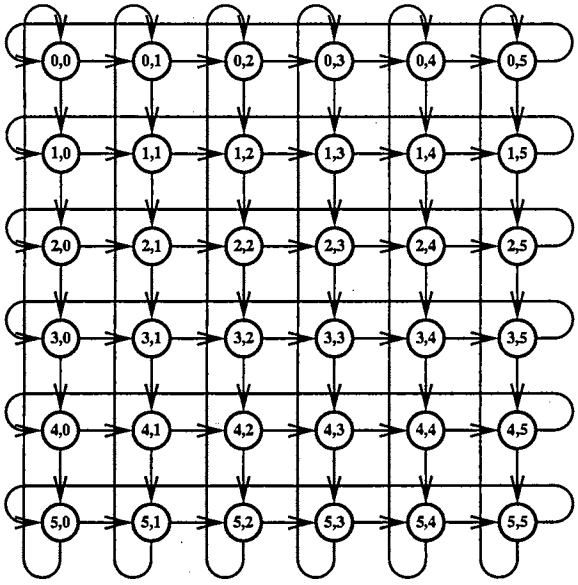

Figure 2.3: A Highway Transfer Network.

lines, may be available in optical networks.)

The two-connected toroidal topology shown in Figure 2.3 has been used for the Highway Transfer Network [44], the Manhattan Fiber Distributed Data Interface (M-FDDI) Network [38], the Token Grid Network [89], and the Multimesh Network [90]. (The Highway Transfer is a fast packet forwarding technique. The M-FDDI Network extends the FDDI Network. The Token Grid Network uses a hierarchical token passing protocol. The Multimesh Network is a generalization of the Token Grid Network.) As in [7], which briefly introduces [44], we call a network with the topology a Highway Transfer Network (HTN). Previously, in the literature, the HTN has not been studied as a deflection network, perhaps, because of its large deflection penalty (the least upper bound on the number of "extra" hops that a packet has to make after a deflection), which grows with the network size. However, the network also has a high density of so-called *don't care* nodes (from which packets can be transmitted through any outgoing link without being deflected) and hence the number of deflections can

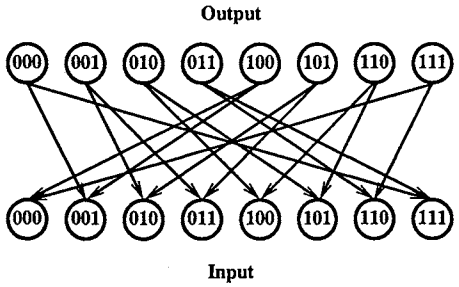

Figure 2.4: A Shuffle Exchange Network.

be kept small. The network is defined for any numbers of rows and columns. The structural requirement is relaxed. The HTN supports a considerably simpler routing scheme than the MSN. A high performance/cost ratio can be expected. The HTN may be used as an inexpensive alternative to the MSN. The performance of the HTN is investigated in a later chapter.

Networks based on the perfect shuffle interconnection have also been studied as deflection networks. The Shuffle-Exchange Network [41, 53] (Figure 2.4) has a smaller diameter ($\log_2 N$, where N is the number of nodes in the network) than the MSN. However, the Shuffle-Exchange Network has a greater deflection penalty than the MSN and is not defined for an arbitrary number of nodes (the number of nodes must be a power of two). Alternate paths are much longer than the shortest paths. Very few nodes have alternate paths that have the same distance to the destinations. The physical layout of the network does not make sense geographically. The network can be used only in a small area. The Shuffle-Exchange Network does not have the characteristics of the MSN that cause deflection routing to perform well. Without buffers (to hold some transit packets), the throughput of the Shuffle-Exchange Network is lower than that of the MSN [55]. The network is highly structured and hence expansion (node addition) is difficult (requires re-wiring of links and re-addressing of

nodes).

Bidirectional extensions of the MSN have also been studied in the literature [1, 2, 3, 4, 11, 12, 13, 14, 91]. The use of bidirectional links doubles the maximum attainable throughput of the MSN. However, the complexity of routing decisions is not just twice of that in the MSN. For networks with unidirectional links, there are only $2^2 = 4$ possible preferred packet moves at a node, 2 of which require contention resolutions. For bidirectional extensions, the number of possible packet moves at a node increases to $3^4 = 81$ [11]. Node construction becomes increasingly difficult. Deflection routing may be difficult to implement for the networks with a high degree of connectivity. A large number of possible switching configurations in high connectivity networks makes cost efficient, high performance implementations challenging. The distance measures of the MSN and the bidirectional extensions are fairly close, yet only half the number of transmitters and receivers of the bidirectional extensions is required in the MSN [23]. One advantage of the bidirectional extensions is that the in-degree and the out-degree of each node are always the same even after link failures. No protocol (that involves multiple nodes) to recover from a link failure is required.

Toroidal networks can be naturally extended into higher dimensions. (A three-dimensional network is shown in Figure 2.5.) A notable multidimensional extension is the Multidimensional Manhattan Street Network (MMSN) [20, 22] (Figure 2.6). Although the MMSN spans multidimensionally, it remains two-connected. Unfortunately, routing and node addition in the MMSN become more difficult than those in the MSN. As in the case of general multidimensional extensions, the reduction of the diameter from the MSN to the MMSN is not as great as that from the loop network

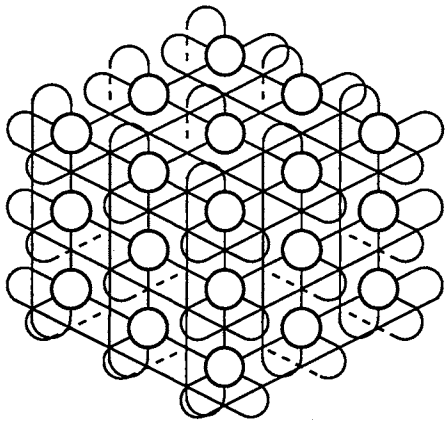

Figure 2.5: A 3D-torus network.

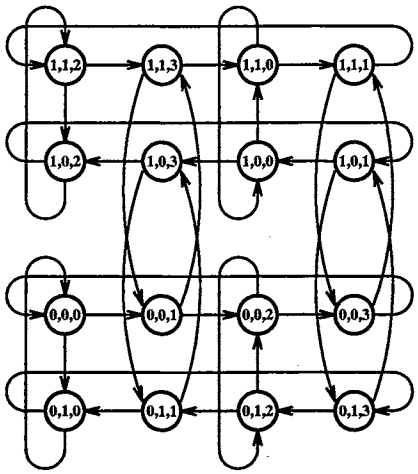

Figure 2.6: A Multidimensional (3D) Manhattan Street Network.

to the MSN (i.e., from one to two dimensions).

2.3.3 Node and Link Failures

Node failures considered here are those that caused by local power outage, which is the most common cause of node failures. In physical implementations, it is assumed that a non-switching relay is used to bypass a failed node. The relay is open when there is power and closes (so that packets will pass straight through the node) when power is lost [54]. (Packets that would have made a turn at the failed node are regarded as deflected packets.)

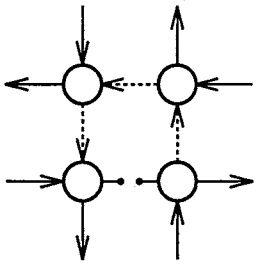

Figure 2.7: A link failure in the MSN.

Nodes in a typical deflection network (slotted network) transmit data continuously to reduce timing acquisition problems [57]. Continuous data transmission also enables the detection of link failures (and also node failures) by downstream nodes. A node recognizes an incoming link failed when no data are coming from the link. Then, the node may stop transmitting data from one of its outgoing links [54]. This procedure is illustrated in Figure 2.7 for the MSN. The equality of the in- and out-degrees of nodes will be preserved and hence once the procedure has completed, no packets will be lost. The deflection mechanism bypasses failed links as it tries alternate paths.

Note that a link is merely a cable and only physical destructions, which are less likely than node failures caused by power outage, can cause link failures.

In practical networks, packets that cannot reach their destinations due to node or link failures must be removed from the networks. Packets that have spent a certain amount of time in the networks without reaching their destinations should be removed. This also removes packets that are livelocked. Acknowledgments are required to recover the lost packets [56].

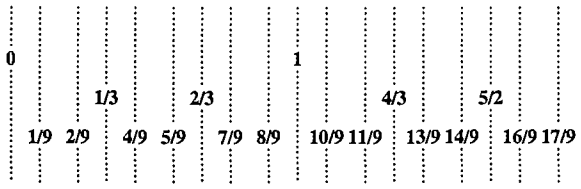

Figure 2.8: Fractional addresses.

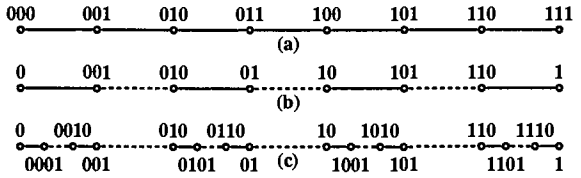

Figure 2.9: The reduced-binary addressing scheme.

2.3.4 Network Expansion and Node Addressing

Node addressing is an issue that needs to be addressed for network expansion. Generally, routing algorithms in structured networks assume that the nodes in the networks are properly addressed.

For the MSN, the fractional addressing scheme [54] and the reduced-binary addressing scheme [47] have been proposed. Both schemes allow new nodes to be inserted into the network without changing the addresses of the existing nodes. The schemes support gradual and modular growth of the MSN.

Figure 2.8 shows fractional addresses in the MSN. The first two rows (or columns) are labeled 0 and 1. Rows are added in pairs and are labeled as two fractions, 1/3 of the way between two other rows. New rows that are added between 1 and 0 are considered to be between 1 and 2.

The reduced-binary addressing scheme in the MSN is illustrated in Figure 2.9. Axis (a) is a binary number axis. Axis (b) shows reduced-binary addresses. Each number in (b) is obtained from the corresponding number in (a) by eliminating the

bit in the least significant position if it is equal to the previous bit. Axis (c) shows the axis after 8 nodes are added to axis (b). The shorter address string needs to be extended to the same length as the longer address string by repeating its last bit before any operations are performed on the address strings. The axis is piecewise-continuous. New nodes can be inserted only in continuous segments (solid liens). No node can be inserted in non-continuous segments (dashed liens). The reduce-binary addressing scheme is designed for high-speed hardware implementation.

Naturally, both the fractional addressing scheme and the reduced-binary addressing scheme can be applied to the HTN and the bidirectional extensions of the MSN.

In the MSN, new node additions may degrade the performance of the routing algorithm. (The address transformation algorithm [54] may displace the destinations from the center of the network. As a result, packets may follow longer paths to the destinations.) A care should be taken for new node additions (e.g., new nodes should be evenly distributed over the grid).

For most networks, if fractional addresses are used, the determination of the shortest paths becomes difficult (simple arithmetic operations on node addresses will no longer give correct distances between nodes). Further more, if irregularities are introduced in the network topology (as a result of network expansion), the given routing scheme may no longer work on the network. Apparently, network irregularities, make routing a difficult task [65, 66, 67].

2.4 Specification Methods

The logic of UNITY (Unbounded Nondeterministic Iterative Transformations) [17, 61, 62, 79] has been chosen as the language for the specification of deflection networks in this research. It is closely related to temporal logic [43], which defines temporal relationships of predicates over infinite sequences of states. In the UNITY model, a program is viewed as a mathematical object and not in terms of its possible executions. The elimination of operational reasoning makes UNITY logic a formal method. UNITY enables us to develop descriptive, non-operational specifications for programs.

The computational model of UNITY separates the concept of termination, which is central in traditional transformational programs, from the problem solving process. The UNITY model captures parallel and distributed programs that are ongoing (*reactive*) and nonterminating.

UNITY logic is surprisingly simple and compact. Its design decisions avoided introducing notational artifacts.

In his forward to Chandy and Misra's work of UNITY [17], Hoare states that a complete theory of programming includes methods for

- specifying programs,
- reasoning about specifications,
- developing correct programs, and
- transforming programs for executions on available machines.

Dijkstra's work [32] provides the methods for sequential programming. Chandy and Misra's UNITY does the same for parallel and distributed programming.

The UNITY methodology has been applied to a variety of design and specification problems [26, 27, 28, 39, 40, 72, 85, 86]. Its versatility has been demonstrated.

The UNITY logic operators are formally defined in a later chapter. In the rest of this section, other notable specification methods for parallel and distributed systems are cited. Hoare's CSP (Communicating Sequential Processes) [37], which is a process-based formalism, has been studied widely. Milner initiated the algebraic approach through his CCS (Calculus of Communicating Systems) [60]. Besides UNITY, several temporal logic based methods [46, 50, 51] have been proposed in recent years. The studies of temporal logic in computer programming can be traced back to the works of Burstall [16] and Pnueli [73] in the late 1970's. (Historical remarks on the development of temporal logic itself can be found in [43].) Automata have also been used for describing the behavior of concurrent systems [49]. Methods based on Petri Net, which is an extension of automata, have also been studied [59]. ISO (International Organization for Standardization) has developed a standardized language LOTOS (Language for Temporal Ordering Specifications) for concurrent, distributed, and nondeterministic systems. The specification of dynamic behaviors in LOTOS is predominantly based on CCS, while the treatment of concurrency and parallelism has been strongly influenced by CSP [88].

CHAPTER III

PERFORMANCE EVALUATION

In this chapter, the performance of deflection routing in two-connected toroidal deflection networks: the Manhattan Street Network (MSN) and the Highway Transfer Network (HTN) is investigated. Both the MSN and the HTN have two incoming links and two outgoing links at each node and hence have the same degree of connectivity (the same number of transmitters and receivers) as the bidirectional loop network. Several contention resolution methods are examined. A routing algorithm for the HTN is proposed.

This chapter is organized as follows. In Section 3.1, the topological measures of the MSN, the HTN, and also the bidirectional loop network are examined and the maximum attainable throughputs of these networks are estimated. Section 3.2 gives the descriptions of the routing schemes for the MSN and the HTN. The dynamic behavior of the schemes has been studied by simulation. The results are reported. Routing in irregular networks and random routing are examined in Section 3.3 and Section 3.4. Section 3.5 summarizes this chapter.

Table 3.1: Diameter and Average Inter-nodal Distance

Network	Diameter	Average Distance
MSN ($n \times n$)	approx. n	approx. $n/2$
HTN ($n \times n$)	$2(n-1)$	$n^2/(n+1)$
BLN (N)	$N/2$	approx. $N/4$

3.1 Topological Measures

Topological measures can be used to estimate the performance of networks. In this section, the topological measures of the MSN, the HTN, and the bidirectional loop network are summarized and the maximum attainable throughputs of the networks are estimated. The definition of the *don't care node* is also given in this section.

3.1.1 Distance Measures

The *inter-nodal distance* between two nodes is the shortest path length (the smallest number of hops) between the nodes. The *diameter*, which is the maximum inter-nodal distance between any two nodes, and the average inter-nodal distance are directly related to packet delays in communication networks (though delays are not solely determined by distances). Table 3.1 shows the diameters and the average inter-nodal distances of the MSN, the HTN, and the bidirectional loop network (BLN). Note that the values for the MSN are approximate and are due to [21, 76].

The *deflection penalty* of a network is (the least upper bound on) the number of "extra" hops that a packet has to make after a deflection. For $n \times n$ HTNs, the deflection penalty is n, whereas it is 4 (which is constant) for MSNs of all sizes.

3.1.2 Attainable Throughput

The *throughput* of a network is the average number of packets accepted into the network per slot (one unit of time). The maximum attainable throughput of a network can be estimated based on its topological properties. If all links are one slot in length (i.e., every packet travels one link for each time slot), the average inter-nodal distance gives the average amount of time a packet must spend in the network before it reaches its destination. In other words, the average inter-nodal distance gives the average cost per packet transmission in terms of time or the number of links. Given the following assumptions:

- the network is symmetric,
- all links are one slot in length,
- all packets are routed along the shortest paths to their destinations,
- an infinite amount of buffer space is available at each node so that no packets are deflected or lost,
- the traffic is uniform (i.e., for each packet, any node is equally likely as a destination), and
- the packet arrival rate is the same at every node,

the maximum attainable throughput T can be expressed as

$$T = \frac{L}{\bar{h}}, \tag{3.1}$$

where L is the total number of links in the network and $\bar{h}$ is the average inter-nodal distance [7].

Table 3.2: Maximum Attainable Throughput

MSN ($n \times n$)	HTN ($n \times n$)	BLN
$4n$	$2n$	8

The maximum attainable throughputs (approximate values) of the MSN, the HTN, and the bidirectional loop network (BLN) are shown in Table 3.2. The values for the MSN and the HTN increase with the network size, whereas the value for the bidirectional loop network is constant.

From formula 3.1, the *nodal throughput*, which is the throughput per node, can also be estimated (T/N, where N is the number of nodes in the network). Naturally, a network cannot be operated at the packet arrival rate to a node exceeding the maximum attainable nodal throughput.

