The Case for Chaotic Adaptive Routing
Technical Report CSE-94-02-04

Kevin Bolding, Melanie L. Fulgham and Lawrence Snyder
Dept. of Computer Science and Engineering
University of Washington
Seattle, Washington

Abstract

Chaotic routers are randomizing, non-minimal adaptive packet routers designed for use in the communication networks of parallel computers. Chaotic routing is reviewed along with other contemporary network routing approaches, including the state-of-the-art oblivious routers. Each routing approach is evaluated for its effectiveness as a multicomputer message router. The results indicate that the Chaos router is the most effective of known routing methods.

1 Introduction

In spite of the fact that network routing has been an active research area in recent years, leading to many diverse proposals, practical experience with routers is extremely limited. The routers used in most implemented parallel computers are all from a single class, known as oblivious routers. Most of the non-oblivious routers have appeared only in single instance machines such as the HEP, CM-2, and CM-5 computers, making it difficult to separate fundamental properties of the routers from artifacts of the specific instances. Researchers are naturally reluctant to implement a router design for several reasons. First, being a fundamental component of a parallel computer that must be fast, routers demand aggressive designs that are notoriously difficult to get perfect. Second, once the router is working it is useless unless it is incorporated into a communication network, which is itself incorporated into a parallel computer. After the computer is operational, it is necessary to write, debug, run, and measure numerous application programs to gather performance statistics. No one would undertake such a monumental effort without some certainty that the router materially improves on the alternatives. The purpose of this paper to compare the Chaos router with state-of-the-art routers, as well as other recent proposals.

2 Terminology

Of the many router designs, most can be classified as either *oblivious* or *adaptive*, depending on whether the path selection is statically determined based on the network topology or if dynamic information about congestion, priorities, or faults is considered in path selection. Adaptive algorithms can be broken down into two broad categories as well: *minimal* or *non-minimal*. Minimal adaptive algorithms allow only topologically minimal-length paths, while non-minimal algorithms allow a larger set of paths, possibly even of infinite length. A wide variety of mechanisms exist within each of the categories to provide reliability and performance.

Another distinguishing feature of routers is the method of *flow control* used. Flow control deals with the way that individual bits of data in a message are grouped into units which can be transmitted through the interconnection network. Messages may be sent directly through the

network, or may be broken up into fixed size *packets*. Also, buffering of many sorts may be added to the network.

Regardless of the method used, the goals or routers are similar. All routers attempt to correctly deliver the greatest number of messages in the least amount of time. The rate at which a network delivers data is known as its *throughput*. The time for a message to travel from its source to its destination is known as *latency*. In general, latency is desired to be small, and throughput large.

3 Flow Control Techniques

Flow control concerns the method in which messages are transmitted and buffered within a network. Specifically, the method of flow control in a network describes how messages are broken up into individual bits to be transmitted and how these bits are assigned to buffers and channels in the network. There are many differing forms of flow control present in network routers, so only the most basic forms are described here.

In its simplest form, network communication can be accomplished in two basic manners: *circuit-switched* and *packet-switched* communication. In circuit-switched routing, the entire path from source to destination is acquired first, then communication proceeds. The path is relinquished only after the message has been completely transmitted. The advantages of circuit-switching include simplicity of transmission once the path is set up, high utilization of the bandwidth during transmission, and low overhead for routing information. The drawbacks include wasted bandwidth during the acquiring/relinquishing phases, the need to acquire the *entire* path before transmission, and the inability for paths to change during a message transmission.

On the other hand, in packet-switched store-and-forward communication, messages are broken up into fixed size *packets*, each independently routed towards their destination. Each link in the path (a *hop*) is acquired only when needed and relinquished when the packet has traversed that link. The packet is stored in any intermediate nodes and then re-transmitted to the next node in its path from source to destination. Store-and-forward communication has the advantage of only requiring link resources when they are actually in use and allowing changes in paths during transmission if necessary. The primary disadvantage is the need to receive and retransmit packets at each hop. Other disadvantages are the overhead time for packetization and reassembly, the additional overhead needed to replicate routing information in each packet, and the space needed to buffer the packets at each intermediate node.

Both circuit-switched and store-and-forward communication have variants which incorporate the advantages of the other. Circuit-switched communication has evolved into *wormhole* routing with *virtual channels*. Wormhole routing is an extension of circuit-switching which allows incremental claiming and relinquishing of links and buffers in a network [Dally 87]. Messages are transmitted immediately behind the header of the message as the path in the network is established, blocking if necessary. In wormhole routing, data is transmitted in units of *flits*, where each flit is typically the amount of data that can be transmitted over a channel during a single cycle. Resources are acquired in flit-sized units, allowing buffer sizes to be independent of the message sizes. Unlike circuit switching, the path from source to destination is destroyed immediately behind the tail of the message. Thus, only the resources of the network presently transmitting or buffering messages are reserved at any given time. Virtual channels add buffering to wormhole routing so that messages may temporarily relinquish ownership of links due to blocking, priorities, or other reasons and allow non-blocked or higher priority messages to pass them [Dally 92].

Store-and-forward communication has been augmented by *virtual cut-through*, which relaxes the requirement that an entire packet be stored in a node before it is forwarded towards its destination [Kermani & Kleinrock 79]. In a virtual cut-through router, arriving packets which have a desired outgoing link available may proceed as soon as the header arrives without having to wait for the entire packet to arrive. If no desired outgoing links are immediately available, the packet is stored in the node until a link becomes available.

The distinctions between virtual cut-through and wormhole routing with virtual channels are less clear than between store-and-forward and circuit switched communication. Aside from implementation details, the primary differences are that cut-through requires packetization and the accompanying overhead, as well as enough storage at each node to hold the entire packet when it becomes blocked. Virtual cut-through, however, retains the advantages of using fixed-size packets which include a wider variety of deadlock-avoidance mechanisms and independent routing of individual packets. In the following sections, most of the path-selection algorithms work equally well with either virtual cut-through or wormhole routing. The primary exception to this is that the packet-exchange method of deadlock avoidance requires packet-switched communication.

4 Network Routing

Routing in an interconnection network is concerned with determining a path for a message to travel in getting from a source node to a destination node. In general, the route a message takes can be described by specifying an ordered list of nodes, $(N_{source}, \ldots, N_{dest})$, which describe a path starting at the source node and ending at the destination node. Routing techniques differ in how they respond to congestion and faults within a network, as well as the degree of flexibility in choosing a path that is allowed within "normal" operation.

4.1 Oblivious Routing

In an *oblivious* router, the path taken by a message is determined solely by its source and destination addresses, without regard to the current state of the network. Thus, all messages which have the same source and destination will follow the same predetermined path through the network. This path is usually chosen to be a topologically shortest path in the network. Some oblivious routers may allow reconfiguration of the routing algorithm at startup time to account for faulty links in the network, but this is not allowed during normal operation.

4.1.1 Oblivious Path Selection

Oblivious routers have gained wide acceptance because they provide reasonable service with great simplicity. For common networks, such as the k-ary n-cube family, topologically shortest paths are found easily. In this family, n dimensions of k nodes each form a network of $N = k^n$ nodes. Included are hypercubes, meshes, and tori of all dimensions. A shortest path for a message in such a network is a path in which each hop takes the message closer to its destination in exactly one dimension. Several shortest paths may exist because the ordering of the hops with respect to dimension does not effect the outcome; the path remains a topologically shortest path regardless of the order in which the dimensions are traversed.

Dimension order routing is a shortest-path algorithm which mandates that a message traverse the network dimensions in dimension order. A message will not traverse a link in dimension d until all dimensions earlier than d in the order have displacements of zero. This provides a simple distributed incremental path selection algorithm which can be implemented by finding the lowest-order dimension with a non-zero displacement from its destination and routing the message along the link which reduces this displacement. When all displacements are zero, the message has arrived at its destination.

4.1.2 Oblivious Deadlock Prevention

Dimension order routing can also be shown to be deadlock-free for open-ended k-ary n-cubes (those without "wrap-around" edges) using simple resource-ordering arguments [Dally 87]. Deadlock cannot occur within a dimension because the open-ended network does not allow cycles within a dimension. Since messages only change the dimension being traversed in a pre-determined order, no cycles can exist between dimensions, either. Assuming that all messages which arrive at their destination are eventually removed from the network, the lack of cyclic dependencies guarantees freedom from deadlock. In order to provide deadlock-freedom for dimension order routing in networks with wrap-around links (such as torus networks), additional mechanisms must be provided to alleviate deadlock within a single dimension. Dally and Seitz provide a solution to this using virtual channels which employs extra buffering to remove the possibility of cyclic dependencies within a single dimension [Dally & Seitz 87]. This mechanism perturbs the network's balance, though, resulting in non-uniform traffic flow through the network and degraded performance [Bolding 92, Adve & Vernon 93].

4.1.3 Congestion Avoidance

Since oblivious routers by nature ignore congestion in the network, it might be expected that they do not perform well under highly congested loads. In fact, Borodin and Hopcroft have shown that, for any N node, degree d network, oblivious routing requires $\Omega(\sqrt{N}/d^{3/2})$ time steps to route a permutation [Borodin & Hopcroft 85]. This bound was tightened to $\Omega(\sqrt{N}/d)$ by Kaklamanis, Krizanc and Tsantilas [Kaklamanis et al. 90].

Valiant and Brebner avoided this bottleneck by introducing a randomized oblivious scheme for the hypercube – messages are first sent to a randomly chosen intermediate node and then forwarded to their destination [Valiant & Brebner 81]. Although this probabilistically removes the theoretical worst-case bottleneck, it does so at the cost of doubling the average path lengths of messages in the network.

For certain traffic patterns, though, oblivious routing performs very well. Nearest-neighbor communication patterns produce very little congestion and are served very well by oblivious routing. Uniform random traffic in networks with no "center" nodes (*i.e.* tori and hypercubes) produces uniform random congestion for which oblivious routing works well. On the other hand, networks with "center" nodes, such as open-ended meshes, have more congestion in the center. Here, one might expect oblivious routing to perform poorly. For dimension-order oblivious routing, though, the scheme spreads the traffic out and helps reduce the effects of the central hot spot, producing very good results [Pertel 92].

For non-uniform traffic, though, oblivious routers do not perform as well. In the presence of hot spots, some nodes in the network will become much more congested than others. However,

since oblivious routers are do not respond to hot spots, paths will be chosen through congestion even when equal-length paths with no congestion exist as alternatives. This is a serious drawback to oblivious routing since many software applications create hot spots or other non-uniform traffic [Pfister & Norton 85].

4.1.4 Buffering

Buffering can be applied to oblivious routing in many ways. Two basic types of buffering are available: blocking and non-blocking. Blocking buffering is simply in-place buffering of messages. In blocking buffering, messages do not give up the resources they hold when they cannot move in the network. The most common way to add simple blocking buffering is to add FIFO's at any desired point in any path in the network. Non-blocking buffering adds storage to the router that is off of the main data paths. In non-blocking buffering, messages that are buffered do not consume critical resources and can be bypassed by other messages which have resources available to them. Non-blocking buffering can be implemented using a shared buffer pool for virtual cut-through routing or by virtual channels with FIFO's for wormhole routing [Dally 92].

Adding blocking buffering to a network by inserting FIFO queues between links can help performance by allowing packets to relinquish control of links to other messages earlier. Dias and Jump showed that, for delta multistage networks, adding just a few buffers between stages could double the throughput of the network [Dias & Jump 81]. However, adding buffers also increases latency by adding additional complexity to the routing path. Dally examined non-blocking buffering by adding virtual channels to wormhole routing and found significant performance advantages for a range of network sizes when compared to networks with equal amounts of blocking buffering [Dally 92]. In general, adding a moderate amount of buffering to an oblivious router can significantly expand its throughput capacity with only a small amount of additional latency per message [Konstantinidou 91].

4.2 Minimal Adaptive Routing

Since most networks provide multiple shortest paths between many, if not most, source-destination pairs, it is intuitive to design a router that allows flexibility in choosing among these paths. Routers which allow some choice among minimal-length paths based on local or temporal conditions are known as *minimal adaptive* routers.

In the "basic" minimal adaptive router, the router computes the set of all outgoing links which lie on some minimal path for an incoming packet. These are known as *profitable* links. Of the profitable links, some may be currently unavailable due to traffic, routing restrictions, or faults in the network; these are removed from the set. The packet is then sent out on one of the remaining profitable links. If none is available, the packet waits for one to become available, in either a blocking or non-blocking manner depending on the router.

When more than one profitable link is available, the router must somehow choose from the alternatives. There are several alternative methods: random, dimension-order, *zigzag*, and *no-turn* [Glass & Ni 91]. The first two methods are obvious in implementation. The *zigzag* method attempts to maximize the number of minimal paths still available at any time by choosing the dimension in which the packet has the largest displacement from its destination. Although this helps to preserve the largest set of profitable hops as the packet progresses towards its destination, its usefulness is marginal at best [Ngai 89]. *No-turn* is based on the observation that when a packet "turns" or

changes dimensions, it blocks packets traveling in two directions, while it would block packets only in one direction otherwise. Thus, the no-turn model chooses the link which minimizes the number of turns when possible.

A problem which arises in minimal adaptive routing is deadlock. The constraints which prevented deadlock in dimension-order routing are no longer present in adaptive routing, so other techniques must be added to ensure freedom from deadlock. Routers based on the turn model [Glass & Ni 92] restrict routing by forbidding certain turns. The Zenith router [Konstantinidou 90] and planar-adaptive routing [Chien & Kim 92] require extra buffering as well as extra constraints on packet paths in order to prevent deadlock. Other routers [Pifarré et al. 91, Cypher & Gravano 92, Linder & Hardin 91, Felperin et al. 91, Berman et al. 92, Boppana & Chalasani 92] are *fully-adaptive minimal*, in that all minimal paths are allowed[1], although some amount of extra buffering is required. One disadvantage of some of these algorithms [Chien & Kim 92, Linder & Hardin 91, Berman et al. 92] is that different buffer capacities for different nodes in the network are required, creating non-uniformities which disturb the network's performance.

However the main problem with minimal adaptive routers is that, in general, the number of paths available decreases as the distance to the destination decreases. Packets which have a non-zero displacement in only one dimension have only one path to choose from and can no longer avoid congestion or faults. Even when there is flexibility, packets near their destinations lose their ability to maneuver around congestion. This is especially evident in a result by Chinn, Leighton, and Tompa where the worst case routing time of a permutation on an N-node mesh for a class of minimal adaptive packet routing algorithms is shown to be $\Omega(N/k^2)$ when k is the number of packets that can be buffered in a single node [Chinn et al. 93].

4.3 Non-minimal Adaptive Routing

While oblivious and minimal adaptive routers require a packet always be routed on a minimal length path, non-minimal adaptive routers allow packets to be routed on paths from the source to destination with paths that are not always restricted to shortest paths. There are two primary motivating factors for allowing packets to follow longer than necessary paths: congestion avoidance and fault-tolerance.

If all minimal paths for a packet are heavily congested, but longer paths are uncongested, it may be possible for a packet to travel out of its way on a longer path but still arrive more quickly. Consider a highway analogy: although the shortest path to where you want to go may be on a freeway, local streets may be faster during rush hour when the freeway is congested even though the distance is further. Also, when packets detour around a congested area, they avoid adding to and worsening the congestion. Moreover, if hardware faults result in a link becoming permanently unavailable, there will be some pairs of nodes in networks using minimal routers which can no longer communicate. By allowing non-minimal paths, these nodes can be avoided and communication is still possible.

Non-minimal routers face different problems in deadlock prevention and must deal with prevention of *livelock*. Solutions to these are presented below. Two performance issues also arise: the time lost in taking potentially longer paths and delay in making routing decisions due to the greater complexity of the decision. These issues are discussed in Section 5.

[1] This is not strictly true for the Linder-Harden algorithm with even size dimensions for wrapped k-ary n-cubes.

4.3.1 Derouting

The decision to route a packet on an outgoing link *away* from its destination is known as *derouting*[2]. Of the routers presented in the literature, there are many differing methods of deciding when and which packet to deroute. However, based on their buffering characteristics, the three basic classes of non-minimal adaptive routers are: *deflection* routers, *queueing* routers, and *wormhole* routers.

Deflection routers [Smith 81, Maxemchuk 89, Fang & Szymanski 91, Smitley 89], also known as "hot-potato" or desperation routers, use a synchronous approach and a time step long enough to transmit an entire packet. At each step the incoming packets are paired with outgoing channels and are transmitted in the next step. Since deflection routers use a synchronous approach, it is guaranteed that all outgoing links will be available for use on the next routing cycle – the routing logic must attempt to find the best mapping of packets to outgoing links. In some cases, though, two or more packets may desire the same outgoing link. One packet will be given its desired link, while the other packets are assigned one of the remaining links. Thus, the "losing" packets are derouted immediately, without being buffered in the node.

Queueing non-minimal routers [Ngai & Seitz 89, Konstantinidou & Snyder 90, Coates et al. 93] differ from deflection routers by the presence of a central buffer which holds packets awaiting free outgoing links. In general, packets move into a central buffer from incoming links and wait there until a preferred outgoing link (a link on a minimal path) becomes available. If the central buffer becomes full, a packet is selected to be derouted on the next free outgoing link. Router designs may allow packets to bypass the queue altogether if an outgoing link is immediately available (cut-through), and there may be FIFO buffering on the input and output channels. The selection of which packet to deroute also depends on the implementation.

While both deflection and queueing non-minimal adaptive routers require that messages be broken into fixed-size packets, wormhole routers do not require this; and allow arbitrary size messages in the network. However this makes deadlock prevention more complex and requires either restrictions on routing [Glass & Ni 92, Dally & Aoki 92], multiple classes of virtual channels [Linder & Hardin 91, Boppana & Chalasani 92], or both [Dally & Aoki 92].

4.3.2 Deadlock Prevention

In fully adaptive non-minimal routers, any packet has the possibility of going out any free channel from its current location[3]. As a result, it is simple to show that deadlock cannot occur through path dependencies because there are no explicit paths. However, mechanisms to guarantee that the links themselves do not deadlock are still necessary.

