Routing without Flow Control

Costas Busch
Rensselaer Polytechnic Institute
buschc@cs.rpi.edu

Maurice Herlihy
Brown University
herlihy@cs.brown.edu

Roger Wattenhofer
Microsoft Research
rogerwa@microsoft.com

ABSTRACT

We present the first dynamic hot-potato routing algorithm that does not require any form of explicit flow control: a node may inject a message into the network ($n \times n$ mesh) whenever a link is free. In the worst case, a node may have to wait an expected $O(n)$ time before it has a free link. If destinations are chosen uniformly at random, this algorithm guarantees delivery in an expected $O(n)$ time steps. Both measures are optimal up to a constant factor.

1. INTRODUCTION

A packet routing algorithm is *dynamic* if packets may be continually injected into the network. Most dynamic routing algorithms rely on some kind of *flow control* to prevent the network from becoming overloaded. For our purposes, flow control is any kind of feed-back mechanism that inhibits a node from injecting a packet into the network even when it can. For example, a node might wait for an acknowledgment before sending a packet, or it may simply wait for some duration between packets. Because flow control techniques are conservative, they do not fully exploit the network capacity, especially in the presence of load fluctuations.

We present a new dynamic hot-potato routing algorithm for the $n \times n$ mesh. A novel aspect of this algorithm is that it uses no flow control: a node may send a message whenever a link is free. Moreover, a link will always be free in an expected $O(n)$ time steps, which is asymptotically optimal. This algorithm delivers packets in $O(n)$ expected time when destinations are chosen uniformly at random. Since the distance to a randomly chosen destination is $\Omega(n)$, this complexity is also asymptotically optimal.

Our algorithm exploits techniques for randomized hot-potato routing ("home-runs") first described by Busch et al. [10]. That algorithm, like most prior hot-potato algorithms [12, 5, 17, 2, 7, 10, 9], is *static*: all packets are injected at time zero, and the analysis examines the time needed to deliver them. (Static algorithms, by definition, need not be concerned with flow control.) Our new algorithm, like only a few others [8, 11], is dynamic: nodes may inject packets into the network repeatedly over a long duration. The new algorithm's principal contribution is how it supports dynamic analysis without the need for flow control.

1.1 Hot-Potato Routing

In *hot-potato* (or *deflection*) routing [3], the network nodes have no buffers to store packets in transit: any packet that arrives at a node other than its destination must immediately be forwarded to an adjacent node. Hot-potato routing algorithms can be realized easily in hardware, since there is no need for buffering or queuing. Moreover, hot-potato routing algorithms have been observed to work well in practice. Hot-potato algorithms have been used in the HEP multiprocessor [24], the Connection Machine [15], the CalTech Mosaic C [23], and in high-speed communication networks [20]. Hot-potato routing algorithm are also well-suited for use in optical switching networks [1, 14].

Our algorithm is *greedy*: whenever possible packets are sent closer to their destination. Greedy algorithms are adaptive: when loads are light, packets follow short paths. Non-greedy algorithms typically impose detours that delay packets unnecessarily. Simulation results [20] show that greedy hot-potato routing is effective; packets almost never move away from their destinations.

Our algorithm is also *local*: Nodes are stateless, making routing decisions exclusively on the basis of the packets received at that time step.

1.2 The Mesh Network

We consider the $n \times n$ rectangular mesh network. Each node (except at the edge) is connected to its neighbors by four *links*, denoted *up*, *down*, *left* and *right*. Each node has coordinates (x, y), where x is a column and y a row. The lower-left corner has coordinates $(0, 0)$ and the upper-right corner $(n-1, n-1)$. The *distance* between nodes (x_0, y_0) and (x_1, y_1) is the quantity

$$|x_0 - x_1| + |y_0 - y_1|.$$

Nodes take steps *synchronously*: time is discrete and at each time step, a node receives at most one packet on each incoming link, routes the packets, and sends at most one packet on each outgoing link. The links are bidirectional. The distance between two nodes corresponds to the minimum time

needed to send a packet from one node to the other.