3.1.3 Don't Care Node

The number of deflections that packets get should be as small as possible so that the packets travel shorter distances and arrive at their destinations in the order of their transmissions. A node is a *don't care node* for a packet if all outgoing links of the node lie on the shortest paths to the packet's destination node. A packet cannot be deflected at its *don't care* nodes (the larger the number of *don't care* nodes, the smaller the number of deflected packets). The density (fraction) of *don't care* nodes for a packet is approximately $1/2$ in MSNs [35] and it is $(n-1)^2/n^2$ in $n \times n$ HTNs.

3.2 Routing

This section gives routing schemes for the MSN and the HTN. The scheme for the MSN is based on [54]. The schemes with several contention resolution (CR) methods

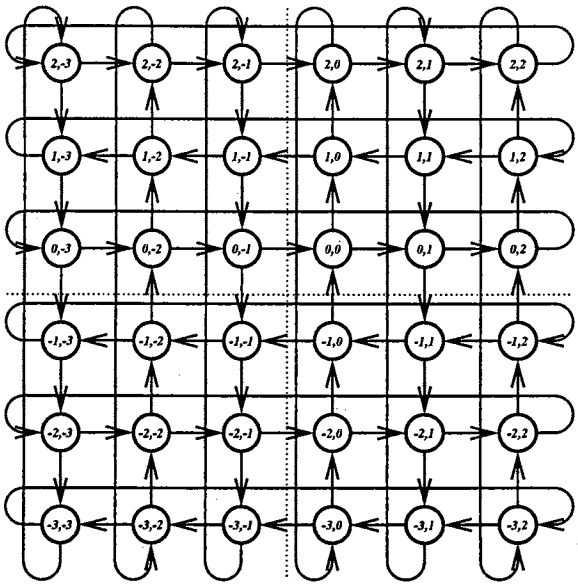

Figure 3.1: Relative addresses in an MSN.

were tested by simulation. The results are reported. It is assumed that the links of a network are locally labeled at each node with a globally consistent orientation and the nodes are sequentially numbered (addressed) according to the orientation (i.e., the nodes are integer row-column addressed). An outgoing link of a node is a *preferred link* of a packet at the node if the link lies on a shortest path to the packet's destination node. The schemes try to route packets along their preferred links.

3.2.1 Routing Scheme for MSN

Routing in the MSN involves address transformation. For each packet, the relative address of the current node, at which the packet is in transit, is calculated with respect to the destination node, which is considered to be in the center of the network and has the address $(0, 0)$. Figure 3.1 shows relative addresses in an MSN. The destination node is located at the lower left corner in the upper right quadrant with its outgoing links directed toward increasing numbered nodes.

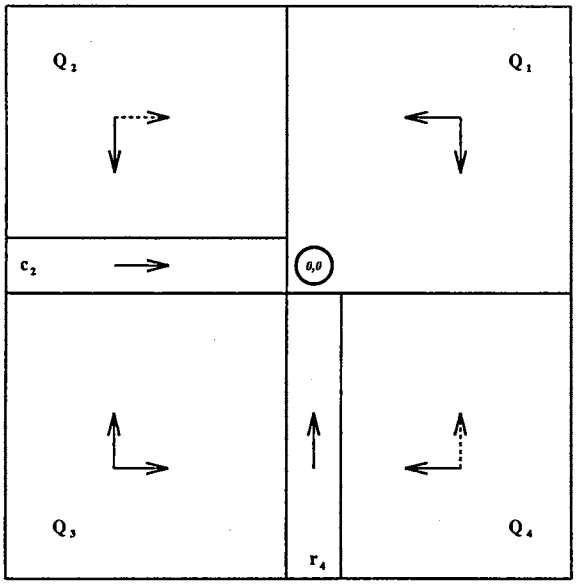

Figure 3.2: Routing (link selection) rules for the MSN.

Outgoing links for packets are selected based on the relative address of the current node. The rules are given in Figure 3.2. A node with relative address (r, c) is in Q_1 if $r \geq 0$ and $c \geq 0$; Q_2 if $r > 0$ and $c < 0$; c_2 if $r = 0$ and $c < 0$; Q_3 if $r < 0$ and $c < 0$; Q_4 if $r < 0$ and $c > 0$; r_4 if $r < 0$ and $c = 0$. Solid arrows indicate preferred links and dashed arrows indicate alternate links. For example, if the relative address of the current node is in Q_1, a link directed to the left and a link directed down are preferred links. An alternate link in Q_2 (Q_4) may be selected for a packet if the current node does not have a preferred link. Note that the direction of a link in Figure 3.1 may not be the same as the direction of the link before address transformation. The detailed algorithm, from which the simulator program is built, is given in Figures 3.9 and 3.10. In Figure 3.9, a packet is currently at (r_s, c_s) and its destination is (r_d, c_d). The variable N_r (N_c) is the number of rows (columns) in the network. The vertical (horizontal) link is a preferred link if $0 \in S$ (if $1 \in S$). In Figure 3.10, the variable S_0 (S_1), which is calculated by the algorithm in Figure 3.9,

indicates the preferred links of the packet arrived from the vertical (horizontal) link. For an empty slot, S_0 (or S_1) is assumed to be $\emptyset$. The variable $config$ indicates one of the configurations in Figure 3.14. The variable *random* is a random variable that is either 0 or 1. Contentions are resolved in a random manner in the algorithm.

3.2.2 Routing Scheme for HTN

Routing in the HTN is extremely easier than in the MSN. The routing scheme does not require calculation of relative addresses and quadrants or examination of link directions.

For typical source (current) and destination nodes in an HTN, both two outgoing links of the source (current) node lie on the shortest paths to the destination node. A packet does not have any preference in moving directions until it reaches one of its *care* nodes (*critical nodes*), which are on the same row or column as the destination node. Hence, a packet can be routed in a random manner until it reaches a *care* node. Upon reaching a care node, the packet must go straight to its destination. Otherwise, the packet gets a large deflection penalty. Figure 3.3 illustrates this routing scheme. The pseudocode for the scheme is given in Figures 3.11 and 3.12. In Figure 3.11, a packet is currently at (r_s, c_s) and its destination is (r_d, c_d). The vertical (horizontal) link is the (only) preferred link if $S = 0$ (if $S = 1$). In Figure 3.12, the variable S_0 (S_1), which is calculated by the algorithm in Figure 3.11, indicates the preferred link of the packet arrived from the vertical (horizontal) link. For an empty slot, S_0 (or S_1) is assumed to be 2. The variable $config$ indicates one of the configurations in Figure 3.14. The variable *random* is a random variable that

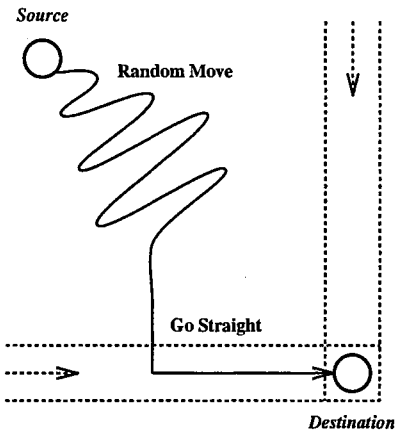

Figure 3.3: A path from a source to a destination in the HTN.

is either 0 or 1. Contentions are resolved in a random manner in the code. In the HTN, the distinctions of links are important only at *care* nodes that are on the same rows or columns as destination nodes. This fact has produced the simple switching algorithm. The routing scheme is expected to inherit the properties of random routing [54] (i.e., it is easy to implement and is tolerant of network irregularities). Note that this routing scheme does not depend on the sequential ordering of the nodes. The scheme only assumes that the nodes on the same row (column) have the same row (column) address.

Because a packet may be deflected only while it is moving straight to its destination, shortening the distance of the final straight move will reduce the probability of deflection. The switching algorithm developed based on this idea is shown in Figure 3.13, which replaces the last else-block of the algorithm in Figure 3.12. In Figure 3.13, two packets are currently at their common *don't care* node (r_s, c_s). Nodes (r_0, c_0) and (r_1, c_1) are the destinations of the packets arrived from the vertical link and the horizontal link, respectively. The variable N_r (N_c) is the number of rows (columns) in the network. The variable $config$ indicates one of the configurations in

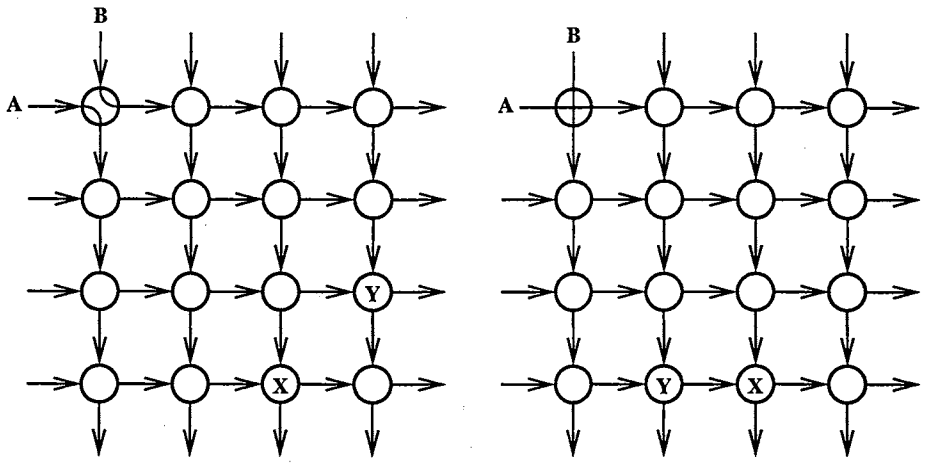

Figure 3.4: Switching in the HTN DA algorithm.

Figure 3.14. The variable *random* is a random variable that is either 0 or 1. For the sake of brevity, the case that only one packet is in transit is omitted in Figure 3.13. This algorithm is referred to as the Deflection Avoidance (DA) algorithm. At each node, the algorithm tries to forward a packet to the node in the direction that does not shorten the (shortest) distance to a *care* node. It routes the packets along their *don't care* nodes as long as possible so that deflections are avoided. Note that there may be a conflict between two packets coming into a node. Examples of switching is shown in Figure 3.4. (Nodes X and Y are the destinations of packets A and B, respectively.)

As an alternative, one may consider using simple zig-zag routing, which forwards packets in alternating directions at every hop (as long as possible). Such a scheme has been tested. The results show no significant performance improvement over the scheme with random packet moves.

Table 3.3: Throughput and Delay of Bidirectional Loop

Size	Nodal Throughput	Average Delay
64	.115	16.503
100	.075	25.536
144	.053	36.526
196	.040	49.482
256	.031	64.501

3.2.3 Simulation Results

The performance of the routing schemes was studied through simulation. At a low link utilization (under a light traffic load), fewer deflections are expected and so are shorter delays. We are interested in the performance of the networks at a high link utilization (under a heavy traffic load). The networks were operated at the link utilization of 100%, which may not be achieved in store-forward networks without creating congestion. A packet with a random destination (uniform traffic) was injected for every empty slot obtained by nodes. All links were assumed to be one slot in length. Delays were measured in slots (hops). (Note that the duration of a (time) slot may differ in different types of networks.) Although the use of a small amount of buffer (queue) space for each link (to hold a few packets) could reduce the number of deflections, no such space was used. At the level of current technology, buffering may be difficult in optical networks without electro-optic and optic-electro conversions, which prevent high speed nodal processing [7].

Table 3.3 shows the nodal throughput and the average delay of the bidirectional loop network. The results are close to the maximum attainable values of the network. The nodal throughput decreases linearly as the number of nodes increases. A packet generated for an empty slot was injected only if the packet would be forwarded along the shortest path to its destination. Otherwise, the generated packet was discarded.

Table 3.4: Nodal Throughput and Average Delay

Size $n \times n$	Random CR		Hop Counter CR		Deflection Counter CR	
	Nodal Throughput	Average Delay	Nodal Throughput	Average Delay	Nodal Throughput	Average Delay
Manhattan Street Network						
8×8 (64)	.201	9.941	.217	9.208	.205	9.739
10×10 (100)	.164	12.227	.179	11.149	.169	11.812
12×12 (144)	.140	14.250	.154	12.971	.146	13.734
14×14 (196)	.123	16.290	.136	14.678	.129	15.550
16×16 (256)	.111	18.057	.123	16.281	.116	17.248
20×20 (400)	.093	21.597	.103	19.383	.098	20.471
24×24 (576)	.080	24.877	.090	22.316	.085	23.502
28×28 (784)	.071	28.087	.080	25.140	.076	26.472
32×32 (1024)	.064	31.142	.072	27.896	.068	29.299
Highway Transfer Network						
8×8 (64)	.199	10.053	.202	9.888	.198	10.094
10×10 (100)	.154	12.941	.157	12.709	.154	12.974
12×12 (144)	.126	15.826	.129	15.551	.126	15.893
14×14 (196)	.107	18.747	.109	18.390	.106	18.853
16×16 (256)	.092	21.618	.094	21.210	.092	21.787
20×20 (400)	.073	27.492	.074	26.892	.072	27.699
24×24 (576)	.060	33.255	.061	32.598	.060	33.590
28×28 (784)	.051	39.101	.052	38.230	.051	39.517
32×32 (1024)	.044	44.968	.046	43.906	.044	45.431

Hence, the network was operated at a link utilization slightly lower than 100%. In the bidirectional loop network, the delays remain the same under lower traffic loads since no deflections can occur in the network.

Several contention resolution (CR) methods have been proposed for the MSN in the literature. In this study, random CR, hop counter CR, and deflection counter CR have been tested for both the MSN and the HTN in various sizes. (The performance of 8×8 MSNs with those CR methods has been reported in [75]. As can be seen from Table 3.4 and Figures 3.5–3.6, for the network size 8×8, there is not much difference in performance between the MSN and the HTN.) Hop counter CR and deflection counter CR deflect the packets with lower counter values, which are increased by 1 for each hop or deflection. Tie-breaking is done by using randomization. The nodal throughputs and the average delays of the networks with those CR methods

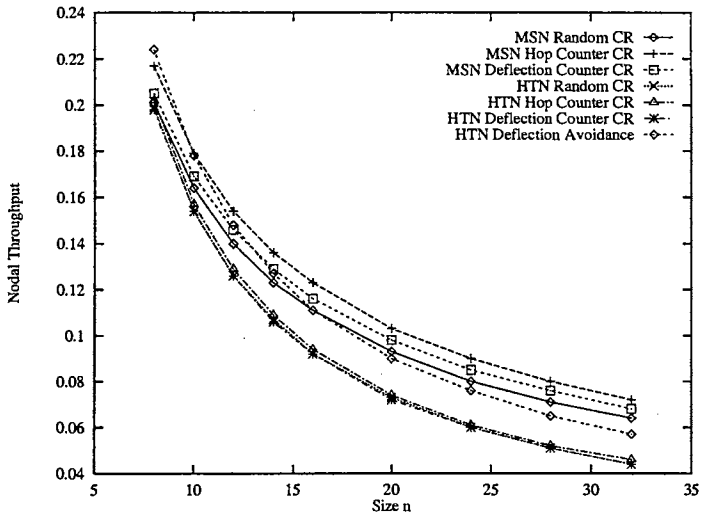

Figure 3.5: Nodal throughput.

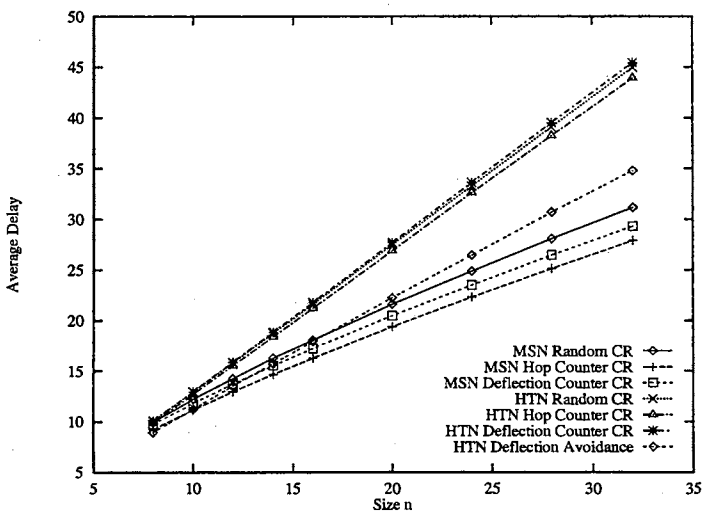

Figure 3.6: Average delay.

are shown in Table 3.4 and are also plotted in Figures 3.5 and 3.6. (Unfortunately, the nodal throughputs of both networks decrease with increasing numbers of nodes under uniform traffic; however, their (total) throughputs increase as shown in Figure 3.7. Note that all networks tested in the simulation have even numbers of rows and columns since the MSN is defined only for even numbers of rows and columns.) Tables 3.5–3.10 show the probability density of the number of deflections experienced by packets. The maximum delays observed are also shown in the tables.