Deflection routers achieve this trivially: there are no buffers and all network links are always free at the beginning of each routing cycle. All packets which enter a node during a routing cycle will exit during the next routing cycle, so all packets are always moving about.

Queueing routers usually rely on the packet-exchange protocol [Ngai & Seitz 89] which ensures that each bi-directional link in the network will not deadlock. Essentially, the protocol mandates that two nodes connected by a bi-directional link and having packets to send to each other must

[2]In some networks, there may be links available which take a packet neither closer nor further away from its destination. We will call these "no-progress" hops deroutes as well.

[3]In some livelock protection schemes, there may be some packets which cannot be derouted, but, as a general rule, this principle applies.

accept the other node's packet. In other words, node a cannot send a packet to node b while at the same time denying b's request to send a packet to a.

Other methods of deadlock prevention based on restricted resource claiming have also been presented [Coates et al. 93].

4.3.3 Livelock Prevention

A distinct problem which plagues non-minimal adaptive routers is the possibility of packets having infinite-length paths. If there is no bound on the number of times a packet may be derouted it is conceptually possible that a packet never makes progress towards its destination, even though it is continually moving. This is known as *livelock*. There are three basic methods of dealing with livelock: "do nothing," priorities, and randomness.

One method of dealing with livelock is to ignore it. The argument is that livelock is rare and pathological, so its likelihood is not great enough to worry about. Although this seems naive, many routers have been proposed without protection from livelock. Since livelock-causing traffic patterns exist, no matter how pathological, this is an unattractive solution.

There are two basic variations of the priority method: timestamps and battle scars. Timestamp protocols [Ngai 89] require that each packet be stamped with the time it was injected into the network. Whenever a router must choose a packet to deroute, it ensures that the oldest packet is not chosen. Thus, the oldest packet in the network is guaranteed to be routed on a minimal path to its destination; when it is delivered, there is a new "oldest" packet which is not derouted. Thus, infinite livelock is prevented since all packets will eventually be delivered or become the oldest packet and be guaranteed delivery.

Battle scar methods [Smith 81] place in each packet information which indicates how many times it has been derouted. When choosing a packet to deroute, the one with the smallest battle scar is selected – thus, packets which have been previously derouted are less likely to be continually derouted. Once a packet reaches some pre-determined maximum battle scar, it is routed on an Eulerian path through the network toward its destination which is guaranteed to be livelock-free. A similar method by Dally and Aoki [Dally & Aoki 92], allows messages to be derouted freely up to a maximum number of times, at which point they are routed deterministically by dimension order. A dynamic version of the algorithm does not have a fixed maximum number of deroutes but restricts messages to wait only on messages with a larger number of deroutes than itself. When this is impossible the message switches to oblivious routing as in the static version of the algorithm.

Priority methods suffer from two problems: complexity and overhead. The routing decision at each node becomes more complicated due to the necessity of comparing priorities when derouting. Also, the timestamp or battle scar must be transmitted in the header of the packet consuming bandwidth that could provide other crucial routing information.

The third solution is randomization. Chaotic routers [Konstantinidou & Snyder 90] choose randomly among queued packets when selecting a packet to deroute. By introducing randomness into the network in this manner, the very regular cycles, typical of livelock, decay with time. Livelock-freedom is provided only in a probabilistic sense: the probability a packet is in the network for a time greater than t goes to zero as t approaches infinity. Since the likelihood of livelock is low to start with, this provides a practical, though not a deterministic solution.

5 Comparisons

Since there are so many competing designs for multicomputer routers, it becomes difficult to determine which is the *best* design. Most implemented massively parallel machines use state-of-the-art oblivious routers with minimal buffering, so these routers obviously have merit. However, the alternative provided by adaptive routing seems very attractive due to its ability to provide better performance in the presence of congestion.

Because speed is of such importance in router design, complexity is very costly. The nature of the decisions which must be made by different classes of routers requires designs of varying complexity. At the heart of the design of a router is the "basic routing decision": assigning incoming packets to outgoing channels. This decision can be made with very little information for oblivious routers, but requires more work for adaptive routers.

The Oblivious Decision. For oblivious routers using dimension order routing, path selection is simple because it is defined in advance. All that is necessary is for the router to see if the header in the current dimension is non-zero: if so, the message "goes straight" along the same dimension, otherwise it "turns" to either the next dimension or the processor channel. It is not necessary to consider more than one dimension at a time or to consider whether buffers are full or links available when making the routing decision.

The Adaptive Decision. The adaptive routing decision is more complicated than the oblivious decision because the path taken must be computed based on dynamically changing information. Each message may have several profitable channels to choose from, some of which may be unavailable. The adaptive decision involves computing the profitable channels for an incoming message, masking out channels which are unavailable due to contention or faults, and selecting a single outgoing channel from those remaining.

The adaptive decision requires first decoding the header to find which outgoing links are desired. If all of the header information can be encoded into a single flit, the decoding can proceed immediately. Since each dimension can be decoded in parallel, there is no time lost over the oblivious decode step. However, whereas the oblivious router has finished its choice at this point, adaptive routers must continue on.

To make an informed choice among outgoing links, the router must use information on which of the outgoing links are available. Although this is simple, it requires information which is often widely distributed about the routing chip, and thus causes some delay. More importantly, as incoming messages claim outgoing links, the status changes. Thus, the decision must either be serialized or controlled by more complex logic to deal with contention. Finally, once the profitable, available channels are identified, one must be selected from the set. This can be done at random or by one of many ordering methods (see Section 4.2).

By closely examining the competing routers, we can dismiss several alternatives as inferior. We will then compare in detail the remaining contenders.

5.1 Oblivious – The One to Beat

Oblivious routers are simple and of mature design – a combination that produces a very fast router upon which it is hard to improve. A good example of this is the Caltech Mesh Routing Chip (MRC)

[Flaig 87, Seitz & Su 93]. The MRC is a simple oblivious wormhole routing chip for open-ended two-dimensional mesh networks. Simplicity is the rule: deadlock is prevented by dimension order routing, messages block in place when links are unavailable, and wormhole routing reduces overhead to a minimum. The routing decision for such a router can be broken down into two cascaded stages, each making the simple decision "go straight or turn" for a single dimension.

The CalTech MRC has undergone several generations of design and implementation, resulting in a finely-tuned routing chip. The routing decision is extensively pipelined so that the bandwidth of the routing channels is determined only by the chip-to-chip delay and not affected by the logic internal to the chip. Within the chip, the logic is kept to a minimum making input-to-output latency very low as well. Thus, the simplicity of oblivious dimension-order routing with minimal buffering results in a high bandwidth, low latency router.

Extensions to oblivious routers include simple FIFO buffering, non-blocking buffering using FIFO's and virtual channels [Dally 92], and virtual channels to provide deadlock prevention in torus networks[Dally & Seitz 86]. Each of these enhancements provides potential advantages at the cost of greater complexity.

The randomized oblivious router of Valiant [Valiant & Brebner 81], while addressing a theoretical bottleneck, does so at a high cost: doubling the average message path length in direct networks. The performance gain of randomization becomes irrelevant compared to this loss, so this extension is not competitive.

5.2 Minimal Adaptive Routers

Minimal adaptive routers must not only make the adaptive routing decision, but also implement a deadlock-prevention protocol. Although the protocols in the literature vary widely, most rely on multiple classes of buffers or queues in the node to hold messages at different stages in their path from source to destination. The router must compute and select which buffer class a message belongs to before assigning a buffer. This may involve a computation on the destination of the message or on information tagged to the message. Either way, the additional complexity of the decision, as well as the hardware space required for the extra buffers results in longer times to route messages through a node.

Some minimal adaptive designs restrict the minimal paths [Konstantinidou 90, Chien & Kim 92, Glass & Ni 92] or the order in which virtual channels may be taken [Cypher & Gravano 92, Boppana & Chalasani 92], causing uneven channel use and further diminishing the performance of the router relative to other fully-adaptive routers. Preliminary studies by Nguyen and Snyder have shown the Chaos torus to be superior to the Cypher-Gravano fully-adaptive minimal algorithm [Nguyen & Snyder 94]. The Cypher-Gravano algorithm restricts the order of virtual channel use causing bottlenecks in the first set of channels while the others experience little congestion. A possible intuitive explanation is that packets in adaptive routers "discover" congestion when all forward paths are blocked. For minimal adaptive routers this is "too late," since the minimum distance constraint forces the packet to wait, adding to the congestion. Nonminimal adaptive routers, however, allow messages to "go backwards" and possibly bypass the congestion. Another common problem with minimal adaptive routers is they often use many virtual channels. The more virtual channels needed the more complex the channel arbitration logic. Algorithms such as Linder-Harden [Linder & Hardin 91] and the hop-based schemes [Boppana & Chalasani 92] use a large number of virtual channels and are clearly impractical. In the former case $O(n2^n)$ and in

the latter O(nk) virtual channels per channel for a k-ary n-cube. Others such as 4-classes and *-channels use more modest number of virtual channels per physical channel, but the complexity necessary may not justify the performance gains.

5.3 Deflection Non-minimal Adaptive Routers

Deflection routers require that all incoming packets arrive at a node during a single routing cycle, and all packets leave on the next routing cycle. This constraint has several consequences on complexity and performance.

Once again, the adaptive routing decision must be made. However, since all packets must be routed at the same time, the decision must be made in parallel for all channels. Essentially, the decision is transformed from "find a free profitable channel for one packet" into "map all incoming packets to outgoing channels in the *best* manner." This decision may be implemented in several ways, from simple greedy algorithms to multiple-pass algorithms with priorities. Although the decision is simplified because all outgoing channels will be free (until assigned), the decision may be very complex. Furthermore, the complexity grows with the node degree, limiting the usefulness of deflection routing in high-degree networks[4].

The second consequence of the deflection technique is that the headers of all packets must arrive at and leave a node on the same cycle. Thus, all packets must be synchronized together so that their headers are aligned. The result is that neither virtual cut-through nor wormhole routing may be used because these methods would cause mis-alignment of the headers: store-and-forward is the only applicable method. While this is of no consequence if an entire packet can be transmitted in a single cycle, limits on the bandwidth of channels cause most multicomputer networks to have multi-flit packets which must incur the large latency penalties of pure store-and-forward routing when using deflection routing. For larger area networks with high-bandwidth cables or fiber-optic channels and less critical latency requirements, deflection routing may be applied more usefully.

Thus, although deflection routing provides two simplifying benefits: guaranteed availability of outgoing channels and no need to deal with internal buffering, the complexity of the mapping decision and the inability to use virtual cut-through routing limit its use as a multicomputer router. Simulation studies show that chaotic routing outperforms deflection routing on torus networks by providing both higher throughput and lower latency at all loads for all commonly used message sizes [Bolding 93].

5.4 Wormhole Non-minimal Adaptive Routers

Most wormhole non-minimal adaptive algorithms are straightforward extensions of the minimal adaptive wormhole versions of the algorithms[Linder & Hardin 91, Boppana & Chalasani 92, Glass & Ni 92]. As with their minimal versions, the Linder-Harden and Boppana-Chalasani algorithms suffer from the complexity of numerous virtual channels, while the Glass and Ni turn-based algorithms severely restict routing opportunities in a manner which depends upon the location of the node in the network.

The Dally and Aoki non-minimal adaptive wormhole algorithms, described briefly in Section 4.3.3, allow messages to be derouted. The static version requires a number of virtual channels

[4]The complexity of the deflection router used in the Tera computer resulted in changing the network, a degree 6 3D torus, to a degree 4 network (by removing X direction and Y direction wires in alternating planes) in order to make the routing decision within the 2.5 ns clock cycle.

per channel proportional to the maximum number of deroutes allowed, making the router complex to build. The dynamic version does not have this requirement, but has an additional routing restriction for waiting messages. However the main problem with both algorithms is they may deroute too often since messages are derouted immediately when all profitable channels are blocked, without waiting, even briefly, for the congestion to clear.

5.5 Queueing Non-minimal Adaptive Routers

Queueing non-minimal adaptive routers must perform the adaptive routing decision, determine when packets enter and leave the queue, and, when necessary, select which packet to deroute in a manner that does not cause livelock.

In a typical queueing router, packets arriving at the node are buffered in a central buffer pool or queue and wait for free profitable channels on a FIFO basis. Typically, this buffer is relatively small, ranging in size from five packets [Bolding & Snyder 92] to nineteen packets [Coates et al. 93] in implemented designs. When the internal queue becomes full and there are no profitable channels available for any packets in the queue, a packet is selected, regardless of its destination, from the queue to be derouted to the next available channel[5]. The method of selecting which packet to deroute varies among designs, primarily depending on what livelock prevention mechanism is used (see Section 4.3.3). The presence of the internal queueing buffer allows for a fully adaptive router where waiting packets do not block channel resources.

5.5.1 Livelock prevention

The primary distinction between different queueing non-minimal routers is in the method of livelock prevention as described in Section 4.3.3. Priority routers [Ngai & Seitz 89] use timestamps, chaotic routers [Konstantinidou & Snyder 90] use randomization, and others do not guard against livelock at all [Coates et al. 93].

The priority router is strictly more complex than a chaotic router because determining which packet among colliding packets is oldest significantly complicates the routing logic. The chaotic router does not need to make such decisions, giving it lower node latency and better overall performance [Konstantinidou 91]. Also, priority adaptive routers require the inclusion of a creation timestamp in each packet, using up bandwidth which could be used for data. Furthermore, the timestamp data must be in the header of a packet since the information is needed before the derouting decision can be made; this may extend the time to transmit the header to more than a single cycle, adding to the node latency.

Thus, within the class of queueing non-minimal routers, chaotic routing provides the lowest complexity, livelock-free design at the cost of providing only probabilistic protection. However, since the finite bounded delivery time of deterministic livelock protection provides little benefit due to the enormous magnitude of the bound, chaotic routing provides, in a practical sense, the same function at a smaller cost.

5.5.2 Queue management

Management of the central queue is a very complex and difficult task when compared with the simple "go straight or turn" oblivious routing decision. Packets must be allowed to enter the queue

[5]Packets are not derouted to the processor channel.

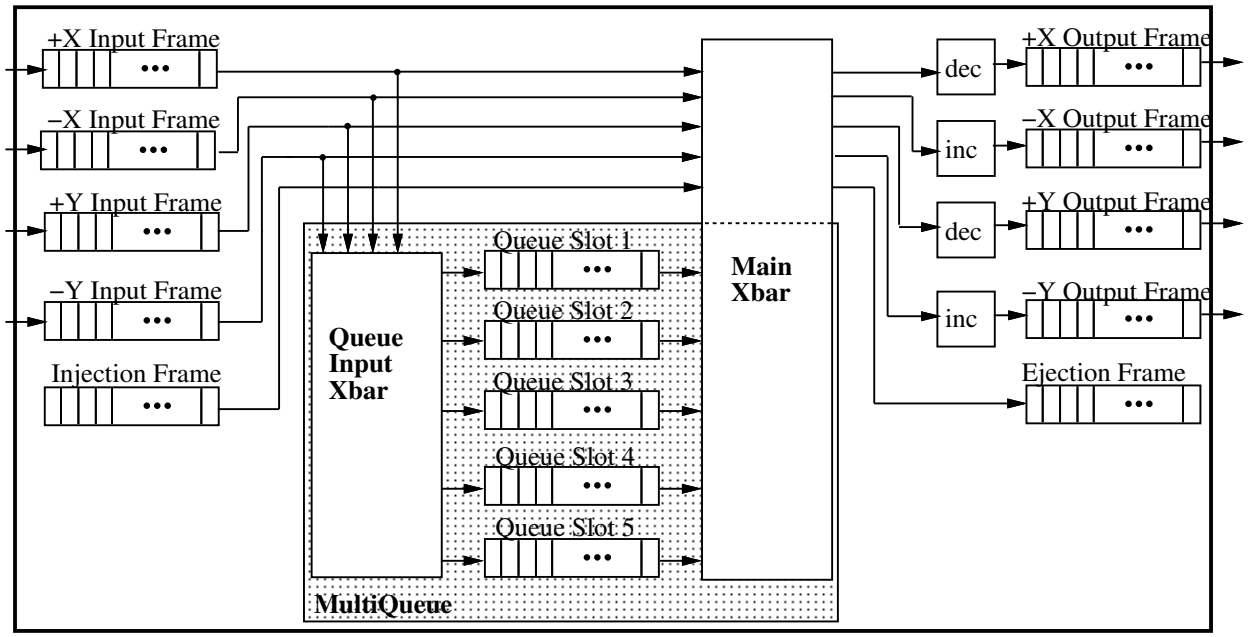

Figure 1: Two-dimensional Chaos router diagram.

in any order, but be removed from the queue in FIFO order by outgoing channels. In other words, when an outgoing channel becomes free, the oldest packet in the queue which can profitably use it must be assigned to it. When the queue fills up, selection of which packet to deroute must take into account the policy for derouting as well. Add to this the need to implement virtual cut-through, which complicates matters because the queue may be filled with some complete packets and some partial packets, and the complexity becomes overwhelming.

Chaotic routing. Chaotic routing attempts to minimize the impact of the queue management overhead by eliminating it from the critical path of the routing decision[Konstantinidou & Snyder 91]. In Chaos routers, single packet buffers are placed on both the input and output sides of each channel (see Figure 1). In "normal" operation, packets enter the node into an input frame, wait for an output frame of a profitable outgoing channel to become available, and move to the output frame in a virtual cut-through fashion. At the output frame, packets wait for the bi-directionally shared channel to become available and advance to the next input frame when it becomes free. Virtual cut-through minimizes the time spent in the buffers when channels are available. Thus, the "core" of a Chaos router looks like a minimal adaptive router without the need for multiple classes of queues for deadlock prevention.

