# Lecture Notes on Message Routing in Parallel Machines [1]

Martin Tompa

Technical Report #94–06–05

June 22, 1994

Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, Washington, U.S.A. 98195



[1]This material is based upon work supported in part by the National Science Foundation under Grants CCR-9002891 and CCR-9301186.

# Contents

# Preface

These are the lecture notes from CSE 522, a graduate algorithms course I taught at the University of Washington in Spring 1994. The topic of the course was Message Routing in Parallel Machines. These notes are not intended to be a survey of that area, however, as there are numerous important results that I would have liked to cover but did not have time.

I am grateful to Allan Borodin, Tom Leighton, Prabhakar Raghavan, Baruch Schieber, and Hisao Tamaki, who helped me both with overview and with technical points in the proofs. I am particularly thankful for the students who attended faithfully, served as notetakers, asked embarassing questions, made perceptive comments, solved problems I claimed were open, and generally make teaching exciting and rewarding.

— Martin Tompa

# Lecture 1

# Overview of Routing in Parallel Machines

March 29, 1994
Notes: M. L. Fulgham

## 1.1. Responsibilities

1. Rotating notetaking: Notes are to be written up in LaTeX, and sent to tompa@cs. See the instructions in /homes/june/tompa/522/DIRECTIONS. Notes are due 10 a.m. the day following lecture, so that they may be distributed at the next lecture. The instructor will insert any citations for bibliography entries that don't already appear in the bibliography file.
2. Assignments (which will be relatively open-ended)
3. Class participation

## 1.2. Prerequisites

1. Analysis of Algorithms (CSE 521)
2. Basic probability theory (expectation, conditional probability and expectation, independent and mutually exclusive events)

## 1.3. Review of Order Notation

- $f(n) = O(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ |f(n)| \leq cg(n)$
- $f(n) = \Omega(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ f(n) \geq cg(n)$
- $f(n) = \Theta(g(n))$ if and only if $(\exists c)(\exists c')(\exists n_0)(\forall n \geq n_0)\ cg(n) \leq f(n) \leq c'g(n)$
- $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f(n) = \omega(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

## 1.4. Routing Context

The overview of routing problems in this section is derived in large part from the survey article of Leighton [13]. Other excellent surveys include those of Borodin [3] and Valiant [20], and Leighton's book [11].

The types of systems of parallel processors we will discuss include many possible arrangements of tightly-coupled processors. For instance, the system can be *synchronous*, in which processors execute commands simultaneously in "lock-step", or *asynchronous*, in which processors operate independently unless an explicit synchronization command is performed. The memory modules of the system can either be *attached*, with each processor having a share of the memory, or *separate*, with some nodes consisting of processors and others containing memory.

In any of these cases, nodes are connected to other nodes by an *interconnection network*, which can be modeled as a directed graph with nodes representing processors and memory modules, and edges representing direct communication links. For practicality, the nodes must have low degree, and the network must have a path between any two nodes.

How do processors simultaneously get the data they need? In order to communicate and share data, nodes send messages to each other. Since a node is not directly connected to all other nodes, a message is moved from one node to another until the message reaches its destination. The hardware responsible for directing messages is called the *router*. The router implements the *routing algorithm*, that is, rules by which messages travel through the network. Real machines devote a large portion of their resources (such as time and hardware) to this communication. Therefore, our goal is to route messages efficiently.

The most common abstract parallel model is the parallel random access machine (PRAM). The PRAM is composed of processors with a single shared memory. In one time step any processor can read or write to any memory location. Although this model is not realistic because the cost of data movement is ignored, it has the advantage of being easy to program. Because of this, there is already an extensive library of PRAM algorithms in the literature. In order to implement one of these algorithms on a real machine, the data needs to be routed among processors explicitly.

There is a wealth of variations on the routing problems in theory and practice. This is what makes routing an attractive and open area of research. Some of these variations are described below, both to give an idea of the scope of the field, and to introduce specific concepts and terms.

1. Topology: how are the nodes interconnected? (Some of these topologies will be defined in later lectures.)
   (a) Trees
   (b) Arrays (meshes) of various dimensions
   (c) Hypercubes
   (d) Butterflies

2. Flow control: how are messages transmitted from node to node?
   (a) Store and forward: Messages are broken up into packets and each packet is sent individually. Each packet is completely received in a node before it is forwarded to the next node. Only one packet is allowed to traverse each link per step.
   (b) Circuit-switching: The entire path from source to destination is reserved. After the path is established, the message is sent. When message transmission is finished, the links are released. This is commonly used in telephone switching networks, but is rarer in parallel machines.
   (c) Worm routing: This is a cross between store and forward and circuit-switching. Messages are divided into a sequence of flits (about a byte or two of data). All flits follow the first flit in a pipelined fashion, so that consecutive flits are located in the same or adjacent nodes. Again, only one flit is allowed to traverse each link per step.
3. Processor clocks
   (a) Synchronous: costly, but easier to reason about than asynchronous
   (b) Asynchronous
4. Types of routes
   (a) Minimal (messages are required to take a shortest path from source to destination) vs. nonminimal
   (b) Oblivious (a packet's route depends only on the source and destination of the message) vs. adaptive (the route may depend on the state of the network as the packet traverses it, including other packets encountered)
5. Queueing discipline at nodes (in routing, the word "queue" is used interchangeably with "buffer", and does not necessarily connote FIFO):
   (a) First-in-first-out (FIFO)
   (b) Farthest-first
   (c) Nearest-first
   (d) "Hot potato" or "deflection" (no queue at all)
   (e) Incoming queues: queues associated with the input link (i.e., heads of directed edges)
   (f) Outgoing queues: queues associated with the output link (i.e., tails of directed edges)
   (g) Central queues: queues associated with the node as a whole rather than with its links
6. Types of routing problems
   (a) Static: all messages are known and ready to be sent at the first step.
      i. (Partial) permutations: at most one message is sent from and sent to each processor.
      ii. Multibroadcast or one-to-many: a single node may send messages to many nodes.
      iii. Many-to-one: many nodes may send messages to a single node.
   (b) Dynamic: messages are injected by processors continually.

7. Performance measures
   (a) Deadlock: there exists a set of messages none of which can move.
   (b) Livelock: there exists a message that can move, but never reaches its destination.
   (c) Total time to route all messages (in the case of static routing) or delay per message (in the case of dynamic routing)
   (d) Queue size per node
   (e) Hardware per node, e.g., area or number of pins
   (f) Switching time per node
   (g) Worst case vs. average case
   (h) Fault-tolerance: routing with faulty links or nodes

## 1.5. Greedy Algorithms on the Mesh

One of the simplest routing algorithms to describe is the "greedy" algorithm on the $n \times n$ mesh [11]. The following are the assumptions made for and a description of this algorithm.

- Topology: There are $n^2$ nodes arranged in an $n \times n$ array. Each node is adjacent to its row neighbors and its column neighbors, so has indegree at most 4 and outdegree at most 4. The mesh is popular since it is an easy topology to build.
- Network assumptions: Store and forward flow control is used. The nodes are synchronous and have unbounded queues. Routing is restricted to partial permutations, with messages that fit in a single packet.
- Routing algorithm: First correct the column position by moving in the source row to the destination column. Then correct the row position by moving in the destination column to the destination. Note in general that some packets will be moving along rows while others are already moving along their destination columns. This algorithm is minimal and oblivious.
- Queueing discipline (to resolve link contention): Prefer the packet with the farthest to go in the direction of the link under contention.

# Lecture 2

# Greedy Routing on the Mesh

March 31, 1994
Notes: Brendan Mumey

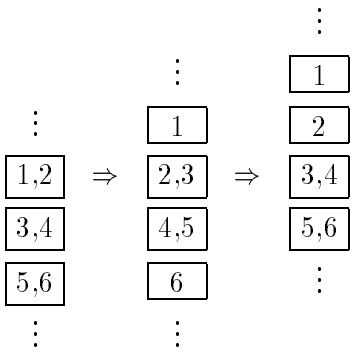
## 2.1. Greedy Static Permutation Routing

**Theorem 2.1:** On the $n \times n$ mesh with unbounded queues, the greedy algorithm using farthest-first discipline can route all packets for any permutation in at most $2n - 2$ steps.

**Note:** This is optimal, since $2n - 2$ is the diameter of the $n \times n$ mesh.

**Proof:** Within the rows there is no contention for links, so every packet arrives at its column destination in at most $n - 1$ steps. Within the columns, northbound and southbound packets do not contend for the same links, so we restrict attention to northbound packets. Number the rows $1, 2, \ldots, n$ from north to south.

Because of the farthest-first discipline, we would like to argue that a packet destined for row $i$ can be delayed only by those destined for rows $1, 2, \ldots, i-1$, and by each of those only once. The total time would then be $i - 1$ steps of delay plus $n - i$ steps of link traversals. This argument is incorrect because it is possible for a given packet to delay another more than once. An example that illustrates this is given below. Numbers indicate destination rows, and the packet destined for row 5 delays the one destined for row 6 more than once.




Taking this example into account we can state a correct proof. Let an *i-priority* packet be a packet destined for one of rows $1, 2, \ldots, i$. For each $i$, we will show that every $i$-priority packet is in rows $1, 2, \ldots, i$ within another $n - 1$ steps. The result will follow from this, since the packet destined for row $i$ will stop as soon as it gets there.

Ignore packets that are not $i$-priority; they cannot delay $i$-priority packets because of the farthest-first discipline. Call the first time when all packets are in their destination columns time 0. We need to be done by time $n-1$. Consider the northernmost $i$-priority packet in a given column at time 0, breaking ties by the one that has farthest to go. It moves undelayed to its destination and so reaches rows $1, 2, \ldots, i$ within $n-i$ steps. Next consider the second northernmost packet at time 1. It can be delayed at most one step at time 0, but from then on is undelayed and so reaches rows $1, 2, \ldots, i$ in at most $1+(n-i)$ steps. Continuing in this fashion, the $i$th northernmost $i$-priority packet at time $i-1$ can be delayed at most $i-1$ steps and travels at most $n-i$ steps to reach rows $1, 2, \ldots, i$. Thus by time $n-1$ all $i$-priority packets reach rows $1, 2, \ldots, i$. □

Scott Hauck observed a simpler proof in class. It rests on the fact that the longest distance left to travel by any packet in a given column always decreases by one at each step. Since this distance is bounded initially by $n-1$, the result follows directly.

**Exercise 2.2:** Show that some permutation causes some queue to have $\Theta(n)$ size. Keep in mind that some packets can already be moving along their destination columns while others are still moving along their source rows.

**Exercise 2.3:** Consider a one-dimensional array with $n$ nodes and unbounded queues, and the routing problem with $n$ packets having distinct destinations, but not necessarily distinct sources. Show that there is a routing algorithm that delivers each packet in $O(d)$ time, where $d$ is the distance from that packet's source to its destination.

**Exercise 2.4:** Consider the permutation routing problem on the $n \times n$ mesh with unbounded queues. Show that there is a routing algorithm that delivers each packet in $O(d)$ time, where $d$ is the distance from that packet's source to its destination. (Hint: Use the greedy algorithm with nearest-first queueing. Unlike the proof of Theorem 2.1, you must take care about packets entering the columns while the packet you are considering is already advancing along its destination column. That makes this proof more complicated than that of Exercise 2.3.)

**Open Problem 2.5:** Consider the permutation routing problem on the $n \times n$ mesh with queues of size $q$. Using the greedy algorithm with nearest-first queueing, what is the time to deliver a packet whose distance from its source to its destination is $d$, as a function of $d$ and $q$? Can you achieve time $O(d)$ with constant queue size using this or some other more practical queueing discipline?

## 2.2. Greedy Dynamic Routing

In this section we examine a result of Leighton [11, 12] concerning the behavior of the same algorithm on the same network in the dynamic setting. At each step each node injects a packet with probability $\lambda$ and uniformly distributed destination. These events are all independent.

**Proposition 2.6:** $\lambda \leq 4/n$ is a necessary condition for stability.

**Proof:** If $\lambda > 4/n$ the expected number of packets injected into the left half of the mesh that are destined for the right half is $\frac{n^2}{2} \cdot \lambda \cdot \frac{1}{2} > n$. The delay will grow without bound because there are only $n$ links from the left half to the right half. □

Let $\rho = \frac{\lambda n}{4} \leq 1$ denote the *load*.

**Theorem 2.7:** On an $n \times n$ mesh, consider the greedy algorithm with farthest-first discipline. Suppose packets are injected dynamically with load $\rho < 0.99$. Then

1. the probability a given packet is delayed $\Delta$ steps is $e^{-\Omega(\Delta)}$,
2. for any window of $T$ steps, the maximum delay incurred by any packet injected during those steps is $O(\log T + \log n)$ with probability at least $1 - O\left(\frac{1}{Tn}\right)$, and
3. in any window of $T$ steps, the maximum queue size is $O\left(1 + \frac{\log T}{\log n}\right)$ with probability at least $1 - O\left(\frac{1}{Tn}\right)$.

Despite the fact that Theorem 2.7 is dealing with the dynamic case, the time and queue size are considerably better than those given for the static case by Theorem 2.1 and Exercise 2.2. The reason is that the random destinations give Theorem 2.7 more of an average case flavor.

**Definition 2.8:** $X$ is a *Bernoulli random variable* if and only if $Pr(X = 0) + Pr(X = 1) = 1$.

We next state a Chernoff-style bound that will be used in the proof of Theorem 2.7.

**Lemma 2.9:** Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables satisfying $\Pr(X_h = 1) \leq P_h$ for $1 \leq h \leq n$. Let $\beta > 1$, $X = X_1 + X_2 + \cdots + X_n$, and $P = P_1 + P_2 + \ldots + P_n$. Then

$$\Pr(X \geq \beta P) \leq e^{(1 - \frac{1}{\beta} - \ln \beta)\beta P}.$$

**Note:** For $\beta \geq 1$, $1 - \frac{1}{\beta} - \ln \beta$ decreases as $\beta$ increases: this can be verified by noting that the derivative with respect to $\beta$ is negative. Therefore, the exponent in Lemma 2.9 is negative for all $\beta > 1$.

**Note:**

$$E(X) = E\left(\sum_{h=1}^{n} X_h\right) = \sum_{h=1}^{n} E(X_h) \leq \sum_{h=1}^{n} P_h = P,$$

so Lemma 2.9 says that the probability that $X$ exceeds its expectation decays exponentially.

**Proof:** The proof appears in Leighton [11, Lemma 1.7]. □

**Exercise 2.10:** Show that, for any network with $N$ nodes, if each of $N$ packets is assigned a destination randomly and independently, then the maximum number of packets destined for any node is $O\left(\frac{\log N}{\log \log N}\right)$ with probability $1 - O(1/N)$.

# Lecture 3

# Greedy Dynamic Routing on the Mesh

April 5, 1994
Notes: Sung-Eun Choi

In this lecture, we begin the proof of part (1) of Theorem 2.7. We begin by introducing the "wide-channel" model.

The wide-channel model states that certain links in the network can be traversed by an arbitrary number of packets at each time step, that is, there are no delays on those links. Though not practical, this model will be a useful device in the proof of the theorem. In particular, for analyzing delays of packets while traveling along rows, we will use the *wide-row model* to indicate that there are no delays on any row edge. Likewise, for analyzing delays while traveling along columns, we will use the *wide-column model* to indicate that there are no delays on any column edge, but the normal restriction of one packet per link per step on any row edge. The term *wide-channel model* will refer to either of these two models.

In broad outline, the proof of part (1) of Theorem 2.7 is as follows. We will show that, if a packet is delayed a long time in the standard model, then

1. some edge $e$ has traffic on it at every step for a long time in the standard model, and so
2. $e$ has a lot of traffic on it for a long time in the wide-channel model.

We will show that this last event is unlikely by the Chernoff bound (Lemma 2.9).

Lemmas 3.1 and 3.2 below deal simultaneously with row delays and column delays.

**Lemma 3.1:** If packet $P$ crosses row (column) edge $e$ at step $T$ in the wide-row (respectively, wide-column) model, and crosses that same edge $e$ at step $T + \delta$ in the standard model, then a packet crosses $e$ at every step in the interval $[T, T + \delta]$ in the standard model.

**Proof:** The proof is by induction on $\delta$.

Basis ($\delta = 0$): The packet $P$ crosses edge $e$ at time $T$ in both models.

Induction ($\delta > 0$): Consider the configuration $C$ (see Figure 3.1) immediately after the first step in which $P$ is delayed in the standard model, but not in the wide-channel model. Suppose that packet $Q$ delayed $P$, that is, $Q$ used the link denied $P$. By the farthest-first discipline, we know that $Q$ will also cross edge $e$. It does so at time $T$ in the wide-channel model and at some time in the interval $[T, T + \delta - 1]$ in the standard model. By applying the induction hypothesis to

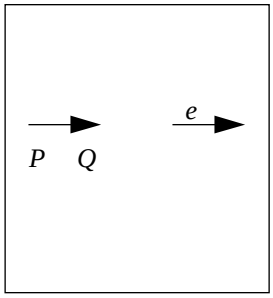



Figure 3.1: Configuration $C$.

$Q$, some packet crosses $e$ at time $T$ in the standard model. Starting the wide-channel model from configuration $C$, $P$ will cross $e$ at time $T+1$. In the standard model, $P$ still crosses $e$ at time $(T+1)+(\delta-1)$. By applying the induction hypothesis to $P$, some packet crosses $e$ at every step in $[T+1, T+\delta]$ in the standard model. □

**Lemma 3.2:** For any $T$, any $\Delta$, and any $x \leq \Delta$, if $x$ packets cross row (column) edge $e$ during the interval $[T+1, T+\Delta]$ in the standard model, then there is a $t \geq 0$ such that at least $x+t$ packets cross $e$ during $[T+1-t, T+\Delta]$ in the wide-row (respectively, wide-column) model.