1.3 Packet Generation and Delivery

As noted, most earlier hot-potato algorithms consider only one-shot (static) problems [12, 5, 17, 2, 7, 10, 9]. By contrast, our algorithm and analysis is *dynamic*, nodes may inject packets repeatedly over a long duration. (We are aware of only two other dynamic hot-potato algorithms [8, 11].)

All the packets have random destination, distributed uniformly over the n^2 nodes in the network. As the analysis will show, when a packet is injected into the network, it is delivered to its destination node in $O(n)$ expected time, which is optimal up to constant factors.

Each node has an associated *application*, a program that periodically sends and receives packets. The application communicates with the node, which is responsible for the low-level network routines: routing arriving packets, injecting the application's outgoing on free links, and delivering its incoming packets.

An application and its node communicate by a simple handshake protocol. When the application sends a packet, it blocks while the node's buffer is full. When the application asks to receive a packet, it blocks until the node delivers one. We guarantee that the node will be able to inject a new packet in $O(n)$ time steps worst case (and four packets per time step best case).

1.4 No Flow Control

The most unusual aspect of our algorithm is that it uses *no explicit flow control*: A node can inject a packet into the network any time a link is free. All known routing algorithms and real-world protocols use some sort of flow control to make sure that the network does not become overloaded. (Overloaded networks perform poorly). Typical flow control methods include:

- Nodes must negotiate network bandwidth before they are allowed to inject packets.
- Nodes must wait for a long deterministic/adversarial [22, 6] or random [19, 16, 21, 25] duration between injections.
- Nodes must await acknowledgments of previous packets before injecting new packets.

Current real-world flow control mechanisms use the first and third approach. An overview of the current state of the art can be found in the books of Gouda [13] and Keshav [18].

For the mesh, algorithms with flow control restrict nodes from injecting more than one packet per $\Omega(n)$ time steps. A node is not allowed to inject a packet whenever it has a free link, and therefore there are many free links in the network. As a consequence, algorithms with flow control do not fully utilize the packet capacity of the network. At any time only a small fraction of the network is filled with packets. Our algorithm, on the other hand, utilizes the network capacity to its maximum.

Routing without flow control proves to be especially useful for situations where not all nodes generate packets at the same rates. For example, consider the scenario where only $O(1)$ nodes want to inject packets into the network. In the most efficient routing algorithms for the mesh with flow control [8], the injecting nodes have to wait $\Omega(n)$ time between two injections, which means that only a very poor fraction of $O(1/n^2)$ of the network is utilized. Without flow control the $O(1)$ nodes can take full advantage of all their links.

An algorithm without flow control gracefully supports applications that change packet generation rates over time. The packet injection rate at a node does not have to be negotiated with other nodes in the network but will adapt to the present situation. For example, if all other nodes magically stop injecting packets at the same time, a remaining node u can start injecting packets at the maximum rate (up to four packets per time step). If all other nodes again start injecting packets, the node u is guaranteed to at least inject one packet every $O(n)$ expected time steps.

Hot-potato routing is the ideal routing paradigm for realizing routing without flow control. In traditional store-and-forward routing algorithms, the nodes use queues to store the packets in transit. In these networks flow control is crucial, since without flow control queues would naturally build up and the time for a packet delivery would go to infinity. On the other hand, in hot-potato routing there are no queues. At any time, there are at most as many packets as links in the network.

Even though our algorithm is asymptotically optimal, we can only guarantee rather large constant factors, and critics might argue that algorithms with sophisticated flow control mechanisms will be more efficient after all. Note that our constants are the result of various safe worst-case assumptions. New simulation results [4] show that greedy algorithms indeed have excellent performance with constant factors that are at least as good as the best non-greedy (and static) algorithms [12, 17]. Moreover, flow control obviously introduces additional meta information packets. In the Internet community it is widely believed that meta packets are a significant fraction of the total Internet traffic. Having no flow control (and therefore no meta traffic) therefore improves the total throughput.