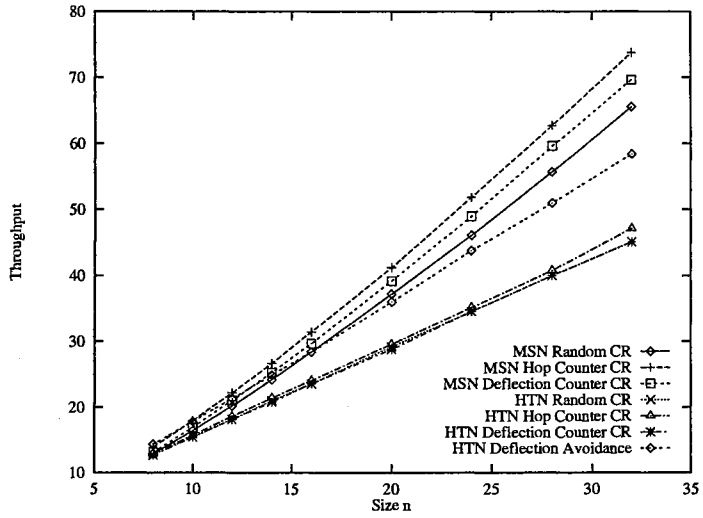

Figure 3.7: Throughput.

Although it is better to keep the networks in square shapes, they tend to perform well even if they are non-square. Table 3.11 shows the nodal throughputs and the average delays of some non-square networks (Hop Counter CR). The results are close to those of square 256 and 400 node networks though the difference becomes larger for networks with a smaller row/column ratio.

Note that if links operate at 1Gb/s, the nodal throughput of 0.1 corresponds to the data rate of 100Mb/s to a node.

Although the maximum attainable throughput of the MSN is approximately twice as much as that of the HTN, the difference in the actual throughput is much smaller for the range of network size in Table 3.4.

Counter based CR methods reduce the maximum delays observed under random CR. For both networks, hop counter CR performs better than other CR methods though the differences are small in the HTN. The effect of the CR methods in the HTN can be seen in Tables 3.6, 3.8, and 3.10.

It is interesting to see that deflection counter CR in the MSN keeps the max-

imum number of deflections experienced by packets smaller than hop counter CR (Tables 3.7 and 3.9) though the performance of hop counter CR is better (higher throughput and lower delay) than that of deflection counter CR. Deflection counter CR deflects packets regardless of the distances they have traveled. In the MSN, a packet has to pass through a *care* node at every other hop while moving inside a quadrant. Hence, the packets destined for distant nodes have higher chances of getting deflected. In the networks with hop counter CR, the packets destined for distant nodes tend to be deflected fewer times than the packets destined for closer nodes. As a result, the average delay is reduced.

As can be seen from Tables 3.6, 3.8, and 3.10, the probability density of the number of deflections experienced by packets in the HTN is much the same for the range of network size in the tables, whereas it varies in the MSN for the same range of network size. (The majority of the packets in the HTN reach their destinations without getting deflected.)

Figure 3.8 shows the relative throughputs (the actual throughput divided by the maximum attainable throughput) of the MSN and the HTN (with hop counter CR). The value indicates the efficiency of the routing scheme (the higher the relative throughput, the more efficient the routing scheme). The relative throughput of the MSN increases with the network size. On the other hand, the relative throughput of the HTN decreases as the number of nodes increases although it is higher than that of the MSN for the range of network size in Figure 3.8.

Table 3.12 shows the nodal throughput and the average delay of the DA algorithm with hop counter CR. The throughput and the delay are also plotted in Fig-

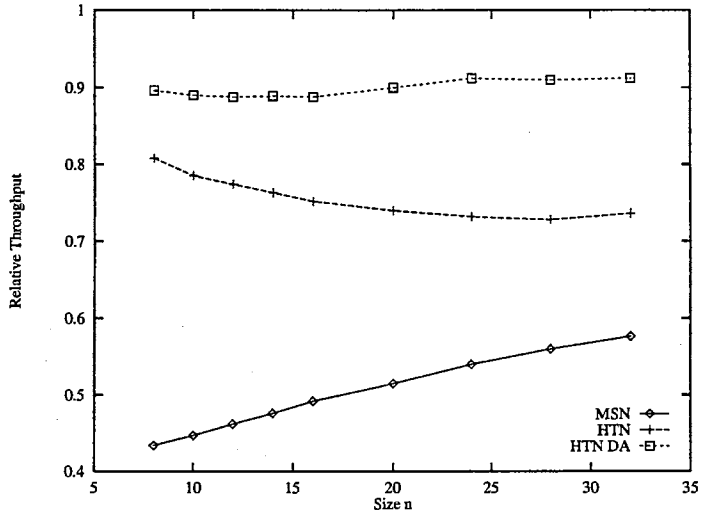

Figure 3.8: Relative throughput.

ures 3.5 and 3.6. (The (total) throughput is plotted in Figure 3.7.) The deflections density is shown in Table 3.13. Although the scheme also increases the complexity of routing, the improvement in performance is remarkable. The number of packets that were delivered without any deflection increases with the network size. In a 256×256 network, more than 98% of packets were delivered to their destinations without a deflection. As can be seen from Figure 3.8, the DA algorithm (gradually) increases the relative throughput of the HTN with the number of nodes.

It is noticed that formula 3.1 with $\bar{h}$ as the average delay approximates the actual throughputs of both the MSN and the HTN.

The values reported in Tables 3.3, 3.4, and 3.12 (except the last three rows of Table 3.12) are the averages of measurements from 5 independent simulation runs. In each run, a network was simulated for 5000 slots and the first 500 slots of data were discarded. Each value lies within the 95% confidence interval of width less than 3% of the value. (For most results, the width of the 95% confidence interval is less than 1% of the value.) Each value in Tables 3.5–3.10, 3.13 and the last three rows

```
{ address transformation }
r := (r_s + N_r + N_r/2 − (c_d mod 2) − r_d) mod N_r;
c := (c_s + N_c + N_c/2 − (r_d mod 2) − c_d) mod N_c;
if r_d mod 2 = 0 then c := c − N_c/2
else c := N_c/2 − 1 − c;
if c_d mod 2 = 0 then r := r − N_r/2
else r := N_r/2 − 1 − r;

{ link selection }
S := ∅;
if (r = 0) and (c < 0) then S := {1}
else if (r < 0) and (c = 0) then S := {0}
else if (r ≥ 0) and (c ≥ 0) then begin
  if r mod 2 ≠ 0 then S := {1};
  if c mod 2 ≠ 0 then S := S ∪ {0}
end
else if (r > 0) and (c < 0) then begin
  if c mod 2 ≠ 0 then S := {0}
  else if r mod 2 = 0 then S := {1}
end
else if (r < 0) and (c < 0) then begin
  if r mod 2 = 0 then S := {1};
  if c mod 2 = 0 then S := S ∪ {0}
end
else if (r < 0) and (c > 0) then begin
  if r mod 2 ≠ 0 then S := {1}
  else if c mod 2 = 0 then S := {0}
end
```

Figure 3.9: A routing (link selection) algorithm for the MSN.

of Table 3.12 is based on a 10 000 slot simulation run with the first 500 slots of data discarded.

3.3 Routing in Irregular Topologies

As noted earlier, deflection routing also works in irregular topologies, which may be incrementally constructed. This section presents simulation results for deflection routing in irregular HTNs shown in Figure 3.16. (Arrows indicating the directions of links are omitted in the figure.)

The networks were created in a random manner. Each of networks 1 and 2 was

```
if ((S0 = {0,1}) and (S1 = {1})) or
   ((S0 = {0})   and (S1 = {0,1})) or
   ((S0 = {0})   and (S1 = {1})) or
   ((S0 = {0})   and (S1 = ∅)) or
   ((S0 = ∅)     and (S1 = {1})) then
  config := 0
else
if ((S0 = {0,1}) and (S1 = {0})) or
   ((S0 = {1})   and (S1 = {0,1})) or
   ((S0 = {1})   and (S1 = {0})) or
   ((S0 = {1})   and (S1 = ∅)) or
   ((S0 = ∅)     and (S1 = {0})) then
  config := 1
else
  config := random
```

Figure 3.10: An algorithm to select a switching configuration in the MSN.

```
if cs = cd then S := 0
else if rs = rd then S := 1
else S := 2
```

Figure 3.11: A routing (link selection) algorithm for the HTN.

created by deleting randomly-chosen 144 nodes from a 20×20 network. For each of networks 3 and 4, 320 nodes were deleted from a 24×24 network. All the networks have 256 nodes. (The topologies follow the definition of the quasi-torus given in [18], except that the irregular HTNs use unidirectional links.) Networks 1 and 2 (3 and 4) are denser (sparser) than networks 3 and 4 (1 and 2).

The performance of two algorithms (with hop counter CR) under random traffic is shown in Table 3.14. The basic algorithm is the one that involves random packet moves presented earlier. The second algorithm, which is referred to as the lookahead algorithm in Table 3.14, is shown in Figure 3.15. In Figure 3.15, a packet is currently at (r_s, c_s) and its destination is (r_d, c_d). The vertical and horizontal outgoing links of (r_s, c_s) are connected to (r_n, c_s) and (r_s, c_n), respectively. The vertical (horizontal)

```
if ((S0 = 0) and (S1 = 1)) or
   ((S0 = 0) and (S1 = 2)) or
   ((S0 = 2) and (S1 = 1)) then
   config := 0
else
if ((S0 = 1) and (S1 = 0)) or
   ((S0 = 1) and (S1 = 2)) or
   ((S0 = 2) and (S1 = 0)) then
   config := 1
else
   config := random
```

Figure 3.12: An algorithm to select a switching configuration in the HTN.

link is a preferred link if $0 \in S$ (if $1 \in S$). The switching algorithm for the lookahead algorithm is the same for the MSN. The algorithm requires each node to know their downstream nodes.

As expected, the basic algorithm did not perform well in sparse networks (networks 3 and 4). The lookahead algorithm performed well in all the networks. The results are comparable to those of non-irregular 16×16 networks (256-node networks).

Each value in Table 3.14 is based on a 10 000 slot simulation run with the first 500 slots of data discarded.

Note that the basic algorithm and the lookahead algorithm work in the same manner in non-irregular networks.

The DA algorithm (with or without the lookahead algorithm) may not work well in irregular networks. In certain cases, the DA algorithm may cause packets to loop. Consider the network in Figure 3.17. Suppose there is a packet at node $(1, 1)$ destined for node $(2, 2)$. At node $(1, 1)$, if the lookahead algorithm is used, neither outgoing link will be considered as a preferred link of the packet. (Since both the

$x_0 := (c_0 - c_s + N_c) \bmod N_c$;
$y_0 := (r_0 - r_s + N_r) \bmod N_r$;
$x_1 := (c_1 - c_s + N_c) \bmod N_c$;
$y_1 := (r_1 - r_s + N_r) \bmod N_r$;
if $((x_0 < y_0)$ **and** $(x_1 > y_1))$ **or**
  $((x_0 > y_0)$ **and** $(x_1 > y_1)$ **and** $(y_0 > y_1))$ **or**
  $((x_0 < y_0)$ **and** $(x_1 < y_1)$ **and** $(x_0 < x_1))$ **or**
  $((x_0 < y_0)$ **and** $(x_1 = y_1))$ **or**
  $((x_0 = y_0)$ **and** $(x_1 > y_1))$ **then**
  $config := 0$
else
if $((x_0 > y_0)$ **and** $(x_1 < y_1))$ **or**
  $((x_0 > y_0)$ **and** $(x_1 > y_1)$ **and** $(y_0 < y_1))$ **or**
  $((x_0 < y_0)$ **and** $(x_1 < y_1)$ **and** $(x_0 > x_1))$ **or**
  $((x_0 > y_0)$ **and** $(x_1 = y_1))$ **or**
  $((x_0 = y_0)$ **and** $(x_1 < y_1))$ **then**
  $config := 1$
else
  $config := random$

Figure 3.13: The deflection avoidance algorithm for the HTN.

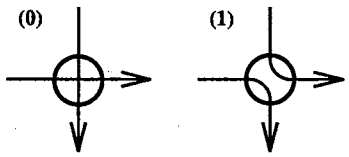

Figure 3.14: Switching configurations.

links are non-preferred links of the packet, the DA algorithm will not be used at node $(1,1)$.) The packet may be forwarded to either node ($(1,3)$ or $(3,1)$). If the lookahead algorithm is not used at node $(1,1)$, both outgoing links will be considered as preferred links of the packet. The DA algorithm will find that neither of the links is better (for the packet) than the other (i.e., the packet can be forwarded to either node). (The DA algorithm uses the distances given by the node addresses.) In either case, if the packet is forwarded to node $(1,3)$, the DA algorithm will route the packet to node $(1,0)$ and then back to $(1,1)$. Analogously, if the packet is forwarded to node $(3,1)$, the DA algorithm will route the packet to node $(0,1)$ and then back to $(1,1)$.

$S := \emptyset$;
if $(c_s < c_n)$ **and not** $((c_s \leq c_d)$ **and** $(c_d < c_n))$ **then**
  $S := S \cup \{1\}$
else if $(c_s > c_n)$ **and** $(c_n \leq c_d)$ **and** $(c_d < c_s)$ **then**
  $S := S \cup \{1\}$;
if $(r_s < r_n)$ **and not** $((r_s \leq r_d)$ **and** $(r_d < r_n))$ **then**
  $S := S \cup \{0\}$
else if $(r_s > r_n)$ **and** $(r_n \leq r_d)$ **and** $(r_d < r_s)$ **then**
  $S := S \cup \{0\}$

Figure 3.15: A routing algorithm for irregular HTNs.

Note that if the DA algorithm is not used, the packet at node $(3, 1)$ may be forwarded to node $(3, 2)$ and the loop is avoided; probabilistically, the packet has a chance to reach its destination.

As noted in [54], the routing scheme for the MSN presented earlier may not work in irregular MSNs. For the MSN, incremental construction of the network is not straightforward and is one of the major disadvantages of the MSN.

3.4 Random Routing

Random routing is not a practical routing method. However, random routing is tolerant of network irregularities and easy to implement. It may be used when certain node or link failures are causing packets to loop (i.e., the primary, non-random routing scheme is not functioning).

Random routing was tested on both the MSN and the HTN. The results are shown in Table 3.15. Each value in Table 3.15 is based on a 50 000 simulation run with the first 500 slots of data discarded. Interestingly, random routing performed better in the HTN than in the MSN even though the MSN connects nodes closer than the HTN. In the MSN, a packet can travel (stay) only in a certain local area of the network, whereas in the HTN, a packet cannot stay only in a certain local area.

Packets explore the network faster in the HTN than in the MSN. This fact resulted a better performance in the HTN although the performance is no better than that of linear topology networks. The performance of the MSN and the HTN tends to degrade sublinearly with the network size.

3.5 Summary

A study of two-connected toroidal deflection networks has been presented. The throughputs and delays of the Manhattan Street Network (MSN) and the Highway Transfer Network (HTN) are examined. Both networks increase their throughputs with the number of nodes. Numerical results carried out by simulation are reported. The results show that the HTN, despite its large deflection penalty, can achieve approximately 80% of the MSN's throughput for networks with a few hundred nodes. The results also show that an effective use of *don't care* nodes can increase the HTN's throughput to 90% of the MSN's. As the network size increases, the scheme decreases the number of deflections. In a 256×256 HTN, over 98% of packets were delivered without any deflection. This fact is notable considering the network was operated at the link utilization of 100%. The MSN increases the relative throughput with the number of nodes, whereas the HTN with a simple routing scheme does not (although the relative throughput of the HTN is higher than that of the MSN for the network sizes examined in the simulation). For the HTN, a switching algorithm that significantly reduces the number of deflections and increases the relative throughput has been proposed and demonstrated.

Routing in irregular topologies has also been studied. A simple routing scheme for irregular HTNs has been proposed and tested. The results are comparable to those

for non-irregular networks.

Random routing, which is tolerant of network irregularities, has also been examined. For pure random routing, the HTN provides a better performance than the MSN.

Although deflection routing may waste the transmission capacity, the stable network behavior provided by deflection routing is very attractive for MANs. Note that in order to avoid congestion, the general store-forward networks cannot be operated at a high link utilization.