**Proof:** Let $t \geq 0$ be the minimum value such that no packet crosses $e$ at time $T-t$ in the standard model. Then $x+t$ packets cross $e$ during the interval $[T+1-t, T+\Delta]$ in the standard model. By Lemma 3.1, in the wide-channel model, all $x+t$ packets cross $e$ after time $T-t$; otherwise, some packet would cross $e$ at time $T-t$ in the standard model. Therefore, they all cross $e$ in the interval $[T+1-t, T+\Delta]$ in the wide-channel model. □

The next two lemmas specialize to delays on row edges.

**Lemma 3.3:** Let $\rho \leq 0.99$ be the load of the network. For any fixed $1 < \alpha \leq 1/\rho$, and for any $T$, $\Delta$, and row edge $e$, let

$$p = \Pr(\text{at least } \alpha\rho\Delta \text{ packets cross } e \text{ during } [T+1, T+\Delta] \text{ in the standard model}).$$

Then $p = O(e^{(1-1/\alpha-\ln\alpha)\alpha\rho\Delta})$.

**Proof:** Let

$$q = \Pr((\exists t \geq 0) \text{ at least } \alpha\rho\Delta + t \text{ packets cross } e \text{ during } [T+1-t, T+\Delta] \text{ in the wide-row model}).$$

By Lemma 3.2, $p \leq q$.

Suppose without loss of generality that $e$ is the edge from node $(i,j)$ to node $(i,j+1)$. For a packet $P$ to cross $e$ at step $\tau$ in the wide-row model, $P$ would have to be injected at some node $(i,k)$, where $1 \leq k \leq j$, during step $\tau-(j-k)-1$ and have destination in one of the columns $j+1, j+2, \ldots, n$. (See Figure 3.2.)

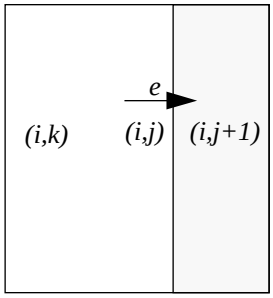



Figure 3.2: A packet injected at node $(i,k)$ destined for the shaded region must cross edge $e$.

For fixed $\tau$ and $k$, the probability of this event is $\lambda(n-j)/n$, where $\lambda$ is the injection rate. For fixed $\tau$, the expected number of such packets is

$$\frac{\lambda(n-j)j}{n} \leq \frac{\lambda(\frac{n}{2})(\frac{n}{2})}{n} = \frac{\lambda n}{4} = \rho.$$

Note that $(T+1-t) \leq \tau \leq (T+\Delta)$.

In Lemma 2.9, let the number of independent random variables be $j(\Delta+t)$, let $P_h = \frac{\lambda(n-j)}{n}$, and let $P = \frac{j(\Delta+t)\lambda(n-j)}{n}$. Define $\beta$ so that $\beta P = \alpha\rho\Delta + t$, that is,

$$\begin{aligned}
\beta &= \frac{(\alpha\rho\Delta+t)n}{\lambda j(n-j)(\Delta+t)} \\
&\geq \frac{4(\alpha\rho\Delta+t)}{\lambda n(\Delta+t)} \\
&= \frac{\alpha\rho\Delta+t}{\rho(\Delta+t)} \\
&\geq \frac{\alpha\rho(\Delta+t)}{\rho(\Delta+t)} \\
&= \alpha.
\end{aligned}$$

Then, by Lemma 2.9,

$$\begin{aligned}
p \leq q &\leq \sum_{t=0}^{\infty} e^{(1-1/\beta-\ln\beta)(\alpha\rho\Delta+t)} \\
&\leq \sum_{t=0}^{\infty} e^{(1-1/\alpha-\ln\alpha)(\alpha\rho\Delta+t)} \qquad \text{(Note following Lemma 2.9)} \\
&= e^{(1-1/\alpha-\ln\alpha)\alpha\rho\Delta} \sum_{t=0}^{\infty} e^{(1-1/\alpha-\ln\alpha)t}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{e^{(1-1/\alpha-\ln\alpha)\alpha\rho\Delta}}{1-e^{1-1/\alpha-\ln\alpha}} \\
&= O(e^{(1-1/\alpha-\ln\alpha)\alpha\rho\Delta}).
\end{aligned}
$$

□

# Lecture 4

# Greedy Dynamic Routing (Continued)

April 7, 1994
Notes: Rakesh K Sinha

We will complete the proof of parts (1) and (2) of Theorem 2.7. A packet can be delayed along both row and column edges. We will bound these delays separately.

## 4.1. Row Delay

**Lemma 4.1:** The probability that a given packet $P$ is delayed at least $\Delta$ steps before reaching its destination column is $e^{-\Omega(\Delta)}$.

**Proof:** Let $e$ be the last row edge $P$ crosses before reaching its destination column. Assume $P$ crosses $e$ at time $T$ in the wide-row model, and at time $T+\Delta$ or later in the standard model. By Lemma 3.1 some packet crosses $e$ at each step in $[T, T+\Delta]$ in the standard model.

By Lemma 3.3, with $\alpha = 1/\rho$, the probability of this event is $e^{(1-\rho+\ln\rho)(\Delta+1)} = e^{-\Omega(\Delta)}$. □

This proves part (1) of Theorem 2.7 for the delay along row edges. Next, we will consider the delay along column edges.

## 4.2. Column Delay

Suppose, without loss of generality, that $e$ is the column edge from node $(i,j)$ to node $(i+1,j)$.

**Lemma 4.2:** For any $T, \Delta$, column edge $e$, and constant $\epsilon > 0$, let $p$ be the probability that there is a packet $P$ that crosses edge $e$ during $[T+1, T+\Delta]$ in the wide-column model after a row delay of $\delta \geq \epsilon\Delta$. Then $p = e^{-\Omega(\Delta)}$.

**Proof:** For this event to happen, $P$ must be injected at some node in the first $i$ rows during the interval $[T+1-l-\delta, T+\Delta-l-\delta]$, where $l$ is the distance from $v$ to $e$, with destination in the last $n-i$ rows of column $j$.

At any particular time step, the probability of $P$ getting injected in the first $i$ rows is $\lambda in$; the probability of it having a destination in the last $n-i$ rows of column $j$ is $\frac{n-i}{n^2}$; and, by Lemma 4.1, the probability of $P$ getting delayed for $\delta$ steps before reaching its destination column is $e^{-\Omega(\delta)}$.

Summing over all possible time steps that packet $P$ can be injected and all possible values of the delay $\delta$, we have

$$\begin{aligned}
p &\leq \sum_{\delta \geq \epsilon\Delta} \Delta\lambda in \frac{n-i}{n^2} e^{-\Omega(\delta)} \\
&\leq \Delta\lambda \frac{(\frac{n}{2})(\frac{n}{2})}{n} \sum_{\delta \geq \epsilon\Delta} e^{-\Omega(\delta)} \\
&\leq \Delta(\frac{\lambda n}{4}) \sum_{\delta \geq \epsilon\Delta} e^{-\Omega(\delta)} \\
&< \Delta \sum_{\delta \geq \epsilon\Delta} e^{-\Omega(\delta)} \qquad (\rho = \tfrac{\lambda n}{4} < 1) \\
&= \Delta \frac{e^{-\Omega(\epsilon\Delta)}}{1 - e^{-\Omega(1)}} \\
&= e^{-\Omega(\Delta)}.
\end{aligned}$$

□

**Lemma 4.3:** Let $0 < \epsilon < \frac{1-\rho}{\rho}$ and $1 < \alpha \leq \frac{1}{\rho(1+\epsilon)}$ be constants. (Note that $\rho(1+\epsilon) < 1$.) For any $T, \Delta, t$, and column edge $e$, let $p$ be the conditional probability that at least $\alpha\rho(1+\epsilon)\Delta + t$ packets cross $e$ during $[T+1-t, T+\Delta]$ in the wide-column model, given that all such packets have a row delay at most $\epsilon(\Delta + t)$. Then $p \leq e^{(1-1/\alpha-\ln\alpha)(\alpha\rho(1+\epsilon)\Delta+t)}$.

**Proof:** This is similar to the proof of Lemma 3.3. There are $in$ nodes where such a packet could be injected, and $n-i$ (out of the $n^2$) possible destinations, if it is to cross $e$. This packet has to cross $e$ within a time interval of length $\Delta + t$ and can get delayed at most $\epsilon(\Delta + t)$ many steps in the rows, so there are at most $(\Delta + t) + \epsilon(\Delta + t) = (1+\epsilon)(\Delta + t)$ distinct time steps during which it could be injected.

In Lemma 2.9, let the number of random variables be $in(1+\epsilon)(\Delta + t)$, $P_h = \lambda(n-i)/n^2$, and $P = \lambda(n-i)i(1+\epsilon)(\Delta + t)/n$. Define $\beta$ so that $\beta P = \alpha\rho(1+\epsilon)\Delta + t$, that is,

$$\begin{aligned}
\beta &= \frac{n\alpha\rho(1+\epsilon)\Delta + t}{\lambda(n-i)i(1+\epsilon)(\Delta + t)} \\
&\geq \frac{\alpha\rho(1+\epsilon)\Delta + t}{\rho(1+\epsilon)(\Delta + t)} \\
&\geq \frac{\alpha\rho(1+\epsilon)(\Delta + t)}{\rho(1+\epsilon)(\Delta + t)} \\
&= \alpha.
\end{aligned}$$

There is a subtle point here: the Chernoff bound (Lemma 2.9) only applies to independent random variables, but the events of the various packets crossing edge $e$ during $[T+1-t, T+\Delta]$ are not

independent because of possible interaction on the rows. However, the probabilities given are simply those for injection with appropriate destinations at appropriate times, which are independent. That is, the independent events we consider are the injections of all packets that could possibly cross $e$ in the right interval, rather than those that actually do so.

By Lemma 2.9, then, $p \leq e^{(1-1/\beta-\ln\beta)\beta P} \leq e^{(1-1/\alpha-\ln\alpha)(\alpha\rho(1+\epsilon)\Delta+t)}$. □

**Lemma 4.4:** For any $T$, $\Delta$, and column edge $e$, let $p$ be the probability that $\Delta$ packets cross $e$ during $[T+1, T+\Delta]$ in the standard model. Then $p = e^{-\Omega(\Delta)}$.

**Proof:** Let $q$ be the probability that there is a $t \geq 0$ such that at least $\Delta + t$ packets cross $e$ during $[T+1-t, T+\Delta]$ in the wide-column model. By Lemma 3.2, $p \leq q$. Let $\epsilon = \frac{1-\rho}{2\rho}$, and $\alpha = \frac{1}{\rho(1+\epsilon)} = \frac{2}{1+\rho}$. Let $W_t$ be the event that $\Delta + t$ packets cross $e$ during $[T+1-t, T+\Delta]$ in the wide-column model, and let $R_t$ be the event that all such packets have row delay at most $\epsilon(\Delta + t)$. Then

$$\begin{aligned}
p \leq q &\leq \sum_{t=0}^{\infty} \Pr(W_t) \\
&= \sum_{t=0}^{\infty} (\Pr(W_t \wedge R_t) + \Pr(W_t \wedge \neg R_t)) \\
&= \sum_{t=0}^{\infty} (\Pr(W_t | R_t) \Pr(R_t) + \Pr(W_t | \neg R_t) \Pr(\neg R_t)) \\
&\leq \sum_{t=0}^{\infty} (\Pr(W_t | R_t) + \Pr(\neg R_t)) \\
&\leq \sum_{t=0}^{\infty} e^{(1-\frac{1+\rho}{2}+\ln\frac{1+\rho}{2})(\Delta+t)} + \sum_{t=0}^{\infty} e^{-\Omega(\Delta+t)} && \text{(Lemmas 4.2 and 4.3)} \\
&= \sum_{t=0}^{\infty} e^{-\Omega(\Delta+t)} && (\tfrac{2}{1+\rho} > 1) \\
&= e^{-\Omega(\Delta)} \sum_{t=0}^{\infty} e^{-\Omega(t)} \\
&= e^{-\Omega(\Delta)}.
\end{aligned}$$

□

**Lemma 4.5:** Let $p$ be the probability that packet P is delayed for at least $\Delta$ steps in its column before reaching its destination. Then $p = e^{-\Omega(\Delta)}$.

**Proof:** Let $e$ be the last column edge $P$ crosses before reaching its destination. Assume $P$ crosses $e$ at time $T$ in the wide-column model, and at time $T + \Delta$ or later in the standard model.

By Lemma 3.1, $\Delta$ packets cross $e$ during $[T+1, T+\Delta]$ in the standard model. The result then follows from Lemma 4.4. □

## 4.3. Maximum Delay in a Window of $T$ Steps

Lemmas 4.1 and 4.5 prove part (1) of Theorem 2.7. Now we give the proof of part (2).

**Corollary 4.6:** For any window of $T$ consecutive steps, the maximum delay incurred by any packet injected during those steps is $O(\log T + \log n)$ with probability at least $1 - \frac{1}{Tn}$.

**Proof:** In part (1) of Theorem 2.7, we proved that the probability that any particular packet $P$ is delayed $\Delta$ or more steps is $e^{-\Omega(\Delta)}$. For ease of notation, let us choose a constant $c > 0$ such that this probability is at most $e^{-c\Delta}$.

Choose $\Delta = \frac{1}{c}(2\ln T + 3\ln n)$. Then

$$\Pr(P \text{ is delayed at least } \tfrac{1}{c}(2\ln T + 3\ln n) \text{ steps}) \leq e^{-(2\ln T + 3\ln n)} = \frac{1}{T^2 n^3}.$$

Since at most $Tn^2$ packets can be injected in $T$ steps, the probability that any of them is delayed $\frac{1}{c}(2\ln T + 3\ln n)$ or more steps is at most $(Tn^2)\frac{1}{T^2n^3} = \frac{1}{Tn}$. □

# Lecture 5

# Queue Size in Greedy Dynamic Routing

April 12, 1994
Notes: Rakesh K Sinha

## 5.1. Maximum Number of Packets in Any Queue

The remainder of the proof of Theorem 2.7 refers only to the standard model; we are done with the wide-channel model. We will need the following lemma in order to prove part (3) of Theorem 2.7.

**Lemma 5.1:** For any $t$, $\Delta$, $q$, and edge $e$, let $p$ be the probability that the queue at edge $e$ contains at least $q$ packets at time $t$, given that it is empty at least once during $[t-\Delta+1, t]$. Then

$$p = \left(\frac{O(\Delta)}{qn}\right)^q + qe^{-\Omega(\Delta)}.$$

**Proof:** Fix a time $t_0 \in [t-\Delta+1, t]$ when $e$'s queue is empty. Let $v$ be the node at $e$'s tail.

Case 1: $e$ is a row edge. The number of packets in $e$'s queue at time $t$ is at most the number of packets injected by $v$ during $[t_0, t]$.

$$\begin{aligned}
p &\leq \Pr(\text{at least } q \text{ packets are injected by } v \text{ during } [t_0, t]) \\
&\leq \Pr(\text{at least } q \text{ packets are injected by } v \text{ in a window of } \Delta \text{ steps}) \\
&\leq \binom{\Delta}{q}\lambda^q \\
&\leq \left(\frac{\Delta e}{q}\right)^q \left(\frac{4}{n}\right)^q && \text{(Stirling's approximation)} \\
&= \left(\frac{O(\Delta)}{qn}\right)^q.
\end{aligned}$$

Case 2: $e$ is a column edge. A packet will be said to *turn at* $e$ if and only if it enters $e$'s queue either after being injected by $v$, or after entering $v$ from the east or west. The number of packets in $e$'s queue at time $t$ is at most the number of packets turning at $e$ during $[t_0, t]$.

We will consider the following two events:

- $A$ is the event that at least $q$ packets turn at $e$ during $[t_0, t]$.

- $B$ is the event that the first $q$ packets turning at $e$ during $[t_0, t]$ each have row delay at most $\Delta$ steps.

Then

$$\begin{aligned} p &\leq \Pr(A) \\ &= \Pr(A \wedge B) + \Pr(A \wedge \neg B) \\ &= \Pr(A \mid B)\Pr(B) + \Pr(A \mid \neg B)\Pr(\neg B) \\ &\leq \Pr(A \mid B) + \Pr(\neg B). \end{aligned}$$

We already have a bound on $\Pr(\neg B)$: By Lemma 4.1, $\Pr(\neg B) = qe^{-\Omega(\Delta)}$ so all that is left is to prove an upper bound on $\Pr(A \mid B)$.

The first $q$ packets turning at $e$ must be injected at one of the $n$ nodes in $v$'s row during an interval of length at most $2\Delta$. (More specifically, a packet with row delay at most $\Delta$, and originating at distance $d$ from $v$ must have been injected during $[t_0 - d - \Delta, t - d]$.)

For each source node and injection time, the probability of injection with a destination that would cause a turn at $e$ is at most $\frac{\lambda n}{n^2} \leq \frac{4}{n^2}$.

$$\begin{aligned} \Pr(A \mid B) &\leq \binom{2\Delta n}{q}\left(\frac{4}{n^2}\right)^q \\ &\leq \left(\frac{2e\Delta n}{q}\right)^q \left(\frac{4}{n^2}\right)^q \qquad \text{(Stirling's approximation)} \\ &= \left(\frac{O(\Delta)}{qn}\right)^q. \end{aligned}$$

Thus, $p \leq \Pr(A \mid B) + \Pr(\neg B) = \left(\frac{O(\Delta)}{qn}\right)^q + qe^{-\Omega(\Delta)}$ □

We are now ready to prove part (3) of Theorem 2.7.