1.5 Outline

In Section 2, we present our new hot-potato routing algorithm. In Section 3, we give the analysis of our algorithm. We show that packets are delivered to their destinations in expected time $O(n)$, if destinations are chosen at random. In Section 4, we discuss the issues related to packet injection. We show that even under heavy utilization of the network, we can guarantee that with every $O(n)$ expected time step there will a free link at any node. Therefore, in the worst case, a node can inject at least one packet every $O(n)$'th time step. This is as good as the asymptotically optimum solution (with flow control) by [8].

2. THE ALGORITHM

Every packet has a destination node chosen uniformly in random over the n^2 nodes of the network. In a node, a *good*

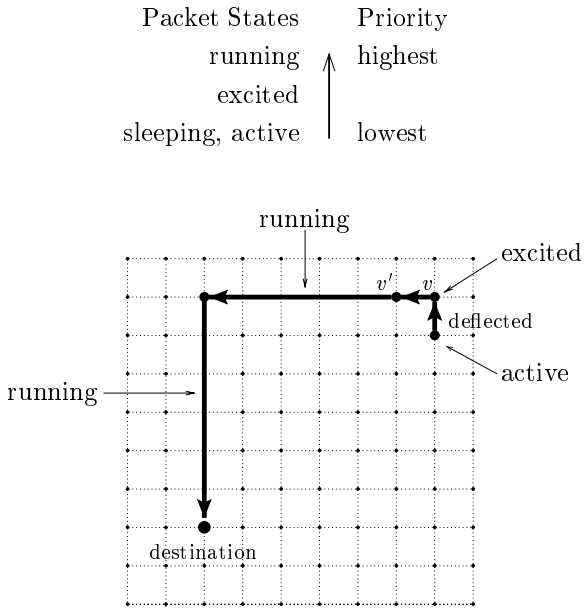

Figure 1: The states and the home run path

link for a packet is one that brings it closer to its destination, and a *bad link* is one that does not.

Our algorithm is *greedy*: in a node, a packet always tries to follow any good link. If a packet cannot advance to its destination (because other advancing packets occupy all the good links), the packet is forced to follow some bad link, in which case we say that the packet is *deflected*.

When two or more packets are competing in the same node for the same links we say that there is a *conflict*. In order to resolve conflicts, our algorithm makes use of *priorities*: each node routes higher-priority packets before routing lower-priority packets. Conflicts between packets of the same priority are resolved in an arbitrary way.

To implement the priority scheme, the packets in our algorithm have states, where each state corresponds to a priority, as shown in the top of Figure 1. A packet π can be in only one of the *sleeping*, *active*, *excited*, or *running* states. A packet changes its state during its lifetime; We assume that the hardware at the nodes supports means to modify the headers of passing packets.

- **Sleeping**. Initially, when the packet π is injected into the network, it is in the sleeping state. In the sleeping state, at any node, packet π simply tries to follow any good link. At each step, the sleeping packet π attempts to change its state, and it becomes an active packet with probability $q = \Theta(1/n)$, and otherwise, it remains sleeping. After packet π leaves the sleeping state it never reenters this state again.

  In order to simplify the analysis, we assume that a sleeping packet cannot be absorbed at its destination. If a sleeping packet reaches its destination, it is sent back into the network. (A packet is absorbed at its destination when it is in a non-sleeping state.)

- **Active**. When packet π is in the active state, it simply tries to follow any good link.