Table 3.5: Deflections Density in MSN (Random CR)

# Deflections	8×8	16×16	32×32
0	.43	.27	.14
1	.22	.20	.14
2	.13	.15	.13
3	.079	.10	.11
4	.047	.078	.098
5	.029	.055	.080
6	.018	.039	.063
7	.011	.027	.049
8	.0067	.019	.038
9	.0040	.013	.029
10	.0025	.0096	.022
11	.0015	.0067	.016
12	.00086	.0050	.012
13	.00056	.0033	.0093
14	.00033	.0024	.0073
15	.00027	.0017	.0051
16	.00010	.0012	.0039
17	.000049	.00088	.0030
18	.000065	.00055	.0022
19	.000024	.00041	.0016
20	.000024	.00032	.0013
21	.0000081	.00021	.00095
22	.0	.00010	.00066
23	.0	.000081	.00049
24	.0000081	.000074	.00038
25	.0	.000074	.00027
26	.0	.000048	.00021
27	.0	.0000074	.00016
28	.0	.000022	.00013
29	.0	.000014	.000099
30	.0000081	.0000074	.000062
31	.0	.000011	.000043
32	.0	.000011	.000038
33	.0	.0000074	.000030
34	.0	.0	.000015
35	.0	.0	.000014
36	.0	.0	.0000095
37	.0	.0000037	.0000095
38	.0	.0	.0000031
39	.0	.0	.0000063
40	.0	.0	.0000047
41	.0	.0	.0000015
42	.0	.0	.0000015
43	.0	.0	.0
44	.0	.0	.0
45	.0	.0	.0
46	.0	.0	.0
47	.0	.0	.0
48	.0	.0	.0000015
49	.0	.0	.0
Max Delay	108	153	176

Table 3.6: Deflections Density in HTN (Random CR)

# Deflections	8×8	16×16	32×32
0	.74	.73	.72
1	.17	.17	.17
2	.053	.061	.065
3	.016	.020	.022
4	.0050	.0072	.0081
5	.0016	.0025	.0030
6	.00046	.00094	.0011
7	.00014	.00031	.00037
8	.000057	.000053	.00016
9	.0000082	.000022	.000039
10	.000016	.000022	.000032
11	.0	.0	.0
12	.0	.0	.0
13	.0	.0	.0000023
14	.0	.0	.0000023
15	.0	.0	.0
Max Delay	84	185	461

Table 3.7: Deflections Density in MSN (Hop CR)

# Deflections	8×8	16×16	32×32
0	.28	.15	.084
1	.30	.20	.12
2	.25	.23	.16
3	.11	.20	.17
4	.027	.12	.16
5	.0033	.053	.12
6	.00022	.016	.082
7	.0000075	.0032	.044
8	.0	.00041	.020
9	.0	.000050	.0075
10	.0	.0000067	.0022
11	.0	.0	.00059
12	.0	.0	.00012
13	.0	.0	.000017
14	.0	.0	.0000042
15	.0	.0	.0
Max Delay	24	37	60

Table 3.8: Deflections Density in HTN (Hop CR)

# Deflections	8×8	16×16	32×32
0	.69	.67	.66
1	.25	.26	.26
2	.041	.056	.064
3	.0014	.0027	.0034
4	.0	.000026	.000033
5	.0	.0	.0
Max Delay	34	74	152

Table 3.9: Deflections Density in MSN (Deflec. CR)

# Deflections	8×8	16×16	32×32
0	.23	.096	.036
1	.29	.14	.057
2	.31	.25	.11
3	.13	.29	.20
4	.016	.16	.28
5	.00045	.032	.22
6	.0	.0021	.070
7	.0	.000088	.0071
8	.0	.0	.00022
9	.0	.0	.0000075
10	.0	.0	.0
Max Delay	29	43	63

Table 3.10: Deflections Density in HTN (Deflec. CR)

# Deflections	8×8	16×16	32×32
0	.67	.64	.62
1	.28	.29	.30
2	.040	.056	.066
3	.0011	.0021	.0029
4	.0	.0000089	.000028
5	.0	.0	.0
Max Delay	38	81	177

Table 3.11: Performance of Non-square Networks

		MSN		HTN	
Network Size	Row : Col Ratio	Nodal Throughput	Average Delay	Nodal Throughput	Average Delay
14×18 (252)	1 : 1.29	.122	16.433	.094	21.368
16×24 (384)	1 : 1.50	.101	19.802	.073	27.344
12×20 (240)	1 : 1.67	.119	16.741	.092	21.840
12×32 (384)	1 : 2.67	.087	23.054	.061	32.717

Table 3.12: Throughput and Delay of HTN (DA)

Size	Nodal Throughput	Average Delay
8×8	.224	8.942
10×10	.178	11.246
12×12	.148	13.534
14×14	.127	15.741
16×16	.111	17.942
20×20	.090	22.243
24×24	.076	26.461
28×28	.065	30.690
32×32	.057	34.784
64×64	.030	67.324
128×128	.015	131.565
256×256	.008	259.370

Table 3.13: Deflections Density in HTN (DA)

# Deflections	8×8	16×16	32×32	64×64	128×128	256×256
0	.78	.83	.88	.93	.96	.98
1	.19	.15	.010	.060	.031	.016
2	.017	.012	.0071	.0034	.0015	.00067
3	.00036	.00019	.000075	.000015	.0000033	.00000083
4	.0	.0000036	.0	.0	.0	.0
5	.0	.0	.0	.0	.0	.0
Max Delay	32	70	129	233	475	910

Table 3.14: Nodal Throughput and Average Delay of Irregular HTN

	Basic		Lookahead	
Network	Nodal Throughput	Average Delay	Nodal Throughput	Average Delay
1	.081	24.661	.099	20.120
2	.081	24.653	.099	20.120
3	.071	28.326	.095	21.005
4	.071	28.307	.097	20.725

Table 3.15: Nodal Throughput and Average Delay of Random Routing

	MSN		HTN	
Network Size	Nodal Throughput	Average Delay	Nodal Throughput	Average Delay
10×10 (100)	.01780	112.321	.02199	90.974
20×20 (400)	.00375	531.513	.00507	393.185
30×30 (900)	.00152	1292.508	.00219	902.429

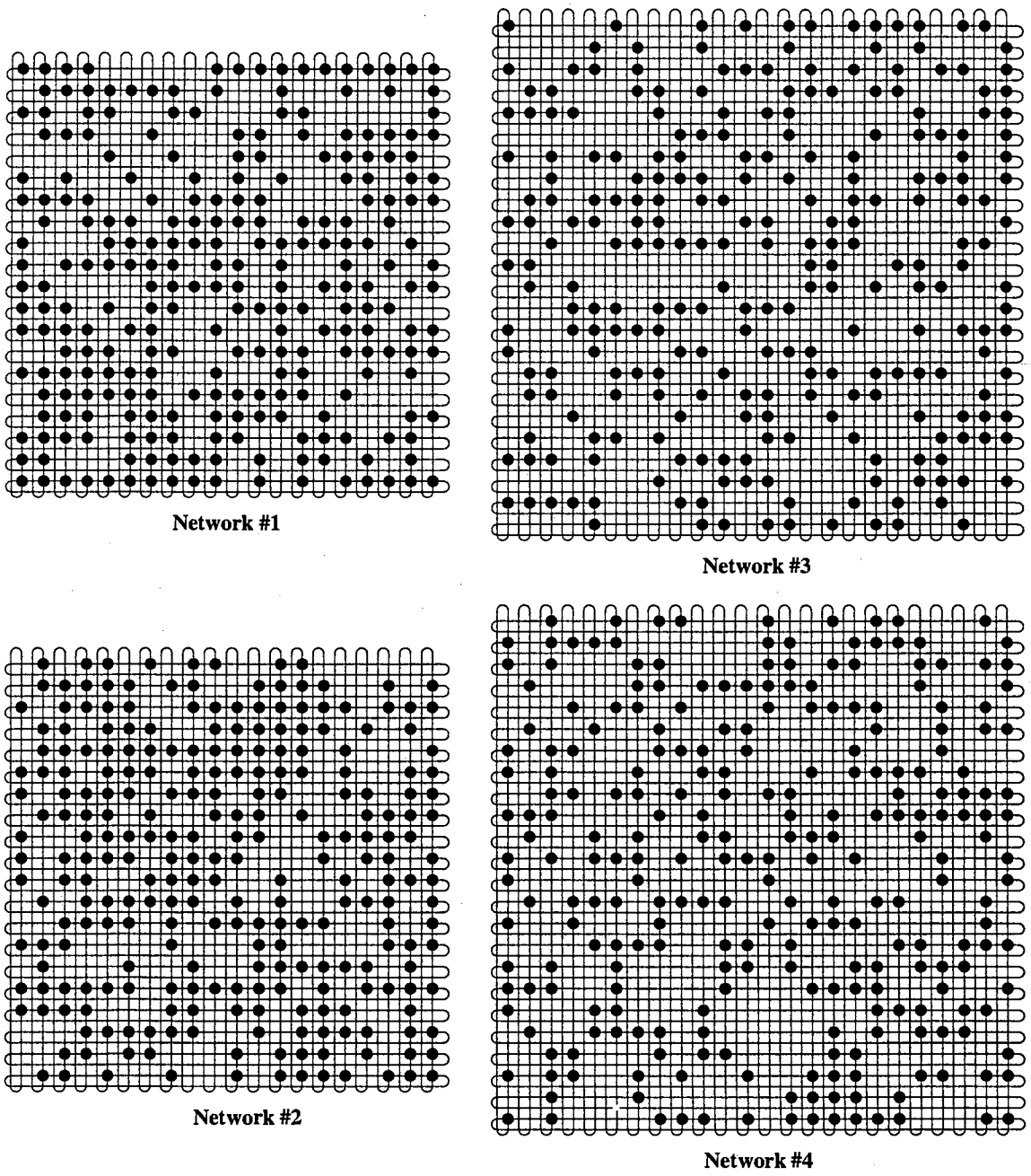

Figure 3.16: Irregular networks.

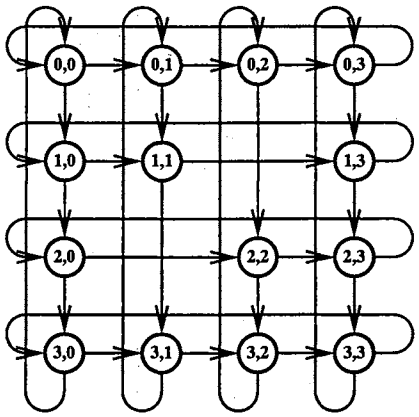

Figure 3.17: An irregular network.

CHAPTER IV

LIVELOCK PREVENTION

One problem uniquely associated with deflection networks is that packets may circulate indefinitely without reaching their destinations. This condition is known as *livelock.*

It has been shown that the use of randomization in contention resolutions provides a probabilistic guarantee that livelock is avoided [15, 56]. This chapter presents a method that provides a deterministic guarantee that livelock is avoided (i.e., every packet that is injected into a network is 100% guaranteed to reach its destination).

One (e.g., [10]) may argue that the use of hop counter values or time stamps (in packet headers) as priorities in contentions will prevent livelock since any packet will eventually have the largest counter value or be the oldest packet (i.e., the highest priority packet) that cannot be deflected by any other packet.

It may be rare, but is conceptually possible to have multiple oldest packets (packets with the same largest counter value) that are endlessly contending against each other and are not reaching their destinations. The proposed method eliminates such conditions. Thus, the eventual delivery of every injected packet is ensured.

We assume two-connected networks in which all links are one slot in length, but do not assume specific topologies (i.e., the method is topology independent). In order for the method to work in higher degree (node connectivity) networks, link assignments (switching) must be carefully done. We discuss this problem in a later section. It is assumed that a packet injected into a network will reach its destination (but not necessarily along a shortest path) if there are no contentions (deflections).

4.1 Method

The method uses two counters in the packet header. One is the hop counter and the other is the deflection counter. The hop counter is used as in previously studied contention resolution methods (e.g., [68]). The counter value is increased by 1 for each hop and the packet with a larger counter value has a higher priority in a contention. The use of the deflection counter differs from that in previous studies. (In previous studies, the deflection counter (if used) was used alone without the hop counter.) The deflection counter value of a packet is increased only when the packet loses a contention against another packet with the same hop and deflection counter values. Tie-breaking is done by using randomization. The loss of a contention against a packet with a larger hop counter value does not increase the deflection counter value. The value of the deflection counter is used in a contention between the packets with the same hop counter value (the packet with a larger deflection counter value will get the link). Initially (when a packet is injected into the network), both the hop counter value and the deflection counter value are 0.

The deflection counter values create classes among the packets with the same hop counter value. Let S be the set of packets with the same largest hop counter

value in a network at a certain time (measured in slots). The packets in S can be deflected only by other packets in S. The packets in S (as long as in the network) always have the hop counter value larger than the hop counter value of any other packet (not in S) at any time. Unless the packets in S continually contend against each other, they will reach their destinations. The packets with the same deflection counter value form a class in S. Contentions among the packets in S (if no packets in S have reached their destinations) will eventually divide S into n classes of size 1, where n is the number of packets in S. The n classes include one packet with the counter value $n-1$, one packet with the counter value $n-2, \ldots$, one packet with the counter value 1, and one packet with the counter value 0. The same two packets in S may contend against each other several times and also some (or even all) packets in S may reach their destinations before S is partitioned into n classes mentioned above. The establishment of the classes of size 1 is the establishment of a total order of the packets. There will be no more continual contentions among the packets when the size of every existing (non-empty) class becomes 1. The priorities of the packets will be based on the total order and hence no two packets will have the same priority. Each of the packets will reach its destination in a finite number of hops.

Clearly, a packet cannot involve in more than one contention at the same time. The proposed method is conceptually equivalent to the following sequential procedure. Imagine that S is a "bag" containing n packets (with the same hop counter value) and the deflection counter values of all packets are 0.

1. Draw a pair of packets (randomly) from S.

2. If both two packets have the same deflection counter value, select one of the

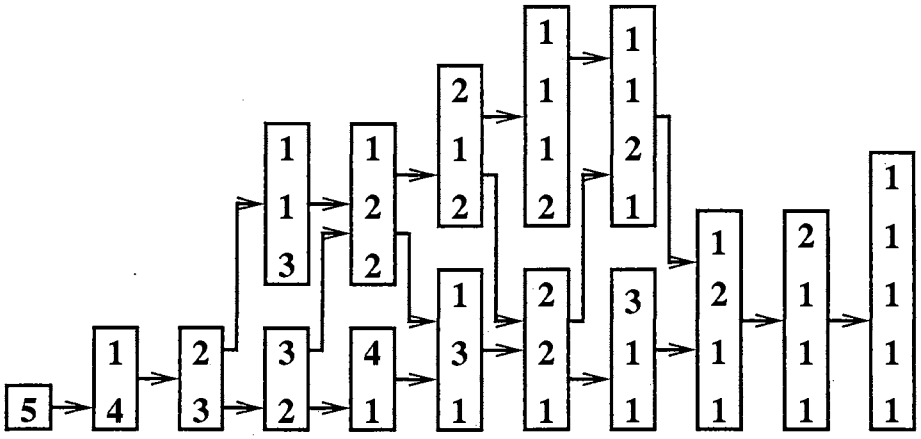

Figure 4.1: Process of partitioning a set of five packets into five classes of size 1.

packets (randomly) and increase the counter value of the packet by 1.

3. Put the packets back in S.

4. Go to Step 1.

Figure 4.1 illustrates the procedure. It shows the process of partitioning a set of five packets into five classes of size 1. A number in a box indicates the number of packets in a class. The number at the bottom of a box indicates the number of packets with the deflection counter value 0, the second number from the bottom indicates the number of packets with the deflection counter value 1, and so on. The existence of numbers at higher positions in a box indicates the existence of higher priority packets. An arrow indicates an occurrence of a contention between two packets in the same class. The procedure operates sequentially although in a real network, several contentions may occur at the same time. Multiple steps in the figure may be one slot in the network (e.g., the first two steps (two contentions) in the figure may happen in a single slot in the network).

Note that drawing in the procedure is not necessarily at random although in

such a case, the procedure may not create n classes of size 1. However, the packets that have not increased their deflection counter values for a certain number of iterations should be considered to be delivered to their destinations in the network and may be ignored in the procedure. A packet with the largest hop counter value will reach its destination within a finite number of hops (no more than the diameter given by the routing scheme) if it will not be deflected by other packets.

Let s_i ($\subseteq S$) denote the set (class) of packets with the deflection counter value i and $\#s_i$ denote the number of packets in s_i.

Clearly, if $\#s_0 = k$ for some $k > 0$, then $\#s_0 \leq k$ after an iteration of the procedure. Each time two packets are drawn from s_0, one of the packets increases its deflection counter value by 1 and hence $\#s_0$ decreases. Since drawing of packets is done at random, a pair of packets belonging to s_0 will eventually be drawn as the execution of the procedure continues. Therefore, $\#s_0$ will continue to decrease and will eventually be 1. The only packet left in s_0 may be drawn as the execution of procedure continues; however, the deflection counter of the packet will not be increased since the other packet (drawn with the packet in s_0) must be drawn from some other class and hence $\#s_0$ will remain 1.

If $\#s_i = 1$ and $\#s_{i+1} = k$ for some $k > 0$, then $\#s_{i+1} \leq k$ after an iteration of the procedure. In order for an iteration of the procedure to increase $\#s_{i+1}$, two packets must be drawn from s_i and hence $\#s_i$ must be at least 2. If $\#s_i = 1$, then an iteration will not increase $\#s_{i+1}$. The iteration may decrease $\#s_{i+1}$ by drawing a pair of packets from s_{i+1} if $\#s_{i+1} \geq 2$. If $\#s_i$ will remain 1 for a substantial number of iterations, then $\#s_{i+1}$ will decrease and will eventually become 1.