Packets in Chaos routers are moved from input buffers into the central queue on two occasions: packet exchanges for deadlock prevention and "stalling" [Bolding & Snyder 92]. The packet-exchange deadlock prevention protocol mandates that if routers on either side of shared link have packets to send to each other, both packets must be sent, rather than only one side sending a packet. In order to guarantee this, the Chaos router implements the following protocol: if a packet is sent to the output frame for channel i and the input frame for channel i has an incoming packet,

the packet in input frame i is moved to the central queue.[6] If there is no space in the queue, a packet is derouted from it to make room for the newly arriving packet. The second method by which packets are moved into the queue is when they have stalled for a "reasonably long" time at an input frame while waiting for a profitable output frame. For performance reasons, stalled packets are moved to the queue to free up the input channel for other packets. The Chaos router determines that a packet is stalled when both its head and tail are buffered in the same input frame, *i.e.* it can no longer cut-through. Packets in the queue have priority over packets in input frames when competing for outgoing channels.

By allowing most packets to bypass the queue altogether, the Chaos router reduces the effect of the complexity of queue management. The critical routing decision is now simply the adaptive routing decision with two additional checks to control moving packets into the queue.

Consequently, chaotic routing should perform better than minimal adaptive routing, deflection routing, and priority routing due to simpler design than minimal and priority routers (deadlock and livelock prevention) and the ability to use virtual cut-through switching which deflection routers cannot use. Thus, chaotic routing is the best contender to compare with state-of-the-art oblivious routers.

6 Oblivious routing vs. Chaotic routing

Having argued that, among adaptive routers, chaotic routing is the best contender due to its simplicity and flexibility, we now compare in detail the Chaos router to an oblivious router. While simplicity of node design affects the critical path of a router, and, therefore, the time spent in each node, this is not the primary performance concern. The response of the network to a user's application is, of course, the best measure of a network's performance and usefulness. Unfortunately, such a study is infeasible because of the huge differences in how applications use computer networks. Instead, we will examine two common performance measures, network latency and throughput, under varying loads and traffic patterns.

6.1 Uncongested Network Performance

When a network is uncongested there are few conflicts for resources, so packets should be able to acquire needed buffers and links with little or no delay at all times. Since the network will have much greater capacity to hold packets than needed, there should be no backup of congestion and throughput should be equal to the applied load in all cases. Therefore, in the uncongested case, there will be no apparent difference in throughput between oblivious and chaotic routing.

Since throughput is not a distinguishing factor between chaotic and oblivious routing when congestion is low, the minimum latency for packets going through the network becomes the figure of merit. Since the chance of conflicts in routing is small, packets will rarely enter the multiqueue in Chaos routers. Because of this, it is almost certain that packets will never be derouted, so packets will almost always follow minimal-length paths from their sources to their destinations. In oblivious routers, the paths are always of minimal length, so a given packet will have to take the same number of network hops in either chaotic or oblivious routing when loading is light. Thus, only the differences in routing latency between oblivious and chaotic routing for packets following paths of the same length are of concern.

[6] Packets from the injection frame do not enter the queue due to deadlock prevention constraints.

Although there is a chance of conflict between two or more packets in the lightly-loaded case, only conflict-free routing is considered in the following analysis. Since chaotic routing allows packets to have a greater number of choices during each routing decision, packets will have fewer chances of conflict when using chaotic routing. Thus, the assumption of conflict-free routing is more likely to hold in chaotic routing. Since conflicts can only delay packets, this assumption tends to bias this analysis slightly in favor of oblivious routing.

The time for a packet to travel a minimum-length path from its source to its destination without conflicts is known as the *minimum network latency* for that packet. This figure includes all of the time from the injection of the packet header into the network until the entire packet is received at its destination. Since the path length is the same for both oblivious and chaotic routing, and conflicts are assumed not to occur, there are only two factors which affect the minimal network latency for a particular packet. These factors are the network cycle time, c, and the delay the message incurs at each node (in cycles), d. The minimum latency for a message of length L flits traveling D hops is then:

$$c \cdot [(D+1)d + L - 1)] \tag{1}$$

since the header of the packet is delayed d cycles for each of the D hops (plus one additional "hop" for delivery) and then the remaining $L-1$ flits follow in cut-through fashion. Since c and d are the only factors which can vary between chaotic and oblivious routers when congestion is small or non-existent, we compare these factors and their effects on the minimum network latency for the two routers.

6.1.1 Cycle Time

The network cycle time, c, is a critical factor in performance, as it is a multiplicative factor affecting the time to transfer each flit of the packet, as well as the time to process the header of the packet at each hop. The network cycle time must be long enough for the slowest function computed in a cycle to be completed. For most of the routing logic, pipelining techniques can be applied to support very short cycle times, although a penalty will be paid in overall latency. However, a fundamental limit on the network cycle time is imposed by the chip-to-chip interface in the network since data cannot get on or off the chip at a rate faster than the I/O pads and wires connecting them can be cycled. For a given technology and interface voltage, the limits on the speed of this interface will be constant regardless of the logic inside the routing chip. Thus, once the interface is defined, there is a maximum speed at which the chip can be run, and there is little point in attempting to make the internal logic run at a faster rate since data cannot get on or off of the chip any faster than the limited speed of the pad drivers.

The question remains as to whether the routers can be built fast enough to be limited only by the off-chip cycle time. With oblivious routers, pipelining of the simple routing decision, as in the Caltech MRC, allows the router to run as fast or faster than the pad drivers[7] [Seitz & Su 93]. Therefore, this limit is clearly achievable for the given technology, a 1.2μ CMOS process with a 5 V interface. For chaotic routing, the design presented in Section 8 achieves this using a multi-stage synchronous pipeline and the same technology. Thus, for both oblivious and chaotic routers, the network cycle time, c, is limited by the same factors and, thus, will be the same value.

[7] The MRC is a self-timed chip, so there are no explicit clocks.

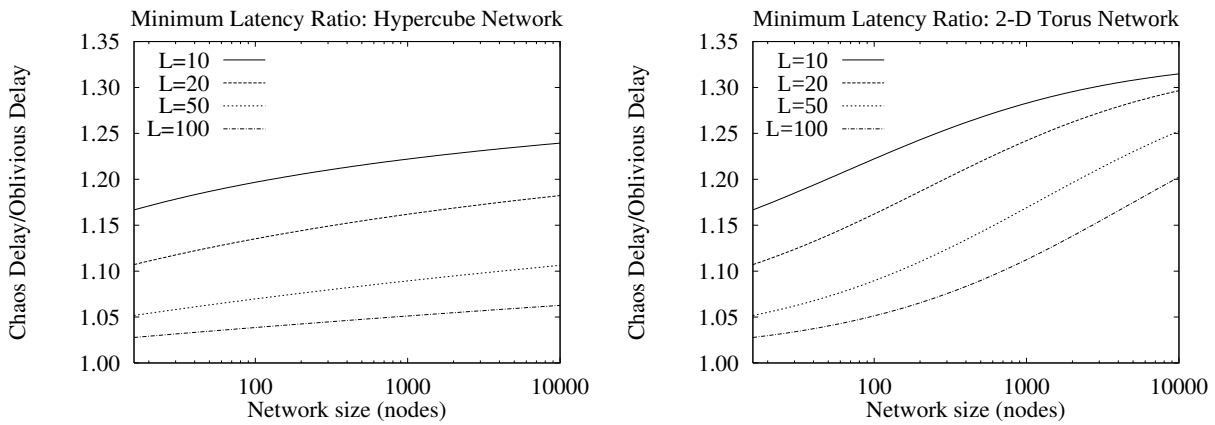

Figure 2: Ratio of minimum average latency with chaotic routing to minimum average latency with oblivious routing: Hypercube and 2-D torus networks.

6.1.2 Routing Latency

The second factor in comparing the minimum latency is d, the routing latency per node. This factor represents the time the header of a packet spends entering a node and waiting to be routed to an outgoing link, when there is no contention. Essentially, d reflects the complexity of the routing decision, and it should be expected to be higher for the Chaos router than for the oblivious router.

For an oblivious router with minimal buffering, the time allowed must be at least one cycle for the header to arrive on the inputs, one cycle for the routing decision to be made, and one cycle for the header to be routed to the outputs and updated. Although it might be possible to combine the second and third steps into a single cycle, this is unlikely given the clock speed constraint from the previous section. As interface technology moves toward lower voltage interfaces, the cycle time will be further reduced, possibly adding additional cycles to the routing decision as the pipeline depth is increased. Nonetheless, this results in a minimum of three cycles, so $d \geq 3$ cycles. The performance exhibited by the Caltech "Elko" MRC is consistent with our observation, having an asynchronous "cycle time" of 14.5 ns and minimum input-to-output latency of about 44.5 ns ($c =$ 14.5 ns and $d = 3$ cycles)[Seitz & Su 93].

With chaotic routing, packets are delayed a total of four cycles by the routing pipeline (see Section 8 for details). When congestion is minimal or absent, incoming packets will immediately enter the routing pipeline and have no extra delay. Thus, for the Chaos router, $d = 4$ cycles.

Therefore, the input-to-input latency for an oblivious router with buffering is three or more cycles, while it is four cycles for a Chaos router. Since this delay is added only to the routing of the header flit, the minimum latency will be only one cycle per hop greater with chaotic routing than with oblivious routing, a factor which becomes negligible as the length of packets becomes large compared to the diameter of the network. Figure 2 shows the ratio of the minimum average latency with a Chaos router to the minimum average latency with an oblivious router for torus and hypercubic networks with different packet sizes.

Both of these graphs are constructed by substituting the average distance a packet must travel in the network, D_{avg}, into Equation 1 and varying the packet length. Since the hypercube network has a much lower diameter than the torus network for same number of nodes, the penalty for chaotic routing is lower. Also, as packet sizes increase, the relative slowdown becomes less apparent because

the additional latency due to chaotic routing is based only on the network size, and is not affected by the packet length.

6.2 Congested Network Performance

As network congestion increases, the probability that packets will be routed without experiencing conflicts for resources diminishes. When the network load is in the range between "light" loading, where packets rarely experience conflicts for resources, and saturation, where the load is equal to the maximum the network can handle, packets will face some delays in routing, but the network throughput will keep up with the applied load.

In this range, a Chaos network is expected to route most packets on minimal-length paths, although some packets may enter the multiqueue and wait for needed resources to free up. Oblivious networks will still, of course, route packets on minimal-length dimension-order paths. Because the Chaos network will allow packets to choose from more than one profitable link at a time, packets should be able to reach their destinations more quickly than with oblivious networks. When the network is operating in this "moderately loaded" range, the latency gains made by the "quicker" oblivious router will be lost and chaotic routing will provide faster delivery of messages.

Once the network loading increases beyond a certain point, the network will reach saturation and will not deliver packets fast enough to keep up with the load. Although it is important to compare latency figures for loads up to saturation, the latency numbers have less meaning at loads above saturation because the load corresponds to a state unachievable in real applications having finite (or non-existent) source queues. Because the adaptive nature of chaotic routing allows packets to have a greater number of choices when routing, a chaotic network should achieve a higher throughput than an oblivious network before saturating. Moreover, as the network nears saturation, Chaos nodes will deroute packets which are not making progress, possibly moving them to less congested paths and improving performance. Overall, the saturation point will be higher with chaotic routing than with oblivious routing.

Due to the complexity of chaotic routing, an analysis of congested performance is not provided. Detailed simulations are presented in Section 7 which confirm these expectations.

6.3 Fault-tolerance

In the presence of faulty links or nodes, routing becomes a much more difficult problem. In oblivious routers, since there is only one possible path between a source a and destination b, any fault on this path will render direct communication between a and b impossible. If the fault is *static*, that is, known to exist ahead of time, communication may be re-established through an intermediate node c such that neither the path from a to c or the path from c to b relies on the faulty component. This solution requires advance knowledge of the faults in the network, construction of an "intermediary" table (which may be non-trivial in the presence of multiple faults) at each injecting node, and that packets avoiding faults are received by the intermediate node and re-injected into the network to travel to their destinations. Although it would be easy to add a "forward" bit in the header of a packet along with a second destination field so that forwarding could be accomplished entirely within a router, this would violate the dimension-order routing properties necessary to prevent deadlock. Thus, oblivious routing may tolerate static faults, but at a high cost.

When transient faults occur or static faults first appear, oblivious routing has no possibility of routing around failed paths and the only solution is to stop the network, map the faults, and

re-start.

In chaotic routing, avoiding faults is more natural. If a link is not functioning, it will appear to the router as a continually busy link and the router will deroute around it just as if congestion were the problem. Since all routes are available, fault-tolerance is established as long as nodes are not completely isolated by faults. Mechanisms to extend this "natural" fault-tolerance to be more efficient and robust at little cost have been outlined [Bolding & Snyder 91].

Thus, chaotic routing provides fault-tolerance for both static and some transient faults without requiring additional overhead. On the other hand, oblivious routing can provide fault-tolerance only for already known static faults at a high overhead cost.

6.4 Out-of-order Packet Delivery

Messages which are broken into packets for network transmission must be reconstructed before being delivered to the receiving node. If the packets arrive in the same order as they are sent, this is trivial, provided there are no lost packets. On the other hand if packets may be lost or arrive in a different order than they were injected, mechanisms must be provided to re-order the packets into the original message.

Oblivious routing always delivers packets in order because there is a single path between a source and destination and packets do not pass each other on the same route[8]. For adaptive routers, this is not true. Since packets may take different paths and face differing levels of congestion, they may arrive out-of-order. Although the router itself does not deal with either packetization or reordering, the cost of using the network is increased because of the extra overhead needed at the ends of the path.

Adaptive routing, unfortunately, does not necessarily deliver messages in the order in which they were sent. Because of this, extra complexity is added to network interface in order to deliver the messages in the correct order. Re-ordering can be done in software, but this adds to the already large time spent in the network interface. A hardware solution to this problem has been proposed by McKenzie in which packets include information on their length and their destination in memory [McKenzie 94]. Packets are placed in their proper place in memory and the processor is signalled as soon as the entire message has arrived.

7 Chaos vs. Oblivious Experiments

While arguments based on design complexity may suggest that chaotic routing performs better than oblivious routing, the actual performance of a router cannot be gauged without employing the router in real traffic. Barring building a multicomputer with the actual router, simulation is the best that can be done. We present the results of simulating chaotic and oblivious routing on several different topologies of 256-node networks under various synthetic traffic workloads.

The chaotic router used is the design described in Section 8. Latency through each node is 4 cycles minimum, and the multiqueue holds 5 packets for the mesh and torus routers and 10 packets for the hypercube router.

The oblivious router is a virtual cut-through packet router with input and output FIFO's. Essentially, it is the same as the chaotic router shown in Figure 1 without the multiqueue. Path

[8] If non-blocking buffering is added to an oblivious router, packets may be able to pass each other and delivery could be out-of-order.

selection is, of course, oblivious, and the delay through the node is 3 cycles instead of the 4 cycles for the Chaos router (see Section 6 for justification).

In the simulations, each packet consists of 20 flits, the first one being the header of the message. The channels between nodes are shared bi-directionally. The details of the simulation methodologies can be found in [Fulgham & Snyder 93].

The traffic patterns considered have been used previously in the literature and are generally thought to be difficult, useful or both. It is assumed that traffic patterns are not known in advance and hence more efficient routing techniques that require precomputing switch settings or traffic specific algorithms cannot be used. Following is a description of the traffic patterns simulated. Let the binary representation of the source node be $a_{n-1}a_{n-2}\ldots a_0$. Also, let $\overline{0} = 1$ and $\overline{1} = 0$.

- *Random*, all destinations including the source are equally likely.
- *4X Hot Spots*, ten randomly selected nodes are distinguished. Destinations are chosen randomly such that the distinguished nodes are four times more likely to be chosen than the undistinguished nodes.
- *Complement*, is a permutation where each source node sends packets to $\overline{a_{n-1}a_{n-2}\ldots a_0}$.
- *Transpose*, is a permutation where each node sends packets to $a_{\frac{n}{2}-1}a_{\frac{n}{2}-2}\ldots a_0 a_{n-1}a_{n-2}\ldots a_{\frac{n}{2}}$.
- *Bit Reversal*, is a permutation where each node sends packets to $a_0 a_1 \ldots a_{n-1}$.
- *Shuffle*, is a permutation where each node sends packets to $a_{n-1}a_{\frac{n}{2}-1}a_{n-2}a_{\frac{n}{2}-2}\ldots a_{\frac{n}{2}}a_0$.
- *Random Leveled*, each node with $i < n/2$ bits set to one sends a packet to a randomly selected node $b_{n-1}b_{n-2}\ldots b_0$ with i one bits satisfying $b_{n-1}b_{n-2}\ldots b_0 \& a_{n-1}a_{n-2}\ldots a_0 = 0$ where & is the bitwise AND operator. Nodes with $i \geq n/2$ one bits simply choose a random destination with i one bits.

All nodes generate packets at the rate specified by the presented load which is measured in messages per cycle. This load is normalized to the maximum load that can be delivered, under any circumstances, for all the traffic patterns except the complement and random leveled traffic. The maximum load is one message every l, $kl/4$, and $kl/2$ cycles for the hypercube, torus, and mesh k-ary n-cube networks respectively, using shared bidirectional channels and messages with an average length of l. This constraint is due to the finite bisection bandwidth of the networks [Thompson 79] and the fact that on average half the messages cross the bisection for all but the complement and the random leveled traffic pattern. However, in order to compare the performance between traffic patterns, the complement and random leveled traffic are normalized to the max load of the other traffic patterns.

When routing a permutation (i.e., all patterns except random and hot spot), a particular source node always generates packets with the same destination. Random and hot spot traffic patterns generate an independent destination for each packet that is created.