**Lemma 5.2:** In any window of $T$ steps, the maximum number of packets in any queue is $O\left(1 + \frac{\log T}{\log n}\right)$ with probability $1 - O\left(\frac{1}{Tn}\right)$.

**Proof:** Fix an edge $e$ and a time step $t$ in the given window of $T$ steps. We define the following two events:

- $A$ is the event that the queue at $e$ is empty at least once during $[t - \Delta + 1, t]$, where $\Delta$ will be determined later.
- $B$ is the event that the queue at $e$ contains at least $q$ packets at time $t$.

We will prove that, for an appropriate choice of $q$, $B$ is very unlikely.

$$\Pr(B) = \Pr(A \wedge B) + \Pr(\neg A \wedge B) = \Pr(B \mid A)\Pr(A) + \Pr(B \mid \neg A)\Pr(\neg A) \leq \Pr(B \mid A) + \Pr(\neg A).$$

$$\begin{aligned}
\Pr(\neg A) &= \Pr(e\text{'s queue is not empty during } [t-\Delta+1, t]). \\
&= \Pr(\Delta \text{ packets cross } e \text{ during } [t-\Delta+1, t]) \\
&= e^{-\Omega(\Delta)} && \text{(Lemmas 3.3 and 4.4)}
\end{aligned}$$

By Lemma 5.1, $\Pr(B \mid A) = \left(\frac{O(\Delta)}{qn}\right)^q + qe^{-\Omega(\Delta)}$, so $\Pr(B) = \left(\frac{O(\Delta)}{qn}\right)^q + (q+1)e^{-\Omega(\Delta)}$.

For sufficiently large $\Delta$, we can choose constants $c_1$ and $c_2$ such that

$$\Pr(B) \leq \left(\frac{c_1\Delta}{qn}\right)^q + (q+1)e^{-c_2\Delta}$$

Choose $\Delta = \frac{3}{c_2}(\ln n + \ln T)$ and $q = 6\left(1 + \frac{\ln T}{\ln n}\right)$. Then $\frac{c_1\Delta}{qn} = \frac{c_1 \ln n}{2c_2 n} \leq \frac{1}{\sqrt{n}}$, for sufficiently large $n$.

$$\begin{aligned}
\Pr(B) &\leq \left(\frac{1}{\sqrt{n}}\right)^{6\left(1+\frac{\ln T}{\ln n}\right)} + \left(7 + \frac{6\ln T}{\ln n}\right) e^{-3(\ln n + \ln T)} \\
&\leq \left(e^{\frac{-\ln n}{2}}\right)^{6\left(1+\frac{\ln T}{\ln n}\right)} + \frac{7 + 6\ln T}{T^3 n^3} \\
&\leq e^{-3(\ln n + \ln T)} + \frac{1}{T^2 n^3} && \text{(for sufficiently large } T\text{)} \\
&\leq \frac{2}{T^2 n^3}
\end{aligned}$$

Since we have $T$ choices for $t$ and $n^2$ choices for $e$, the probability that some edge has more than $6\left(1 + \frac{\ln T}{\ln n}\right)$ packets during the window of $T$ steps is at most $Tn^2\left(\frac{2}{T^2 n^3}\right) = O\left(\frac{1}{Tn}\right)$ □

This completes the proof of Theorem 2.7.

## 5.2. Decreasing Queue Size for Permutation Routing

We now return to the topic of Section 2.1, which was (worst case) permutation routing on the mesh, and we concentrate on improving the queue size. Recall that the greedy algorithm for static permutation routing uses $\Theta(n)$ size queues (Exercise 2.2). Our goal is to find an efficient algorithm with constant size queues.

The first approach to reducing queue size uses randomness, and is based on ideas of Valiant and Brebner [21] (see Lecture 19). It works in two phases: in the first phase, each packet is sent to a randomly chosen nearby node in its source column; in the second phase, we use the greedy algorithm with farthest-first queueing to route the packets to their destinations. This is still oblivious, but nonminimal. (See Section 1.4 for terminology.)

Leighton [11, Section 1.7.3] describes such an algorithm that, with high probability, routes any permutation within $2n + O(\log n)$ steps and uses $O(1)$ size queues. Note that the probability here is with respect to the random decisions made by the algorithm and not the input permutation; the time and queue size bounds hold for the worst case permutation.

**Open Problem 5.3:** Show that this randomized algorithm works well in the dynamic case. It is unreasonable to assume that injected packets have random destinations (since Theorem 2.7 shows that this problem is solved by the simpler greedy algorithm), nor that injected packets have worst case destinations (since degeneracies arise if all have the same destination). One possible assumption is that packets are injected at random times according to a reasonable rate $\lambda$, and destinations are assigned by an adversary that is subject to the restriction that not too many packets with the same destination can be active at any time.

# Lecture 6

# Permutation Routing on the Mesh with Small Queues

April 14, 1994
Notes: Tracy Kimbrel

We continue with the topic of Section 5.2, namely (worst case) permutation routing on the $n \times n$ mesh, using small queues. The goals for the algorithms considered are as follows:

- The running time should be at most $cn$, with $c$ a small constant as close to 2 as possible.
- The queue size should be small (preferably a small constant).

## 6.1. Deterministic Permutation Routing Based on Sorting

Suppose that there are $x$ packets in the mesh, at most one per node, each containing a key. They are to be routed in such a way that, at completion, the first $x$ nodes in column-major order contain the $x$ packets arranged in sorted order according to their keys, one packet per node, and the remaining nodes contain no packets. We will refer to this process simply as *sorting the packets*. We will assume as a primitive that packets can be so sorted in $4n + o(n)$ steps with a queue of size 1 for each link. (See Leighton [11, Section 1.6.3].) Three algorithms based on sorting will be described. All are adaptive and nonminimal (because the sorting phase is adaptive and nonminimal).

The first two algorithms are due to Kunde [10]. The first of these routes permutations using $6n + o(n)$ steps and a queue of size 1 for each link. The algorithm consists of three phases:

> Phase 1: Sort the packets using their destination columns as keys, and breaking ties arbitrarily.
>
> Phase 2: Route each packet along its current row to its destination column.
>
> Phase 3: Route each packet along its destination column to its destination.

After Phase 1 of the algorithm is finished, for any row $i$ and column $j$, there is at most 1 packet in row $i$ destined for column $j$. (If there were two, then at least $n + 1$ packets would be destined for column $j$.) After Phase 2, therefore, there is at most 1 packet in each node. Thus, there is no contention for links in Phase 2 or in Phase 3, and these two phases require at most $2n - 2$ steps. Thus the total time of the three phases is $6n + o(n)$, as claimed, and queues of size 1 are sufficient.

The second algorithm, also due to Kunde, provides a tradeoff between the algorithm above and the greedy algorithm, in an attempt to improve the time bound without sacrificing too much in queue size. Specifically, for any $1 \leq q \leq n$, the algorithm uses $(2 + 4/q + o(1/q))n$ steps and queues of size $2q - 1$. Examining the endpoints of this tradeoff curve, we see that for $q = 1$ the algorithm uses $6n + o(n)$ steps and queue size 1 as does the first algorithm, and for $q = n$ it uses $2n + O(1)$ steps, and queues of size $O(n)$ as does the greedy algorithm. (In fact, these limiting cases of the second algorithm are identical to the algorithm presented above, and the greedy algorithm, respectively.) Assume for simplicity that $q$ divides $n$. The algorithm has two phases:

Phase 1: Partition the mesh into $q^2$ blocks, each of size $\frac{n}{q} \times \frac{n}{q}$. Sort each block independently using the destination column as the key, and breaking ties arbitrarily.

Phase 2: Finish routing using the greedy algorithm with farthest-first queueing discipline.

To prove the claim for the running time, observe that if the queue length were unbounded, by Theorem 2.1, Phase 2 uses only $2n - 2$ steps. Phase 1 uses $4\frac{n}{q} + o(\frac{n}{q})$ steps, for a total of $2n + 4\frac{n}{q} + o(\frac{n}{q})$ steps (with unbounded queues). Now we need only show that, in fact, no queue size ever exceeds $2q - 1$. Queues of size 1 are enough for the sorting phase, so we need consider only the maximum queue size during Phase 2. Notice that the maximum queue size during this phase is at most the maximum number of packets in any row at the end of Phase 1 that are destined for the same column, since the size of the queue for a column edge can increase only by the arrival of a (turning) packet on a row edge. (There is no contention for row edges.) For a given row $i$, suppose that the $k^{th}$ block containing some piece of row $i$ has $r_k$ packets destined for a given column $j$. Note that $\sum_{k=1}^{q} r_k \leq n$. Only $\lceil r_k \frac{q}{n} \rceil$ packets destined for column $j$ can be in row $i$ of the $k^{th}$ block, because of the sorting done within each block in Phase 1. Thus the maximum queue size is at most

$$\begin{aligned}
\sum_{k=1}^{q} \lceil r_k \frac{q}{n} \rceil &< \sum_{k=1}^{q} \left( r_k \frac{q}{n} + 1 \right) \\
&= q + \frac{q}{n} \sum_{k=1}^{q} r_k \\
&\leq q + \frac{q}{n} n \\
&= 2q.
\end{aligned}$$

**Exercise 6.1:** Suppose that $D$ is the maximum, over all packets $P$, of the distance from $P$'s source to $P$'s destination. Modify this algorithm so that it uses $O(D)$ steps and constant size queues. Can you do the same if you do not know the value of $D$?

The third algorithm is more complicated than the first two and will not be presented. It is due to Rajasekaran and Overholt [19], and uses $2n - 2$ steps and queues of size 112. This improves the queue size of an earlier algorithm of Leighton, Makedon, and Tollis [14].

## 6.2. Discussion of Practical Routing on the Mesh

Are these algorithms really practical? Some of the algorithms require a large (albeit constant) queue size. Sorting requires complex logic (implemented either in hardware or software) at each node. The cost (either in hardware or time) may be prohibitive for large networks. Sorting doesn't seem to extend to the dynamic routing problem: in the dynamic case, packets are not all available at the start in order to sort them. Even if the algorithm delayed until some number of packets was ready to sort, what would it do with packets that were injected once the sort phase had begun? Also, the sorting algorithm (and even our analysis of the greedy algorithm) depends on the assumption of a synchronous network. In real applications, synchronization is prohibitively expensive. But then, in real applications, the routing problems encountered are not static permutations; why consider static permutation routing at all? Permutation routing serves as an analyzable benchmark for comparison of routing algorithms, with the hope that performance on permutations is indicative of performance in more practical settings.

**Open Problem 6.2:** Is there a simple and practical algorithm for routing all permutations on the $n \times n$ mesh? The algorithm should meet the following criteria:

- It should be deterministic.
- It should run in time $cn$ for a small constant $c$ (or, even better, a packet with distance $d$ from source to destination should be delivered in at most $cd$ steps).
- The queue size should be bounded by a small constant.
- It should use a simple and fast queueing discipline.
- It should extend to the asynchronous and dynamic settings.

There is a negative result that bears on Open Problem 6.2. A routing algorithm will be called *destination-exchangeable* if and only if it does not depend on any packet's full destination address, but only on knowing which links from the current node take the packet closer to its destination. For instance, the greedy algorithm with FIFO queueing discipline is destination-exchangeable, but the algorithms based on sorting are not. Destination-exchangeability is an attempt to characterize the simplicity of the queueing discipline.

Chinn, Leighton, and Tompa [6] show that any deterministic, minimal, destination-exchangeable routing algorithm on the $n \times n$ mesh requires $\Omega(n^2/q^2)$ steps to route some permutation, where $q$ is the queue size. (Note that the sorting-based algorithms are neither minimal nor destination-exchangeable, which is why they do not contradict this lower bound.)

# Lecture 7

# Deflection Routing on the Mesh

April 19, 1994
Notes: Darren C. Cronquist

Deflection routing is a form of packet routing in which every message is required to traverse a link at every step, preferably moving closer to its destination. Thus, deflection routing does not require queueing at the nodes. More accurately, each node has a queue of size one for each incoming link, and every queue must be emptied at every step. Since the packets are continuously moving from node to node, deflection routing is often called "hot potato" routing. The key motivation for deflection routing is to achieve fast cycle time. Hence, each node's routing decision should be simple and based on local information such as the node's identity and its current messages' sources, destinations, and incoming links.

In this lecture, three deflection routing algorithms will be described that route permutations on the $n \times n$ mesh. They are presented in increasing order of difficulty. Not surprisingly, as the difficulty of the algorithms increases, the time complexity decreases. These results are due to Bar-Noy, Raghavan, Schieber, and Tamaki [1].

## 7.1. An $O(n^2)$ Time Algorithm

At first glance, it is not clear that deflection routing algorithms can avoid livelock. The following theorem should be viewed simply as such an existence proof.

**Theorem 7.1:** There exists an $O(n^2)$ time deflection routing algorithm that routes any permutation on the $n \times n$ mesh, for even $n$.

**Proof:** Fix any directed Hamiltonian cycle of the mesh. Route all packets along this cycle until they reach their destinations. Since $n$ is even, there is such a Hamiltonian cycle. □

**Exercise 7.2:** Give a deflection routing algorithm for any permutation on the $n \times n$ torus that delivers each packet in $O(d^2)$ steps, where $d$ is the distance from the packet's source to its destination.

## 7.2. An $O(n^{1.5})$ Time Algorithm

In this algorithm, each packet moves east and west in its source row from end to end. Whenever a packet reaches its destination column during a westbound pass, it attempts to turn either north or south, independent of its destination row. Packets already traveling in that column have priority for those links, so two such packets using the north and south links can prevent the turn. Once the packet successfully turns into its column, it moves from end to end until its destination is reached, which takes at most $2n$ additional steps. (This algorithm is nonminimal, and is adaptive since a packet's ability to turn into its column is dependent on other packets encountered.)

**Theorem 7.3:** Any permutation will be routed in $O(n^{1.5})$ steps by this algorithm.

**Proof:** The execution of the algorithm is partitioned into intervals of $4n$ steps each. For any fixed column $j$, let $N_i$ be the number of packets destined for column $j$ that have not yet been delivered at the beginning of the $i$th interval. In particular, $N_1 \leq n$.

Let $l$ be the number of packets in column $j$ at any time during the first $2n$ steps. $N_i - l$ packets are blocked from turning into column $j$ during these $2n$ steps, despite the fact that each tries once. A packet is blocked by some pair of the $l$ packets in column $j$, and each such pair blocks at most 2 packets from turning, since the pair meets at most twice before they are delivered. Hence,

$$\begin{aligned} 2\binom{l}{2} &\geq N_i - l, \\ l^2 - l &\geq N_i - l, \\ l &\geq \left\lceil \sqrt{N_i} \right\rceil \qquad \text{(since } l \text{ is an integer).} \end{aligned}$$

Since each of these $l$ packets is delivered before or during the last $2n$ steps of this interval,

$$N_{i+1} \leq N_i - \left\lceil \sqrt{N_i} \right\rceil .$$

**Claim:** Let $x$ be an integer. If $N_i \leq x^2$, then $N_{i+2} \leq (x-1)^2$.

**Proof:** Assume $N_{i+1} > (x-1)^2$, since otherwise the claim is proved. Then,

$$\begin{aligned} (x-1)^2 < N_{i+1} \leq N_i \leq x^2, \\ x - 1 < \sqrt{N_{i+1}} \leq \sqrt{N_i} \leq x, \\ \left\lceil \sqrt{N_{i+1}} \right\rceil = \left\lceil \sqrt{N_i} \right\rceil = x \qquad \text{(since } x \text{ is an integer).} \end{aligned}$$

It follows that $N_{i+1} \leq N_i - \lceil \sqrt{N_i} \rceil \leq x^2 - x$ and $N_{i+2} \leq N_{i+1} - \lceil \sqrt{N_{i+1}} \rceil \leq (x^2 - x) - x < (x-1)^2$, which proves the claim. □

Now the claim will be used to obtain a bound on the number of intervals in terms of $n$. As mentioned earlier, $N_1 \leq n \leq \lceil \sqrt{n} \rceil^2$. By induction on $k$, it follows from the claim that $N_{2k+1} \leq (\lceil \sqrt{n} \rceil - k)^2$, so that $N_{2\lceil \sqrt{n} \rceil + 1} \leq 0$. Hence, the total number of intervals is at most $2 \lceil \sqrt{n} \rceil$. Since each interval takes $4n$ steps, the total time of the algorithm is at most $8n \lceil \sqrt{n} \rceil \leq 8n^{1.5} + 8n$. □

**Exercise 7.4:** Find a permutation that causes this algorithm to use $\Omega(n^{1.5})$ steps.

## 7.3. An $n2^{O(\sqrt{\log n \log\log n})}$ Time Algorithm

The final algorithm for deflection permutation routing on the mesh achieves a time bound of $n2^{O(\sqrt{\log n \log\log n})}$. Notice that this is faster than $n^{1+\Omega(1)}$ but slower than $n\log^{O(1)} n$. Hence, this algorithm is fairly close to being linear in $n$. This lecture will finish with a description of the algorithm, and the analysis will follow in the next lecture.

This algorithm is a recursive version of the algorithm of Section 7.2. The mesh is partitioned into level 1 squares, each of which is partitioned into level 2 squares, etc., for $\beta$ levels ($\beta$ to be determined later).