- **Excited**. Whenever an active packet is deflected, it has a chance to increase its priority and become an excited packet. Let's assume that the active packet π got deflected in the previous time step so that at the current time step it appears in a node v, one link further away from its destination. In node v, the active packet π changes its state with probability $p = \Theta(1/n)$, and it becomes excited, and otherwise it remains active. When packet π becomes excited it will attempt to follow an one-bend path towards its destination which we will call a *home run* (see Figure 1). The first part of the home run path is the row path towards the destination column, and the second part is the path in the destination column towards the destination node.

- **Running**. Let's assume that the excited packet π successfully followed the first link in the home run path, so that the current time step it appears in a node v', one link closer to its destination. In node v', the packet π changes its state (deterministically this time) and it becomes running, and it will remain in the running state for the rest of the home run path. We note that a packet is in the excited state for at most one time step.

Consider now an excited or running packet π. If the packet π becomes excited in the destination column then the home run path consists only from the links towards the destination node. At any time step, the *preferred link* of packet π is always the link in the home run path. If in a node v, packet π is unable to follow its preferred link, because of conflicts with other high priority packets, then at the same time step, packet π loses its high priority and it becomes an active packet, and is treated like that by node v. In this case we say that *packet π is interrupted*.

There are only two nodes where the excited or running packet π can be interrupted. The first is at the beginning of the home run path, at the node where the packet π becomes excited, and it can be interrupted by other excited or running packets. The second is in the node of the destination column where the running packet π turns, and it can be interrupted by other running packets with destinations in the same column. (To simplify the algorithm, we specify that a running packet which is already traversing in the destination column cannot be interrupted by other running packets that try to turn in the same column.) Notice that if packet π is interrupted in the destination column then it has to be deflected.

3. TIME ANALYSIS

In this section, we show that when a packet is injected into the network it is delivered to its destination in $O(n)$ expected time (which is optimal up to constant factors).

For the analysis, we need to define the good and bad conditions of the columns. At any time step, we say that a column x of the network is in *good condition* if there are

less than $10n$ non-sleeping packets in the network that have their destination in the column; else the column x is in *bad condition*.

We will show that a packet reaches its destination in $O(n)$ expected time if its destination column is in good condition. We will show that columns are almost always in good condition. The condition of a column is a random process which is determined by the random destinations of the packets. We will argue that in steady state (after long enough time), when a packet is injected in the network at a random time with high probability the packet faces good conditions. The rare event in which a packet faces bad conditions when injected in the network, doesn't hurt the time analysis of our algorithm, since we only argue about expected time bounds for packets injected at random times.

In this section we proceed as follows. In Subsection 3.1, we show that a packet π can be routed to its destination in $O(n)$ time, whenever the destination column of π is in good condition. In Subsection 3.2, we give several basic preliminaries for the subsequent subsections.

In Subsection 3.3 we show that any column remains in good condition for a long time. In particular, we divide the time in consecutive time periods, each of length $6n$, and we examine the condition of a column (if it is good or not) at the beginning of each time period. We show that if at the beginning of a time period a column x is in a good condition then at the beginning of the next time period, column x will remain in good condition with extremely high probability (at least $\Omega(1-\frac{1}{e^n})$).

In Subsection 3.4, we show that if a column departs from the good condition, then it returns back to good condition in very short expected time. In particular, we show that if a column x enters a bad condition then it will again reach a good condition in at most $4n$ time periods, with extremely high probability (at least $\Omega(1-\frac{n}{e^n})$).

Finally, in Subsection 3.5, we combine all the above results and we show that any packet is delivered to its destination in expected $O(n)$ time (under any conditions).

In the analysis we will often make use of the following inequalities. For all n, t, such that $n \geq 1$ and $|t| \leq n$,

$$e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t. \quad (1)$$

For all p, k, such that $0 < p < 1$ and $k \geq 1$,

$$1 - p \leq \left(1 - \frac{p}{k}\right)^k \quad (2)$$

3.1 Time for one Packet

In this subsection we prove that any packet π is delivered to its destination in $O(n)$ time steps in the expected case (Theorem 3.9), assuming that the destination column is in good condition.[1] (We will assume throughout this subsection that the destination column of π is in good condition.)