The traffic patterns illustrate different features. As mentioned earlier the random traffic is simply a standard benchmark used in network routing studies. The 4X hot spot traffic models cases where references to program data, such as synchronization locks, bias packet destinations towards a few nodes. The complement is a particularly difficult permutation since it requires all packets to cross the network diameter in the hypercube and the network bisection in both topologies. Given

Saturation						
	Hypercube		Torus		Mesh	
Traffic	obliv	Chaos	obliv	Chaos	obliv	Chaos
Random	0.60	0.70	0.65	0.95	0.80	0.80
Transpose	0.10	0.70	0.55	0.55	0.55	0.70
Bit reversal	0.15	0.70	0.40	0.85	0.55	0.80
Shuffle	0.35	0.75	0.55	0.70	0.70	0.70
Random level	0.20	0.70	0.50	0.55	0.65	0.55
Complement	0.50	0.55	0.45	0.35	0.50	0.35

Table 1: Minimum load at which saturation is detected.

x and y axes through the center of a torus network, the complement destination is the composition of the x and y axes reflection of the source. Transpose and bit reversal are important because they occur in practical computations and can cause worst case behavior in hypercubic oblivious routers [Leighton 92]. The shuffle converts row major indexing to shuffled row major indexing on the mesh or torus topologies. This indexing scheme can be used for efficient sorting [Thompson & Kung 77]. Random leveled traffic can cause worst case behavior for oblivious hypercubes [Valiant & Brebner 81].

7.1 Saturation

The first set of results simply identifies the saturation points for the different traffic patterns. The saturation point reported is the first normalized applied load, using intervals of .05, that saturates the network. See Table 1. Saturation does not necessarily occur at the knee of the throughput, latency, or delay curves since the effects of saturation can alter system behavior before, during, or after saturation depending on the particular conditions.

After saturation some system statistics such as network delay are no longer valid since they do not have a limiting distribution. The load where the system saturates is an important measure since after saturation it is not possible to predict the delivery time of messages.

The traffic patterns exhibit considerable diversity in the throughput they are able to sustain. In all cases on the hypercube, except several of the hot spot traffic patterns which are discussed in more detail in Section 7.3, the Chaos router saturates at a higher load than the oblivious router. The Chaos torus also saturates at an equal or higher load than the oblivious torus for all the traffic patterns except for the complement permutation. In both cases the difference is modest. On the other hand, with the mesh network, the oblivious router achives equal or higher saturation levels for all permutation-like traffic but the transpose and bit reversal. This phenomonon is due to the a fortuitous combination of the shape of the mesh network and the routes messages take in dimension-order oblivious routing.

Pertel has compared oblivious routing with minimal adaptive routing on mesh networks and found the oblivious router to perform better under random traffic [Pertel 92]. The main underlying reason for this is that, in a mesh-connected network with random traffic, there are many more possible paths which cross the center than the edges of the network. With minimal adaptive routing, severe congestion will form in the center of the mesh, resulting in degraded performance. When dimension-order routing is used, packets follow 'L'-shaped paths and are not as likely to use

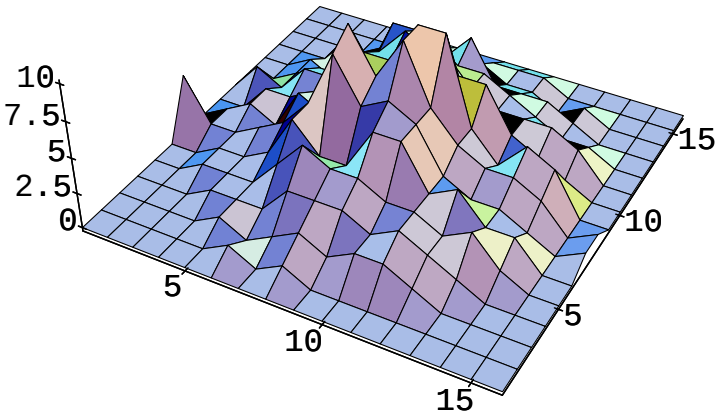

Figure 3: Average injection delay for a 256-node mesh using chaotic routing.

the center paths as intensively as in minimal adaptive routing (though the center paths will still be more congested than the edge paths). Thus, the hot spot in the center is not as "hot" when using oblivious routing as when using minimal adaptive routing, so performance is potentially better. Similar results are observed for chaotic adaptive routing. The center becomes a very "hot" hot spot, resulting in severe congestion and excessive derouting. This congestion can be seen in Figure 3 where the average time to inject a message from each node in the network is plotted. Because the mesh hot spot creates so many difficulties, and since the addition of only a few extra links to create a torus doubles the bisection bandwidth and halves the network diameter, we will concentrate on torus and hypercubes for the remaining discussion.

On torus and hypercube networks, diffences in saturation levels between the Chaos and oblivious routers can be great, as illustrated by the hypercube transpose permutation, or indistinguishable, as in the torus transpose. For the torus transpose, no benefit is gained from relaxing the requirement of dimension order routing. The transpose on the torus is a reflection of the source about the line $y = -x$ given a coordinate system through the center of the network. This pattern causes a continuous hot spot along the diagonal of the network for both the Chaos and oblivious torus routers.

The behavior of the complement permutation is especially interesting. On the oblivious hypercube the complement achieves an unusually high throughput when compared to the other non-uniform traffic patterns. To understand why this happens note that for the oblivious router, dimensions are traversed from lowest to highest. This implies that input frame i has packets destined for output frame $i + 1$. At loads close to saturation most of the input frames should be occupied since the complement packets traverse all dimensions and hence use all channels equally. Therefore when the oblivious router is selecting a packet to use output channel i, it almost always finds a packet; specifically, the packet in input frame $i - 1$. More importantly, no other packet in the node has a conflict with the selected packet because only packets in input frame i need output frame $i + 1$. The complement on the Chaos torus is more complex. Each packet is destined for a node that is the composition of the x axis and y axis reflection of the source in a coordinate system passing through the center of the network. This causes a hot spot in the center of the network and about the wrap around links at the ends of each of the axes, (if viewed in 3-D, this would be like having two centrally located hot spots at opposite sides of the network), preventing the Chaos torus from reaching the maximum possible throughput. However, for the complement, the

oblivious torus excels over the Chaos because it has fewer conflicts. Each packet suffers conflicts only with packets being injected in the current direction of travel and at the one node where the packet turns from correcting the x dimension to the y dimension.

When comparing the complement with the other non-uniform traffic patterns, for both the hypercube and torus, the major limiting factor of the complement permutation is the bisection bandwidth. In the complement pattern, all packets must cross the network bisection, while this is not true of the other traffic patterns. In addition, on the hypercube the all packets in the complement pattern must also travel the full diameter of the network.

7.2 Throughput and Latency

In this section the behavior of the two routers is compared by examining expected throughput and expected latency for the seven traffic patterns. The graphs in Figures 4 and 5 display the presented load versus either throughput or latency for all non-hotspot traffic patterns. Detailed comparisons are found in Appendix A and hotspot results are found in Appendix B. Ninety-five percent confidence intervals for these statistics are also shown. Confidence intervals are not visible for measurements with very small error. The first method we use to compare the two routers is to examine their throughputs.

Throughput for the Chaos router is greater than or equal to the oblivious for both topologies for all traffic patterns at all loads except for a few of the hot spot traffic patterns on the hypercube. However the difference is small, less than a few percent of the normalized throughput. See Figures 6 and 10 in Appendix A. A more detailed discussion of hot spot traffic follows in Section 7.3.

The throughputs for the transpose on the Chaos hypercube and the bit reversal permutation on the Chaos torus are especially noteworthy. The transpose on the Chaos hypercube saturates with a normalized throughput of 68%, whereas the oblivious saturates at a throughput of 9%. The throughput of the oblivious router at the load where the Chaos saturates is only 16%. With the bit reversal permutation, the throughput of the torus Chaos router at saturation is more than double the throughput of the oblivious router from about 39% to 82% respectively.

After saturation, the throughput of the Chaos hypercube degrades for the bit reversal, transpose, shuffle, and random leveled traffic patterns. However, the throughput still remains higher than the oblivious router. Additional data not shown here, shows that optimal queue size depends upon the traffic pattern and the particular goals of the router. In general a multiqueue of size $d + 1$ for a degree d router performs well for all the traffic patterns simulated. Larger multiqueues are able to sustain greater throughputs, but increase latency and latency variance after saturation. Latency is unaffected before saturation due to the cut-through feature of the multiqueue.

Throughput on the oblivious hypercube router does not degrade significantly above saturation. This is the primary strength of the oblivious hypercube router. In a saturated oblivious system, messages are forced to wait for needed channels. We believe the derouting capability of the Chaos router is no longer beneficial when the network is saturated and very congested. In this case, waiting for the desired channels is more effective than using bandwidth to deroute messages. The oblivious torus router does not share this strength. On the torus the oblivious throughput degrades after saturation with the complement, random, shuffle, and random leveled traffic patterns. This is most likely due to the asymmetry in the oblivious network introduced by the virtual channels used for deadlock protection in the torus [Bolding 92, Adve & Vernon 93].

Derouting also affects latency. In general the torus latencies have three phases. When the

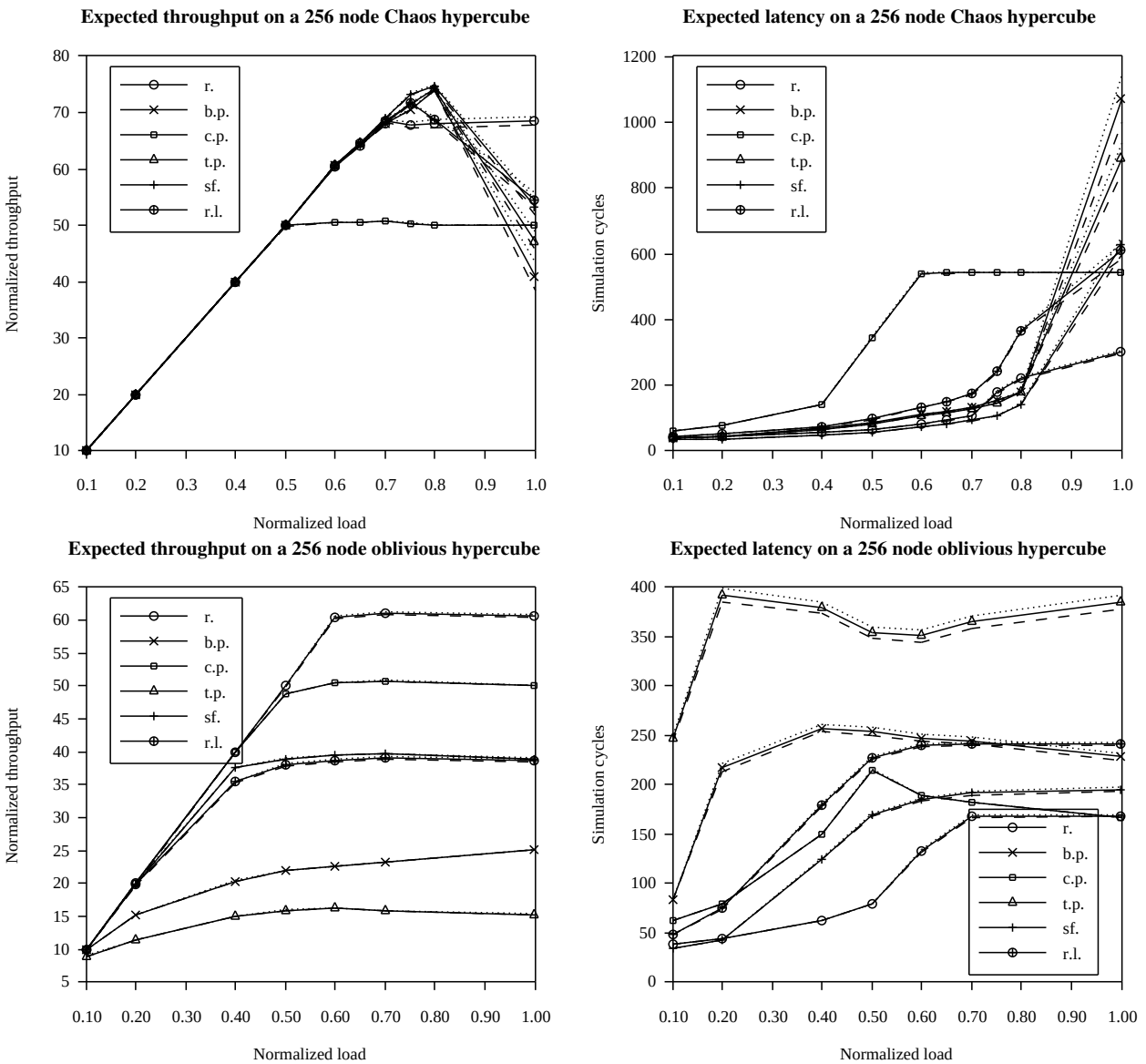

Figure 4: Throughput and latency for the Chaos and oblivious hypercube. Traffic legend is r = random, b.p. = bit-reversal permutation, c.p. = complement permutation, t.p. = transpose permutation, sf. = shuffle permutation, r.l. = random-leveled traffic.

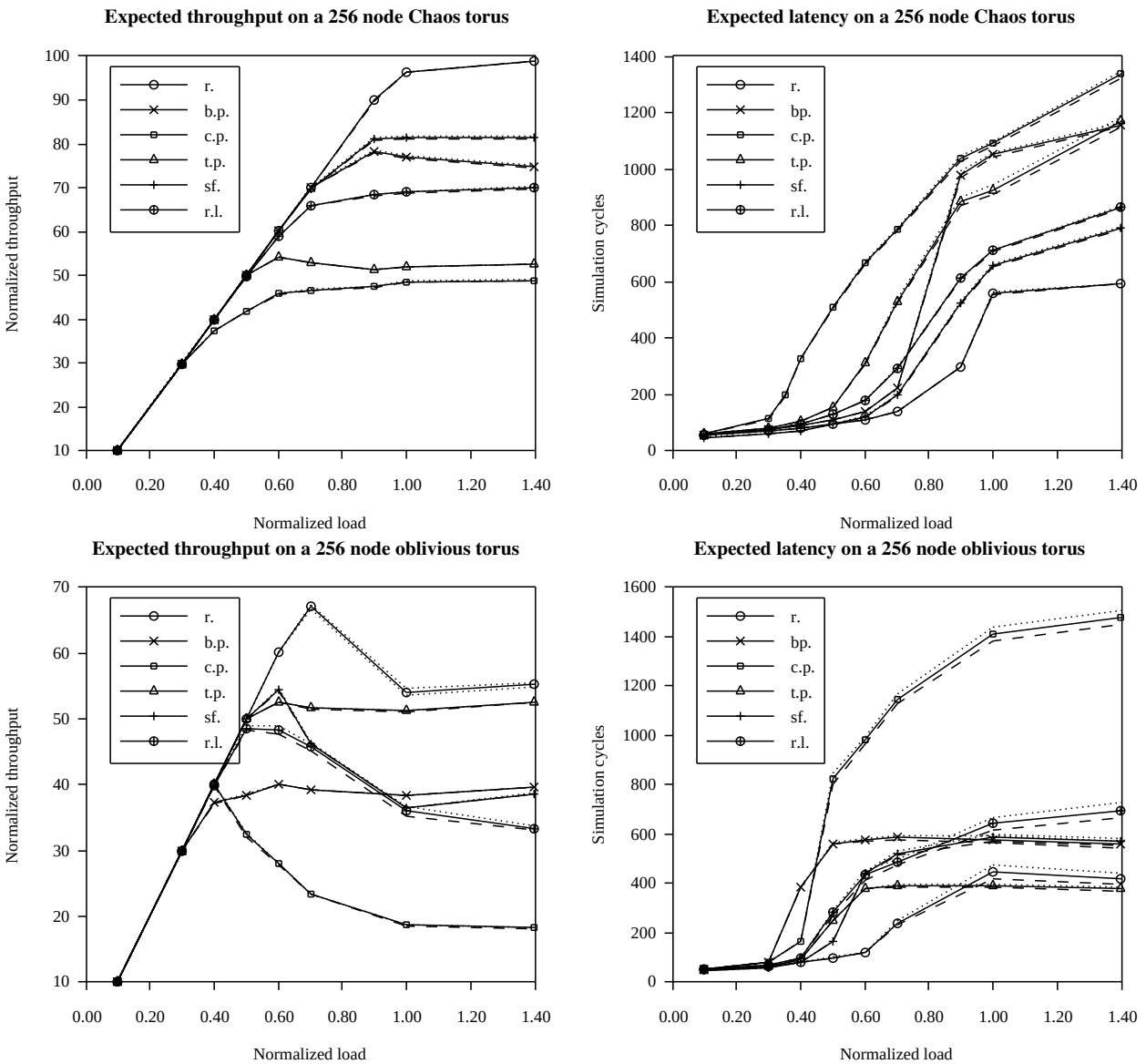

Figure 5: Throughput and latency for the Chaos and oblivious torus. Traffic legend is r = random, b.p. = bit-reversal permutation, c.p. = complement permutation, t.p. = transpose permutation, sf. = shuffle permutation, r.l. = random-leveled traffic.

load is below the neighborhood of the saturating load of the oblivious torus, latency for the Chaos torus router is slightly greater for all the traffic patterns. This is due to the higher latency charged to cross the Chaos node. After the oblivious torus saturates, the Chaos torus experiences lower latency until it saturates. After saturation the Chaos torus router experiences much greater latency than the oblivious router due to both the higher throughput of the Chaos and the increased path lengths of the messages caused by derouting. The one exception is the complement where it is still more beneficial to deroute than to wait.

Latency on the Chaos hypercube has two phases. Before the Chaos hypercube saturates, the Chaos hypercube has a lower or comparable latency than the oblivious hypercube. The greater adaptivity in the hypercube paths allows the Chaos hypercube to hide the larger latency needed to cross the Chaos node. After saturation the Chaos hypercube router experiences much greater latency for the same reasons as the Chaos torus. See Figures 7 and 11 in Appendix A.

Before either of the routers saturate, the delays are comparable to the latencies. Therefore for the Chaos torus, delays are slightly higher than the oblivious torus. On the hypercube, the Chaos has equivalent or slightly lower delays than the oblivious router. After either router reaches saturation, delay comparisons are meaningless, they grow towards infinity as the simulation time increases.