For any packet $P$, let *$P$'s level $i$ square* be the level $i$ square that contains $P$'s destination. The goal of a packet at level $i$ is to get into its level $i+1$ square, assuming it is already in its level $i$ square.

To prevent packets from being deflected out of the level $i$ square after entry, do the following. Any row or column whose index mod $\beta$ equals $i$ will be said to be "colored" $i$. The rows and columns colored $i$ are dedicated to level $i$ packets, and all packets in level $i$ but not in level $i+1$ travel on rows and columns colored $i$.

The algorithm will concentrate first on *aligned packets*, which are those whose source row is colored 0 and whose destination column is colored $\beta - 1$. The routing of other packets will be described later.

Level 0 is the whole $n \times n$ mesh. The level 0 square is divided into $n^2/m^2$ level 1 squares, each of size $m \times m$ ($m$ to be determined later). Assume for simplicity that $m \mid n$ and $\beta \mid m$.

The goal of an aligned packet at level 0 is to get into a row colored 1 in its level 1 square. This process is divided into two stages.

1. *Column Turn Stage*: Let *$P$'s slice* be the $n \times m$ submesh containing its level 1 square. $P$ travels east and west in its source row from end to end. Whenever it reaches any of the $m/\beta$ columns colored 0 in its slice during a westbound pass, it attempts to turn either north or south, independent of its destination row. Any packets already traveling in those columns have priority for the north/south links.

2. *Row Turn Stage*: Once in its slice, $P$ travels north and south in its current column from end to end. Whenever it reaches any of the $m/\beta$ rows colored 1 in its level 1 square during a southbound pass, it attempts to turn either east or west, independent of its destination column. Any packets already traveling in those rows have priority for the east/west links.

Once $P$ is in its level 1 square, it continues recursively, using rows and columns colored 1, and the same algorithm restricted to its $m \times m$ level 1 square.

At level $\beta - 1$, the $O(n^{1.5})$ algorithm of Section 7.2 is used to complete the route. Note that that algorithm will keep $P$ on rows and columns colored $\beta - 1$, since an aligned packet must be destined for a column colored $\beta - 1$.

# Lecture 8

# Deflection Routing (Continued)

April 21, 1994
Notes: Donald Chinn

In this lecture, we analyze the running time of the algorithm described in Section 7.3, when restricted to the aligned packets only.

**Lemma 8.1:** For any permutation, the algorithm of Section 7.3 delivers all aligned packets in $n \cdot 2^{O(\sqrt{\log n \log\log n})}$ steps.

**Proof:** Let $T(n, \beta)$ be the time to deliver all aligned packets on the $n \times n$ mesh with $\beta$ levels. Fix any $n \times m$ slice $S$. At most $nm/\beta$ aligned packets are destined for $S$. We will show that they are all delivered in the required number of steps, and the lemma will follow from the fact that $S$ is an arbitrary slice.

Partition the algorithm's execution for the aligned packets destined for $S$ into phases: for $t \geq 1$, *phase* $t$ consists of steps in which the number of undelivered packets destined for $S$ is contained in the half open interval $(nm/(2^t\beta)\ ,\ nm/(2^{t-1}\beta)]$. Each phase is further divided into *subphases* of $4n - 4 + 2T(m, \beta - 1)$ steps each. Our goal is to bound the number of subphases.

Consider the first $2n-2$ steps of a subphase. Let $r$ be the number of aligned packets that travel in any of the columns colored 0 of $S$ at any time during these $2n-2$ steps. Call these *slice packets*.

Case 1: At least $nm/(2^{t+1}\beta)$ packets destined for $S$ fail to turn into $S$ in the first $2n-2$ steps. We will first argue, as in Theorem 7.3, that many packets must have turned into $S$ already to prevent so many others from turning. (Case 2, in which at least $nm/(2^{t+1}\beta)$ packets are in $S$ after step $2n-2$, will be simpler, since this first argument is unnecessary.)

Suppose some column colored 0 in $S$ has $l$ aligned packets, each on it sometime during the first $2n-2$ steps. Each pair of these packets can meet at most twice in $2n-2$ steps, and each time prevents at most one packet from turning. There are at least $nm/(2^{t+1}\beta)$ packets that have not turned, and each one will attempt to turn into this column once in the $2n-2$ steps. Thus,

$$\begin{aligned} 2\binom{l}{2} &\geq \frac{nm}{2^{t+1}\beta} \\ l^2 - l &\geq \frac{nm}{2^{t+1}\beta} \\ l &\geq \sqrt{\frac{nm}{2^{t+1}\beta}} \end{aligned}$$

Since the same argument holds for each of the $m/\beta$ columns colored 0 in $S$, then $r \geq (m/\beta)^{1.5}\sqrt{n/2^{t+1}}$.

We now argue the same way for these $r$ slice packets during the row turn stage of the algorithm, using the next $2n-2$ steps of the subphase.

For $1 \leq i \leq n/m$, let $d_i$ be the number of slice packets destined for the $i$th level 1 square of slice $S$, and let $r_i$ be the number of aligned packets that travel in any of the rows colored 1 of this square at any time during the first $4n-4$ steps of the subphase. The goal now is to obtain a lower bound on $\sum r_i$. Such a bound will yield an upper bound on the number of subphases in phase $t$, since these $\sum r_i$ packets will be delivered by the end of the subphase.

CASE 1.1: $r_i \leq d_i/2$, i.e., at least $d_i/2$ packets fail to turn into a row colored 1 during the first $4n-4$ steps. Suppose some row colored 1 in the $i$th level 1 square has $l$ aligned packets, each on it sometime during these $4n-4$ steps. Each pair of these packets can meet at most $(4n-4)/(m-1)$ times during the $4n-4$ steps, since the square is $m$ nodes wide. Each of at least $d_i/2$ packets tries to turn into this row twice during these steps. Then using the same reasoning as above,

$$\begin{aligned} \frac{4n-4}{m-1}\binom{l}{2} &\geq 2 \cdot \frac{d_i}{2} \\ l^2 - l &\geq \frac{d_i(m-1)}{2n-2} \geq \frac{d_i m}{4n} \\ l &\geq \sqrt{\frac{d_i m}{4n}} \end{aligned}$$

Since this is true for each of the $m/\beta$ rows colored 1 in this square, $r_i \geq (m^{1.5}/(2\beta))\sqrt{d_i/n}$.

CASE 1.2: $r_i \geq d_i/2$.

In either case, $r_i \geq \min\left\{(m^{1.5}/(2\beta))\sqrt{d_i/n}\ ,\ d_i/2\right\}$.

Recall that we want to obtain a lower bound on $\sum r_i$, and we are subject to the following constraints:

1. $d_i \leq m^2/\beta$, since the square has only that many nodes in columns colored $\beta - 1$, and
2. $\sum_{i=1}^{n/m} d_i = r \geq (m/\beta)^{1.5}\sqrt{n/2^{t+1}}$.

The minimum value of $\sum r_i$ is achieved when $\sqrt{n/(m\beta 2^{t+1})}$ of the $d_i$'s are $m^2/\beta$ and the rest are zero, because $\min\left\{(m^{1.5}/(2\beta))\sqrt{x/n}\ ,\ x/2\right\}$ is a concave downward function in $x$: if $0 < d_j \leq d_i < m^2/\beta$, then $r_i + r_j$ cannot increase if $d_i$ is increased by $\Delta$ and $d_j$ is decreased by $\Delta$.

For $d_i = m^2/\beta$, we have $d_i/2 \geq (m^{1.5}/(2\beta))\sqrt{d_i/n}$ (since we can assume that $n \geq 2m$), so

$$\sum_{i=1}^{n/m} r_i \geq \sqrt{\frac{n}{m\beta 2^{t+1}}} \cdot \frac{m^{2.5}}{2\beta^{1.5}\sqrt{n}} = \frac{m^2}{\beta^2 \cdot 2^{0.5t+1.5}}. \tag{8.1}$$

Since $\sum r_i$ packets are in their level 1 squares after $4n-4$ steps and $2T(m, \beta-1)$ steps remain in the subphase, all of them are delivered by the end of the subphase. (The extra factor of 2 is for skipping over rows and columns colored 0; a factor of $\beta/(\beta-1)$ would be more accurate.)

The number of subphases in phase $t$ is at most the number of packets that need to be delivered before phase $t+1$ begins, divided by the number of packets delivered per subphase, which is at most

$$\frac{nm}{2^t\beta} \div \frac{m^2}{\beta^2 \cdot 2^{0.5t+1.5}} = \frac{n\beta}{m \cdot 2^{0.5t-1.5}}. \tag{8.2}$$

The number of steps to deliver all aligned packets is the sum over all phases of the number of subphases in that phase, times the number of steps per subphase:

$$\begin{aligned} T(n,\beta) &< \sum_{t=1}^{\infty} \frac{n\beta}{m \cdot 2^{0.5t-1.5}} \left(4n + 2T(m,\beta-1)\right) \\ &= \frac{4n^2\beta + 2n\beta T(m,\beta-1)}{m} \cdot \frac{2\sqrt{2}}{\sqrt{2}-1} \\ &< \frac{28n^2\beta}{m} + \frac{14n\beta}{m} \cdot T(m,\beta-1) \end{aligned} \tag{8.3}$$

Before we solve this recurrence relation, we must look at Case 2.

Case 2: At least $nm/(2^{t+1}\beta)$ packets are in $S$ after step $2n-2$. In the worst case, all these packets are not yet in their level 1 squares, i.e., $r \geq nm/(2^{t+1}\beta)$. The analysis is similar to Case 1, with differences as follows:

- The minimum value of $\sum r_i$ is achieved when $n/(m2^{t+1})$ of the $d_i$'s are $m^2/\beta$ and the rest are zero.
- The right hand side of Equation 8.1 (the sum of the $r_i$'s) is

$$\frac{m^{1.5}\sqrt{n}}{\beta^{1.5} \cdot 2^{t+2}}.$$

- The right hand side of Equation 8.2 (the number of subphases in phase $t$) is

$$4\sqrt{\frac{n\beta}{m}}.$$

- The right hand side of Equation 8.3 (the recurrence relation) is

$$4\sqrt{\frac{n\beta}{m}} \cdot \left(4n + 2T(m,\beta-1)\right)\left(\log_2 \frac{nm}{\beta}\right).$$

**Solving the Recurrence Relation.** In either case, for sufficiently large $n$ and provided $m = o(n/\log^2 n)$ (which will turn out to be the case),

$$T(n,\beta) \leq \frac{28n^2\beta}{m} + \frac{14n\beta}{m} \cdot T(m,\beta-1). \tag{8.4}$$

To solve this recurrence relation asymptotically, choose $m$ so that $T(m, \beta - 1) = n$, and let $C_\beta$ be defined by $T(n, \beta) = C_\beta \cdot n^{1+1/(\beta+1)}$. That is, we guess that $T(n, \beta)$ is of this form, and then we will verify that the guess is correct.

Note that $n = T(m, \beta - 1) = C_{\beta-1} \cdot m^{1+1/\beta} = C_{\beta-1} \cdot m^{(\beta+1)/\beta}$, so that

$$m = \left(\frac{n}{C_{\beta-1}}\right)^{\frac{\beta}{\beta+1}} = \left(\frac{n}{C_{\beta-1}}\right)^{1-1/(\beta+1)}.$$

Thus,

$$\begin{aligned} C_\beta \cdot n^{1+1/(\beta+1)} = T(n,\beta) \le \frac{42n^2\beta}{m} &= 42n^2\beta\left(\frac{C_{\beta-1}}{n}\right)^{1-1/(\beta+1)} \\ &\le 42\beta C_{\beta-1} \cdot n^{1+1/(\beta+1)}. \end{aligned}$$

That is, $C_\beta \le 42\beta C_{\beta-1}$. This has solution $C_\beta \le 42^\beta \beta!$, noting for the basis that $C_1 \le 42$ (where $C_1$ is the constant in Theorem 7.3). Since $C_\beta$ is independent of $n$, we have verified our guess of the form of $T(n, \beta)$.

Now choose

$$\beta = \sqrt{\frac{\log n}{\log\log n}}$$

to balance $C_\beta$ and $n^{1/(\beta+1)}$ asymptotically. This gives us

$$\begin{aligned} T(n,\beta) &= C_\beta \cdot n^{1+1/(\beta+1)} \\ &= n \cdot 2^{\beta\log_2\beta + O(\beta)} \cdot 2^{(\log_2 n)/(\beta+1)} \\ &= n \cdot 2^{O(\sqrt{\log n \log\log n})}. \end{aligned}$$

Note that $m < n/C_{\beta-1} = o(n/\log^2 n)$, as promised. □

We still need to show how to route the unaligned packets, and we will do that in the next lecture.

# Lecture 9

# Deflection Routing (Continued)

April 26, 1994
Notes: Ruben Ortega

In this lecture, we conclude the algorithm begun in Section 7.3 by describing the routing and analyzing the running time of the remaining unaligned packets.

An unaligned packet for the most part meanders around, getting deflected by aligned packets until it is its turn to get aligned. It will turn out that the unaligned packets do not interfere with the analysis from Lemma 8.1 of the aligned packets, since unaligned packets will be routed on any free link in deference to aligned packets, which have higher priority.

## 9.1. Algorithm for Unaligned Packets

Assign $\beta^2$ different priorities to the packets according to the colors of the source row and destination column. A packet with destination column color $d$ moves from end to end in its current column, trying to turn into any row colored $(d+1) \bmod \beta$ whenever it is in a southbound pass. From there it uses the algorithm of Section 7.3, with the change that at level $i$ it uses rows and columns colored $(i+d+1) \bmod \beta$. (This puts it on the correct row and column color at level $\beta - 1$, namely $d$.)

Call a packet *aligned* if it is executing the algorithm of Section 7.3, and *unaligned* if it is trying to turn into its level 0 rows.

If two packets of different priority contend for a link, the one of lower priority is deflected and starts over as an unaligned packet. If packets of equal priority contend for a link, the unaligned ones, if any, are deflected. Finally, if unaligned packets of equal priority contend for a column link, the one (if any) that entered the node via a column link has priority. If an unaligned packet is deflected onto a row link, it tries to turn into any column to resume its attempt to get to a level 0 row.

A *priority packet* is an undelivered packet of highest priority among all undelivered packets.

This algorithm has the nice property that there is no global control, which might be used, for instance, to keep lower priority packets from being injected until all priority packets are delivered. Instead, each packet optimistically proceeds as if it was a priority packet.

Note that there are initially $n^2/\beta^2$ packets of each priority.

**Lemma 9.1:** Suppose at time $T$ that there are $p$ priority packets, where $0 < p \leq n^2/\beta^2$. Then after $n \cdot 2^{O(\sqrt{\log n \log\log n})}$ additional steps, at least $p/(4\beta+8)$ of them are delivered.

**Proof:** Let $d$ be the destination column color of the priority packets. Let $r$ be the number of aligned priority packets at time $T + 2n - 2$. For some row $i$ colored $(d + 1)$ mod $\beta$, at most $r\beta/n$ priority packets travel along row $i$ sometime during the interval $[T + 1, T + 2n - 2]$ of $2n - 2$ steps, because they cannot become unaligned and change rows once aligned.

How many unaligned priority packets try to turn into row $i$ during these $2n - 2$ steps? Any unaligned priority packet that is never deflected from its column during these steps will try to turn into row $i$, so we will bound the number of such packets. First notice that, at any step in $[T + 1, T + 2n - 2]$, at most $p/2$ unaligned priority packets travel on row links, since otherwise some node would route more unaligned priority packets on row links than total priority packets on column links, which contradicts the fact that the unaligned packets are trying to turn into any column. Thus, at any step in $[T + 1, T + 2n - 2]$, at least $p/2 - r$ unaligned priority packets travel on column links. Each of the at most $r$ aligned priority packets $P$ can deflect at most $\beta$ of these $p/2 - r$ unaligned packets from their columns, once for each time $P$ succeeds in a column turn stage. Thus, at least $p/2 - (\beta + 1)r$ unaligned priority packets each try to turn into row $i$ once during $[T + 1, T + 2n - 2]$, and at least $p/2 - (\beta + 2)r$ of them fail.

As in Lemmas 7.3 and 8.1, every pair of aligned priority packets on row $i$ can prevent at most two such turns during these $2n - 2$ steps, so

$$
\begin{aligned}
2\binom{r\beta/n}{2} &\geq \frac{p}{2} - (\beta+2)r \\
\frac{r^2\beta^2}{n^2} &\geq \frac{p}{2} - (\beta+2)r \\
r^2 + \frac{(\beta+2)n^2}{\beta^2}r - \frac{pn^2}{2\beta^2} &\geq 0 \\
r &\geq -\frac{(\beta+2)n^2}{2\beta^2} + \sqrt{\frac{(\beta+2)^2n^4}{4\beta^4} + \frac{pn^2}{2\beta^2}} \\
&= \frac{(\beta+2)n^2}{2\beta^2}\left(\sqrt{1 + \frac{2p\beta^2}{(\beta+2)^2n^2}} - 1\right) \\
&\geq \frac{(\beta+2)n^2}{2\beta^2}\left(\left(1 + \frac{p\beta^2}{2(\beta+2)^2n^2}\right) - 1\right) \qquad \text{(since } 0 < p \leq n^2/\beta^2\text{)} \\
&= \frac{p}{4(\beta+2)}.
\end{aligned}
$$

By Lemma 8.1, these $r$ aligned priority packets will all be delivered within $n \cdot 2^{O(\sqrt{\log n \log\log n})}$ additional steps, as they cannot be deflected by lower priority or unaligned packets. □

**Theorem 9.2:** For any permutation, this algorithm delivers all packets in $n \cdot 2^{O(\sqrt{\log n \log\log n})}$ steps.