Inductively applying the arguments above, we can show that for any i, $\#s_i$ will eventually become 1.

Note that increasing the deflection counter value by 1 every time a packet is deflected as in a conventional method will not create the classes mentioned above; the packets in S may continually contend against each other and indefinitely increase their deflection counter values.

4.2 Simplified Method

It is possible to assign a node id and a link id to each packet. Generally, a packet stores its source node id in its header. Likewise, the id of the link from which the packet is injected into the network can also be stored in the header. (Each outgoing link at a node should have its own unique number that locally identifies the link at the node.) Since only one packet can be injected into the network through a link at a time, the combination of a node id and a link id uniquely identifies packets among the packets injected into the network at the same time (i.e., the packets with the same time stamp or equivalently the same hop counter value). A total order (the priorities) of the packets can be defined based on the the combination of the id numbers.

This method is static (i.e., the priorities of packets that are injected into the network at the same time will not change while they are in the network). In contentions, the packets from certain nodes will always be discriminated against. However, the costs of implementation and nodal operation can be reduced and livelock is still prevented.

4.3 Link Assignments for Higher Degree Networks

In the earlier discussion, we assumed that the networks are two-connected. In order for the proposed methods to work in higher degree networks, link assignments (switching) must be carefully done. In this section, we discuss this problem.

Although in higher degree networks, preferences may be given to certain non-shortest path links for optimized switching (some non-shortest path link may be more preferable to another non-shortest path link), in this discussion, no preferences will be given to non-shortest path links (no preference of one over another).

Packets are assigned to (possibly their preferred) links one by one according to their priorities, which are based on the hop and deflection counter values of the method given in an earlier section. Hence, no preferred links may be available for lower priority packets (i.e., lower priority packets may be deflected). (Tie-breaking (priorities of packets and preferences of links) is done by using randomization.)

First, the highest priority packet will be assigned to its preferred link. If there are several packets with the same highest priority, one of them will be selected randomly. Likewise, if the (selected) highest priority packet has several preferred links, one of them will be selected randomly. Then the second highest priority packet (or one of the remaining highest priority packets) will be assigned to its preferred link. If no preferred link is available, one of the unassigned links (non-shortest path links for the packet) will be selected randomly for the packet. The remaining packets will be assigned to the remaining links in the same manner.

In higher degree networks, a routing algorithm (in order to increase the performance) may select a switching configuration that minimizes the total remaining

distance (number of hops) of packets [4, 14]. In general, there is no guarantee that livelock is avoided by such a method.

4.4 Summary

Livelock is a problem unique to deflection networks. We have studied a method that prevents livelock in this chapter. The method will dynamically establish a total order of the packets that have been in the network longest. The packets have priorities based on the established total order. As a result, no two packets have the same priority. This stops continual contentions among the packets. The method guarantees that livelock is avoided. We have also studied a simplified method for low cost implementations and link assignments for higher degree networks.

Aside: The argument in the paper [10], which is discussed in the beginning of this chapter, indeed motivated this study. However, the importance of the problem was not realized until the development of the specification for the networks, which is presented in the following chapter, was half done. The experiences gained from the earlier simulation studies were giving the author an impression that the problem is less important. It was found that the problem must be solved for completeness of the specification and the solution was produced. It will be noticed that the conceptual algorithm used to show the correctness of the proposed method is influenced by the computational model of UNITY, which is used for developing the specification presented in the following chapter. *End of aside.*

CHAPTER V

FORMAL SPECIFICATION

We model a deflection network as a closed system and develop a specification for the network using the logic of UNITY (Unbounded Nondeterministic Iterative Transformations) [17, 61, 62, 79].

In [26, 27], formal specifications for a static (fixed routes) wormhole message router for a multiprocessor interconnection network (a grid of $N \times M$ switches, where N is the number of input lines and M is the number of output lines) are studied. The router is modeled as a closed system in [26]. Whereas, [27] attempts to model it as an open system. Our closed-system assumption in modeling deflection networks is influenced by [26].

The goals of this chapter are to develop a formal specification for deflection networks and to identify methodological elements that provide a common foundation for the design and specification of data networks.

This chapter is organized as follows. Section 5.1 gives an overview of the UNITY computational model and logic; we re-define the operators of [61, 62], which are derived from [17], using the notion of the *strongest invariant* [79]. A formal specification for deflection networks is developed in Section 5.2. The developed specification

is mapped to a UNITY program (a network simulator in pseudocode) in Section 5.3. Section 5.4 summarizes this chapter.

5.1 Overview of UNITY

This section gives an overview of the UNITY computational model and logic.

5.1.1 UNITY Computational Model

The UNITY computational model (program model) is built upon a traditional imperative foundation and a state-transition system.

A UNITY program consists of a declaration/initialization of variables and a set of atomic, terminating, deterministic, guarded, multiple-assignment statements. A UNITY program has no control statements. In each step of execution, a statement is selected nondeterministically and executed. (Executing a statement whose guard is false does not change the values of the variables.) Nondeterministic selection is constrained by the *fairness* rule; every statement is selected infinitely often.

The execution of an assignment statement corresponds to the transition from one state to the next. An execution sequence will be either infinite or end in a state in which no statement leading to another state exists (i.e., a *fixed point* of the program is reached).

Fairness is an important hypothesis in the study of parallel programs. It guarantees that the computations exhibit all behaviors manifested by the execution of programs. In a multiprocess program, different processes (represented by the statements in a UNITY program) will be individually allowed to proceed [5, 34, 50, 61].

5.1.2 UNITY Logic

The verification of a sequential program involves placing predicates at specific points; the predicates hold when the control reaches the points. This is not the case for a UNITY program since UNITY does not have the notion of program control. The properties that must be satisfied are associated with the entire program.

5.1.2.1 Notation A quantified expression is written in the form

$$\langle Op\ x : R(x) : T(x) \rangle,$$

where Op is an associative and commutative operator (e.g., $\wedge$, $\vee$, $+$, etc.), x is a list of dummy variables whose scope is delimited by the angle brackets, $R(x)$ is a predicate giving the ranges of dummy variables over which the quantification is to be done, and $T(x)$ is the term of the quantification. (When $T(x)$ is a predicate, we write $\forall$ instead of $\wedge$ and $\exists$ instead of $\vee$. Note that $R(x)$ may be omitted if the ranges (domains) of dummy variables are understood.)

The *Hoare triple* [36] has the form

$$\{p\}s\{q\},$$

where p and q are predicates and s is a program statement. Its meaning is that if s is executed in a state where p holds, then q holds after the execution of s.

An inference rule is written as

$$\frac{P}{Q},$$

where P and Q are lists of properties. Its meaning is that if P holds, then we may infer that Q holds as well.

A set consisting of all elements x that satisfy the property P is written in the form

$$\{x \mid P\}.$$

A finite set may be specified by enumerating its elements between curly brackets. The cardinality of a finite set A is denoted by $\#A$.

The operators that we use are summarized below, ordered by increasing binding power.

$$\equiv$$
$$\Leftarrow, \Rightarrow$$
$$\textbf{initially}, \textbf{co}, \textbf{stable}, \textbf{constant}, \textbf{invariant}, \textbf{transient}, \mapsto$$
$$\wedge, \vee$$
$$\neg$$
$$=, \neq, <, \geq, >, \leq, \in, \notin$$
$$+, -, \min, \max$$
"." (function application)

The definitions of operators **initially**, **co**, **stable**, **constant**, **invariant**, **transient**, and $\mapsto$ are given later. All other operators have their usual meanings.

5.1.2.2 UNITY Logic Operators We adopt the notion of the *strongest invariant* [79] and re-define the operators of [61, 62]. Note that although the operators of [61, 62] are derived from the original work of UNITY [17], they are developed for generic (discrete) *action systems* (which consist of a number of actions that may change the state of the system) and are not specific to UNITY.

A predicate p is stronger than predicate q if $p \Rightarrow q$. The strongest invariant SI of a program is the strongest predicate X that satisfies the following condition:

$$(\text{initial condition} \Rightarrow X) \wedge \langle \forall s : s \in F : \{X\}s\{X\} \rangle,$$

where F is the program (a set of program statements) and s is a program statement. The strongest invariant characterizes the set of states that are reachable during some execution of a program.

- **initially** p means that p holds for the initial state of every execution sequence:

$$\textbf{initially } p \equiv \text{initial condition} \Rightarrow p.$$

- p **co** q (p constrains q) means that if p holds for some reachable state, then q holds for the next state:

$$p \textbf{ co } q \equiv \langle \forall s : s \in F : \{SI \wedge p\} s \{q\} \rangle.$$

- **stable** p means that if p holds for some reachable state, then p continues to hold for all succeeding states:

$$\textbf{stable } p \equiv p \textbf{ co } p.$$

(In the program model, once p is established, it is preserved by every statement.)

- **constant** p means that p is true for all reachable states if p is initially true and false for all reachable states if it is initially false:

$$\textbf{constant } p \equiv (\textbf{stable } p) \wedge (\textbf{stable } \neg p).$$

- **invariant** p means that p holds initially and continues to hold for all succeeding states:

$$\textbf{invariant } p \equiv (\textbf{initially } p) \wedge (\textbf{stable } p)$$

or simply

$$\textbf{invariant } p \equiv SI \Rightarrow p.$$

(In the program model, p is preserved by every statement.)

- **transient** p means that if p holds for some reachable state, then $\neg p$ holds (p being falsified) for a later state:

$$\textbf{transient } p \equiv \langle \exists s : s \in F : \{SI \wedge p\} s \{\neg p\} \rangle.$$

(In the program model, if p holds at a point, there is at least one statement whose execution falsifies p and that statement is going to be selected for execution due to the fairness rule of the model.)

- $p \mapsto q$ (p leads to q) means that if p holds for some reachable state, then q holds for a later state (within a finite number of execution steps). Formally $p \mapsto q$ holds if and only if it can be derived by a finite number of applications of the following three inference rules:

1. (Basis)

$$\frac{p \wedge \neg q \textbf{ co } p \vee q, \textbf{ transient } p \wedge \neg q}{p \mapsto q},$$

2. (Transitivity)

$$\frac{p \mapsto q, \ q \mapsto r}{p \mapsto r},$$

3. (Disjunction) For any set S of predicates,

$$\frac{\langle \forall p : p \in S : p \mapsto q \rangle}{\langle \exists p : p \in S : p \rangle \mapsto q}.$$

The operators **co**, **stable**, **constant**, and **invariant** are used to specify *safety* properties, which claim that undesirable state transitions will not occur during the execution of the program. The operators **transient** and $\mapsto$ (leads-to) are used for *progress* properties, which claim that the program performs useful work.

Note that UNITY has introduced only two basic operators **co** and **transient**; all other operators are defined in terms of those two operators.

5.2 Specification

We are interested in developing a specification that clarifies the structures of deflection networks and can aid the construction of physical networks. We develop a topology independent, packet-level specification for two-connected networks. The specification is based on a global observation. A network is modeled as a closed system.

The development of the specification proceeds with the following principles: The specification for a system should be sufficiently strong to rule out any undesired behaviors. At the same time, the specification should be sufficiently weak to provide implementers with the freedom to satisfy the specification in the most convenient and efficient way. In other words, it should avoid overspecifying the elements that are not essential to producing the desired system [86].

5.2.1 Requirements

Before we proceed to the development of the specification, we summarize the general requirements of packet communication networks below.

- The values of packets must not be changed, except the values for routing control purposes.
- Packets must not be lost.
- Packets must eventually be delivered to their destinations.

5.2.2 Basic Model and Topology

We start with the general graph model and make refinements to the model so that sufficient details of deflection routing can be specified.

A network is a directed graph $G = (V, E)$, where V is a set of nodes (vertexes) and E is a set of directed links (edges). The set E is a binary relation on the set V. (Note that the general graph model does not allow multiple links connecting two nodes in the same direction. That is the case for most general data networks.)

We identify each node in a network by a unique integer. Let n be the number of nodes in the network. Then the set V is defined as

$$V \equiv \{v \mid v \in \mathbf{N},\ 1 \leq v \leq n\},$$

where $\mathbf{N}$ denotes the set of natural numbers. Self-loops are excluded from E:

$$\langle \forall v : v \in V : (v, v) \notin E \rangle.$$

A deflection network has the following topological properties:

$$\langle \forall v, w : v, w \in V : (v, w) \in E^{+} \rangle,$$

where E^{+} denotes the transitive closure of E, and

$$\langle \forall v : v \in V : \#\{w \mid (w, v) \in E\} = \#\{w \mid (v, w) \in E\} \rangle.$$

The first property specifies that a deflection network is strongly connected. The second property states that each network node has equal in- and out-degrees, which are denoted as d_v for each node v:

$$d_v \equiv \#\{w \mid (w, v) \in E\}$$

or equivalently

$$d_v \equiv \#\{w \mid (v,w) \in E\}.$$

For the rest of this chapter, $d_v = 2$. Note that the transitive closure R^+ of a relation R can be constructed by the following rules:

$$\frac{(a,b) \in R}{(a,b) \in R^+},$$

$$\frac{(a,b) \in R,\ (b,c) \in R}{(a,c) \in R^+}.$$

A path from node v to node w is a non-null sequence of links such that the first link in the sequence is directed away from v; the last link in the sequence is directed toward w; for every successive pair (a,b), (c,d) of links in the sequence, $b = c$. The length of a path is the number of links in the path. The distance from node v to node w is given by the function Δ (from E^+ to $\mathbf{N}$), such that

$$\Delta.(v,w) = \begin{cases} \text{the length of the minimum-length path from } v \text{ to } w & \text{if } v \neq w; \\ 0 & \text{if } v = w. \end{cases}$$

The topological properties are static. We assume that there are no topological changes during the execution of the system. The following statements are assumed:

$$\langle \forall v :: \textbf{constant } v \in V \rangle,$$

$$\langle \forall v, w :: \textbf{constant } (v,w) \in E \rangle.$$

5.2.3 I/O Queues and Network Medium

In order to model the behavior of a deflection network, we must be able to distinguish the locations of packets in the network precisely according to the topology and the nodal structure of the network. We abstract the inputs and outputs as unbounded

queues of packets. Let S, D, and M be the set of locations in the input (source) queues, the set of locations in the output (destination) queues, and the set of locations in the network medium. We define the sets as follows:

$$Q \equiv \{(t,v,i,k) \mid t,i,k \in \mathbf{N},\ v \in V,\ 1 \le t \le 2,\ 1 \le i \le d_v,\ 1 \le k\},$$

$$S \equiv \{(t,v,i,k) \mid (t,v,i,k) \in Q,\ t = 1\},$$

$$D \equiv \{(t,v,i,k) \mid (t,v,i,k) \in Q,\ t = 2\},$$

$$M \equiv \{(t,v,i) \mid t,i \in \mathbf{N},\ v \in V,\ 1 \le t \le 3,\ 1 \le i \le d_v\}.$$

The components v, i, and k are a node id, a link id, and a position number. The component t of an element in M indicates the location of a packet inside a node (Figure 5.1). The details appear later.

The sets S and D have countably infinite elements. An order of elements in the sets may be given by Cantor numbering. The order of locations is important only in the same queue. The component k determines the order of locations in a queue.

Instead of writing locations in the forms (t,v,i,k) or (t,v,i) in the specification, we often use the following more intuitive notations of the forms: $src^v{}_i k$ denoting $(1,v,i,k)$, $dst^v{}_i k$ denoting $(2,v,i,k)$, $in^v{}_i$ denoting $(1,v,i)$, $sw^v{}_i$ denoting $(2,v,i)$, and $out^v{}_i$ denoting $(3,v,i)$.

Figure 5.1 shows the locations in a node. (The node id is omitted in the figure.) We assign an input queue and an output queue to each link. This clarifies the nodal processing of deflection networks.

The symbol $\vdash$ is used to indicate the connection of nodes (which outgoing link of a node is connected to which incoming link of another node). The expression

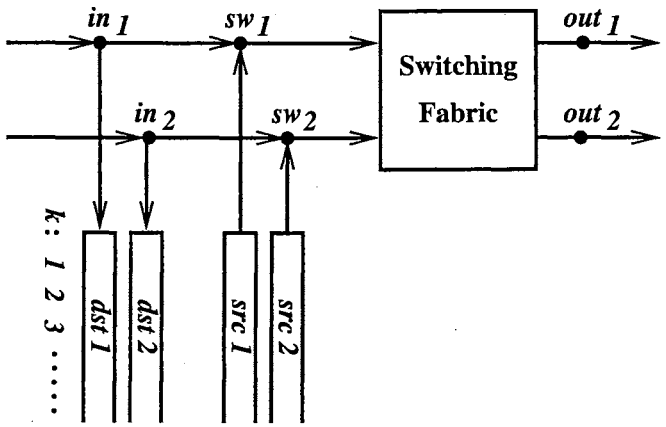

Figure 5.1: Locations in a node.