7.3 Hot Spots

For hot spot traffic, placement of the hot spots affects performance, particularly on the oblivious routers which cannot route around the hot spots. Simulations were run for six torus and eight hypercube hot spots arrangements. See Appendix B for the location of the hot spots and for the arrangement of the hot spots on the torus. The hot spot patterns were chosen to demonstrate the variety of behavior in hot spot traffic.

We are primarily interested in the case when all the hot spots are on distinct nodes. In this case, the oblivious torus does the best when the hot spots are evenly distributed resulting in a peak of about 60% of the normalized throughput (case 4). However, when the hot spots are clustered, the throughput degrades slightly (cases 2, 5, 6) and when arranged in a linear fashion, the oblivious torus throughput degrades significantly, peaking at only 47% (case 3). The lack of adaptivity, especially for packets that must traverse a row or column of hot spots, is particularly detrimental to oblivious throughput. The Chaos torus does much better than the oblivious torus reaching between 85% and 90% of the normalized throughput for all the hot spot cases simulated. See Figure 19 in Appendix B.

When two of the hot spots are on the same node, the difference between the two routers is not so dramatic (case 1). The Chaos torus does about 5% better than the oblivious torus and reaches 55% of the normalized throughput. In this case, throughput is affected by the delivery capacity of the double hot spot node.

For the hypercube, the arrangement of the hot spots is more difficult to visualize. We consider two experiments, the first with the standard delivery rate and the second with the delivery rate of each node increased by the hot spot factor. With the standard delivery rate, the hot spot nodes become a bottleneck since they cannot accept packets destined for them fast enough. Using chaotic routing, this causes packets waiting to be delivered to a hot spot to deroute and consume network bandwidth. The standard delivery rate results in slightly worse throughput than on the oblivious hypercube.

With the faster (quadrupled) delivery rate, the Chaos hypercube peaks at a throughput of about 50% before degrading slightly below 40% for all the hot spot cases except case 2, which peaks at about 42%. The oblivious hypercube does almost as well for the randomly placed hot spots reaching throughput between 36% and 38%. However when the hot spots form a tight cluster (case 7) or are connected in two contiguous paths (case 8), the oblivious router cannot even reach 30% of the normalized throughput. See Figures 13 and 14 in Appendix B.

When there is more than one hot spot at a hypercube node, the two routers have lower throughput with similar performance. However the Chaos hypercube does slightly better than the oblivious with the standard delivery and the oblivious hypercube does better with the quadrupled delivery. The Chaos peaks at a higher throughput than the oblivious hypercube if the delivery rate of the double hot spot is increased (by eight times) to match the expected arrival rate of the packets.

Latencies follow the same general phases as for the other traffic patterns and will not be discussed here. See Figures 15, 16, and 20 in Appendix B for more details.

Saturation						
	Hypercube		Torus		Mesh	
Traffic	obliv	Chaos	obliv	Chaos	obliv	Chaos
1	0.20	0.15	0.55	0.55	0.75	0.80
1 4X d.r.	0.25	0.35	0.60	0.95		
1 8X d.r.	0.25	0.40				
2	0.25	0.20	0.50	0.90	0.65	0.80
2 4X d.r.	0.35	0.50				
3	0.25	0.20	0.50	0.90	0.65	0.80
3 4X d.r.	0.35	0.55				
4	0.25	0.25	0.65	0.90	0.80	0.80
4 4X d.r.	0.40	0.55				
5	0.25	0.25	0.55	0.90	0.75	0.80
5 4X d.r.	0.40	0.60				
6	0.25	0.25	0.55	0.90	0.70	0.80
6 4X d.r.	0.35	0.55				
7	0.25	0.25				
7 4X d.r.	0.30	0.55				
8	0.25	0.25				
8 4X d.r.	0.25	0.55				

Table 2: Minimum load at which saturation is detected for various hot spot traffic.

Saturation figures are as expected. The more difficult the hot spot arrangement is for the router the lower the saturation point becomes. The Chaos router saturates at a higher load for all the hot spots cases except three (case 1, 2, and 3) with the standard delivery rates. This inferior performance is a result of delivery bottlenecks at the hot spot nodes, as explained above. When the delivery rate of each node is quadrupled (4X d.r.), chaotic routing saturates at a higher load than oblivious routing in all the hot spot cases.

8 Chaos Router Design

While oblivious routers have been in use for many years, few adaptive routers have been built to date. Of those that have, little technical information is available on most of them due to proprietary design considerations. One exception to this is the Post Office router, of which a preliminary version has been built and runs at around 20Mhz [Coates et al. 93]. A two-dimensional (mesh or torus) Chaos router has been built with the design described below and is designed to run at 66MHz [Bolding et al. 93]. Ongoing testing has confirmed operation only to 25MHz in this prototype. The design goals have centered around being able to match the cycle time of an oblivious router with minimum additional latency, and this has been, in simulation, achieved through careful design and extensive pipelining.

As outlined in Section 6, to compete with a state-of-the-art oblivious router, the Chaos router must first of all have comparable throughput capability at all loads and match the minimum latency experienced during light loads. The goal of equal throughput resulted in a design in which the channels can be driven at the maximum rate allowed by the technology without lengthening the cycle time due to the router design. If the channels were slowed down, raw throughput would suffer. This is achieved by straightforward pipelining techniques at the expense of adding additional latency: data is read into the router on one cycle, but not processed until the next. Oblivious routers make use of this technique as well.

8.1 Pipeline Design

Because the router must run at a clock speed to match the maximum speed of the pads, the design must be pipelined. The pipeline, which operates only on the header of each packet, has four primary stages:

1. Read the header into the input frame across the network channel.
2. Decode the header to identify profitable output channels for this packet.
3. Select a single output frame to route this packet to.
4. Move the header across the crossbar to the output frame and update the header to reflect the routing.

Thus, the minimum latency through the router is four cycles.

The pipeline design is complicated by new packets arriving in input frames and others leaving output frames constantly during the routing decision. Add to this the fact that there may be several packets desiring multiple output frames at once, and the complexity of the routing decision can be seen [Konstantinidou 91]. In order to reduce the complexity, the constraint that only one new route may be set up per cycle is added. Furthermore, during a cycle, a single output channel is selected to be considered as the destination for newly routed packets and only packets which can profitably use this output channel are considered. With these constraints, the complexity becomes manageable since the complex parallel decision of matching all packets to all destinations at once has been serialized so that the router only has to choose a single route to a fixed destination during a cycle.

The serialization constraints are enforced by the use of the *ActionDim* vector, which indicates the output frame that is currently being considered for the destination for newly routed packets.

ActionDim is cyclically incremented through the output frames, with the routing logic setting up routes from input frames to the output frame indicated by *ActionDim* when possible.

To avoid "wasting" cycles when *ActionDim* points to an output frame which no packets can use, *ActionDim* is only cycled through *interesting* directions. A direction i is interesting whenever output frame i is available and there is a packet in some input frame which can profitably use it, or when a packet in input frame i is *stalled* and should be read into the multiqueue.

Once *ActionDim* is chosen, the router must set up the appropriate route to move a packet to the output frame specified by *ActionDim*. First, the multiqueue is searched for packets which can profitably use this output frame. If none are found, then the input frames are searched. Regardless of whether such a packet is found, any packet in the input frame corresponding to ActionDim is moved directly into the multiqueue. All of these actions complete within a single cycle except selecting a packet in the multiqueue to be moved to an output frame. This process takes three or more cycles depending on traffic conditions, with an average of 6.43 cycles for the worst traffic [Bolding 93].

8.2 Design Consequences

Although the serialization of the routing decision causes the Chaos router to be able to route less than one new packet per cycle, this constraint does not seriously effect the performance. Throughput through the node would be affected if this serialization caused a backlog of packets to queue up. However, because the packets are composed of multiple flits, there is a limit on how frequently a new packet can arrive on a channel. In order to keep up with newly arriving packets, all that is necessary is that each channel is visited at least as often as a new packet can arrive on a channel. With worst-case traffic, each channel of the 2-D router will be visted only once every 25.7 cycles [Bolding 93]. However, since in order for the worst-case load to exist, the bi-directional channels must be utilized equally in both directions, new packets can arrive only once every $2L$, or 40 cycles. Thus, the router should be able to provide sufficient throughput even in the worst-case situation.

9 Conclusions

Although dozens of designs for multicomputer routers have been proposed over the last several years, none has currently displaced the simple-but-fast oblivious routers in any significant manner. The reasons for this are many, but boil down to usefulness and complexity. Any adaptive router is more complex than a straightforward dimension-order oblivious router. To compete, this complexity must be dealt with carefully to produce a router that can deliver messages with the low node-to-node latency of an oblivious router. When network loads are high, the advantages of adaptivity become more apparent. At this point, non-minimal adaptive routers with limited derouting perform better than minimal adaptive routers and deflection routers.

Chaotic routing is a queueing non-minimal adaptive routing technique which lends itself to a relatively simple critical path by using cut-through techniques to enhance low-load performance. When loading increases, packets are stored in a non-blocking buffer where they await channels to become free. When loads become very high and the buffer becomes full, derouting is employed to distribute the load and prevent deadlock. By careful design of the low-load critical path, a per-node routing latency comparable with an oblivious router is achieved in the Chaos router. When the

load increases, the per-node latency increases, but the throughput achieved is much higher than in oblivious routing. Thus, chaotic routing is presented as a contender to replace oblivious routing as the router of the next generation of multicomputers.

References

[Adve & Vernon 93] V. S. Adve and M. K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Transactions on Parallel and Distributed Systems*, 1993.

[Berman et al. 92] P. Berman, L. Gravano, G. Pifarré, and J. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. In *ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[Bolding & Snyder 91] K. Bolding and L. Snyder. Overview of fault handling for the chaos router. In *Proceedings of the 1991 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 124–127. IEEE, November 1991.

[Bolding & Snyder 92] K. Bolding and L. Snyder. Mesh and torus chaotic routing. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pages 333–347, March 1992.

[Bolding 92] K. Bolding. Non-uniformities introduced by virtual channel deadlock prevention. Technical Report UW-CSE-92-07-07, University of Washington, Seattle, WA, July 1992.

[Bolding 93] K. Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router*. PhD dissertation, University of Washington, Seattle, WA, July 1993.

[Bolding et al. 93] K. Bolding, S.-C. Cheung, S.-E. Choi, C. Ebeling, S. Hassoun, T. A. Ngo, and R. Wille. The chaos router chip: Design and implementation of an adaptive router. In *Proceedings of VLSI '93*, pages 8.2.1–8.2.10, September 1993.

[Boppana & Chalasani 92] R. Boppana and S. Chalasani. New wormhole routing algorithms for multicomputers. Technical Report ECE-92-7, Univ. Wisc., Madison, October 1992.

[Borodin & Hopcroft 85] A. Borodin and J. E. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[Chien & Kim 92] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. 19th Intl. Symposium on Computer Architecture*, pages 268–277, May 1992.

[Chinn et al. 93] D. Chinn, F. Leighton, and M. Tompa. Lower bounds for minimal adaptive routing on the mesh with bounded queues. In *to be published*, 1993.

[Coates et al. 93] B. Coates, A. Davis, and K. Stevens. The post office experience: Designing a large asynchronous chip. In *Proceedings of the HICSS*, 1993.

[Cypher & Gravano 92] R. Cypher and L. Gravano. Adaptive, deadlock-free packet routing in torus networks with minimal storage. In *Proc. 1992 Intl. Conf. on Parallel Processing*, pages 204–211, 1992.

[Dally & Aoki 92] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 1992.

[Dally & Seitz 86] W. Dally and C. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1(3), 1986.

[Dally & Seitz 87] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[Dally 87] W. Dally. Wire-efficient VLSI multiprocessor communication networks. In P. Losleben, editor, *Proceedings of the Stanford Conference on Advanced Research in VLSI*, pages 391–415. MIT Press, March 1987.

[Dally 92] W. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[Dias & Jump 81] D. M. Dias and J. R. Jump. Analysis and simulation of buffered delta networks. *IEEE Trans. on Computers*, C-30(4):273–282, April 1981.

[Fang & Szymanski 91] C. Fang and T. Szymanski. An analysis of deflection routing in multidimensional regular mesh networks. In *Proceedings of IEEE INFOCOM '91*, pages 859–868. IEEE, April 1991.

[Felperin et al. 91] S. Felperin, L. Gravano, G. Pifarré, and J. Sanz. Fully–adaptive routing: Packet switching performance and wormhole algorithms. In *Supercomputing*, pages 654–663, 1991.

[Flaig 87] C. Flaig. VLSI mesh routing systems. Master's thesis, California Institute of Technology, May 1987.

[Fulgham & Snyder 93] M. L. Fulgham and L. Snyder. Performance of chaos and oblivious routers under non-uniform traffic. Technical Report CSE-93-06-01, University of Washington, Seattle, WA, June 1993.

[Glass & Ni 91] C. Glass and L. Ni. Adaptive routing in mesh–connected networks. Technical Report MSU-CPS-ACS-45, Mich. State Univ, October 1991.

[Glass & Ni 92] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proceedings of the 19th International Symposium on Computer Architecture*. IEEE, 1992.

[Kaklamanis et al. 90] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proceedings of the 2nd Symposium on Parallel Algorithms and Architectures*, pages 31–35, 1990.

[Kermani & Kleinrock 79] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[Konstantinidou & Snyder 90] S. Konstantinidou and L. Snyder. The chaos router: A practical application of randomization in network routing. In *Proceedings of the 2nd Symposium on Parallel Algorithms and Architectures*, pages 21–30. ACM, 1990.

[Konstantinidou & Snyder 91] S. Konstantinidou and L. Snyder. Chaos router: Architecture and performance. In *Proceedings of the 18th International Symposium on Computer Architecture*, pages 212–221. IEEE, May 1991.

[Konstantinidou 90] S. Konstantinidou. Adaptive, minimal routing in hypercubes. In *Proc. 6th MIT Conf. On Advanced Research in VLSI*, pages 139–153, 1990.

[Konstantinidou 91] S. Konstantinidou. *Deterministic and Chaotic Adaptive Routing in Multicomputers.* PhD dissertation, University of Washington, Seattle, WA, May 1991.

[Leighton 92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.

[Linder & Hardin 91] D. H. Linder and J. C. Hardin. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, C-40(1):2–12, January 1991.

[Maxemchuk 89] N. F. Maxemchuk. Comparison of deflection and store-and-forward techniques in the manhattan street and shuffle-exchange networks. In *Proceedings of IEEE INFOCOM '89*, pages 800–809. IEEE, 1989.

[McKenzie 94] N. McKenzie. A network interface circuit design. Unpublished draft, University of Washington, January 1994.

[Ngai & Seitz 89] J. Y. Ngai and C. L. Seitz. A framework for adaptive routing in multicomputer networks. In *Proceedings of the Symposium of Parallel Algorithms and Architectures*, pages 1–9. ACM, 1989.

[Ngai 89] J. Y. Ngai. *A Framework for Adaptive Routing in Multicomputer Networks.* PhD dissertation, California Institute of Technology, Pasadena, CA, May 1989.

[Nguyen & Snyder 94] T. Nguyen and L. Snyder. Performance of minimal adaptive routers. Submitted for Publication, 1994.

[Pertel 92] M. J. Pertel. A critique of adaptive routing. Technical Report CS-TR-92-06, California Institute of Technology, Pasadena, CA, June 1992.

[Pfister & Norton 85] G. F. Pfister and V. A. Norton. "Hot spot" contention and combining in multistage interconnection networks. *IEEE Trans. on Computers*, C-34(10), October 1985.

[Pifarré et al. 91] G. Pifarré, L. Gravano, S. Felperin, and J. Sanz. Fully-adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks. In *Proc. of the Third ACM Symp. on Parallel Alg. and Arch.*, pages 278–290, 1991.

[Seitz & Su 93] C. L. Seitz and W.-K. Su. A family of routing and communication chips based on the Mosaic. In *Symp. on Integrated Systems: Proc. of the 1993 Washington Conf.*, pages 320–337, 1993.

[Smith 81] B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proceedings of SPIE*, pages 241–248, 1981.

[Smitley 89] D. Smitley. Design tradeoffs for a high speed network node. Technical Report SRC-TR-89-007, Supercomputing Research Center Institute for Defense Analysis, Bowie, Maryland, July 1989.

[Thompson & Kung 77] C. Thompson and H. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, 1977.

[Thompson 79] C. Thompson. Area-time complexity for VLSI. In *Annual Symposium on Theory of Computing*, pages 81–88, May 1979.

[Valiant & Brebner 81] L. G. Valiant and G. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 263–277, 1981.