**Proof:** By Lemma 9.1, all priority packets are delivered in

$$n \cdot 2^{O(\sqrt{\log n \log\log n})} \frac{\ln\left(\frac{n^2}{\beta^2}\right)}{\ln\left(\frac{4\beta+8}{4\beta+7}\right)} \leq n \cdot 2^{O(\sqrt{\log n \log\log n})}(2(4\beta+8)\ln n)$$

steps, using the identity $\ln\left(\frac{1}{1-x}\right) \geq x$. Thus all packets are delivered in

$$\beta^2 n \cdot 2^{O(\sqrt{\log n \log\log n})}(8\beta+16)\ln n = n \cdot 2^{O(\sqrt{\log n \log\log n})} \tag{9.1}$$

steps, recalling from the proof of Lemma 8.1 that $\beta = \sqrt{\frac{\log n}{\log\log n}}$. □

By making the dependence on the parameter $\beta$ explicit, we can show that there is a spectrum of deflection routing algorithms, generalizing Theorem 9.2:

**Theorem 9.3:** For $\beta \geq 2$, there is an algorithm that delivers all packets in any permutation on the $n \times n$ mesh in time

$$\beta! 2^{O(\beta)} n^{1+\frac{1}{\beta+1}} \log n.$$

Larger values of $\beta$ lower the exponent of $n$ but drastically increase its coefficient.

## 9.2. Is This Algorithm Practical?

In contrast to the $8n^{1.5}$ time algorithm of Section 7.2, this one may not be as practical. The big-Oh in the running time hides $\log n$ factors and large constants. The logic for the routing nodes must calculate alignment, priority, and distance to destination.

Is the algorithm practical for $\beta = 2$? This is the choice for $\beta$ in any case as long as $n \leq 2^{16}$ (that is, a $65,536 \times 65,536$ mesh). By choosing $m = (n/2)^{2/3}$ and using Equations (8.4) and (9.1), a crude upper bound on the number of steps is

$$(28 \cdot 2 \cdot 2^{2/3} + 14 \cdot 2 \cdot 2^{2/3} \cdot 8((1/2)^{2/3})^{3/2}) \cdot 4 \cdot 32 \cdot n^{4/3} \ln n \approx 34,135 n^{4/3} \ln n.$$

**Exercise 9.4:** Make the $O(n^{4/3})$ algorithm more practical by tightening the analysis or the algorithm itself. In particular, derive a better bound on the constant and eliminate the $\ln n$ factor.

**Exercise 9.5:** Under certain conditions, a node cannot tell whether one of its packets is aligned or unaligned, unless packets carry this information explicitly. Determine what those conditions are, and whether the analysis still stands if the node simply assumes such a packet is aligned.

# Lecture 10

# Deflection Worm Routing on the Torus

April 28, 1994
Notes: Brendan Mumey

## 10.1. Worm Routing

Bar-Noy *et al.* [1] also discuss worm routing on the $n \times n$ torus. The authors use the term "worm" instead of the common alternative "wormhole", and we will do the same. In this model each message (or "worm") consists of $k$ "flits", where only one flit can cross a link per step, and each flit must follow the one in front of it, maintaining a distance of at most one link between them. We consider the problem of deflection routing for permutations. If queues are allowed worms may possibly compress and decompress, but in the deflection case consecutive flits must remain exactly one link apart.

We can have at most $\frac{4n^2}{k}$ worms in the torus at any given time, since there are only $4n^2$ links. This means that some method is needed for controlling worm injection. (One new danger in this case, in addition to the usual danger of livelock, is that four worms simultaneously converge on a node when that node is in the process of injecting a new worm.) As there may be $n^2$ worms and each may have to travel $\Omega(n)$ distance, $\Omega(kn)$ is a lower bound on the time to route arbitrary permutations.

The *head* of a worm is its first flit. We adopt a *head-to-head discipline*, in which two worms contend for a link only at their heads.

## 10.2. How To Turn a Deflection Packet Router into a Deflection Worm Router

The first deflection worm routing method uses a novel reduction of worm routing to packet routing. The version in this section is due to Melanie Fulgham, who generalized a proof of Bar-Noy *et al.* [1] so that it would work for any packet router.

**Theorem 10.1:** For any deflection packet router that routes arbitrary permutations on the $n \times n$ torus in time $T(n)$, there is a deflection worm router that does so in time $O\left(k^3 T\left(\frac{n}{\sqrt{k}}\right)\right)$.

**Proof:** Assume for simplicity that there is an integer $h$ such that

1. $h = O(\sqrt{k})$,
2. $h \geq 2\sqrt{k}$, and
3. $h \mid n$.

Partition the torus into $\frac{n^2}{h^2}$ squares each of size $h \times h$, and color the nodes of each square distinctly with colors $1, 2, \ldots, h^2$ in row-major order.

There are $h^4$ rounds. In round $(i, j)$, where $1 \leq i, j \leq h^2$, inject all worms with source node colored $i$ and destination node colored $j$. For this $(i, j)$, choose a set of *column paths* and a set of *row paths* satisfying the following properties:

1. Each pair of nodes colored $j$ that are at distance $h$ apart in some column (row) are connected by a column path (row path) in the torus of length exactly $k$.
2. These paths are pairwise edge-disjoint.

One way of accomplishing this is to divide each $h \times h$ square into four quadrants and grant exclusive meandering rights in one of the quadrants to one of the row paths and in another quadrant to one of the column paths. We leave the details to the reader. We will assume that every column path has the same "shape", in the sense that each is a translation of every other one, and similarly for the row paths.

Round $(i, j)$ begins by routing every worm head from its source node to the closest node on a column or row path in the same square, and from there along the column or row path to the node colored $j$ in the same square. By assuming again that each of these initial routes within its source square is a translation of every other one, it will be the case that every worm head reaches the node colored $j$ at the same time. We then simulate the deflection packet router on the $\frac{n}{h} \times \frac{n}{h}$ torus of nodes colored $j$, with column and row paths simulating the column and row links. Note that this obeys the head-to-head discipline, since routing decisions are only made every $k$ steps, when worm heads arrive at nodes colored $j$. If $T(n)$ is the time used by the packet router, this simulation uses time $O(h^4 k T(\frac{n}{h})) = O\left(k^3 T\left(\frac{n}{\sqrt{k}}\right)\right)$.

□

**Corollary 10.2:** There are deflection worm routers that route arbitrary permutations on the $n \times n$ torus in time $O(k^{2.25} n^{1.5})$ and $k^{2.5} n \cdot 2^{O(\sqrt{\log n \log\log n})}$, respectively.

**Proof:** Apply Theorem 10.1 to the deflection packet routers of Theorems 7.3 and 9.2. □

**Open Problem 10.3:** Design an efficient worm routing algorithm on the torus that will handle worms of varying lengths.

# Lecture 11

# Deflection Worm Routing (Continued)

May 3, 1994
Notes: Sushil Aryal

## 11.1. An $O(kn^{1.5})$ Time Deflection Worm Router

In this lecture, an $O(kn^{1.5})$ algorithm for deflection routing of length $k$ worms on the $n \times n$ torus is presented. This improves one of the algorithms of Corollary 10.2, and is again due to Bar-Noy *et al.* [1].

It is assumed for simplicity that $k \mid n$. A *diagonal* of the torus is a set of nodes on a northwest/southeast line. One of the diagonals is distinguished as the *main diagonal*. An *eastbound (westbound) sliding diagonal* is a diagonal that moves one node east (respectively, west) at each step.

For each $c$ satisfying $0 \leq c \leq n/k - 1$, the *eastbound (westbound) sliding diagonal of color c* is the one that leaves the main diagonal at times that are equivalent to $ck \bmod n$.

A worm routing algorithm is said to be a *diagonal preserving algorithm* if and only if it satisfies the following two properties:

1. The head of a worm $W$ can only be injected at a node $v$ using $v$'s east or north link (west or south link) when one of the colored eastbound (respectively, westbound) sliding diagonals contains $v$. The worm $W$ has *color c* if its head is injected on a sliding diagonal of color $c$.
2. A worm $W$ moving east or north (west or south) turns west or south (respectively, east or north) only at the main diagonal. That is, worms never change color once they are injected.

These two conditions guarantee the following properties of the algorithm:

1. The routing obeys the head-to-head discipline, since the colored diagonals are $k$ nodes apart. (It is permissible for worms to intersect each other as long as they are not contending for the same links.)
2. Worms contend for a link only if they are of the same color, since the two sliding diagonals that meet at the main diagonal have the same color.

We can now describe the $O(kn^{1.5})$ algorithm, which is a particular diagonal preserving algorithm:

1. A node with an uninjected worm $W$ will inject $W$ onto its east link (on an eastbound colored sliding diagonal) or its south link (on a westbound colored sliding diagonal) the first time there is no contention for that link.

2. Whenever an eastbound worm reaches its destination column, it tries to turn north. Only a northbound worm already in the column (which has a higher priority for that north link) can prevent it from doing so. Once a worm is moving north in its destination column, it continues to its destination.

3. Whenever a southbound worm reaches its destination row, it tries to turn west. Only a westbound worm already in the row (which has a higher priority for that west link) can prevent it from doing so. Once a worm is moving west in its destination row, it continues to its destination.

4. Whenever an eastbound (southbound) worm reaches the main diagonal, it turns south (respectively, east).

**Lemma 11.1:** For any $T$, $\Delta$, $p$, and $c$ with $0 \leq c < n/k$, suppose that there are $p$ worms such that, for each one, there is a time during the interval $[T+1, T+\Delta]$ when it has color $c$. Then more than $2\sqrt{p} - 2$ worms of color $c$ are delivered during $[T+1, T+\Delta+3n]$.

**Proof:** Let $N$ (respectively, $W$) be the number of worms colored c that are moving north (respectively, west) sometime during $[T+1, T+\Delta+2n]$. Then $p - N - W$ worms colored $c$ try and fail to turn both north and west during $[T+1, T+\Delta+2n]$.

Each pair of worms, one moving north on column $j$ and one moving west on row $i$, prevents at most one worm from its two turns, namely the worm with destination $(i, j)$. (Recall that worms of color other than $c$ cannot prevent either of these two turns.) Therefore $NW \geq p - N - W$, which implies that $NW + N + W \geq p$.

Suppose that $N + W \leq 2\sqrt{p} - 2$. Then $NW \leq (\sqrt{p} - 1)^2$, so $NW + N + W \leq (p - 2\sqrt{p} + 1) + (2\sqrt{p} - 2) = p - 1$. Since this contradicts the earlier condition $NW + N + W \geq p$, it must be the case that $N + W > 2\sqrt{p} - 2$.

All of these $N + W$ worms are delivered within another $n$ steps, that is by time $T + \Delta + 3n$, as claimed. □

**Theorem 11.2:** The algorithm described above routes any permutation of length $k$ worms on the $n \times n$ torus in $O(kn^{1.5})$ time.

**Proof:** The proof is divided into two parts. In Part A, it is shown that the time until all but 15 worms are injected is $O(kn^{1.5})$. In Part B, it is proved that the additional time until the delivery of all the remaining worms (including the 15 uninjected worms from Part A) is $O(n^{1.5})$.

We start with the proof of Part A. Suppose that $2^{m+1} \leq n < 2^{m+2}$. Divide the execution of the algorithm into $m$ phases. For $1 \leq t \leq m$, phase $t$ consists of the steps in which the number of uninjected worms lies in the half-open interval $(n^2/4^t, n^2/4^{t-1}]$.

Let $[T+1, T+n]$ be an interval contained in phase $t$. Since the number of uninjected worms is greater than $n^2/4^t$, the number of rows plus columns containing uninjected worms is greater than twice the side of the smallest square containing $n^2/4^t$ nodes, which is $n/2^{t-1}$.

Consider the two sliding diagonals of color $c$ during $[T+1, T+n]$. More than $n/2^{t-1}$ worms have color $c$ during $[T+1, T+n]$, because the eastbound (westbound) sliding diagonal of color $c$ must have as many worms as there are source rows (respectively, columns) of uninjected worms.

By Lemma 11.1, then, more than $2\sqrt{n/2^{t-1}} - 2$ worms of color $c$ are delivered during $[T+1, T+4n]$. Among all colors, the number delivered in these $4n$ steps is greater than

$$\frac{n}{k}\left(2\sqrt{\frac{n}{2^{t-1}}} - 2\right) \quad \geq \quad \frac{n}{k}\sqrt{\frac{n}{2^{t-1}}},$$

since $n \geq 2^{m+1} \geq 2^{t+1}$.

The time to end phase $t$ is at most the time to deliver $3n^2/4^t$ worms, which is at most

$$\frac{3n^2}{4^t} \div \frac{\frac{n}{k}\sqrt{\frac{n}{2^{t-1}}}}{4n} = \frac{3kn^{1.5}}{\left(2\sqrt{2}\right)^{t-1}}.$$

The time for all $m$ phases is thus at most

$$\begin{aligned} 3kn^{1.5}\sum_{t=1}^{\infty}\frac{1}{\left(2\sqrt{2}\right)^{t-1}} &= \frac{24+6\sqrt{2}}{7}kn^{1.5} \\ &< 5kn^{1.5}. \end{aligned}$$

At the end of phase $m$ the number of uninjected worms is at most $n^2/4^m < 4^{m+2}/4^m = 16$.

This concludes the proof of Part A of the theorem.

# Lecture 12

# Deflection Routing on Arbitrary Networks

May 5, 1994
Notes: Sung-Eun Choi

## 12.1. Conclusion of Theorem 11.2

For Part B of the proof of Theorem 11.2, divide the remaining time into intervals of $3n$ steps each. For any color $c$, let $N_i$ be the number of undelivered worms of color $c$ at the beginning of the $i$th interval. Note that $N_1 \leq 4n$.

By Lemma 11.1,

$$N_{i+1} < N_i - 2\sqrt{N_i} + 2,$$

so

$$N_{i+1} \leq N_i - \lfloor 2\sqrt{N_i} \rfloor + 1,$$

since $N_i$ and $N_{i+1}$ are integers.

**Claim:** For any integer $x$, if $N_i \leq x^2$, then $N_{i+1} \leq (x-1)^2$.

**Proof:** Assume that $N_i > (x-1)^2$, since otherwise the claim is proved.

Case 1: $\sqrt{N_i} = x$. Then

$$\begin{aligned} N_{i+1} &\leq N_i - \lfloor 2\sqrt{N_i} \rfloor + 1 \\ &= x^2 - 2x + 1 \\ &= (x-1)^2. \end{aligned}$$

Case 2: $x - \frac{1}{2} \leq \sqrt{N_i} < x$. Then $\lfloor 2\sqrt{N_i} \rfloor = 2x - 1$, so

$$\begin{aligned} N_{i+1} &< x^2 - (2x-1) + 1 \\ &= x^2 - 2x + 2, \\ \text{so } N_{i+1} &\leq (x-1)^2. \end{aligned}$$

Case 3: $x - 1 < \sqrt{N_i} < x - \frac{1}{2}$. Then $\lfloor 2\sqrt{N_i} \rfloor = 2x - 2$, so

$$N_{i+1} < (x^2 - x + \frac{1}{4}) - (2x-2) + 1$$

$$\begin{aligned} &= x^2 - 3x + 3.25, \\ \text{so } N_{i+1} &\leq x^2 - 3x + 3 \\ &\leq (x-1)^2 \qquad \text{(since } x \geq 2 \text{ in Case 3).} \end{aligned}$$

□

From the Claim, the number of intervals is at most $\lceil \sqrt{N_1} \rceil \leq \lceil 2\sqrt{n} \rceil$. Thus, the total Part B time is at most $\lceil 2\sqrt{n} \rceil 3n \leq 6n^{1.5} + 3n$.

The 15 uninjected packets from Part A add another $O(n)$ steps for injection and delivery. □

**Exercise 12.1:** Find a permutation of worms showing that Theorem 11.2 is tight. Try starting with the case $k = 1$.

**Exercise 12.2:** What would happen if worms were injected west instead of south and tried to turn south into their destination columns?

**Open Problem 12.3:** Extend these deflection algorithms to the dynamic case.

## 12.2. Other Results on Deflection Routing on Meshes and Tori

Bar-Noy *et al.* [1] also present a probabilistic, diagonal preserving, deflection worm router that routes any permutation in $O(kn)$ time with probability $1 - O(\frac{1}{n})$.

Returning to deflection *packet* routing, Newman and Schuster [17] present a $7n + o(n)$ time deflection packet routing algorithm for any permutation. Although this improves the results of Theorems 7.3 and 9.2, it is based on sorting. (See Section 6.2 for a discussion of drawbacks of routers based on sorting.)

Ben-Dor, Halevi, and Schuster [2] present an $O(n\sqrt{k})$ time deflection packet routing algorithm that routes any $k$ packets on the $n \times n$ mesh in a static (but not necessarily permutation) problem. We will improve on this result in Theorem 14.4. We will consider this same static setting, parameterized by the number $k$ of packets, in the next few sections, but on other networks.

## 12.3. Deflection Packet Routing on General Networks

Consider an arbitrary network with indegree($v$) = outdegree($v$) for all nodes $v$, so that deflection routing makes sense. Consider an arbitrary static routing problem of $k$ packets (not necessarily a partial permutation), where the number of packets with source $v$ is at most outdegree($v$). The results in this section are due to Hajek [8].

Define *one-pass deflection routing* as follows. At each node and each step, there is some ordering of the packets, and a packet $P$ is deflected if and only if the outgoing links on all the shortest paths to $P$'s destination are assigned to packets earlier in the ordering.