$out^v{}_i \vdash in^w{}_j$ is true if output link i of node v is connected to input link j of node w. The operator has the same binding power as $=$, $\neq$, and the like. The following statements hold:

$$\langle \forall v, w :: (v, w) \in E \Leftrightarrow \langle \exists i, j :: out^v{}_i \vdash in^w{}_j \rangle \rangle,$$

$$\langle \forall v, w, i, j, i', j' :: out^v{}_i \vdash in^w{}_j \wedge out^v{}_{i'} \vdash in^w{}_{j'} \Leftrightarrow i = i' \wedge j = j' \rangle.$$

As mentioned earlier, topological properties are static. We assume the following property:

$$\langle \forall v, w, i, j :: \textbf{constant}\ out^v{}_i \vdash in^w{}_j \rangle.$$

5.2.4 Packet Representation

A packet has seven components: the source node id, the destination node id, the queue (link) id, the packet (position) number, the hop counter value, the deflection counter value, and the data portion. (The third and fourth components above are not necessary in implementations. We augment the packets with those components in order to uniquely identify each packet in the model.)

Using the static components of packets (source node id, destination node id, queue (link) id, packet (position) number, and data portion), we define the set P^* (the set of all *logical* packets) as follow:

$$P^* \equiv \{(s, r, i, k, z) \mid s, r \in V,\ i, k \in \mathbf{N},\ 1 \leq i \leq d_s,\ 1 \leq k,\ z \in \Lambda\},$$

where Λ is the set of all strings (data that can be represented by computers).

Let P be the set of all *physical* packets. The physical packets are the packets that actually exist in the system. The set P is a subset of P^*. In the specification, a packet is represented by a variable (α or β). The components of a packet are accessed through the following access functions: *source*, *destination*, *queue*, *number*, and *data*. For the variable components of packets (hop counter and deflection counter), we use two functions (from P to $\mathbf{N}$) *hcount* and *dcount*. Throughout the execution of the model, the set P is unchanged.

Packet Existence

$$\langle \forall \alpha : \alpha \in P^* : \textbf{constant } \alpha \in P \rangle \tag{5.1}$$

Property 5.1 states that no packets will be created or destroyed in the model. Only packets that initially exist continue to exist.

The location of a packet in the system is given by the function Θ, which is defined as

$$\Theta : P \rightarrow S \cup M \cup D.$$

We define a predicate *empty* as

$$empty.x \equiv \langle \forall \alpha : \alpha \in P : \Theta.\alpha \neq x \rangle.$$

The predicate *empty*.x is true if there is no packet at the location x and false otherwise.

For the rest of this chapter, the domain of packets is P and may be omitted in an expression.

Packet Location

$$\textbf{invariant } \langle \forall \alpha, \beta : \alpha \neq \beta : \Theta.\alpha \neq \Theta.\beta \rangle \tag{5.2}$$

Property 5.2 means that no two packets have the same location (i.e., the function Θ is a one-to-one function). (This implies that a packet cannot move to a location that is occupied by another packet.)

Network Initialization

$$\textbf{initially } \langle \forall \alpha :: \Theta.\alpha \in S \rangle \tag{5.3}$$

Property 5.3 means that initially all packets are in the input queues and there are no packets in the output queues and in the network medium.

Packet Validity

$$\textbf{initially } \langle \forall \alpha, v, i, k :: \Theta.\alpha = src^v{}_i k \Leftrightarrow$$

$$source.\alpha = v \wedge queue.\alpha = i \wedge number.\alpha = k \rangle \tag{5.4}$$

$$\textbf{initially } \langle \forall \alpha :: \langle \exists v :: destination.\alpha = v \rangle \rangle \tag{5.5}$$

$$\textbf{initially } \langle \forall \alpha :: \langle \exists z :: data.\alpha = z \rangle \rangle \tag{5.6}$$

$$\langle \forall \alpha, v, w, i, k, z :: \textbf{stable } source.\alpha = v \wedge$$

$$destination.\alpha = w \wedge queue.\alpha = i \wedge$$

$$number.\alpha = k \wedge data.\alpha = z \rangle \tag{5.7}$$

Properties 5.4–5.7 state that each packet must have valid component values which must not be changed during the execution of the system. Note that properties 5.5–5.7 simply follow property 5.1; they are merely repeated here.

The position number of a packet is unique in its input queue. The number is assigned to each packet based on the initial location of the packet. The triple (s, i, k), where s, i, and k are a source node id, a queue (link) id, and a position number, uniquely identifies a packet in the model. This allows the existence of multiple packets that have the same source node id, the same destination node id, and the same data value in the model (i.e., in the set P).

Note that the hop and deflection counters, which should be included in the packet header in implementations, are not included in packets in this model; they are given by functions from P to $\mathbf{N}$ (*hcount* and *dcount*).

5.2.5 Packet Moves

The most fundamental property that must be implemented is

$$\langle \forall \alpha :: \Theta.\alpha \in S \mapsto \Theta.\alpha \in D \rangle.$$

Every packet in input queues must eventually move into some output queue. More precisely,

$$\langle \forall \alpha, v :: \mathit{destination}.\alpha = v \mapsto \langle \exists i, k :: \Theta.\alpha = dst^v{}_i k \rangle \rangle.$$

Every packet must eventually reach its destination. We define the detailed properties of packet moves below.

Queue Move

$$\langle \forall \alpha, v, i, k : k > 1 : \Theta.\alpha = src^v{}_i k \ \mathbf{co}\ \Theta.\alpha = src^v{}_i k \vee \Theta.\alpha = src^v{}_i k - 1 \rangle \tag{5.8}$$

$$\langle \forall \alpha, v, i, k : k > 1 : \Theta.\alpha = src^v{}_i k \ \mapsto \Theta.\alpha = src^v{}_i k - 1 \rangle \tag{5.9}$$

$$\langle \forall \alpha, v, i, k : k > 0 : \Theta.\alpha = dst^v{}_i k \ \mathbf{co}\ \Theta.\alpha = dst^v{}_i k \vee \Theta.\alpha = dst^v{}_i k + 1 \rangle \tag{5.10}$$

$$\langle \forall \alpha, v, i, k : k > 0 : \Theta.\alpha = dst^v{}_i k \ \mapsto \Theta.\alpha = dst^v{}_i k + 1 \rangle \tag{5.11}$$

Properties 5.8–5.11 define the packet moves in input and output queues. Property 5.8 states that a packet at the position k in an input queue will either stay at the same position or move to the position $k-1$ and there are no other possible moves. Property 5.9 guarantees that the packet will move to the position $k-1$ in a finite number of execution steps. Properties 5.10 and 5.11 specify the analogous moves for packets in output queues.

Generally, a pattern of system behavior is specified by a pair of properties (a safety property and a progress property).

Injection

$$\langle \forall \alpha, v, i :: \Theta.\alpha = src^v{}_i 1 \ \mathbf{co}\ \Theta.\alpha = src^v{}_i 1 \vee$$

$$(\Theta.\alpha = sw^v{}_i \wedge hcount.\alpha = 0 \wedge dcount.\alpha = 0) \rangle \tag{5.12}$$

$$\langle \forall \alpha, v, i :: \Theta.\alpha = src^v{}_i 1 \mapsto \Theta.\alpha = sw^v{}_i \rangle \tag{5.13}$$

Properties 5.12 and 5.13 specify that the packet at the head of an input queue must be injected into the network within a finite number of execution steps and there are no other moves. Property 5.13 must be implemented for property 5.9. Recall that no

two packets can be at the same location (property 5.2). Hence, the packet at the head of an input queue must be injected into the network so that the following packets in the queue can make their moves. (All packet moves must satisfy property 5.2.)

Note that for any packet in an input queue, there are only finitely many packets ahead of it in the queue. The packet will be out of the input queue within a finite number of execution steps.

The following properties define the packet moves in the network medium.

Node-to-Node Hop

$$\langle \forall \alpha, v, i, k, l :: \Theta.\alpha = out^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l \ \mathbf{co}$$

$$(\Theta.\alpha = out^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l) \vee$$

$$(\langle \exists w, j :: out^v{}_i \vdash in^w{}_j \wedge \Theta.\alpha = in^w{}_j \rangle \wedge hcount.\alpha = k + 1 \wedge dcount.\alpha = l) \rangle \quad (5.14)$$

$$\langle \forall \alpha, v, i :: \Theta.\alpha = out^v{}_i \mapsto \langle \exists w, j :: out^v{}_i \vdash in^w{}_j \wedge \Theta.\alpha = in^w{}_j \rangle \rangle \quad (5.15)$$

Property 5.14 specifies that the hop counter value increases by 1 every time a packet makes a hop, while the deflection counter value remains the same.

Delivery

$$\langle \forall \alpha, v, i :: \Theta.\alpha = in^v{}_i \wedge destination.\alpha = v \ \mathbf{co} \ \Theta.\alpha = in^v{}_i \vee \Theta.\alpha = dst^v{}_i 1 \rangle \quad (5.16)$$

$$\langle \forall \alpha, v, i :: \Theta.\alpha = in^v{}_i \wedge destination.\alpha = v \mapsto \Theta.\alpha = dst^v{}_i 1 \rangle \quad (5.17)$$

Transit

$$\langle \forall \alpha, v, i, k, l :: \Theta.\alpha = in^v{}_i \wedge destination.\alpha \neq v$$

$$\wedge\ hcount.\alpha = k \wedge dcount.\alpha = l\ \mathbf{co}$$

$$(\Theta.\alpha = in^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l) \vee$$

$$(\Theta.\alpha = sw^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l)\rangle \tag{5.18}$$

$$\langle \forall \alpha, v, i :: \Theta.\alpha = in^v{}_i \wedge destination.\alpha \neq v \mapsto \Theta.\alpha = sw^v{}_i \rangle \tag{5.19}$$

Switching

$$\langle \forall \alpha, v, i, k, l :: \Theta.\alpha = sw^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l\ \mathbf{co}$$

$$(\Theta.\alpha = sw^v{}_i \wedge hcount.\alpha = k \wedge dcount.\alpha = l) \vee$$

$$(\langle \exists j :: \Theta.\alpha = out^v{}_j \rangle \wedge hcount.\alpha = k \wedge (dcount.\alpha = l \vee dcount.\alpha = l + 1))\rangle \tag{5.20}$$

$$\langle \forall \alpha, v, i :: \Theta.\alpha = sw^v{}_i \mapsto \langle \exists j :: \Theta.\alpha = out^v{}_j \rangle \rangle \tag{5.21}$$

In an implementation, the routing algorithm at each node determines the value of j in properties 5.20 and 5.21. The deflection counter value may be increased by 1 during switching. We will make refinements to property 5.21 later for a specific implementation.

Up to this point, we have specified only individual packet moves. Now, we must model and specify synchronized packet moves. First, we define an invariant for packet locations. Then the synchronized packet moves are specified in terms of the number of packets at a node.

Relative Packet Location

$$\langle \forall t, v, i :: \textbf{invariant}\ \neg empty.(t, v, i) \Rightarrow \langle \forall s, j : s \neq t : empty.(s, v, j) \rangle \rangle \tag{5.22}$$

Property 5.22 specifies the relative locations of packets in a node. For example, if there is a packet at $in^v{}_i$ for some i, no other packets can be at $sw^v{}_j$ and $out^v{}_k$ for all j and k at the same time.

Synchronized Packet Move

$$\langle \forall t, v, k : k > 0 : \#\{i \mid \neg empty.(t, v, i)\} = k \textbf{ co}$$

$$\#\{i \mid \neg empty.(t, v, i)\} = k \vee \#\{i \mid \neg empty.(t, v, i)\} = 0\rangle \tag{5.23}$$

Property 5.23 implies that the switching and transmission of packets need to be synchronized. The number of $in^v{}_i$'s (i ranging from 1 to d_v) that have packets will remain the same or drop to zero. This implies that the packets at $in^v{}_i$'s must move at once. The same property holds for packets at $sw^v{}_i$'s and $out^v{}_i$'s. Although the property is written in terms of the number of packets in a node, it globally synchronizes the node-to-node packet moves in the network. Packets coming into a node must be coming at the same time. Since all those packets are coming from different nodes, the transmission of the packets at the nodes must be synchronized.

The packet at $src^v{}_i 1$ (the head of an input queue of a node) can be injected into the network if there are no other packets in the node or it may be injected at the same time that the packet at $in^v{}_i$ is moved to $dst^v{}_i 1$ (i.e., an incoming packet on link i is extracted). There are no other cases that the packet at the head of an input queue can be injected into the network. Transit packets have higher priorities than source packets. This means that if a node always receives transit packets, the node cannot inject its source packets. In other words, property 5.13 may not hold. Certain injection control mechanisms must be implemented to prevent this situation, which

is generally known as *source lockout.* We will deal with this problem later.

The basic network hardwares such as cables, transmitters, and receivers should support the properties specified above for packet moves although the functions of the hardware components alone cannot guarantee all the properties. (It appears that the properties for the packet moves can be guaranteed solely by the network hardware components; however, many of the properties cannot be guaranteed by the functions of the hardware components alone.) In the following subsection, we specify a routing scheme for general two-connected networks.

5.2.6 Selecting Switching Configurations

The following properties specify shortest path routing if there is no contention.

Shortest Path Routing

$$\begin{aligned}\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge \\ out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge \\ ((\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge \\ \Delta.(w_1, destination.\beta) \geq \Delta.(w_2, destination.\beta)) \vee \\ (\Delta.(w_1, destination.\alpha) \leq \Delta.(w_2, destination.\alpha) \wedge \\ \Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta))) \\ \mapsto \Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \rangle \end{aligned} \tag{5.24}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$((\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$
$$\Delta.(w_1, destination.\beta) \leq \Delta.(w_2, destination.\beta)) \vee$$
$$(\Delta.(w_1, destination.\alpha) \geq \Delta.(w_2, destination.\alpha) \wedge$$
$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta)))$$
$$\mapsto \Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \rangle \tag{5.25}$$

$$\langle \forall \alpha, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge empty.sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \mapsto \Theta.\alpha = out^v{}_1 \rangle \tag{5.26}$$

$$\langle \forall \alpha, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge empty.sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \mapsto \Theta.\alpha = out^v{}_2 \rangle \tag{5.27}$$

$$\langle \forall \alpha, v, w_1, w_2, i_1, i_2 :: empty.sw^v{}_1 \wedge \Theta.\alpha = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \mapsto \Theta.\alpha = out^v{}_1 \rangle \tag{5.28}$$

$$\langle \forall \alpha, v, w_1, w_2, i_1, i_2 :: empty.sw^v{}_1 \wedge \Theta.\alpha = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \mapsto \Theta.\alpha = out^v{}_2 \rangle \tag{5.29}$$

Properties 5.24 and 5.25 may be simplified for faster processing in implementations.

Suppose a packet α destined for node b is at node a ($a \neq b$) and node a has two adjacent nodes c_1 and c_2 ($c_1 \neq c_2$) (i.e., $(a, c_1), (a, c_2) \in E$). For α to reach b, it must pass through either c_1 or c_2. The shortest distance from c_1 to b is $\Delta.(c_1, b)$ by definition. Analogously, the shortest distance from c_2 to b is $\Delta.(c_2, b)$. Therefore,

$$\Delta.(a.b) = 1 + (\Delta.(c_1, b) \ \min \ \Delta.(c_2, b)).$$

Hence,

$$\Delta.(a.b) > \Delta.(c_1, b) \ \min \ \Delta.(c_2, b).$$

The routing scheme decreases the remaining distance to the destination. A packet injected into a network will eventually reach its destination if there is no contention.

Note that the operator min is defined as

$$a = b \min c \equiv (a = b \vee a = c) \wedge a \leq b \wedge a \leq c.$$

5.2.7 Contention Resolution and Livelock Prevention

We specify the livelock prevention method of Chapter IV. When two packets want to use the same output link, the packet with the larger hop counter value gets the link and the other packet is deflected. Similarly, when two packets with the same hop counter value want to use the same output link, the packet with the larger deflection counter value gets the link and the other packet is deflected. If both two packets have the same hop and deflection counter values, one of them should be selected randomly for the link and the deflection counter value of the other should be increased.