A Permutation-like Traffic

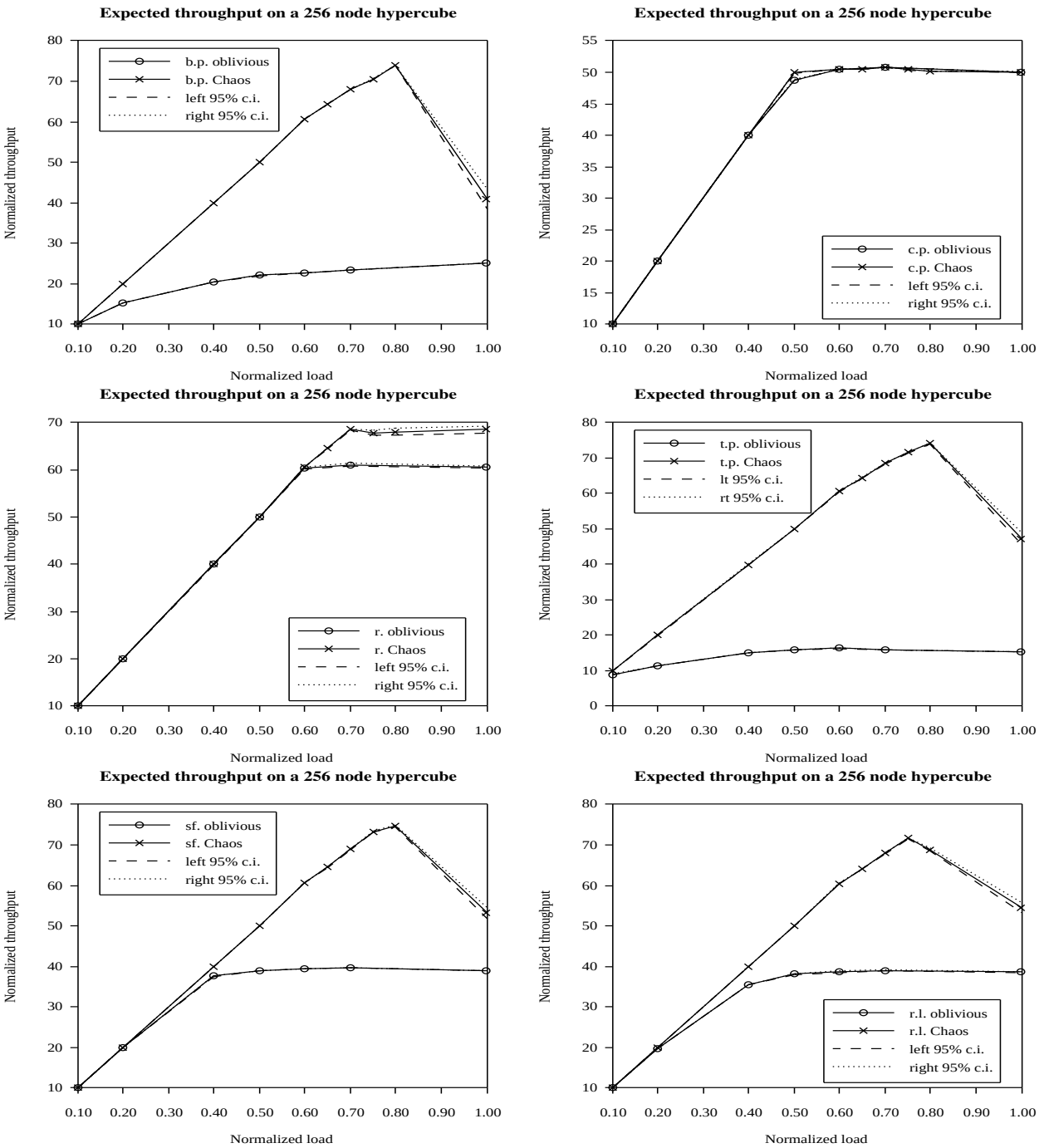

Figure 6: Hypercube throughput for bit reversal, complement, random traffic, transpose, shuffle, and random leveled traffic.

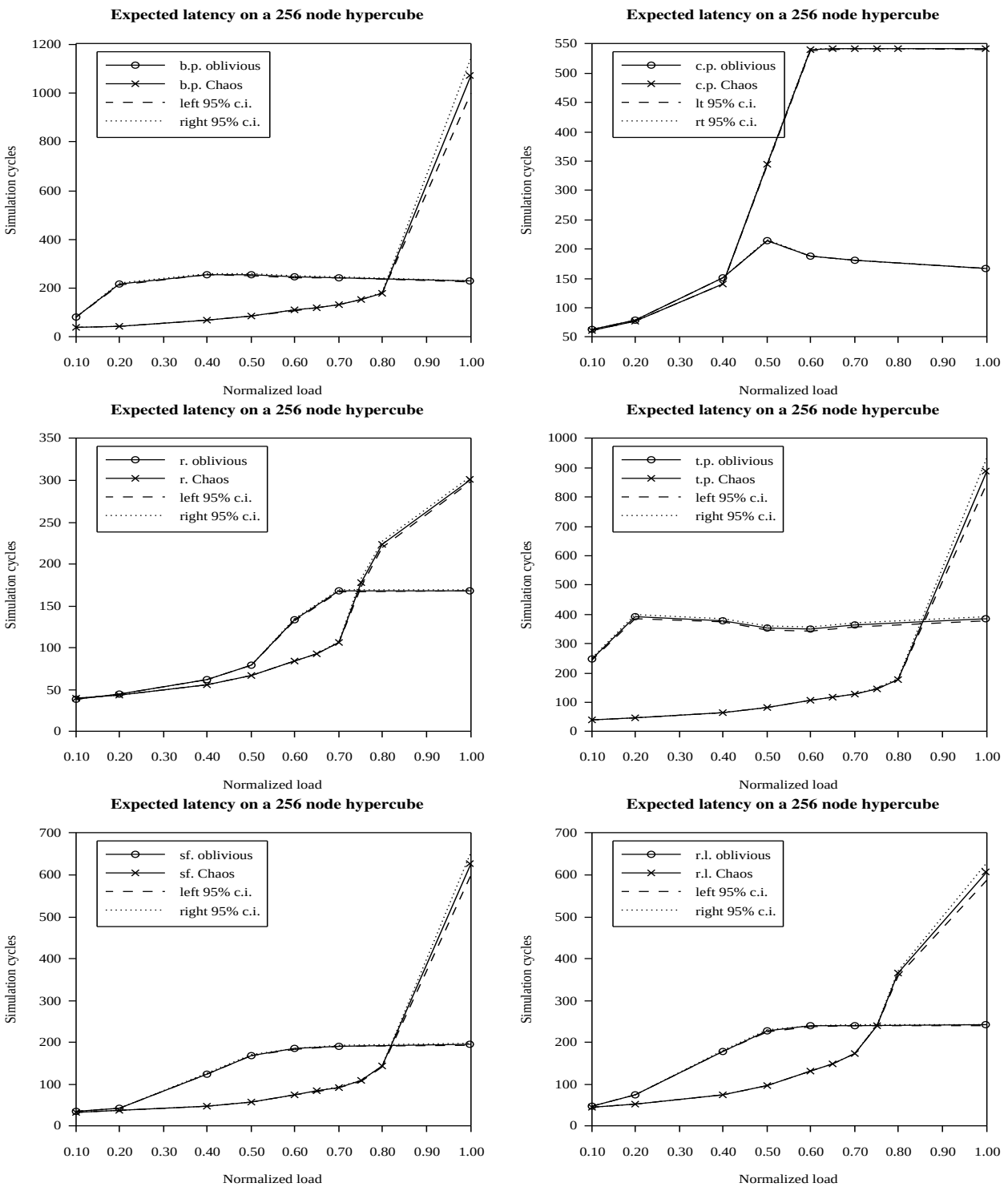

Figure 7: Hypercube latency for bit reversal, complement, random traffic, transpose, shuffle, and random leveled traffic.

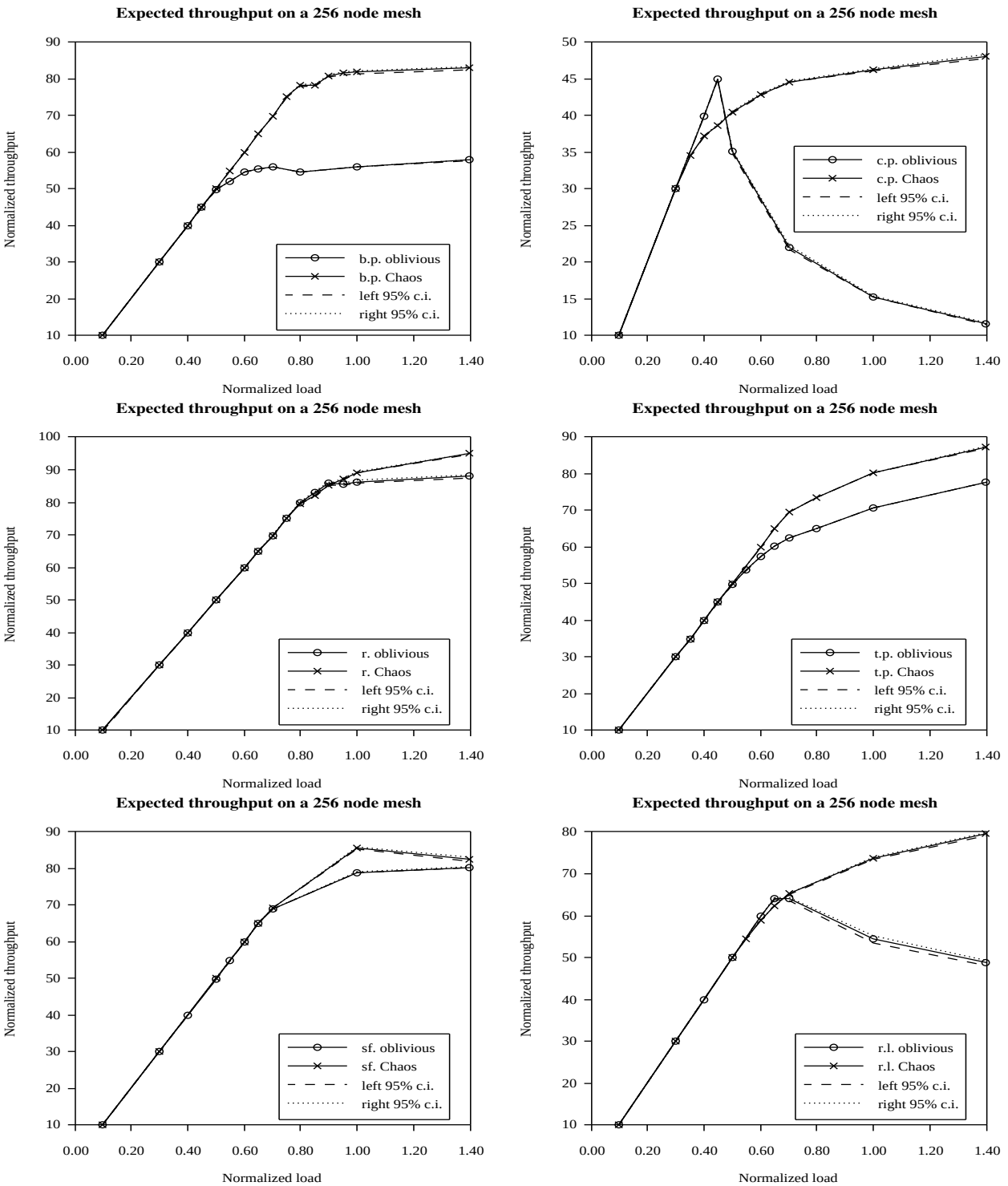

Figure 8: Mesh throughput for bit reversal, complement, random traffic, transpose, shuffle, and random level traffic.

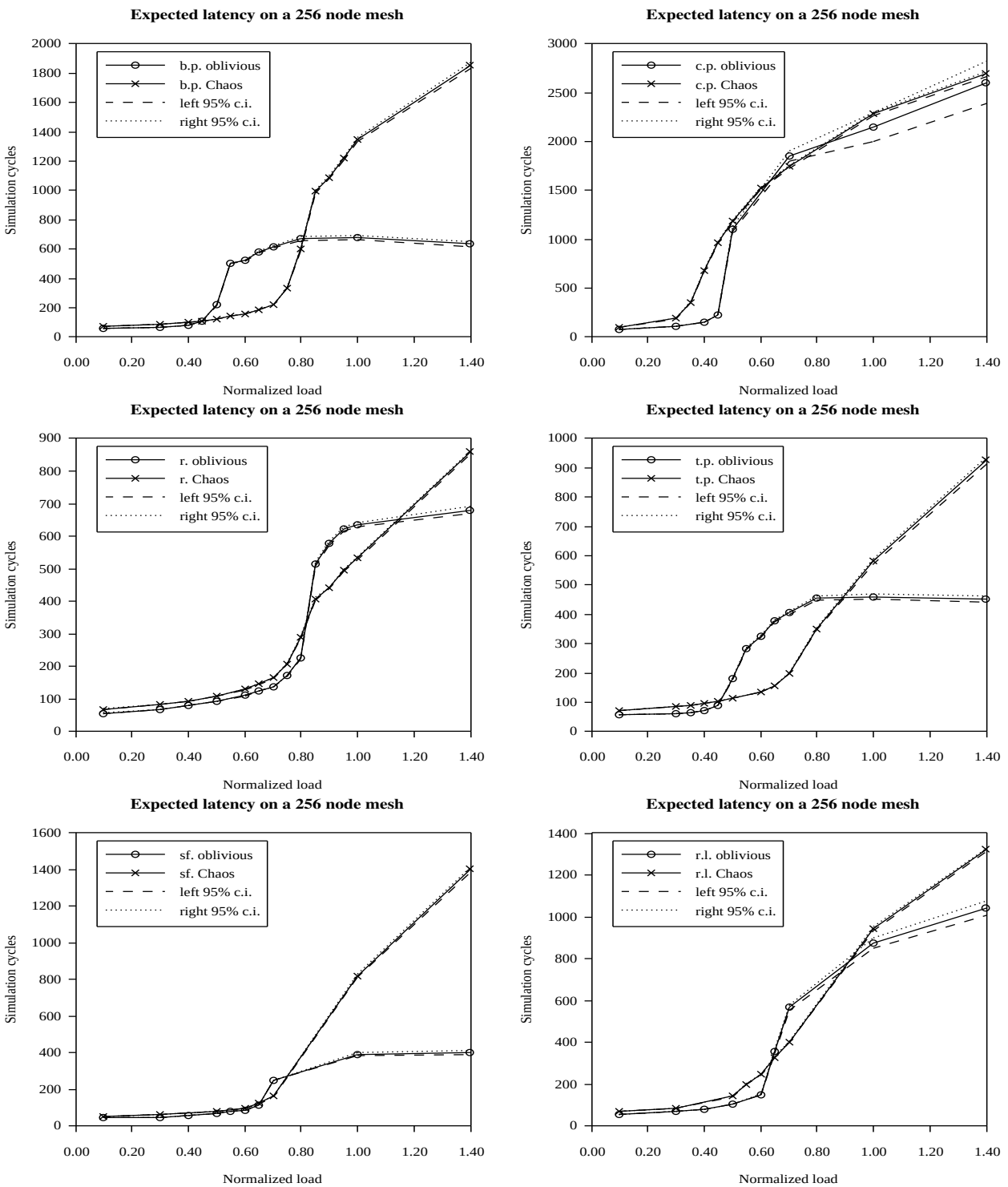

Figure 9: Mesh latency for bit reversal, complement, random traffic, transpose, shuffle, and random level traffic.

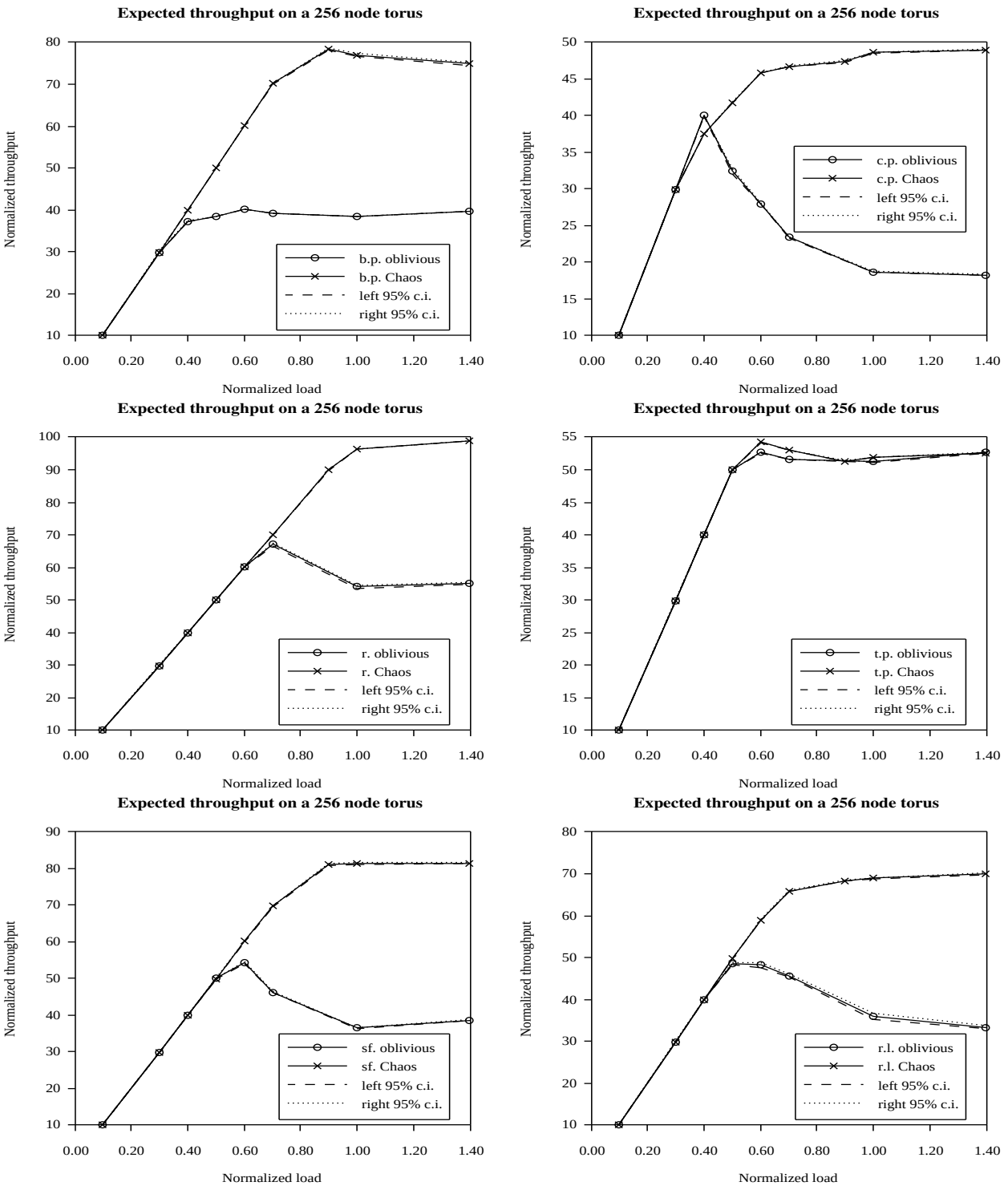

Figure 10: Torus throughput for bit reversal, complement, random traffic, transpose, shuffle, and random leveled traffic.