Such routing algorithms can produce a livelock situation. In Figure 12.1, $P_i$ denotes a packet destined for node $i$. Livelock can occur if $P_1$ and $P_4$ have highest priority in the configuration depicted on the left and are chosen to move along the links indicated. The resulting configuration is shown on the right. Now if $P_2$ and $P_3$ have the highest priority and are chosen to move along the links indicated, the network will return to the first configuration.

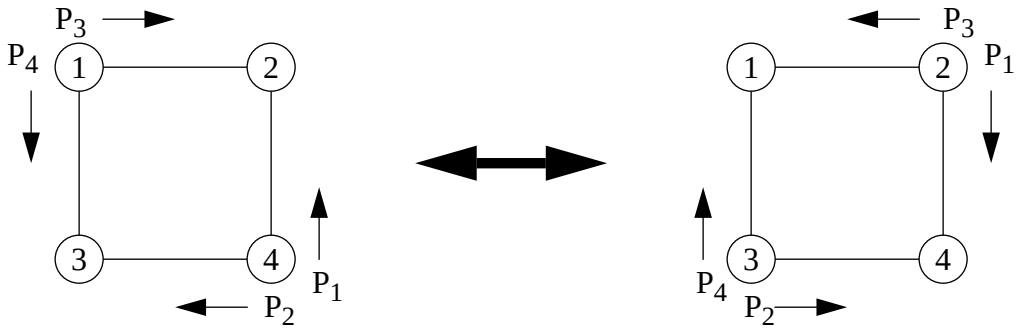



Figure 12.1: Example of a possible livelock situation.

To prevent livelock, define *nearest-first deflection routing* to be a subclass of one-pass deflection routing in which packets are considered in nondecreasing order of distance to destination.

**Theorem 12.4:** Suppose there are $k$ packets to be routed. If any nearest-first deflection routing algorithm is used, all packets are delivered in at most $\delta k$ steps, where $\delta$ is the diameter of the network.

**Proof:** At any step, consider all packets closest to their destinations. One of those is first in its node's ordering, so it will not be deflected. Thus, the minimum distance to destination over all undelivered packets decreases by one each step until some packet is delivered, which must happen in $\delta$ steps. The result follows by repeating this argument for each of the remaining $k - 1$ packets. □

**Open Problem 12.5:** Can you do better than Theorem 12.4 on an arbitrary network, by using a scheme in which a packet about to be deflected can be swapped with another, provided this decreases the number of deflections?

**Open Problem 12.6:** The nearest-first ordering based on distance to destination may introduce too much delay to be practical. What happens if a random ordering is used instead of the nearest-first ordering?

# Lecture 13

# Deflection Routing on the Hypercube

May 10, 1994
Notes: Jim Fix

**Definition 13.1:** The *n-dimensional hypercube* $H_n$ consists of the vertex set $\{0,1\}^n$ and the edge set

$$\{(u,v) \mid (\exists i)(u_i \neq v_i \;\&\; (\forall j \neq i)(u_j = v_j))\}.$$

That is, two vertices are adjacent if and only if they differ in exactly one position. (See Figure 13.1 for an example.) Note that the number of vertices is $2^n$, the indegree and outdegree of each vertex is $n$, and the diameter $\delta$ is $n$.

For the remainder of this section, we consider any nearest-first deflection routing of $k$ packets on the $n$-dimensional hypercube $H_n$. Our goal is to improve Theorem 12.4 to show that deflection routing on $H_n$ can be performed in $O(\delta + k) = O(n + k)$ steps. This result is due to Hajek [8]. (See Theorem 14.4 for a closely related result.)

**Definition 13.2:** Suppose packet $P$ has destination $w$ and is about to traverse the link from $u$ to $v$. $P$'s *conditional distance* is $1 + \text{dist}(v,w)$, where $\text{dist}(v,w)$ is the length of a shortest path from $v$ to $w$.

On the hypercube, the conditional distance is either $\text{dist}(u,w) + 2$ or $\text{dist}(u,w)$, depending on whether or not $P$ is deflected.

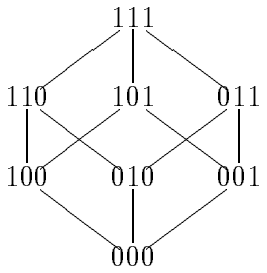



Figure 13.1: The 3-dimensional hypercube.

**Lemma 13.3:** Suppose a node $v$ of the hypercube has at least $r$ undelivered packets, each with distance at most $j$ to its destination, where $r \leq j-1$. If any nearest-first deflection algorithm is used, then at least $r$ packets will depart $v$ with conditional distance at most $j$.

**Proof:** Let the order in which packets at $v$ are considered be $P_1, P_2, \ldots, P_r, \ldots$. Let $1 \leq i \leq r$, and let $w$ be $P_i$'s destination. Let $d = \text{dist}(v, w)$. Since $i \leq r$ and packets are considered in nondecreasing order of distance to destination, $d \leq j$. Note that $d$ links from $v$ are each on shortest paths from $v$ to $w$, since $v$ and $w$ differ in exactly $d$ positions.

CASE 1: $j-1 \leq d \leq j$. The packets $P_1, P_2, \ldots, P_{i-1}$ use at most $i-1 \leq r-1 \leq j-2$ links from $v$, so $P_i$ will not be deflected and its conditional distance is $d \leq j$.

CASE 2: $1 \leq d \leq j-2$. Whether or not $P_i$ is deflected, its conditional distance is at most $d+2 \leq j$.

In either case, the conditional distance is no more than $j$. □

In order to understand the routing, we next introduce a graph that captures the trajectories of the $k$ packets on the hypercube $H_n$.

**Definition 13.4:** The *time-expanded graph* $G$ has vertices $(t, v)$, where $t \geq 0$ and $v$ is a vertex of $H_n$, and an edge $e$ from $(s, u)$ to $(t, v)$ if and only if $t = s+1$ and some packet $P$ traverses the link from $u$ to $v$ at time $s$. The *label* on $e$ is the conditional distance of $P$ at time $s$.

**Definition 13.5:** The *traces of* $G$ are the directed paths $T_1, T_2, \ldots, T_k$ in $G$ defined as follows. Suppose $T_1, T_2, \ldots, T_{m-1}$ have been constructed, where $1 \leq m \leq k$. $T_m$ begins at any vertex $(0, v)$ that has an outgoing edge that is not in $T_1, T_2, \ldots, T_{m-1}$. Repeatedly add to $T_m$ the minimum labeled edge directed out of $T_m$'s head (the last vertex added) that is not in $T_1, T_2, \ldots, T_{m-1}$. Stop when $T_m$ reaches a vertex $(t, v)$ such that the number of traces entering $(t, v)$, including the trace $T_m$, is at most the number of packets with destination $v$ that were delivered at time $t$. (That is, if $p$ packets had destination $v$ and were delivered at time $t$, then the first $p$ traces to reach $(t, v)$ will stop there, and any later traces will continue.) Let length$(T_m)$ be the number of edges in $T_m$.

**Lemma 13.6:** If length$(T_m) \geq t$, the label of the $t$-th edge of $T_m$ is at most $n + 2m - t - 1$.

**Proof:** The proof is by induction on $m$ and $t$.

BASIS : Suppose $t = 1$ and $1 \leq m \leq k$.

CASE 1: $m = 1$. Then for any node $v$, the conditional distance of some packet at node $v$ is at most $n = n + 2m - t - 1$, since the first packet considered at $v$ is not deflected.

CASE 2: $m \geq 2$. The conditional distance of any packet at any node is at most $n + 1 \leq n + 2m - 2 = n + 2m - t - 1$.

INDUCTION : Suppose the lemma holds for all $m' < m$ and all $1 \leq t \leq \text{length}(T_{m'})$, and also for $m' = m$ and all $t' \leq t$. We want to show that the lemma holds for the $(t+1)$-th edge of $T_m$. Let $(t, v)$ be the vertex at the tail of that edge.

Let

$$M = \{T_i \mid 1 \leq i \leq m \text{ and the } (t+1)\text{-th edge of } T_i \text{ is directed out of } (t, v)\}.$$

Note that $T_m \in M$. By the induction hypothesis, the $t$-th edge of any $T_i \in M$ has label at most $n + 2i - t - 1$. That means that there are at least $|M|$ packets at $v$ at time $t$, each with distance between 1 and $j = n + 2m - t - 2$ to its destination.

Applying the induction hypothesis again and the fact that all edge labels are positive, for any $1 \leq i < m$ we have $n + 2i - \text{length}(T_i) - 1 \geq 1$, so $\text{length}(T_i) \leq n + 2i - 2$. For $T_i \in M$, $\text{length}(T_i) \geq t+1$, so $M \subseteq \{T_i \mid (t-n+3)/2 \leq i \leq m\}$. Therefore $|M| \leq m-(t-n+1)/2 = (j+1)/2$.

Case 1: $j = 1$. Then $|M| \leq (j+1)/2 = 1$, so $M = \{T_m\}$. At least $|M| = 1$ packet at $v$ at time $t$ is at a distance between 1 and $j = 1$ from its destination. By the nearest-first discipline, some packet with distance 1 to its destination will not be deflected, so has conditional distance 1. Therefore, the label on the $(t+1)$-th edge of $T_m$ is $1 = j = n + 2m - (t+1) - 1$.

Case 2: $j \geq 2$. Then $|M| \leq (j+1)/2$, so $|M| \leq (j-1)$. Applying Lemma 13.3 with $r = |M|$, at least $|M|$ packets depart $v$ at time $t$ with conditional distance at most $j$. Hence, there are at least $|M|$ edges directed from $(t, v)$, each with label at most $j$. In particular, the $(t+1)$-th edge of $T_m$ has label at most $j = n + 2m - (t+1) - 1$. □

**Theorem 13.7:** Suppose $k$ packets are to be routed on an $n$-dimensional hypercube, with at most $n$ packets having any single node as source. If any nearest-first deflection router is used, all $k$ packets are delivered within $n + 2(k-1)$ steps.

**Proof:** By Lemma 13.6 and the fact that all edge labels are positive, for all $m$ satisfying $1 \leq m \leq k$ we have $n + 2m - \text{length}(T_m) - 1 \geq 1$. Therefore,

$$\begin{aligned} \text{length}(T_m) &\leq n + 2(m-1) \qquad (13.1) \\ &\leq n + 2(k-1). \end{aligned}$$

Since the termination of each trace corresponds to the delivery of a distinct packet, all $k$ packets are delivered within $n + 2(k-1)$ steps. □

It is informative to compare the proof of Theorem 13.7 with that of Theorem 12.4. Each one shows that the first packet is delivered within the first $\delta$ steps. Whereas Theorem 12.4 goes on to show that each successive packet is delivered within an additional $\delta$ steps, the specialization to hypercubes in Theorem 13.7 shows that each successive packet is delivered within an additional 2 steps. (See Inequality (13.1).)

**Exercise 13.8:** When applied to permutation routing, Theorem 13.7 only yields an $O(2^n)$ time bound for deflection routing on $H_n$. Find a deflection routing algorithm that improves this bound.

**Open Problem 13.9:** Find a practical deflection routing algorithm that routes any permutation on $H_n$ in $O(2^{n/2})$ steps. (Newman and Schuster [17] have an $O(n^2)$ bound for a deflection router based on sorting. See Section 6.2 for a discussion of the drawbacks of routing algorithms based on sorting.)

# Lecture 14

# Deflection Routing on Multidimensional Meshes

May 31, 1994
Notes: Donald Chinn

In this lecture, we conclude our discussion of deflection packet routing with a recent result of Borodin, Rabani, and Schieber [5]. The goal of their work is to prove a bound of the following form: any packet $P$ with distance $d_P$ from its source to its destination is delivered in time $d_P + 2(k-1)$, where $k$ is the number of packets in the network. Recall that Hajek's result [8] on deflection packet routing on the $n$-dimensional hypercube (Theorem 13.7) is of the same type, with a bound of $n + 2(k-1)$ steps.

Borodin, Rabani, and Schieber achieve their bound for any multidimensional mesh, including hypercubes. They employ a novel accounting scheme by which each deflection of $P$ can be charged to a unique packet. This charging scheme is not specific to meshes, and is the topic of Section 14.1.

## 14.1. Deflection Sequences and Paths

**Definition 14.1:** For any network, any deflection router, and any packet $P$, a *deflection sequence* for $P$ is a sequence $(P_0 = P, P_1, P_2, \ldots, P_l)$ of packets such that, for $0 \leq i < l$, $P_i$ and $P_{i+1}$ meet at node $v_i$ at some time $T_i$, where $T_0 < T_1 < \cdots < T_{l-1}$. Let $v_l$ be the destination of $P_l$. Define the *deflection path* corresponding to this sequence as the path from $v_0$ to $v_l$ that follows the route of $P_i$ from $v_{i-1}$ to $v_i$, for $1 \leq i \leq l$.

**Lemma 14.2:** Suppose $P$ is at node $v$ at time $T_0$, moves to node $w$ at time $T_0 + 1$, and $S = (P_0, P_1, \ldots, P_l)$ is a deflection sequence for $P$ starting at $v$ at time $T_0$. Let $v_l$ be the destination of packet $P_l$. If $\mathrm{dist}(w, v_l)$ is at least the length of the deflection path corresponding to $S$, then $P_l$ cannot be the last packet in any deflection sequence for $P$ that starts at a time after $T_0$.

**Proof:** Suppose $P_l$ is also the last packet in a deflection sequence $S'$ for $P$ that starts at a node $v'$ at a time after $T_0$. Let $T_l$ be the time at which $P_l$ is delivered to its destination $v_l$. Consider the path $\pi$ from $w$ to $v_l$ that follows the route of $P$ to $v'$ and then follows the deflection path corresponding to $S'$ from $v'$ to $v_l$.

The length of $\pi$ is $T_l - (T_0 + 1)$. This is less than the length $T_l - T_0$ of the deflection path corresponding to $S$, contradicting the assumption in the lemma. □

**Definition 14.3:** A *bidirectional* network is one in which the existence of an edge $(u, v)$ implies the existence of the edge $(v, u)$.

The consequence of Lemma 14.2 is that, if $P$ is deflected at time $T_0$, that deflection can be "charged" to $P_l$, and costs $P$ two steps if the network is bidirectional. If such a deflection sequence can be found for each deflection of $P$, then $P$ will be delivered in at most $d_P + 2(k-1)$ steps. We will show how to find such deflection sequences for multidimensional meshes in Section 14.2.

## 14.2. Deflection Routing on Multidimensional Meshes

**Theorem 14.4:** For any $d$ and any $d$-dimensional mesh $G$ with arbitrary dimensions, consider all routing problems on $G$ with $k$ packets, at most one per source node. There is a deflection routing algorithm for all such problems on $G$ that delivers each packet $P$ in time at most $d_P + 2(k-1)$.

**Proof:** The proof is a generalization of the $O(n^{1.5})$ time deflection routing algorithm of Bar-Noy *et al.* [1], described in Section 7.2.

Order the dimensions $1, 2, \ldots, d$ arbitrarily. A packet $P$ traveling along dimension $i$ inductively has corrected its address positions in dimensions $1, 2, \ldots, i-1$ so that they agree with $P$'s destination address. $P$ is trying to turn into the dimension $i+1$ row at its destination value of dimension $i$, preferably toward its destination but, if not, in the other direction. Packets already in that row have priority. If $P$ is moving away from its destination in dimension $i$, it tries to reverse direction at each step, but packets going toward their destinations in dimension $i$ have priority.

The theorem follows from Lemma 14.2, if the appropriate deflection sequence can be identified. When $P$ is deflected from reversing direction in its row, or from turning, it meets some packet $P_1$ moving toward its destination. (Note that $P_1$ may not be the packet using the link that $P$ wanted, since that packet itself might have been deflected from turning. In fact, $P_1$ may be several dimensions beyond $P$.) Similarly, follow $P_1$ until it is either delivered to its destination, or deflected and meets $P_2$ moving toward its destination, etc. Note that, for $i \geq 1$, if $P_i$ is deflected, this occurs while trying to turn into the next dimension it needs to correct. Thus, $P_{i+1}$ is moving along a higher dimension than $P_i$, so that this sequence of packets terminates. This is the deflection sequence $(P_0, P_1, \ldots, P_l)$.

Since the corresponding deflection path is a shortest path from $v$ to the destination of $P_l$, and $P$ is deflected away from the destination of $P_l$, Lemma 14.2 applies. Thus, each deflection of $P$ can be charged to a distinct packet, and each adds 2 to $P$'s delivery time. □

Note that the scheduling of packets to outgoing links at each node can be implemented using one pass over the incoming links in decreasing order of dimension. This scheduling involves very simple decisions, possibly more practical than Hajek's nearest-first scheme [8], which involves sorting the incoming packets according to their distance to destination.

Borodin, Rabani, and Schieber [5] also give a $d_P + 2(k-1)$ algorithm in the "fully loaded" case, where each node begins with a number of packets that can be as great as the outdegree of the node.

**Open Problem 14.5:** Devise an $O(d_P + k)$ deflection algorithm for arbitrary networks.

**Exercise 14.6:** What is the running time of the algorithm of Theorem 14.4 when routing permutations on $d$-dimensional meshes? Show that it is $\omega(\delta)$, where $\delta$ is the diameter of the mesh, when $d \geq 3$. (Hint: Use the method of Theorem 15.7.) Is this also true when $d = 2$?