Contention Resolution By Hop Counter

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge$$
$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta) \wedge$$
$$hcount.\alpha > hcount.\beta \mapsto \Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \rangle \tag{5.30}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$
$$\Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta) \wedge$$
$$hcount.\alpha > hcount.\beta \mapsto \Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \rangle \tag{5.31}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$
$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$
$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge$$
$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta) \wedge$$
$$hcount.\alpha < hcount.\beta \mapsto \Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \rangle \tag{5.32}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$

$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha < hcount.\beta \mapsto \Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \rangle \tag{5.33}$$

Contention Resolution By Deflection Counter

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$

$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha > dcount.\beta$$

$$\mapsto \Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \rangle \tag{5.34}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$

$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha > dcount.\beta$$

$$\mapsto \Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \rangle \tag{5.35}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = {sw^v}_1 \wedge \Theta.\beta = {sw^v}_2 \wedge$$

$${out^v}_1 \vdash {in^{w_1}}_{i_1} \wedge {out^v}_2 \vdash {in^{w_2}}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha < dcount.\beta$$

$$\mapsto \Theta.\alpha = {out^v}_2 \wedge \Theta.\beta = {out^v}_1 \rangle \tag{5.36}$$

$$\langle \forall \alpha, \beta, v, w_1, w_2, i_1, i_2 :: \Theta.\alpha = {sw^v}_1 \wedge \Theta.\beta = {sw^v}_2 \wedge$$

$${out^v}_1 \vdash {in^{w_1}}_{i_1} \wedge {out^v}_2 \vdash {in^{w_2}}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha < dcount.\beta$$

$$\mapsto \Theta.\alpha = {out^v}_1 \wedge \Theta.\beta = {out^v}_2 \rangle \tag{5.37}$$

Tie-breaking and Incrementing Deflection Counter

$$\langle \forall \alpha, \beta, v, k, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$

$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) < \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) < \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha = k \wedge dcount.\beta = k$$

$$\mapsto (\Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \wedge dcount.\alpha = k \wedge dcount.\beta = k+1)$$

$$\vee (\Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \wedge dcount.\alpha = k+1 \wedge dcount.\beta = k)\rangle \qquad (5.38)$$

$$\langle \forall \alpha, \beta, v, k, w_1, w_2, i_1, i_2 :: \Theta.\alpha = sw^v{}_1 \wedge \Theta.\beta = sw^v{}_2 \wedge$$

$$out^v{}_1 \vdash in^{w_1}{}_{i_1} \wedge out^v{}_2 \vdash in^{w_2}{}_{i_2} \wedge$$

$$\Delta.(w_1, destination.\alpha) > \Delta.(w_2, destination.\alpha) \wedge$$

$$\Delta.(w_1, destination.\beta) > \Delta.(w_2, destination.\beta) \wedge$$

$$hcount.\alpha = hcount.\beta \wedge dcount.\alpha = k \wedge dcount.\beta = k$$

$$\mapsto (\Theta.\alpha = out^v{}_2 \wedge \Theta.\beta = out^v{}_1 \wedge dcount.\alpha = k \wedge dcount.\beta = k+1)$$

$$\vee (\Theta.\alpha = out^v{}_1 \wedge \Theta.\beta = out^v{}_2 \wedge dcount.\alpha = k+1 \wedge dcount.\beta = k)\rangle \qquad (5.39)$$

Given that the routing scheme is correct and livelock is prevented, we can say

$$\langle \forall \alpha :: \Theta.\alpha \in M \mapsto \Theta.\alpha \in D \rangle.$$

If we can say

$$\langle \forall \alpha :: \Theta.\alpha \in S \mapsto \Theta.\alpha \in M \rangle,$$

then we have the property (requirement) given earlier:

$$\langle \forall \alpha :: \Theta.\alpha \in S \mapsto \Theta.\alpha \in D \rangle.$$

Recall that $\mapsto$ is transitive. In the next subsection, we deal with the property

$$\langle \forall \alpha :: \Theta.\alpha \in S \mapsto \Theta.\alpha \in M \rangle.$$

5.2.8 Lockout Prevention

As noted earlier, we must prevent potential source lockout and validate property 5.13. Recall that source lockout is a situation that a node is busy routing transit packets all the time and hence (source) packets cannot be injected into the network from the node.

Knowing that in practice, source lockout is unlikely to occur in symmetric networks with uniform traffic patterns and interrelating the problem to livelock prevention, we present the following solution for our model. We force each node to send a packet (dummy packet) to every node in the network once in a while (infinitely often). (When a node receives and extracts a packet addressed to the node, the node can inject one of its source packets into the network (i.e., the node is unlocked). Intuitively, lockout will be prevented if livelock is prevented (i.e., every injected packet reaches its destination in a finite number of hops).) In the model, every input queue contains infinitely many packets addressed to every node. This scheme may not create a probabilistically uniform traffic pattern; nevertheless, there will be traffic between every node pair.

Packet Destination

initially $\langle \forall v, w, i : v \neq w : \langle \exists \alpha, k :: \Theta.\alpha = src^v{}_i k \wedge destination.\alpha = w \rangle \rangle$ (5.40)

$$\textbf{initially } \langle \forall \alpha, v, i, k : \Theta.\alpha = src^v{}_i k :$$

$$\langle \exists \beta, m : m > 0 : \Theta.\beta = src^v{}_i(k+m) \wedge destination.\beta = destination.\alpha \rangle \rangle \quad (5.41)$$

Property 5.40 specifies that each input queue of every node contains packets addressed to all other nodes. Property 5.41 states that in each input queue, there are infinitely many packets addressed to the same node.

Consider tracing a packet injected into a network (perhaps the first packet injected into the network after the initialization) and its succeeding packets (i.e., the packets that will be injected into the network at the nodes receiving the traced packet). Given that the routing scheme is correct and livelock is prevented, a packet injected into a network will reach its destination node in a finite number of hops. The node receiving the packet will inject a packet into the network. Again, the packet will reach its destination node in a finite number of hops and the node receiving the packet will inject a packet into the network. Nodes that have not received any packet after a certain period of execution steps (if such nodes exist) may have been locked out. The existence of such nodes implies that the injection of packets has occurred only at a certain subset of the nodes in the network. If this situation will remain unchanged (i.e., only a subset of the nodes in the network will be injecting packets into the network), then because of properties 5.40 and 5.41, the nodes in the subset will eventually (have to) inject packets addressed to the nodes that may have been locked out. An input queue that continually injects packets will eventually inject a packet addressed to every node in the network. Hence, no node can be indefinitely locked out.

It is possible to implement other methods to prevent lockout. For example, a node that can inject packets may voluntarily stop injecting packets so that downstream nodes will have chances to inject their packets. Property 5.13 says that a node must eventually inject the packet at the head of an input queue into the network; however, the property does not say that a node must inject a packet whenever it can.

Note that although indefinite source lockout can be avoided, temporary lockout may happen at a node. More sophisticated mechanisms may be required to provide fair access (or demand based access) to the network when a uniform traffic pattern is not assumed. Lockout-free does not necessarily mean that fair network access is provided.

It is not difficult to implement fair and guaranteed network access. For example, every node could always have its own tokens (slots marked as tokens for the node) in the network; a token can be used only by its owner. Unfortunately, the throughput under such an implementation will be limited. Only a fraction of the network capacity may be used for the actual communications. It is difficult to provide fair and guaranteed network access for all nodes while also achieving a high throughput level. Fortunately, recent practices of networking and distributed computing encourage balanced traffic loads (patterns), which will naturally provide fair network access for all nodes.

5.3 Constructing A Simulator Program

This section gives an outline of the derivation of a (pseudo) UNITY program (a network simulator in the UNITY computational model) from the specification. Since the complete implementation will be lengthy and perhaps tedious, only important

issues for mapping the specification to a program are described.

5.3.1 Data Types and Representations

We use the data type "packet" as the basic data type in the program. (The data type "packet" is a record. We use the same notation as in the specification to access the fields of "packet" as we need in the program.) The data type of the locations is "packet". In the specification, we have modeled a network with its environments together as a closed system. In the program, we represents the I/O environments by the data type "sequence of packet". The symbol $\perp$ is used to denote a null-packet. The program variables are declared as follows.

declare
  in, *out*, *sw*: array[1..N, 1..2] of packet;
  src, *dst*: array[1..N, 1..2] of sequence of packet;
  net: array[1..N, 1..2] of 1..N;
  dist: array[1..N, 1..N] of 1..N;
{ end of declare-section }

The constant N is the number of nodes in the network. The array *net* is used to represent the node connections (topology), which is represented by the operator $\vdash$ in the specification. If node v is connected to node w by link k, then $net[v, k] = w$. The array *dist* stores the shortest distances between nodes, implementing the function Δ in the specification. The distance from node v to node w is given by $dist[v, w]$.

We use the following operations on sequences:

$$head.s \equiv \text{head element of the sequence } s,$$

$$tail.s \equiv \text{tail sequence of the sequence } s,$$

$$(s; x) \equiv \text{sequence obtained by appending the element } x \text{ at the end of the sequence } s,$$

$$nil.s \equiv s \text{ is an empty sequence.}$$

The basic notation used in this section follows the UNITY program notation in [17]. An assignment statement can be composed by several assignment components separated by the operator ||. The operator [] separates the assignment statements in a program.

5.3.2 Initialization

The initialization of the program is done as follows.

initially
{ Empty Network }
$\langle \| i,j : 1 \leq i \leq N,\ 1 \leq j \leq 2 : in[i,j], out[i,j], sw[i,j] = \bot, \bot, \bot \rangle$ ||
{ Define Topology }
$net[1,1], \ldots, net[N,2], dist[1,1], \ldots, dist[N,N] = \cdots$
{ end of initially-section }

The network is initially empty as specified by property 5.3 (Network Initialization). The arrays *net* and *dist* should be initialized with aprropreate values for the network to be simulated.

5.3.3 Assignment Statements

The assignment statements that compose a UNITY program are shown below.

assign
{ Node-to-Node Hop }
$\langle \| i_1, i_2, j : 1 \leq i_1 \leq N,\ 1 \leq i_2 \leq N,\ 1 \leq j \leq 2 : in[i_2,j], out[i_1,j] := out[i_1,j], \bot$
if $net[i_1,j] = i_2 \wedge out[i_1,j] \neq \bot \wedge \langle \forall k : 1 \leq k \leq 2 : in[i_2,k] = \bot \wedge sw[i_2,k] = \bot \rangle \rangle$

[] { Packet Moves Inside A Node }
$\langle \| i,j : 1 \leq i \leq N,\ 1 \leq j \leq 2 :$
$sw[i,j], in[i,j] := in[i,j], \bot$ { Transit }
if $in[i,j] \neq \bot \wedge destination.in[i,j] \neq i \wedge$
$\langle \forall k : 1 \leq k \leq 2 : sw[i,k] = \bot \wedge out[i,k] = \bot \rangle$ ||
$sw[i,j], src[i,j] := head.src[i,j], tail.src[i,j]$ { Injection }
if $\neg nil.src[i,j] \wedge in[i,j] = \bot \wedge$
$\langle \forall k : 1 \leq k \leq 2 : sw[i,k] = \bot \wedge out[i,k] = \bot \rangle$ ||

$dst[i,j], in[i,j] := (dst[i,j]; in[i,j]), \perp$ { Delivery }
if $in[i,j] \neq \perp \wedge destination.in[i,j] = i\rangle$

$[]$ { Switching }
$\langle \| \, i : 1 \leq i \leq N : out[i,1], out[i,2] :=$
$sw[i,1], sw[i,2]$
if $(dist[net[i,1], destination.sw[i,1]] < dist[net[i,2], destination.sw[i,1]] \wedge$
$dist[net[i,1], destination.sw[i,2]] \geq dist[net[i,2], destination.sw[i,2]]) \vee$
$\cdots\cdots\cdots \sim$ { else }
$sw[i,2], sw[i,1]$
if $(dist[net[i,1], destination.sw[i,1]] > dist[net[i,2], destination.sw[i,1]] \wedge$
$dist[net[i,1], destination.sw[i,2]] \leq dist[net[i,2], destination.sw[i,2]]) \vee$
$\cdots\cdots\cdots\rangle$
{ end of assign-section }

For simplicity and clear presentation (but without loss of generality), the counter manipulations and the details of switching are omitted in the program. (Assignment operations should be done component by component if counter manipulations are implemented.) We assume that the input queues are non-empty.

The first assignment is based on property 5.15 (Node-to-Node Hop). The second assignment is based on properties 5.19 (Transit), 5.13 (Injection), and 5.17 (Delivery). The third assignment is based on properties 5.21 (Switching), 5.24 (Shortest Path Routing), and 5.25 (Shortest Path Routing).

The program satisfies property 5.1 (Packet Existence). No packets will be created or destroied in the program. The number of packets will be unchnaged after the execution of any assignment statement in the program. Since every location in the specification is represented by a valiable in the program, property 5.2 (Packet Location) is cleary satisfied. We assume that the sequence *src* satisfies Properties 5.4–5.6 (Packet Validity). The components of packets, except the counter values, will not be changed by the program. Hence, property 5.7 (Packet Validity) is satisfied. Prop-

erties 5.8–5.11 (Queue Move) are implimented by sequences. Modeling synchronized packet moves is strightforward in the program. Propery 5.23 (Synchronized Packet Move) is implemented by using parallel assignments.

The assignment statements can be executed concurently in the computational model; however, the execution of statements may be ordered (based on the packet moves in the network) and iterrated in a sequential implementation.

5.4 Summary

We have developed a formal specification for deflection networks in UNITY logic. The developed specification then was mapped to a UNITY program (a network simulator in pseudocode).

The specification is descriptive and non-operational. A network is viewed as a mathematical object. The advantage of this approach is that error prone operational reasoning is eliminated in the specification.

The I/O queues as well as the locations in the network medium were represented by sets of distinct locations rather than sequence variables, the use of which may seem to be more natural for communication networks. The set model allows us to use the standard mathematical tools in the specification. This approach may be applied to the specifications of routing schemes in other types of networks.

The development of the specification forced us to realize the logical problems in the network. It appears that all individual properties can be guaranteed solely by the network hardware components; however, many of the properties cannot be guaranteed by the functions of the hardware components alone. The problems of livelock and lockout have been addressed by specification.

A network with its environment together was modeled as a closed system. This formulation avoids dealing with conditional properties, which add a certain degree of complexity to the development of specifications. An open system model imposes an examination of the compositionality of the defined logic.

The notion of the strongest invariant was adopted in defining the UNITY operators. The non-equivalence between the axiomatic and informal operational semantics of the operators in [17] is eliminated. However, the properties specified are weaker than those specified in the original logic of [17] in the sense that the properties hold only for reachable states.

In practice, the difference in formulation of the logic has little effect on the derivation of programs [80]. A recent study of the compositionality of properties and a discussion of the differences in logic formulations can be found in [25].

CHAPTER VI

CONCLUSION

6.1 Contributions

The materials presented in this dissertation have covered many issues of deflection networks. The main contributions of the dissertation include the developments of a high performance routing scheme for the HTN, topology independent methods that prevent livelock, and a formal specification for deflection networks.

The throughput/delay characteristics of the MSN and the HTN with several contention resolution methods have been investigated by intensive simulations. It is demonstrated that the proposed routing scheme for the HTN increases the throughput by effectively using *don't care* nodes. As the network size increases, the scheme decreases the number of deflections. The simulation result shows that in a 256×256 HTN, over 98% of packets were delivered to their destinations without even a single deflection under a complete network saturation condition (i.e., 100% link utilization). The research also provides simulation case studies for routing in irregular networks and random routing in the MSN and the HTN. The results obtained in this research show that the deflection networks, yet simple in structure, are indeed more powerful than popular linear structure networks and suitable as metropolitan area communi-

cation infrastructures.

Livelock prevention is one of key issues of deflection networks. It has been known that in many cases (but not all cases), the introduction of randomization and certain priority mechanisms in contention resolutions will avoid livelock. This study provides a stronger result. The proposed methods guarantee that livelock is deterministically avoided (i.e., every packet that is injected into a network is 100% guaranteed to reach its destination). Furthermore, the methods are topology independent and can be applied to networks with a high degree of connectivity.

A formal specification (a high-level design specification) for two-connected deflection networks has been developed. A network was modeled as a closed system with unbounded I/O queues. The development of the specification, specially the specification of progress properties, forced us to realize the logical problems in the network. The basic network hardwares such as cables, transmitters, and receivers should support the properties in the specification; however, the functions of the hardware components alone cannot guarantee all the properties. The necessity of livelock and lockout prevention mechanisms has been addressed by specification. We specified the livelock prevention method proposed earlier and solved the problem of lockout in the model by interrelating the problem to livelock prevention. The developed specification is weak in the sense that it leaves implementers with the freedom to satisfy the specification in a convenient and efficient way. For example, for improved lockout prevention and fair network access, an implementation may have a mechanism to control the packet injection rate at each node. As more sophisticated schemes are found to be appropriate, they may be implemented without changing the specification. The

study demonstrates the effectiveness of the formal approach to the system design problems. It also shows that the use of a small set of temporal logic operators (in addition to the conventional mathematical tools) is sufficient to build a model for a concurrent communication system. We have dealt with rather fundamental functions of the networks in this study; however, the model and specification can be used as a framework for designing and specifying other more complex functions of the networks (e.g., broadcasting) As we have seen, simulators (and perhaps the actual systems) can be build based on the specification. The simulators that have been used to carry out the performance characteristics of deflection networks follow the specification, except that livelock and lockout prevention issues are not addressed in the simulators. In the specification, we have treated the system as rather a critical system whose functions must be correct all the time. In realistic communication systems, certain errors (e.g., the loss of packets that may be resulted from buffer overflow or perhaps in deflection networks, livelock or lockout prevention) are allowed to occur in general; however, such an assumption, which may allow us to drop dealing with the problem of livelock or lockout in our model, should not be made during the basic system design process. The study yielded better understanding of the formal approach to the system design and specification. The same approach can be applied to the design and specification of other types of data networks.