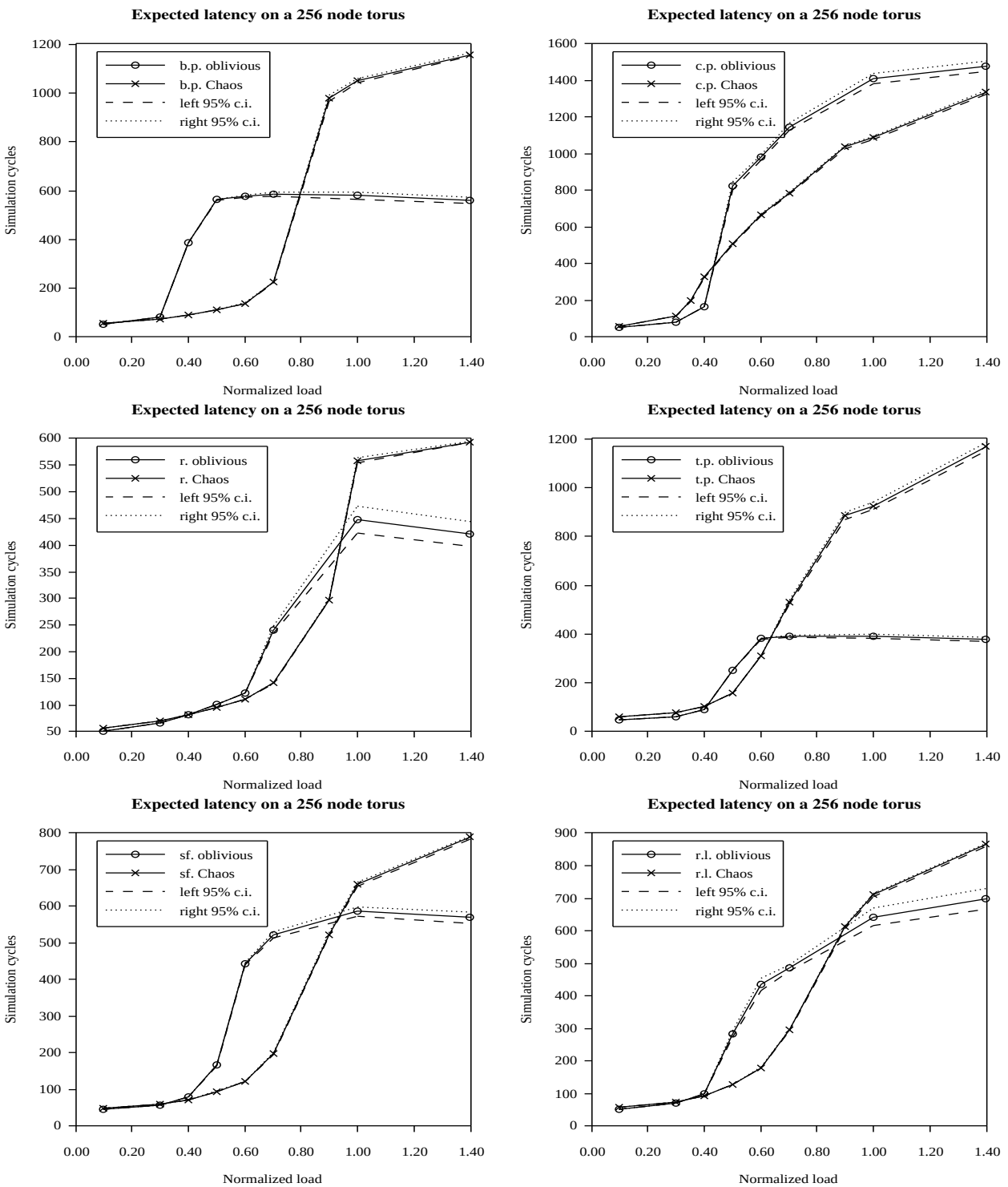

Figure 11: Torus latency for bit reversal, complement, random traffic, transpose, shuffle, and random leveled traffic.

B Hot Spot Traffic

Following are the hot spot nodes for each of the hot spot cases.

1. 146 102 94 51 196 25 107 94 15 224
2. 61 12 8 245 5 27 69 28 98 46
3. 3 239 207 83 6 9 89 125 7 255
4. 77 241 105 197 98 126 223 251 163 52
5. 223 251 163 52 74 220 70 179 55 158
6. 210 225 243 73 149 241 136 227 130 88
7. 0 1 2 4 8 16 32 64 128 3
8. 0 1 3 7 15 129 131 135 143 128

Figure 12 shows the arrangment of the hot spot traffic patterns for the mesh and torus with cases 1-6 arranged left to right, top to bottom.

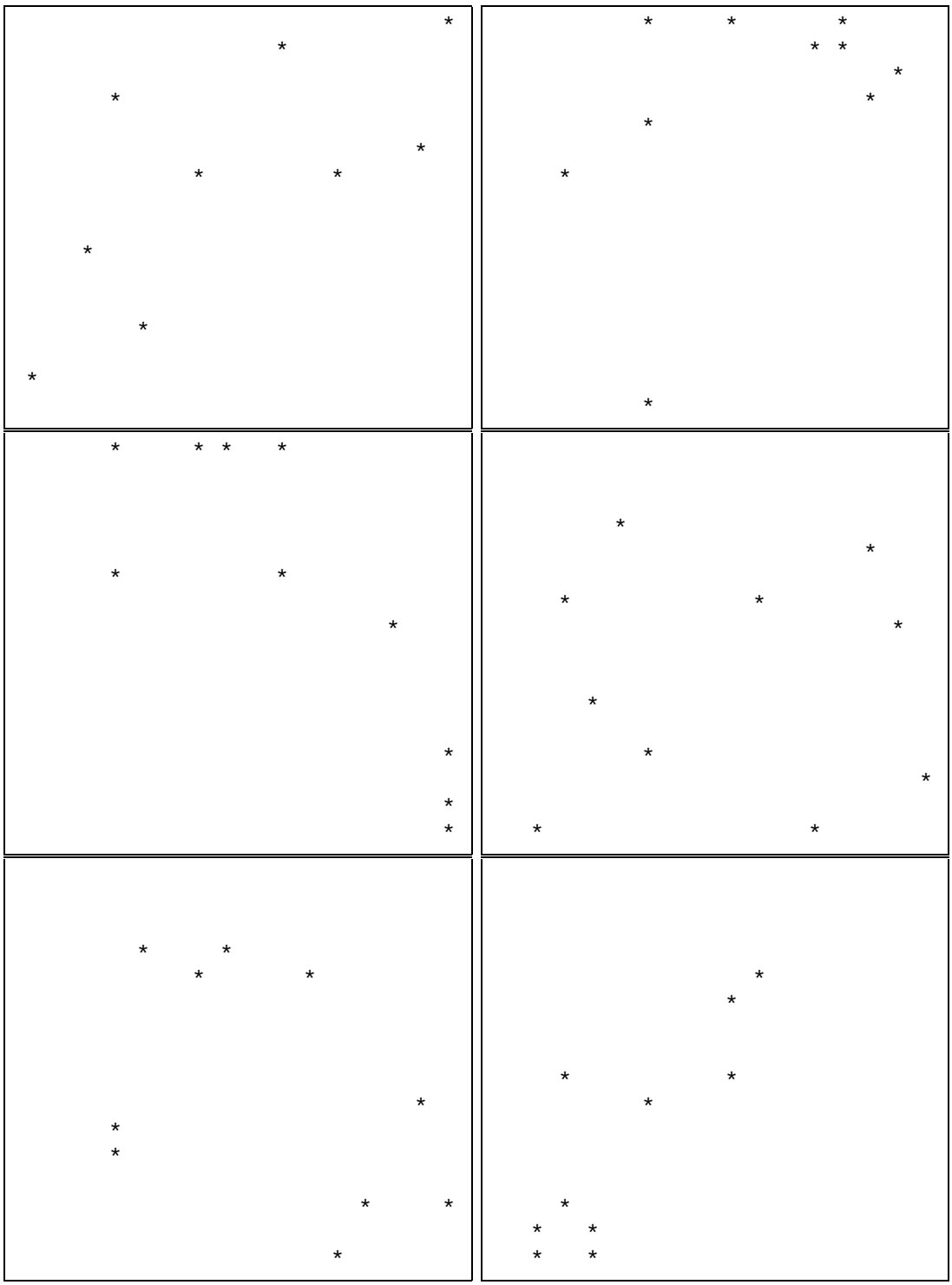

Figure 12: Mesh and torus hot spot locations for cases 1, 2, 3, 4, 5, and 6.

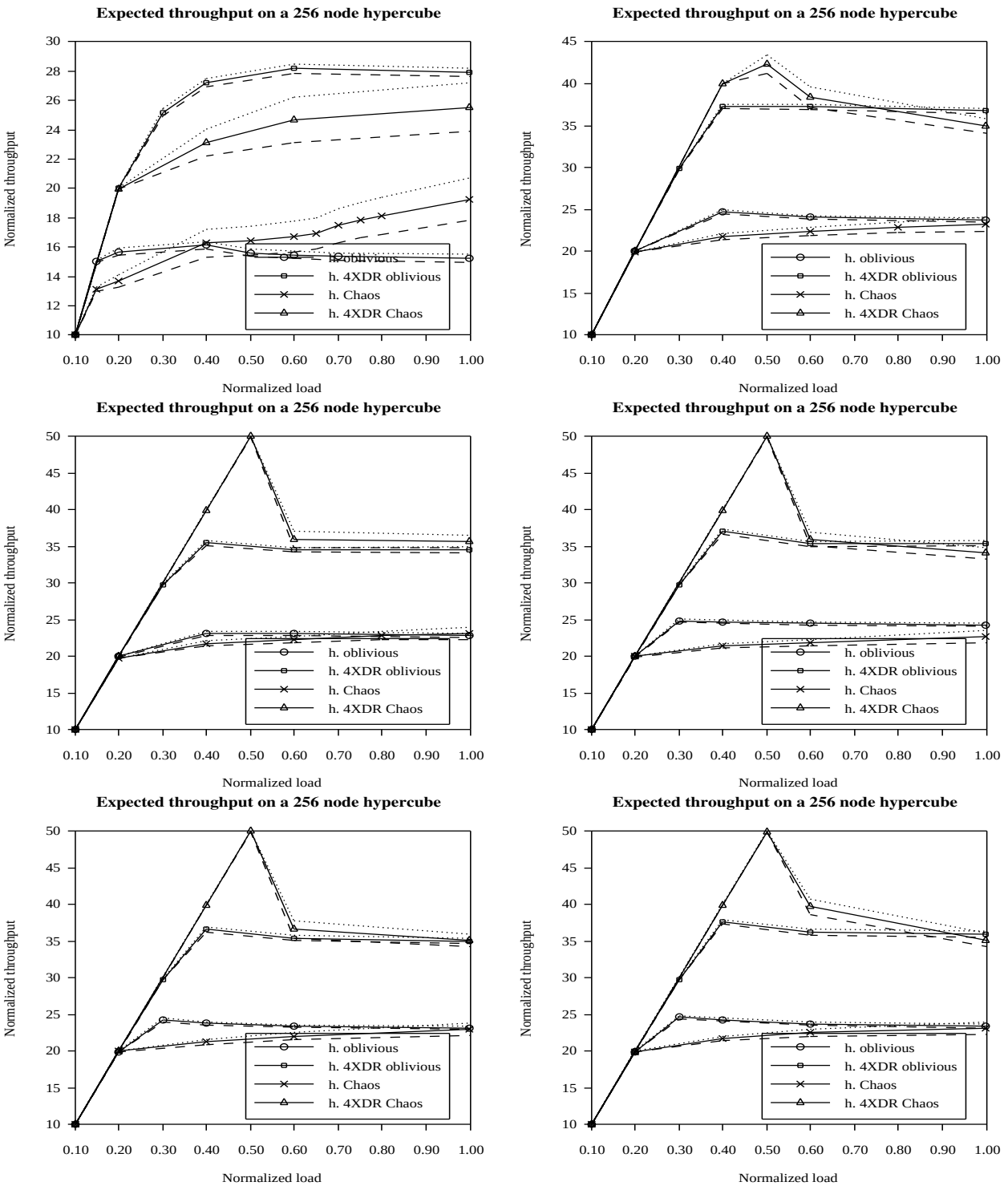

Figure 13: Hypercube hot spots throughput cases 1, 2, 3, 4, 5, and 6.

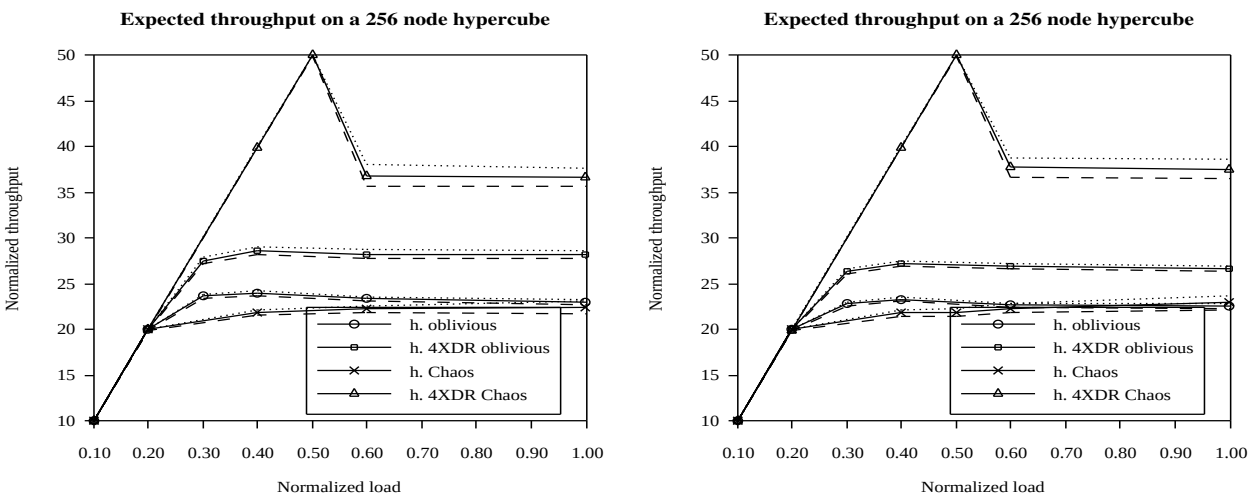

Figure 14: Hypercube hot spots throughput cases 7 and 8.

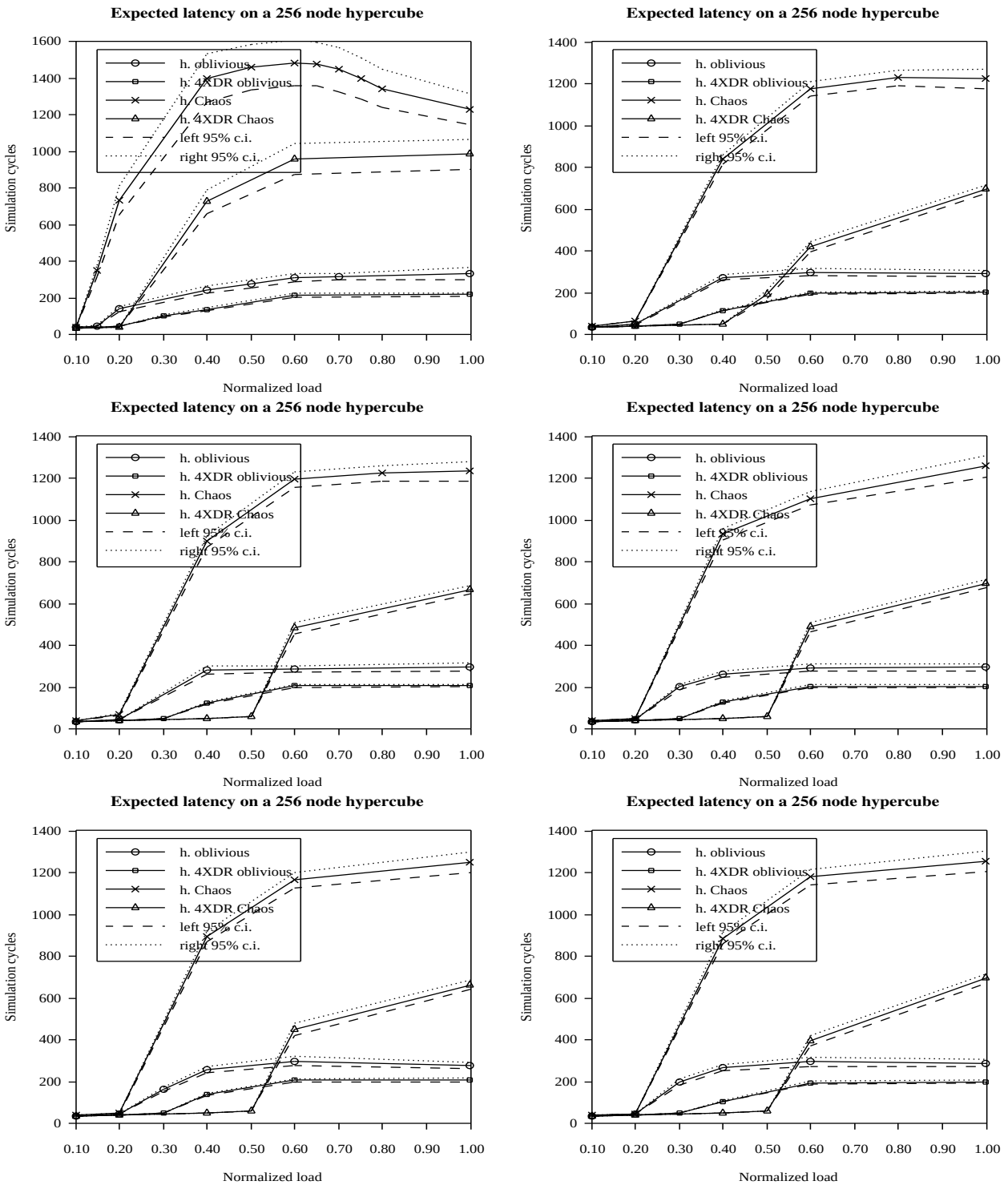

Figure 15: Hypercube hot spots latency cases 1, 2, 3, 4, 5, and 6.