# Lecture 15

# A Lower Bound for Oblivious Routing

May 12, 1994
Notes: M.L. Fulgham

With this lecture, we return from deflection routing to permutation routing with queueing, on the hypercube. We begin with a result on oblivious routing due to Kaklamanis, Krizanc, and Tsantilas [9]. This work strengthens and is based on earlier work of Borodin and Hopcroft [4].

**Definition 15.1:** A routing algorithm is *oblivious* if and only if the route of each packet is determined solely by its source and destination.

In particular, the route cannot depend on other packets encountered, congestion, or the state of the network.

**Example 15.2:** The following "greedy algorithm" on the hypercube $H_n$ is oblivious: correct address positions in which the source and destination addresses differ in increasing order. Use unbounded queues, and any of various queueing disciplines, such as FIFO, farthest-first, or nearest-first.

**Example 15.3:** Although at first blush the notions of "oblivious" and "deflection" seem irreconcilable, there is in fact an oblivious deflection routing algorithm on any network that has a Hamiltonian cycle; see Theorem 7.1.

We will prove a very general lower bound for any deterministic, oblivious routing algorithm on any network, even if it uses unbounded queues.

In what follows, let $G$ be a network with $N$ nodes, each with indegree at most $d$. Assume that $G$ is *strongly connected* (i.e., for every ordered pair $(u, v)$ of nodes, there is a directed path in $G$ from $u$ to $v$), since otherwise packets cannot possibly get from all sources to all destinations.

**Definition 15.4:** Let $t$ be a node of $G$. The *destination graph* $G_t$ is the union, over all $N$ nodes $s$, of the path specified by the oblivious router for a packet with source $s$ and destination $t$.

**Definition 15.5:** An edge $e$ in the destination graph $G_t$ is *$k$-congested* if and only if it belongs to at least $k$ paths with destination $t$, one for each of $k$ source nodes $s$. Let $S_k$ be the set of $k$-congested edges in $G_t$. Let $V(S_k)$ be the set of nodes $v$ such that some edge in $S_k$ is directed into or out of $v$.

Lemma 15.6 bounds the number of nodes that are not in $V(S_k)$ in terms of the number that are.

**Lemma 15.6:** Suppose that $k \leq (N-1)/d$. Then for any destination graph $G_t$,

$$N - |V(S_k)| \leq (k-1)\, d\, |V(S_k)|.$$

**Proof:** There are $N-1$ source nodes other than $t$, each of which uses one of the at most $d$ edges directed into $t$. This guarantees that one of these edges is at least $(N-1)/d$-congested and hence also $k$-congested. Therefore, $t \in V(S_k)$.

For any node $s$ not in $V(S_k)$, consider the path taken by a packet with source $s$ and destination $t$. Let $v$ be the first node on this path that belongs to $V(S_k)$. (Such a node exists since $t \in V(S_k)$.) At most $(k-1)d$ source nodes $s$ can have $v$ as a first such node, since any edge $(u,v)$ used by more than $k-1$ sources is $k$-congested, making $u$ an earlier node for those sources that belongs to $V(S_k)$. Thus, this mapping from nodes not in $V(S_k)$ to nodes in $V(S_k)$ is at most $(k-1)d$–to–1. □

**Theorem 15.7:** In any $N$-node network $G$ with maximum indegree $d$, any deterministic, oblivious routing algorithm requires time $\Omega\left(\sqrt{N}/d\right)$ to route some permutation.

**Proof:** Let $k = \sqrt{N/2}/d \leq (N-1)/d$ (since we can assume $N \geq 2$). Let $G_t$ be any destination graph of $G$. Then

$$\begin{aligned} N &= |V(S_k)| + (N - |V(S_k)|) \\ &\leq |V(S_k)| + (k-1)\, d\, |V(S_k)| && \text{(Lemma 15.6)} \\ &\leq 2|S_k|(1 + (k-1)d) && (|V(S_k)| \leq 2|S_k|) \\ &\leq 2kd|S_k| && (d \geq 1). \end{aligned}$$

Solving for $|S_k|$ shows that each of the $N$ destination graphs has at least $N/(2kd)$ $k$-congested edges. Let the *weight* of an edge $e$ be the number of destination graphs in which $e$ is $k$-congested. The total weight of all edges is then at least $N^2/(2kd)$. Thus, there is some edge $e$ with weight at least

$$\frac{N^2/(2kd)}{Nd} = \frac{N}{2kd^2} = \frac{\sqrt{N/2}}{d}.$$

That is, $e$ is $\sqrt{N/2}/d$-congested in at least $\sqrt{N/2}/d$ destination graphs.

Construct a permutation in which $\sqrt{N/2}/d$ packets must cross $e$, by choosing a distinct source $s$ from each of these $\sqrt{N/2}/d$ destination graphs $G_t$, and injecting a packet at $s$ with destination $t$. Since only one packet can cross $e$ per step, the time to route this permutation is at least $\sqrt{N/2}/d$. □

Note that only a small partial permutation with these $\sqrt{N/2}/d$ packets is necessary to cause this much delay.

On the $n \times n$ mesh or torus, Theorem 15.7 only implies $\Omega(n/4)$ time, which is no better than the obvious lower bound that follows from the fact that the diameter of the network is $\Omega(n)$. However, on the hypercube (and many other small diameter networks), the lower bound is more dramatic:

**Corollary 15.8:** Any deterministic, oblivious packet routing algorithm on the $n$-dimensional hypercube requires time $\Omega\left(\frac{2^{n/2}}{n}\right)$ to route some permutation.

Note that this is much greater than the diameter $n$ of the hypercube.

# Lecture 16

# Greedy Routing on the Hypercube

May 19, 1994
Notes: Tracy Kimbrel

## 16.1. Greedy Static Permutation Routing

In this section, we analyze the greedy routing algorithm on the hypercube, as described in Example 15.2. Recall that the the greedy algorithm corrects the address positions in which the source and destination addresses differ, in increasing order. Note that the hypercube can be viewed as an $n$-dimensional mesh with side length 2 in each dimension. Thus the greedy algorithm of Example 15.2 is the natural extension of the greedy algorithm on the mesh of Section 1.5: it routes first along the first dimension, then the second, and so on.

**Definition 16.1:** If $b \in \{0,1\}$, let $\overline{b} = 1 - b$. If $v = v_1 \cdots v_n \in \{0,1\}^n$, let $v^{(j)} = v_1 \cdots v_{j-1} \overline{v_j} v_{j+1} \cdots v_n$.

**Lemma 16.2:** In the hypercube $H_n$, consider the edge $e$ from node $v = v_1 \cdots v_j \cdots v_n \in \{0,1\}^n$ to node $v^{(j)} = v_1 \cdots \overline{v_j} \cdots v_n$. When the greedy routing algorithm is used, the only packets that contend for $e$ are those with

1. source addresses $u_1 \cdots u_{j-1} v_j \cdots v_n$, for some $u_1, \ldots, u_{j-1} \in \{0,1\}$, and
2. destination addresses $v_1 \cdots v_{j-1} \overline{v_j} u_{j+1} \cdots u_n$, for some $u_{j+1}, \ldots, u_n \in \{0,1\}$.

**Proof:** Address positions are corrected in increasing order. □

**Theorem 16.3:** Consider the greedy routing algorithm with unbounded queues, and any queueing discipline (as long as it never refuses to use a free link). This algorithm routes any permutation on the $n$-dimensional hypercube $H_n$ in $O(2^{n/2})$ steps.

**Proof:** Assume for simplicity that $n$ is even. Consider a packet $P$ that wants to cross the link $e$ from $v$ to $v^{(j)}$. By Lemma 16.2, at most $N_j = \min(2^{j-1}, 2^{n-j})$ packets contend for $e$. Because all queues are unbounded, $P$ will take at most $N_j$ steps to cross $e$ once it reaches $v$. Thus the total time $P$ takes to reach its destination is at most

$$\begin{aligned}\sum_{j=1}^{n} N_j &\leq \sum_{j=1}^{n/2} 2^{j-1} + \sum_{j=n/2+1}^{n} 2^{n-j} \\ &= 2\sum_{j=0}^{n/2-1} 2^j \\ &< 2^{n/2+1}.\end{aligned}$$

□

**Exercise 16.4:** Let $n$ be even, and consider the "transpose" permutation on the $n$-dimensional hypercube, in which every node $u_1 \cdots u_n$ is the source of a packet with destination $u_{n/2+1} \cdots u_n u_1 \cdots u_{n/2}$. Show that the greedy routing algorithm with unbounded queues and any queueing discipline requires $\Omega(2^{n/2})$ steps to route the transpose permutation. What is the maximum number of packets in any queue at one time during the routing?

Kaklamanis, Krizanc, and Tsantilas [9] give an oblivious algorithm that routes any permutation in $O(2^{n/2}/n)$ steps, proving that Theorem 15.7 is tight.

**Open Problem 16.5:** Find a deterministic algorithm that routes any permutation on $H_n$ in $o(2^{n/2}/n)$ steps. You may use unbounded queues, but do not base it on sorting. (By using sorting, permutations can be routed in $O(n \log n)$ time: see Cypher and Plaxton [7] and Leighton [11, Section 3.4.3]. See Section 6.2 for a discussion of the drawbacks of routing algorithms based on sorting.)

Contrast Open Problem 16.5 with Theorem 2.1, which showed that time proportional to the diameter can be achieved on the mesh.

**Exercise 16.6:** Find a deterministic algorithm that routes any permutation on $H_n$ in $o(2^n)$ steps. Use bounded queues, and do not base it on sorting.

**Open Problem 16.7:** Find a deterministic algorithm that routes any permutation on $H_n$ in $O(2^{n/2})$ steps. Use bounded queues, and do not base it on sorting.

## 16.2. Information Dispersal Routing on the Hypercube

The next topic is an algorithm due to Rabin [18] and based on earlier work of Valiant and Brebner [21]. Valiant and Brebner's algorithm is the following: for each packet, choose an intermediate node $\rho$ at random, and use the greedy route from the source to $\rho$, followed by the greedy route from $\rho$ to the destination. Valiant and Brebner prove that, with high probability, the number of

steps to route any permutation on the hypercube $H_n$ is $O(n)$. (See Lecture 19 for details. This same paradigm was discussed in Section 5.2 for the mesh.)

The description of Rabin's information dispersal routing will follow Leighton [11, Section 3.4.8]. In information dispersal, each packet is split into $n$ subpackets which are routed along edge-disjoint paths via a randomly chosen intermediate node. Advantages of this approach include

- balanced communication loads, low congestion, and small queues, and
- fault-tolerance, if subpackets encode the original packet redundantly. (We will return to the algorithm's fault-tolerance after presenting and analyzing the basic information dispersal algorithm.)

A disadvantage is that the number of items (subpackets) to be routed is increased, each containing its own routing information. The original messages must be long (compared to the lengths of node addresses) to make this worthwhile.

In the information dispersal model, the algorithm execution is composed of "supersteps". In the $j$th superstep of Phases 1 and 2 below, each node $v$ sends only those packets that want to go to node $v^{(j)}$. In one superstep,

- each node can split a packet into $n$ subpackets, or combine those subpackets into a single packet,
- each node $v$ can select up to $6n$ of its subpackets that want to go to $v^{(j)}$, and
- up to $6n$ subpackets can traverse any link.

Because subpackets have size about $1/n$ times the original packet size, a superstep takes between $\Theta(1)$ and $\Theta(n)$ ordinary steps. There will be $\Theta(n)$ supersteps, so the total time is between $\Theta(n)$ and $\Theta(n^2)$. (Compare to Theorem 16.3 at $O(2^{n/2})$.)

The algorithm is composed of four phases:

**Phase 0:** Each node $v$ starts with a packet $P(v)$ that has destination $\delta(v)$. The node $v$ chooses a random, uniformly distributed intermediate destination $\rho(v)$ for $P(v)$. It partitions $P(v)$ into $n$ subpackets $P_1(v), P_2(v), \ldots, P_n(v)$, and sends $P_i(v)$ to $v^{(i)}$. This is the first superstep.

**Phase 1:** In the next $n$ supersteps, $P_i(v)$ is routed from $v^{(i)}$ to $\rho(v)^{(i)}$ along the greedy route. At the $j$th of these $n$ supersteps, $P_i(v)$ does not move if $v_j = (\rho(v))_j$.

**Phase 2:** In the next $n$ supersteps, $P_i(v)$ is routed from $\rho(v)^{(i)}$ to $\delta(v)^{(i)}$ as in Phase 1.

**Phase 3:** In the last superstep, $P_i(v)$ is sent from $\delta(v)^{(i)}$ to $\delta(v)$, and the $n$ subpackets are combined to reconstruct $P(v)$.

This algorithm runs in $2n + 2$ supersteps, provided only $6n$ subpackets contend for any link at any time. We will prove in the next lecture that this occurs with very high probability.

# Lecture 17

# Information Dispersal on the Hypercube

May 24, 1994
Notes: Darren C. Cronquist

## 17.1. Contention for Links in Information Dispersal

**Theorem 17.1:** With probability $1 - O(2^{-n})$, at most $6n$ subpackets traverse any link in any step of the information dispersal algorithm on $H_n$.

**Proof:** Consider an edge $f = (w, w^{(j)})$ during Phase 1. By Lemma 16.2, $P_i(v)$ crosses $f$ during Phase 1 only if one of the following conditions holds:

1. $i = j$, and $v$ has suffix $\overline{w_j} w_{j+1} \cdots w_n$ and $\rho(v)$ has prefix $w_1 w_2 \cdots w_j$, or
2. $i > j$, and $v$ has suffix $w_j \cdots w_{i-1} \overline{w_i} w_{i+1} \cdots w_n$ and $\rho(v)$ has prefix $w_1 \cdots w_{j-1} \overline{w_j}$, or
3. $i < j$, and $v$ has suffix $w_j \cdots w_n$ and $\rho(v)$ has prefix $w_1 \cdots w_{i-1} \overline{w_i} w_{i+1} \cdots w_{j-1} \overline{w_j}$.

Hence, for each individual source node $v$, there is at most one $i$ such that $P_i(v)$ crosses $f$ during Phase 1, since $i$ is uniquely determined by the position where the suffixes of $v$ and $w$ disagree and the position where the prefixes of $\rho(v)$ and $w^{(j)}$ disagree. (Restated, this says that all $n$ subpackets of any packet follow paths that are edge disjoint.)

We now examine the contention for $f$ among all source nodes, each of which contributes at most one subpacket to that contention. The source nodes are categorized into three types, depending on the index of the subpacket contributed.

TYPE 1 ($i = j$): At most $2^{j-1}$ sources $v$ have suffix $\overline{w_j} w_{j+1} \cdots w_n$. For each one,

$$\Pr(P_j(v) \text{ crosses } f \text{ during Phase 1}) = 2^{-j},$$

since the randomly chosen node $\rho(v)$ must have prefix $w_1 w_2 \cdots w_j$.

TYPE 2 ($i > j$): At most $(n - j)2^{j-1}$ sources $v$ have suffix $w_j \cdots w_{i-1} \overline{w_i} w_{i+1} \cdots w_n$ for some $j < i \leq n$. For each one,

$$\Pr(P_i(v) \text{ crosses } f \text{ during Phase 1}) = 2^{-j},$$

since $\rho(v)$ must have prefix $w_1 \cdots w_{j-1} \overline{w_j}$.

TYPE 3 ($i < j$): At most $2^{j-1}$ sources have suffix $w_j \cdots w_n$. For each one,

$$\Pr(P_i(v) \text{ crosses } f \text{ during Phase 1, for some } 1 \le i < j) = (j-1)2^{-j},$$

since $\rho(v)$ must have prefix $w_1 \cdots w_{i-1}\overline{w_i}w_{i+1} \cdots w_{j-1}\overline{w_j}$, for some $1 \le i < j$.

Combining Types 1 and 2 yields the following inequality:

$$\begin{aligned}
&\Pr(\text{at least } r \text{ subpackets each with } i \ge j \text{ cross } f \text{ during Phase 1}) \\
&\quad \le \binom{(n-j+1)2^{j-1}}{r} \left(2^{-j}\right)^r \\
&\quad \le \left(\frac{en2^{j-1}}{r}\right)^r \left(2^{-j}\right)^r \qquad \text{(Stirling's approximation)} \\
&\quad = \left(\frac{en}{2r}\right)^r.
\end{aligned}$$

Let $r = en$. Then

$$\Pr(\text{at least } en \text{ subpackets each with } i \ge j \text{ cross } f \text{ during Phase 1}) \le 2^{-en}.$$

Similarly, for sources of Type 3,

$$\begin{aligned}
&\Pr(\text{at least } r \text{ subpackets each with } i < j \text{ cross } f \text{ during Phase 1}) \\
&\quad \le \binom{2^{j-1}}{r} \left((j-1)2^{-j}\right)^r \\
&\quad \le \left(\frac{e2^{j-1}}{r}\right)^r \left(n2^{-j}\right)^r \qquad \text{(Stirling's approximation)} \\
&\quad = \left(\frac{en}{2r}\right)^r.
\end{aligned}$$

Let $r = en$. Then,

$$\Pr(\text{at least } en \text{ subpackets each with } i < j \text{ cross } f \text{ during Phase 1}) \le 2^{-en}.$$

Combining all three types,

$$\begin{aligned}
&\Pr(\text{at least } 2en \text{ subpackets cross } f \text{ during Phase 1}) \\
&\quad \le \Pr(\text{at least } en \text{ subpackets each with } i \ge j \text{ cross } f \text{ during Phase 1}) \\
&\qquad + \Pr(\text{at least } en \text{ subpackets each with } i < j \text{ cross } f \text{ during Phase 1}) \\
&\quad \le 2^{1-en}.
\end{aligned}$$

Since the hypercube $H_n$ contains $n \cdot 2^n$ links,

$$\Pr(\text{at least } 2en \text{ subpackets cross any link during Phase 1}) \le n \cdot 2^{n+1-en}.$$

Through an analogous derivation, it can be shown that

$$\Pr(\text{at least } 2en \text{ subpackets cross any link during Phase 2}) \leq n \cdot 2^{n+1-en}.$$

Hence,

$$\Pr(\text{at least } 2en \text{ subpackets cross any link during Phase 1 or Phase 2}) \leq n \cdot 2^{n+2-en} = O(2^{-n}),$$

from which the theorem follows. □

**Corollary 17.2:** With probability $1 - O(2^{-n})$, the information dispersal algorithm on $H_n$ routes any permutation in $2n + 2$ supersteps.