6.2 Future Research Problems and Discussion

There are several problems that may be studied to improve deflection networks. The problems include bandwidth reservation, connection service, broadcasting, and multicasting.

Bandwidth reservation and connection service are closely related. Given that the packet delay is much smaller than the end-to-end delay required by connection service, the only resource to be managed is the rate of the network access granted to a node [14].

Note that connection service through virtual circuits can hardly be provided in deflection networks without buffering packets at intermediate nodes. The use of the buffering resources increases the complexity of the nodal structures and slows nodal operations down. Moreover, establishing virtual circuits requires (additional) complex resource management.

Although deflection networks generally have regular topologies, which support simple routing schemes, broadcasting in the networks is not straight forward. Broadcasting methods that involve packet replications at branching nodes (such as flooding) are not suitable for deflection networks. Since the availability of links is a random event, the replications of packets may not be possible at branching nodes (immediately after the arrival of the packets).

In [29], Hamiltonian broadcasting (linear broadcasting), which involves no packet replications, has been proposed for the bidirectional MSN (torus network). It may not be difficult to find a Hamiltonian path in a structured network; however, in general networks, the problem of finding a Hamiltonian path is an NP-complete problem. Hence, Hamiltonian-path-based broadcasting is not always possible.

For the same reason for broadcasting being difficult, multicasting that involves packet replications is also difficult in deflection networks. Furthermore, the problem of constructing a minimum cost multicasting tree (the Steiner problem) is an NP-

complete problem even for regular networks such as meshes [82]. Efficient multicasting seems to be inherently difficult in deflection networks.

Note that both broadcasting and multicasting can be done by repeated unicasting at the source node, of course.

All of the above problems come from the unpredictability of deflection routing.

As for formal specification, developing a specification for a large concurrent system remains a difficult task. Research opportunities for providing large but non-cumbersome case studies still exist.

BIBLIOGRAPHY

[1] G. Albertengo, R. Cigno, and G. Panizzardi. The deflection network: A reliable high speed packet network for computer communication. In *Proceedings of the IEEE International Conference on Computer Systems and Software Engineering (COMPEURO '91 the 5th Annual European Computer Conference)*, pages 84–88, Bologna, Italy, May 13–16, 1991.

[2] G. Albertengo, R. Cigno, and G. Panizzardi. Optimal routing algorithms for the bidirectional Manhattan street network. In *Conference Record of the IEEE International Conference on Communications (ICC '91)*, pages 1676–1680, Denver, Colorado, June 23–26, 1991.

[3] G. Albertengo, R. Cigno, and G. Panizzardi. Simplified routing algorithms for the bidirectional Manhattan street network. In O. Spaniol and A. Danthine, editors, *High Speed Networking, III*, pages 127–135. Elsevier Science Publishers B.V. (North-Holland), 1991.

[4] G. Albertengo, P. Civera, R. Cigno, G. Piccinini, M. Zamboni, F. Borgonovo, L. Fratta, and G. Panizzardi. Deflection network: Principles, implementation, services. *European Transactions on Telecommunications*, 3(2):195–206, 1992.

[5] K. Apt and E. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, New York, 1991.

[6] P. Baran. On distributed communication networks. *IEEE Transactions on Communication Systems*, CS-12(1):1–9, 1964.

[7] C. Baransel, W. Dobosiewicz, and P. Gburzynski. Routing in multihop packet switching networks: Gb/s challenge. *IEEE Network*, 9(3):38–61, 1995.

[8] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[9] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 1992.

[10] K. Bolding, M. Fulgham, and L. Snyder. The case for chaotic adaptive routing. *IEEE Transactions on Computers*, 46(12):1281–1291, 1997.

[11] F. Borgonovo and E. Cadorin. HR^4-NET: A hierarchical random-routing reliable and reconfigurable network for metropolitan area. In *Proceedings of the IEEE Conference on Computer Communications, (INFOCOM '87)*, pages 320–326, San Francisco, California, March 31–April 2, 1987.

[12] F. Borgonovo and E. Cadorin. Routing in the bidirectional Manhattan network. In L. Moraes, E. Silva, and L. Soares, editors, *Data Communication Systems and Their Performance*, pages 181–189. Elsevier Science Publishers B.V. (North-Holland), 1988.

[13] F. Borgonovo and E. Cadorin. Locally-optimal deflection routing in the bidirectional Manhattan network. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '90)*, pages 458–464, San Francisco, California, June 3–7, 1990.

[14] F. Borgonovo and L. Fratta. Deflection networks: Architectures for metropolitan and wide area networks. *Computer Networks and ISDN Systems*, 24(2):171–183, 1992.

[15] J. Brassil, A. Choudhury, and N. Maxemchuk. The Manhattan street network: A high performance, highly reliable metropolitan area network. *Computer Networks and ISDN Systems*, 26(6–8):841–858, 1994.

[16] M. Burstall. Program proving as hand simulation with a little induction. In *Proceedings of the IFIP Congress (Information Processing 74)*, pages 308–312, Stockholm, Amsterdam, August 5–10, 1974.

[17] K. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, Reading, Massachusetts, 1988.

[18] Y. Chen and G. Sasaki. Routing in quasi torus networks. *Networks*, 28(4):195–209, 1996.

[19] A. Choudhury and V. Li. An approximate analysis of the performance of deflection routing in regular networks. *IEEE Journal on Selected Areas in Communications*, 11(8):1302–1316, 1993.

[20] T. Chung. *Design and Analysis of Manhattan Street Class of Doubly Linked Networks.* PhD thesis, North Carolina State University, 1989.

[21] T. Chung and D. Agrawal. On network characterization of and optimal broadcasting in the Manhattan street network. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '90)*, pages 465–472, San Francisco, California, June 3–7, 1990.

[22] T. Chung and D. Agrawal. Design and analysis of multidimensional Manhattan street networks. *IEEE Transactions on Communications*, 41(2):295–298, 1993.

[23] T. Chung, N. Sharma, and D. Agrawal. Cost-performance trade-offs in Manhattan street network versus 2-D torus. *IEEE Transactions on Computers*, 43(2):240–243, 1994.

[24] E. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[25] P. Collette and E. Knapp. A foundation for modular reasoning about safety and progress properties of state-based concurrent programs. *Theoretical Computer Science*, 183(2):253–279, 1997.

[26] C. Creveuil and G. Roman. Formal specification and design of a message router. *ACM Transactions on Software Engineering and Methodology*, 3(4):271–307, 1994.

[27] H. Cunningham and Y. Cai. Specification and refinement of a message router. In *Proceedings of the Seventh International Workshop on Software Specification and Design*, pages 20–29, Redondo Beach, California, December 6–7, 1993.

[28] H. Cunningham, V. Shah, and S. Shen. Devising a formal specification for an elevator controller. Technical Report UMCIS-1994-10, Software Methods Research Group, Department of Computer and Information Science, University of Mississippi, 1994.

[29] M. Dècina, V. Trecordi, G. Zanolini, and D. Zucca. Two simple techniques for broadcasting in deflection routing multichannel MANs. *Computer Networks and ISDN Systems*, 26(6–8):1023–1041, 1994.

[30] M. Dickie. *Routing in Today's Internetworks: The Routing Protocols of IP, DECnet, NetWare, and AppleTalk.* Van Nostrand Reinhold, New York, 1994.

[31] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[32] E. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[33] L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, New Jersey, 1962.

[34] N. Francez. *Fairness.* Springer-Verlag, New York, 1986.

[35] A. Greenberg and J. Goodman. Sharp approximate models of deflection routing in mesh networks. *IEEE Transactions on Communications*, 41(1):210–223, 1993.

[36] C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580,583, 1969.

[37] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall, New York, 1985.

[38] S. Kartalopoulos. A Manhattan fiber distributed data interface architecture. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '90)*, pages 141–145, San Diego, California, December 2–5, 1990.

[39] E. Knapp. An exercise in the formal derivation of parallel programs: Maximum flows in graphs. *ACM Transactions on Programming Languages and Systems*, 12(2):203–223, 1990.

[40] E. Knapp. Derivation of concurrent programs: Two examples. *Science of Computer Programming*, 19(1):1–23, 1992.

[41] A. Krishna and B. Hajek. Performance of shuffle-like networks with deflection. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '90)*, pages 473–480, San Francisco, California, June 3–7, 1990.

[42] R. Krishnan and N. Maxemchuk. Life beyond linear topologies. *IEEE Network*, 7(2):48–54, 1993.

[43] F. Kröger. *Temporal Logic of Programs.* Springer-Verlag, Berlin, 1987.

[44] T. Kubo and K. Yoguchi. Highway transfer: A new packet forwarding technique for real-time applications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '90)*, pages 403–408, San Francisco, California, June 3–7, 1990.

[45] J. Kurose and H. Mouftah. Simulation of communication networks. In T. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation*, pages 223–231. Springer-Verlag, New York, second edition, 1994.

[46] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[47] W. Lee and L. Kung. Binary addressing and routing schemes in the Manhattan street network. *IEEE/ACM Transactions on Networking*, 3(1):26–30, 1995.

[48] B. Lester. *The Art of Parallel Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[49] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[50] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1992.

[51] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[52] N. Maxemchuk. The Manhattan street network. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '85)*, pages 255–261, New Orleans, Louisiana, December 2–5, 1985.

[53] N. Maxemchuk. Regular mesh topologies in local and metropolitan area networks. *AT&T Technical Journal*, 64(7):1659–1685, 1985.

[54] N. Maxemchuk. Routing in the Manhattan street network. *IEEE Transactions on Communications*, COM-35(5):503–512, 1987.

[55] N. Maxemchuk. Comparison of deflection and store-and-forward techniques in the Manhattan street and shuffle-exchange networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '89)*, pages 800–809, Ottawa, Ontario, Canada, April 23–27, 1989.

[56] N. Maxemchuk. Problems arising from deflection routing: Live-lock, lockout, congestion and message reassembly. In G. Pujolle, editor, *High-Capacity Local and Metropolitan Area Networks: Architecture and Performance Issues*, pages 209–233. Springer-Verlag, Berlin, 1991.

[57] N. Maxemchuk and R. Krishnan. A comparison of linear and mesh topologies — DQDB and the Manhattan street network. *IEEE Journal on Selected Areas in Communications*, 11(8):1278–1289, 1993.

[58] J. McQuillan, G. Falk, and I. Richer. A review of the development and performance of the ARPANET routing algorithm. *IEEE Transactions on Communications*, COM-26(12):1802–1811, 1978.

[59] G. Michelis, L. Pomello, E. Battiston, F. Cindio, and C. Simone. Formal methods: A Petri nets based approach. In A. Zomaya, editor, *Parallel and Distributed Computing Handbook*, pages 59–88. McGraw-Hill, New York, 1996.

[60] A. Milner. *A Calculus of Communicating Systems (Lecture Notes in Computer Science 92)*. Springer-Verlag, Berlin, 1980.

[61] J. Misra. A logic for concurrent programming: Progress. *Journal of Computer and Software Engineering*, 3(2):273–300, 1995.

[62] J. Misra. A logic for concurrent programming: Safety. *Journal of Computer and Software Engineering*, 3(2):239–272, 1995.

[63] R. Nair. *Analysis of Routing Algorithms for Large Networks.* PhD thesis, University of Maryland, Baltimore County, 1993.

[64] R. Newman, Z. Budrikis, and J. Hullett. The QPSX MAN. *IEEE Communications Magazine*, 26(4):20–28, 1988.

[65] N. Nezu and H. Lu. An asynchronous address updating scheme for expanding torus networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '96)*, pages 407–415, Sunnyvale, California, August 9–11, 1996.

[66] N. Nezu and H. Lu. Incremental design and routing schemes for torus networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97)*, pages 1302–1307, Las Vegas, Nevada, June 30–July 3, 1997.

[67] N. Nezu and H. Lu. Incremental construction of torus networks. In *Proceedings of the ACM Symposium on Applied Computing (SAC '98)*, pages 80–84, Atlanta, Georgia, February 27–March 1, 1998.

[68] N. Nezu and H. Lu. Performance of toroidal deflection networks. In *Proceedings of the Second IASTED International Conference European Parallel and Distributed Systems (Euro-PDS '98)*, pages 103–110, Vienna, Austria, July 1–3, 1998.

[69] N. Nezu and H. Lu. Livelock prevention in deflection networks. In *Proceedings of the ACM Symposium on Applied Computing (SAC '99)*, pages 96–97, San Antonio, Texas, February 28–March 2, 1999.

[70] N. Nezu and H. Lu. Modeling deflection networks: Design and specification. In *Proceedings of the ACM Symposium on Applied Computing (SAC '99)*, pages 66–73, San Antonio, Texas, February 28–March 2, 1999.

[71] R. Perlman. Routing protocols. In D. Lynch and M. Rose, editors, *Internet System Handbook*, pages 157–181. Addison-Wesley, Reading, Massachusetts, 1993.

[72] A. Pizzarello. An industrial experience in the use of UNITY. In J. Banâtre and D. Métayer, editors, *Research Directions in High-level Parallel Programming Languages: Mont Saint-Michel, France, June 17–19, 1991: Proceedings (Lecture Notes in Computer Science, 574)*, pages 39–49. Springer-Verlag, Berlin, 1992.

[73] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Annual Symposium Foundations of Computer Science*, pages 46–57, Providence, Rhode Island, October 31–November 2, 1977.

[74] T. Robertazzi. Toroidal networks. *IEEE Communications Magazine*, 26(6):45–50, 1988.

[75] T. Robertazzi and A. Lazar. Deflection strategies for the Manhattan street network. In *Conference Record of the IEEE International Conference on Communications (ICC '91)*, pages 1652–1658, Denver, Colorado, June 23–26, 1991.

[76] C. Rose. Mean internodal distance in regular and random multihop networks. *IEEE Transactions on Communications*, 40(8):1310–1318, 1992.

[77] F. Ross. FDDI — A tutorial. *IEEE Communications Magazine*, 24(5):10–17, 1986.

[78] P. Roth, M. Ilyas, and H. Mouftah. Simulation: A powerful tool for prototyping telecommunications networks. *Simulation*, 58(2):78–82, 1992.

[79] B. Sanders. Eliminating the substitution axiom from UNITY logic. *Formal Aspects of Computing*, 3(2):189–205, 1991.

[80] B. Sanders. On the UNITY design decisions. In J. Banâtre and D. Métayer, editors, *Research Directions in High-level Parallel Programming Languages: Mont Saint-Michel, France, June 17–19, 1991: Proceedings (Lecture Notes in Computer Science, 574)*, pages 50–63. Springer-Verlag, Berlin, 1992.

[81] M. Schwartz and T. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications*, COM-28(4):539–552, 1980.

[82] A. Shaikh, S. Lu, and K. Shin. Localized multicasting routing. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '95)*, pages 1352–1356, Singapore, November 14–16, 1995.

[83] D. Smitley. Performance of networks using deflection routing. Technical Report SRC-TR-92-082, Supercomputing Research Center, Institute for Defense Analyses, 1992.

[84] W. Stallings. *Local and Metropolitan Area Networks.* Macmillan, New York, fourth edition, 1993.

[85] M. Staskaukas. Formal derivation of concurrent programs: An example from industry. *IEEE Transactions on Software Engineering*, 19(5):503–528, 1993.

[86] M. Staskauskas. The formal specification and design of a distributed electronic funds-transfer system. *IEEE Transactions on Computers*, 37(12):1515–1528, 1988.

[87] A. Tanenbaum. *Computer Networks.* Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1996.

[88] A. Tocher. LOTOS and the formal specification of communication standards: An example. In P. Scharbach, editor, *Formal Methods: Theory and Practice*, pages 5–51. CRC Press, Boca Raton, Florida, 1989.

[89] T. Todd. The token grid network. *IEEE/ACM Transactions on Networking*, 2(3):279–287, 1994.

[90] T. Todd and E. Hahne. Multiaccess mesh (multimesh) networks. *IEEE/ACM Transactions on Networking*, 5(2):181–189, 1997.

[91] J. Wong and Y. Kang. Distributed and fail-safe routing algorithms in toroidal-based metropolitan area networks. *Computer Networks and ISDN Systems*, 18(5):379–391, 1990.

VITA

Nobuyuki Nezu

Candidate for the Degree of

Doctor of Philosophy

Thesis: DEFLECTION NETWORKS: PERFORMANCE EVALUATION, LIVELOCK PREVENTION, AND FORMAL SPECIFICATION

Major Field: Computer Science

Biographical:

Education: Graduated from Gakushuin University, Tokyo, Japan with a Bachelor of Science degree in Mathematics in March 1991; received a Master of Science degree in Computer Science from Oklahoma City University, Oklahoma City, Oklahoma in May 1993. Completed the requirements for the Doctor of Philosophy degree in Computer Science at Oklahoma State University in July 1999.

Professional Experience: Graduate Teaching Assistant; Computer Science Department, Oklahoma State University, January 1994 to December 1998.

Professional Memberships: Association for Computing Machinery, Sigma Xi.