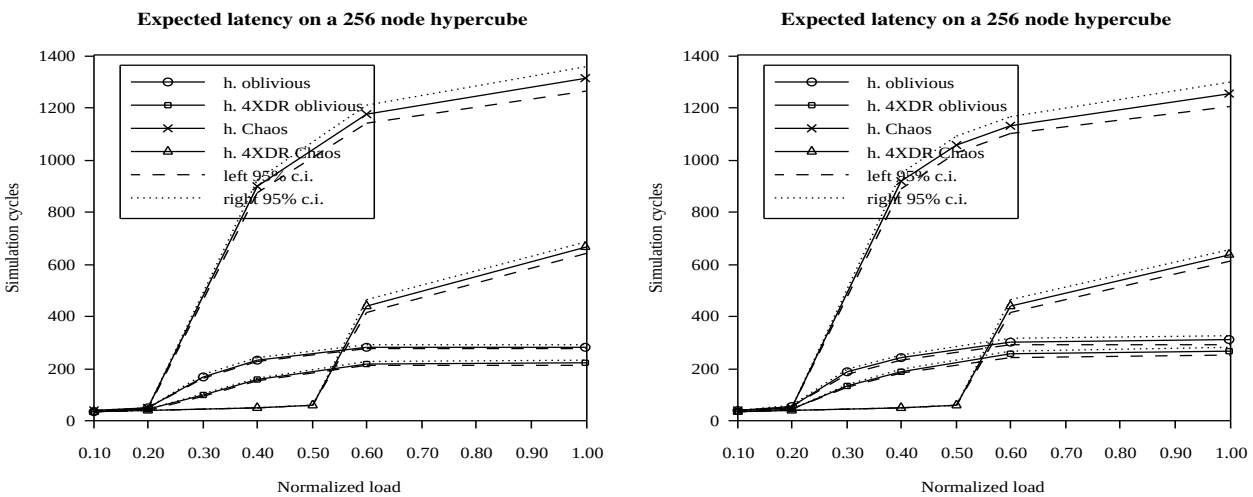

Figure 16: Hot spot latency cases 7 and 8.

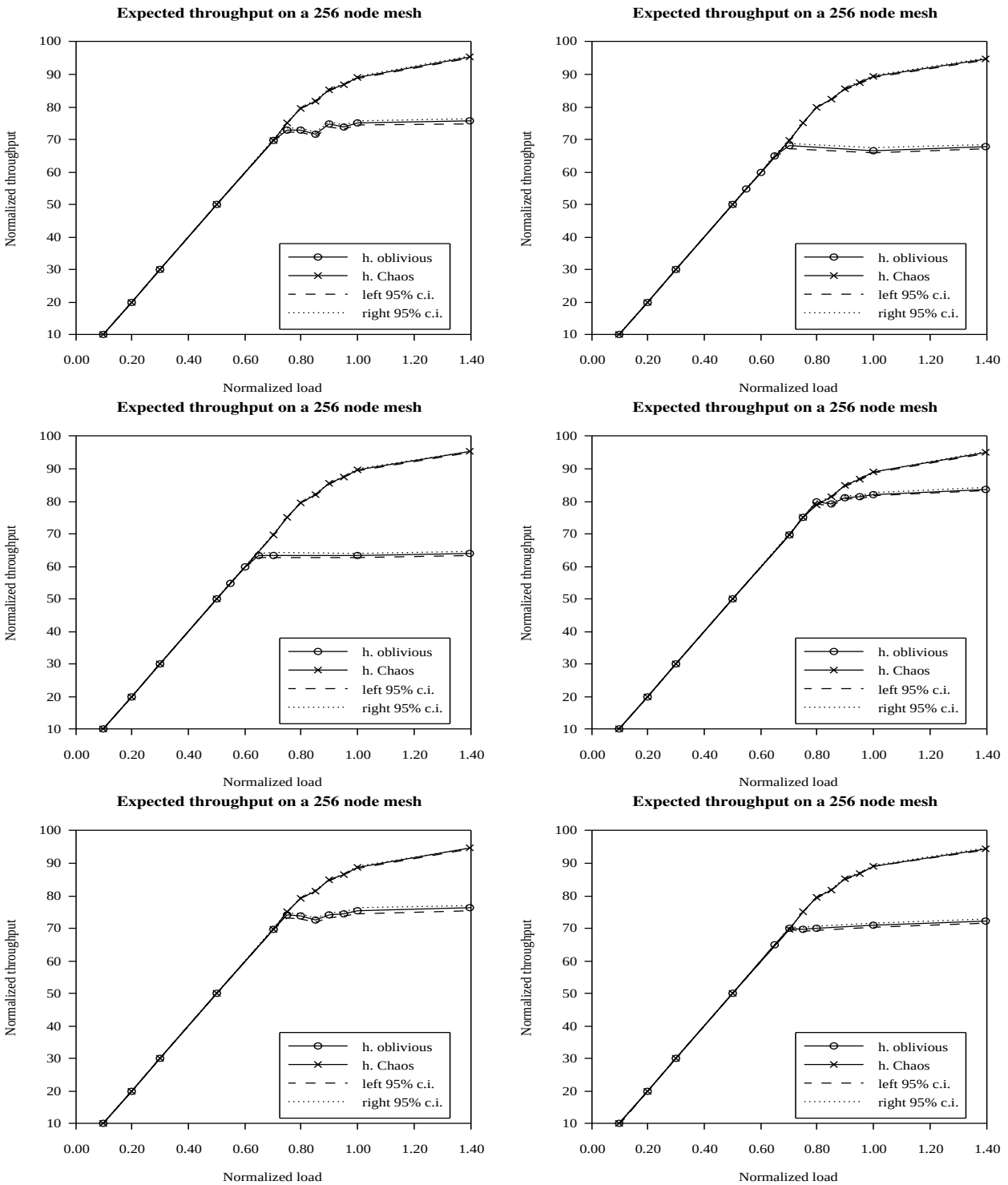

Figure 17: Mesh hot spot throughput for cases 1 - 6.

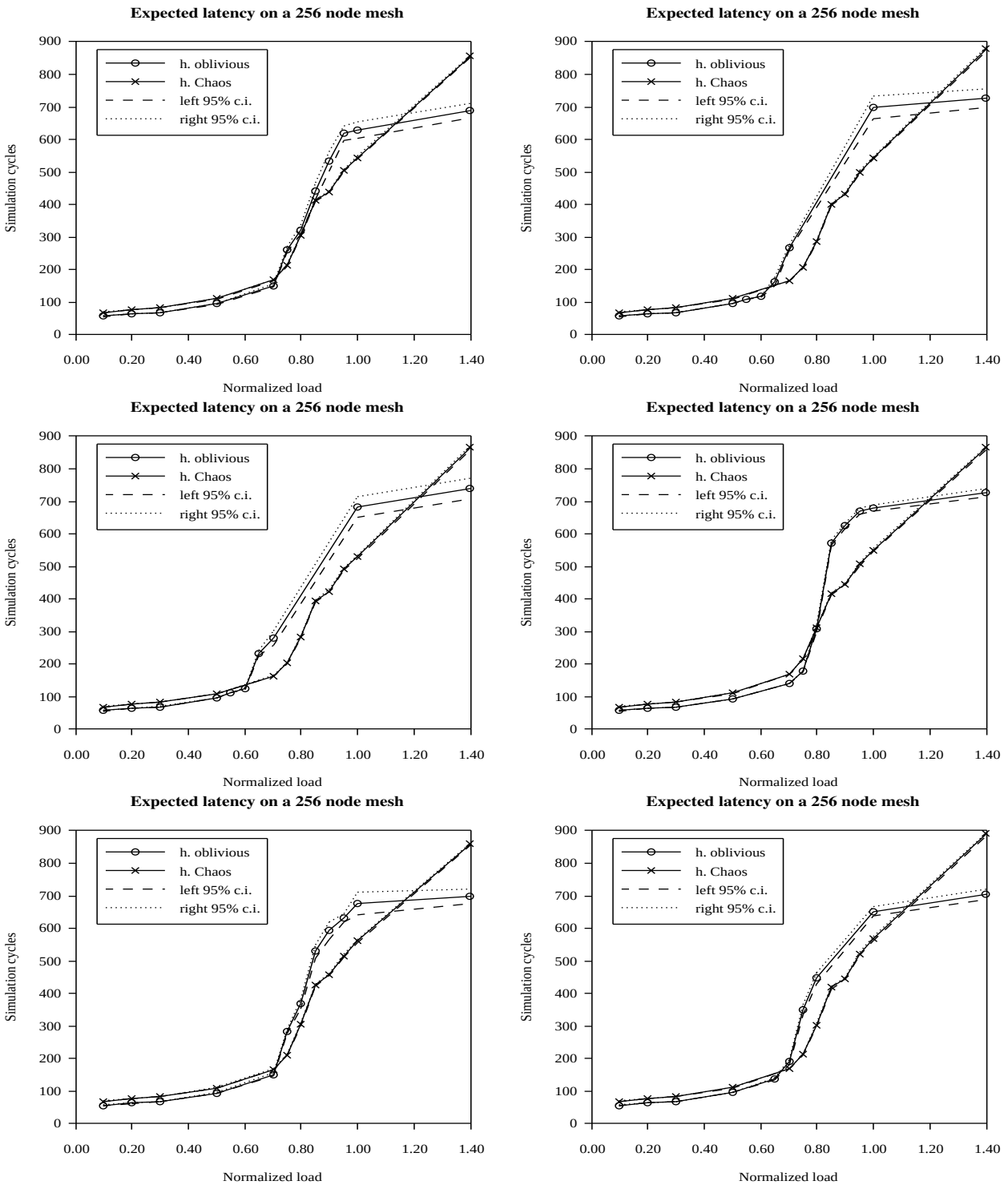

Figure 18: Mesh hot spot latency for cases 1 - 6.

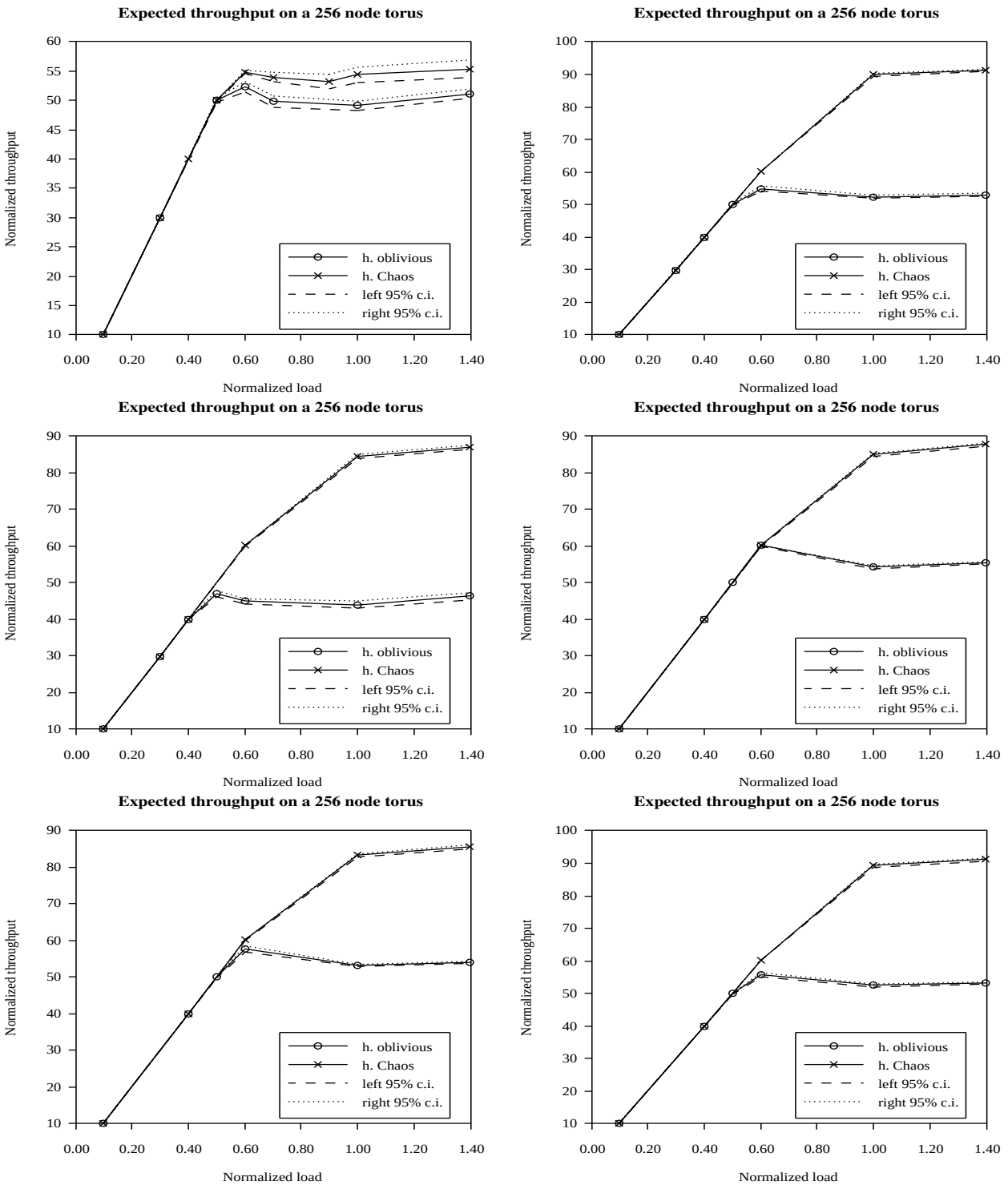

Figure 19: Torus hot spots throughput cases 1, 2, 3, 4, 5, and 6.

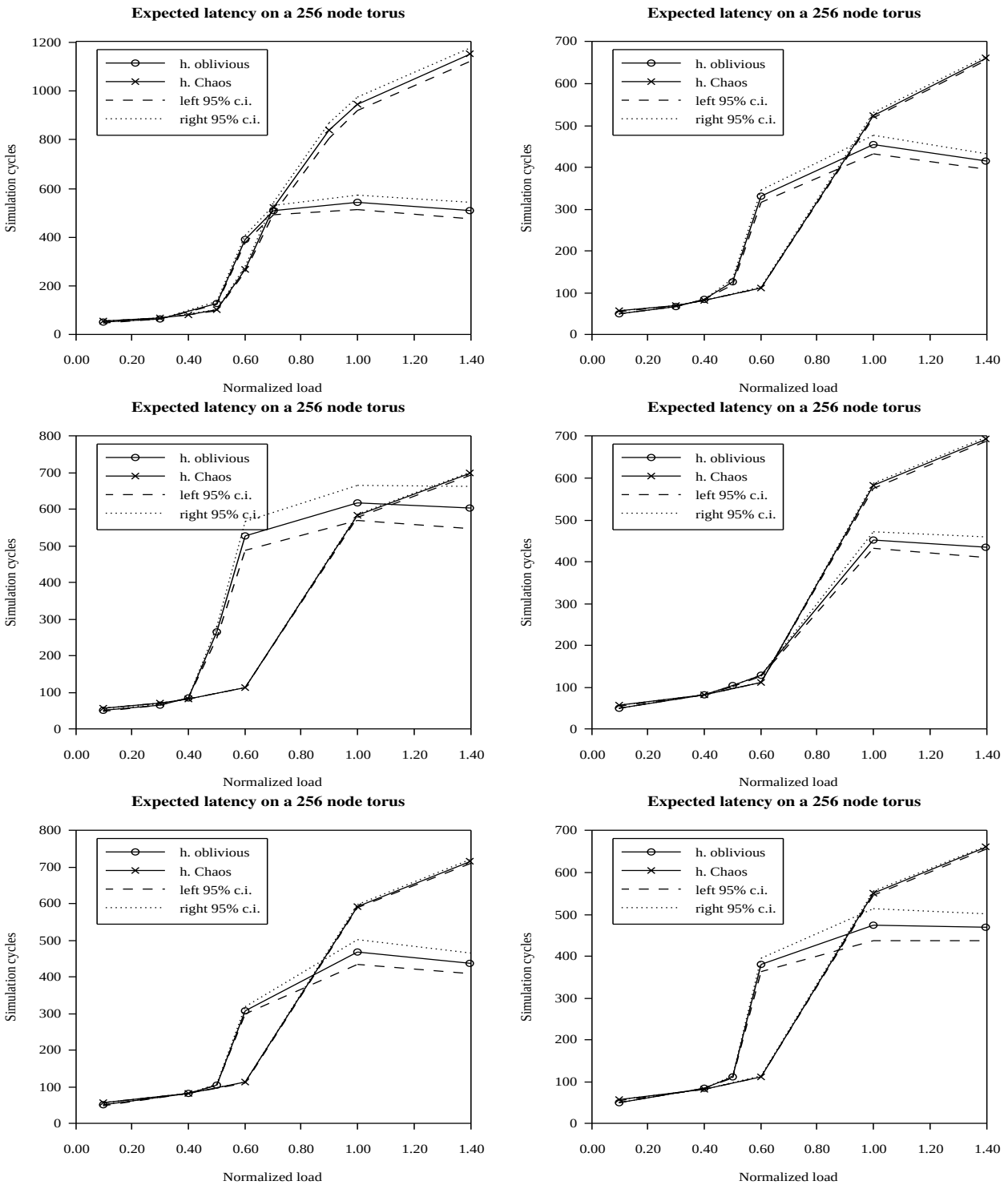

Figure 20: Torus hot spots latency cases 1, 2, 3, 4, 5, and 6.