## 17.2. Fault Tolerance for Information Dispersal

In all the algorithms we have discussed so far, there has been an implicit assumption that components of the network are not faulty. In reality, as networks grow in size this assumption is less likely to be sound. The chief benefit of the information dispersal algorithm is that it provides a certain independence among the subpackets, and this can be used to deliver messages even in the presence of network faults.

Assume for now that each packet $P(v)$ of size $B$ can be divided into $n$ subpackets $P_1(v), P_2(v), \cdots, P_n(v)$, each of size approximately $2B/n$, such that $P(v)$ can be reconstructed from any $n/2$ of these $n$ subpackets. We will return to the question of how to achieve this redundant decomposition after discussing how it is used to achieve fault tolerance in the information dispersal algorithm.

Assume each node and link fails randomly and independently with probability $\sigma \leq \frac{1}{c(n+1)}$, where $c$ is a constant to be determined. A subpacket trying to use such a failed node or link is lost. The goal is thus to show that, with high probability, half of each packet's subpackets are delivered successfully.

# Lecture 18

# Information Dispersal (Continued)

May 26, 1994
Notes: Ruben E. Ortega

## 18.1. Fault Tolerance

The following lemma ensures that a single node's failure cannot affect too many subpackets of a given packet.

**Lemma 18.1:** For any nodes $x$, $y$, and $w$, at most two of the $n$ values of $i$ have greedy routes from $x^{(i)}$ to $y^{(i)}$ that pass through $w$.

**Proof:** By way of contradiction, suppose three distinct values $i_1, i_2, i_3$ of $i$ have greedy routes that pass through $w$. For $1 \leq k \leq 3$, let $j_k$ be the value such that the greedy route from $x^{(i_k)}$ to $y^{(i_k)}$ leaves $w$ on edge $(w, w^{(j_k)})$. Without loss of generality, assume $j_1 \leq j_2 \leq j_3$. By Lemma 16.2, for $1 \leq k \leq 3$,

1. $x^{(i_k)}$ has suffix $w_{j_k} \cdots w_n$, and
2. $y^{(i_k)}$ has prefix $w_1 \cdots w_{j_k - 1} \overline{w_{j_k}}$.

Since $j_1 \leq j_2$,

$$\begin{aligned} x &= (* \cdots * \, w_{j_1} \cdots w_{j_2} \cdots w_n)^{(i_1)} \\ &= (* \cdots\cdots\cdots * \, w_{j_2} \cdots w_n)^{(i_2)} \ . \end{aligned}$$

It follows that $i_2 < j_2$, since $i_1 \neq i_2$.

Since $j_2 \leq j_3$,

$$\begin{aligned} y &= (w_1 \cdots w_{j_2 - 1} \overline{w_{j_2}} * \cdots\cdots\cdots\cdots\cdots *)^{(i_2)} \\ &= (w_1 \cdots w_{j_2 - 1} w_{j_2} \cdots w_{j_3 - 1} \overline{w_{j_3}} * \cdots *)^{(i_3)}. \end{aligned}$$

It follows that $i_2 \geq j_2$, since $i_2 \neq i_3$. This contradicts the earlier conclusion that $i_2 < j_2$, so the assumption of three routes passing through $w$ is false. □

**Theorem 18.2:** Suppose nodes and links fail randomly and independently with probability $\sigma \leq \frac{1}{c(n+1)}$, where $c = 2^{21}e$. (But see Exercise 18.3 below.) Then with probability at least $1 - 2^{-n}$, every packet for which $v$ and $\delta(v)$ do not fail has at least $n/2$ subpackets successfully delivered to $\delta(v)$.

**Proof:** Each of the $n$ subpacket paths from $v$ to $\delta(v)$ uses at most $2n + 2$ links, and at most $2n + 1$ nodes other than $v$ and $\delta(v)$. Thus,

$$\begin{aligned}
&\Pr(\text{at least } n/8 \text{ nodes and links used by subpackets of } P(v) \text{ fail}) \\
&\leq \binom{n(4n+3)}{n/8} \sigma^{n/8} \\
&\leq \left(\frac{4en(n+1)}{n/8}\right)^{n/8} \left(\frac{1}{c(n+1)}\right)^{n/8} \qquad \text{(Stirling's approximation)} \\
&= \left(\frac{32e}{c}\right)^{n/8} \\
&= \left(2^{-16}\right)^{n/8} \\
&= 2^{-2n}.
\end{aligned}$$

By Lemma 18.1, each node that fails loses at most two of $P(v)$'s subpackets in each of Phases 1 and 2. From the proof of Theorem 17.1, the subpackets of $P(v)$ follow edge-disjoint paths, so each link that fails loses at most one of $P(v)$'s subpackets in each of Phases 1 and 2. Thus,

$$\Pr(\text{at least } n/2 \text{ subpackets of } P(v) \text{ are lost}) \leq 2^{-2n},$$

so

$$\Pr(\text{there is a node } v \text{ for which at least } n/2 \text{ subpackets of } P(v) \text{ are lost}) \leq 2^n \cdot 2^{-2n} = 2^{-n}.$$

□

**Exercise 18.3:** Improve the value of $c$ to be more practical. According to Leighton [11, page 608], "it is possible to show that a 1024-node hypercube can withstand dozens of random faults without destroying too many of the subpacket paths for any packet".

## 18.2. Coding Theory, and the Redundant Encoding of Packets

**Theorem 18.4:** Suppose each packet $P$ has $B$ bits of data, excluding the routing information, where $B \geq \frac{1}{2}n \log_2 n$. Then $P$ can be divided into $n$ subpackets $P_0, P_1, \ldots, P_{n-1}$, each of $2B/n$ bits, such that $P$ can be recovered from any $n/2$ of the $n$ subpackets.

**Proof:** Let $s = 2B/n \geq \log_2 n$. Partition the $B$ bits of $P$ into $n/2$ blocks $Q_0, Q_1, \ldots, Q_{n/2-1}$ of $s$ bits each. All operations below are performed over GF($2^s$), the finite field of $2^s$ elements.

Let $x_0, x_1, \ldots, x_{n-1}$ be any $n$ distinct elements of GF($2^s$), which exist since $s \geq \log_2 n$. Let

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n/2-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n/2-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n/2-1} \end{pmatrix}.$$

The subpackets are computed as

$$\begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{n-1} \end{pmatrix} = A \begin{pmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_{n/2-1} \end{pmatrix}. \tag{18.1}$$

Given any $n/2$ subpackets $P_{i_0}, P_{i_1}, \ldots, P_{i_{n/2-1}}$, let

$$P' = \begin{pmatrix} P_{i_0} \\ P_{i_1} \\ \vdots \\ P_{i_{n/2-1}} \end{pmatrix} = V \begin{pmatrix} Q_0 \\ Q_1 \\ \vdots \\ Q_{n/2-1} \end{pmatrix},$$

where $V$ is composed of rows $i_0, i_1, \ldots, i_{n/2-1}$ of $A$. $V$ is a Vandermonde matrix over distinct elements, and is invertible over GF($2^s$) (Lipson [15]). Thus $Q_0, Q_1, \ldots, Q_{n/2-1}$ (and hence $P$) can be recovered by computing the product $V^{-1}P'$. □

The computation of the $n$ subpackets can be done in $O(n \log n)$ operations over GF($2^s$), by choosing a primitive $n$th root of unity $\omega$ and letting $x_i = \omega^i$. Then Equation (18.1) is equivalent to the evaluation of the polynomial $\sum_{i=0}^{n/2-1} Q_i x^i$ at the $n$ values $x_0, x_1, \ldots, x_{n-1}$, which can be accomplished using the fast Fourier transform (Lipson [15]).

Similarly, $V^{-1}P'$ can be computed in $O(n \log n)$ operations over GF($2^s$), since it is equivalent to interpolation of a polynomial through $n/2$ points (Lipson [15]).

# Lecture 19

# Probabilistic Routing on the Hypercube

June 2, 1994
Notes: Jim Fix

In this lecture, we return to the problem of permutation routing with unbounded queues on the hypercube $H_n$. We present the "classical" result of Valiant and Brebner [21], following the exposition of Motwani and Raghavan [16]. (See Section 16.2 for the closely related information dispersal algorithm on the hypercube, and Section 5.2 for a related algorithm on the two-dimensional mesh.)

For each node $v$, suppose that there is exactly one packet $P(v)$ with source $v$, and that its destination is $\delta(v)$. (Valiant and Brebner [21] show how to extend this result to the case of a partial permutation.) Each node $v$ randomly and independently chooses an intermediate node $\rho(v)$. Routing proceeds in two phases:

**Phase 1:** Route $P(v)$ on the greedy route from $v$ to $\rho(v)$. $P(v)$ waits at $\rho(v)$ until time $7n/2$.

**Phase 2:** Route $P(v)$ on the greedy route from $\rho(v)$ to $\delta(v)$.

Every node has a separate unbounded queue for each outgoing link. The exact queueing discipline is immaterial, as long as a link never goes unused when some packets are queued to use it.

How many steps is a packet delayed in Phase 1? For any packet $P(v)$, let $\pi(v) = (e_1, e_2, \ldots, e_k)$ be the sequence of links followed by $P(v)$ during Phase 1 of the algorithm.

**Lemma 19.1:** Suppose in Phase 1 that some packet $P'$ traverses link $e_i$, but the next link that $P'$ traverses is not $e_{i+1}$. Then after $e_i$, $P'$ never again traverses a link in $\pi(v)$.

**Proof:** After two packets separate in a certain dimension, they only visit nodes that disagree in that address position. □

**Definition 19.2:** If a packet is queued for link $e_i$ at step $T$, its *lag* is $T - i$.

Note that each packet's lag is defined in terms of the link number in the path of the distinguished packet $P(v)$, rather than its own path.

**Lemma 19.3:** Let $S$ be the set of packets (other than $P(v)$) whose routes in Phase 1 share at least one link with $\pi(v)$. Then $P(v)$ is delayed at most $|S|$ steps in Phase 1.

**Proof:** We will show that each step at which the lag of $P(v)$ increases can be charged to a distinct member of $S$. Thus, $P(v)$ reaches $\rho(v)$ by step $k + |S|$, since its lag never exceeds $|S|$.

Suppose the lag of $P(v)$ increases from $l$ to $l+1$. Let $T$ be the last step at which any packet in $S$ has lag $l$. (Such a packet exists, since one used the link $P(v)$ wanted when $P(v)$'s lag increased.) That is, at time $T$ some packet in $S$ is queued for link $e_{T-l}$. $P(v)$ cannot traverse $e_{T-l}$ at this step, because $P(v)$'s lag has increased to $l+1$ by this time. Thus, some packet $P' \in S$ must traverse $e_{T-l}$ at time $T$. By the maximality of $T$, $P'$ does not enter the queue for $e_{T-l+1}$ at time $T+1$. The increase in the lag of $P(v)$ can be charged to $P'$ since, by Lemma 19.1, the lag of $P'$ never exceeds $l$. □

**Lemma 19.4:** Pr(there is a packet $P(v)$ that uses more than $7n/2$ steps in Phase 1) $< 3^{-n}$.

**Proof:** Fix some packet $P(v)$. For $w \neq v$, define the Bernoulli random variable

$$X_w = \begin{cases} 1 & \text{if } P(w) \text{ traverses a link in } \pi(v) \text{ during Phase 1} \\ 0 & \text{otherwise} \end{cases}.$$

By Lemma 19.3, the delay of $P(v)$ in Phase 1 is at most $\sum_{w \neq v} X_w$. The random variables $X_w$ are independent, since the nodes $\rho(w)$ are independent and the routing is oblivious. We want to use the Chernoff bound (Lemma 2.9) on $\sum_{w \neq v} X_w$, but to do so we need an upper bound on $\sum_{w \neq v} E(X_w)$.

For any link $e$, let the random variable $T(e)$ be the number of packets that traverse $e$ in Phase 1. By symmetry, $E(T(e)) = E(T(f))$ for any two links $e$ and $f$ of $H_n$. (This would not be true in the general case of a partial permutation.) Since the expected number of positions in which $w$ and $\rho(w)$ disagree is $n/2$,

$$\begin{aligned} 2^n \cdot \frac{n}{2} &= \sum_{w \in H_n} E(\text{dist}(w, \rho(w))) \\ &= E\left( \sum_{w \in H_n} \text{dist}(w, \rho(w)) \right) \\ &= E\left( \sum_e T(e) \right) \\ &= \sum_e E(T(e)) \\ &= n \cdot 2^n \cdot E(T(e)), \text{ for any link } e. \end{aligned}$$

Thus, $E(T(e)) = \frac{1}{2}$. Since every $X_w$ that is 1 contributes 1 to some $T(e_i)$,

$$\sum_{w \neq v} X_w \leq \sum_{i=1}^{k} T(e_i).$$

Thus, the desired upper bound on $\sum_{w \neq v} E(X_w)$ is

$$\sum_{w \neq v} E(X_w) = E\left( \sum_{w \neq v} X_w \right) \leq E\left( \sum_{i=1}^{k} T(e_i) \right) = \sum_{i=1}^{k} E(T(e_i)) = \frac{k}{2} \leq \frac{n}{2}.$$

Using Lemma 2.9,

$$\begin{aligned}\Pr(P(v) \text{ uses more than } 7n/2 \text{ steps in Phase 1}) &\leq \Pr\left(\sum_{w \neq v} X_w \geq 5n/2\right) \\ &\leq e^{(1-\frac{1}{5}-\ln 5)5n/2} \\ &< 6^{-n}.\end{aligned}$$

Hence, $\Pr((\exists v)\ P(v)$ uses more than $7n/2$ steps in Phase 1$) < 2^n 6^{-n} = 3^{-n}$. □

Because Phase 2 is dual to Phase 1, we can show the following:

**Lemma 19.5:** Pr(there is a packet $P(v)$ that uses more than $7n/2$ steps in Phase 2$) < 3^{-n}$.

**Theorem 19.6:** For any permutation routing problem on $H_n$,

$$\Pr(\text{all packets are delivered within } 7n \text{ steps}) \geq 1 - 2^{-n}.$$

**Proof:** By Lemmas 19.4 and 19.5,

Pr(there is a packet $P(v)$ that uses more than $7n$ steps in Phases 1 and 2$) < 2 \cdot 3^{-n} < 2^{-n}$

for $n \geq 2$. (For $n = 1$ the proof is straightforward.) □

In fact, it can be shown that

$$\Pr(\text{all packets are delivered within } 2n + O(n/\log n) \text{ steps}) \geq 1 - 2^{-\Omega(n)}.$$

(See Valiant [20].)

**Exercise 19.7:** What is the maximum queue size (with high probability) used by this algorithm?

**Open Problem 19.8:** Find a probabilistic algorithm that routes any permutation on $H_n$ in fewer than $2n$ steps with high probability.

# Bibliography

## References


[1] A. Bar-Noy, P. Raghavan, B. Schieber, and H. Tamaki. Fast deflection routing for packets and worms. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–86, 1993.

[2] A. Ben-Dor, S. Halevi, and A. Schuster. On greedy hot-potatoe routing. Technical Report PCL Report #9204, CS Department, Technion, Jan. 1993.

[3] A. Borodin. Towards a better understanding of pure packet routing. In *Algorithms and Data Structures, WADS '93 Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 14–25. Springer-Verlag, 1993.

[4] 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.

[5] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. Preprint, 1994.

[6] D. D. Chinn, T. Leighton, and M. Tompa. Minimal adaptive routing on the mesh with bounded queue size. In *Proceedings of the 1994 ACM Symposium on Parallel Algorithms and Architectures*, Cape May, NJ, June 1994.

[7] R. Cypher and G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 193–203, Baltimore, MD, May 1990.

[8] B. Hajek. Bounds for evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.

[9] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, June 1990.

[10] M. Kunde. Routing and sorting on mesh-connected arrays. In *3rd Aegean Workshop on Computing (AWOC)*, volume 319 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1988.

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

[12] T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10, Crete, Greece, July 1990.

[13] T. Leighton. Methods for message routing in parallel machines. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 77–96, Victoria, B.C., Canada, May 1992.

[14] T. Leighton, F. Makedon, and I. Tollis. A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant size queues. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 328–335, July 1989.

[15] J. D. Lipson. *Elements of Algebra and Algebraic Computing*. Addison-Wesley, Reading, MA, 1981.

[16] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1994.

[17] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. Technical Report PCL Report #9201, CS Department, Technion, Nov. 1992.

[18] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, Apr. 1989.

[19] S. Rajasekaran and R. Overholt. Constant queue routing on a mesh. *Journal of Parallel and Distributed Computing*, 15(2):160–166, June 1992.

[20] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 18, pages 943–971. M.I.T. Press/Elsevier, 1990.

[21] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pages 263–277, Milwaukee, WI, May 1981.