[1]We would like to note that the methodology in the proofs of this subsection is similar to the methodology in the proofs of paper [10].

Let the probabilities of a packet becoming excited and active be:

$$p := \frac{1}{16n} \quad \text{and} \quad q := \frac{1}{24n}.$$

Let t_0 be the time when packet π becomes active, and let time t_1 be the time when it is absorbed. Let x be the destination column of packet π. Let $m_\pi(t)$ be the number of non-sleeping packets with destination column x at time t. Let $m_\pi := \max_{t_0 \leq t \leq t_1} m_\pi(t)$. As we will show in Corollary 3.16, while column x is in a good condition, $m_\pi < 12n$ with high probability.

LEMMA 3.1. *The probability that a particular node contains no excited packet is at least* $p' := (1-p)^4$.

PROOF. A packet becomes excited with probability p, only if it was deflected in the preceding step. It will fail to become excited with probability at least $1-p$. Since a node contains at most four packets, all four will fail to become excited with probability at least $(1-p)^4$. □

LEMMA 3.2. *An excited packet π successfully follows its preferred link and enters the running state with probability at least* $(1-p)^{4n}$.

PROOF. Let's assume that the excited packet π is in node $v = (x, y)$ at time t and wishes to follow its preferred link. Our priority assignment guarantees that the excited packet π can be interrupted only by other excited or running packets.

By Lemma 3.1, the probability that node v has no excited packet at time t is at least p'. Therefore, with at least this probability packet π will not be interrupted by another excited packet.

We assume (WLOG) that packet's π preferred link is in the left direction. A conflicting running packet must first have become excited at some node $(x+d, y)$ and time $t-d$, for it to be in the running state at node (x, y) at time t, for $1 \leq d \leq n-x-1$. By Lemma 3.1, the probability that no packet became excited in a node $v' = (x+d, y)$ at time $t-d$ is at least p'. Since there are at most $n-1$ nodes like v', the probability that no packet became excited at any of these nodes and times, is at least p'^{n-1}. Therefore, with at least this probability there will be no conflicting running packet at node v at time t, and packet π will not be interrupted by such packets.

In total, the probability that the excited packet π is not interrupted in node v by other excited or running packets is at least $p' \cdot p'^{n-1} = (1-p)^{4n}$. □

Note that the probability that a packet becomes running after being excited is independent from m_π.

LEMMA 3.3. *A running packet π manages to turn into its destination column with probability at least* $(1-p)^{m_\pi}$.

PROOF. According to the algorithm, the running packet π can be interrupted by other running packets only at node where it turns to its destination column. Let $v = (x, y)$ be the node where the running packet π turns at time t. We will compute the probability that packet π will not be interrupted by any other running packet at node v and time t.

Let σ be a running packet with destination in column x. Let's assume that packet σ conflicts with π in node v at time t. In order for packet σ to be in a running state at time t it must have become deflected at some time $t - d - 1$, and then became excited at time $t - d$, and traversed d links to collide with π, all without being deflected. After the packet σ becomes excited, both packets are in "phase": they are at the same time steps at same distance from node v, and then they meet in v at time t.

The key observation is that there is at most one deflection of σ that could bring both packets in phase. If σ failed to become excited after the deflection that could bring it in phase with π, then σ will never catch up with π in time to interrupt π in node v and time t. Packet σ has one chance to become excited, with probability p, after that deflection. Thus, packet σ fails to become excited, and is not in phase with π, with probability $1 - p$. Therefore, with probability at least $1 - p$, packet σ will not interrupt π.

Because π is already in its destination column, there are at most $m_\pi - 1$ other potential conflicting packets, like σ, which have their destinations in column x. Each such packet will not conflict with π with probability at least $1-p$. Therefore, with probability at least $(1 - p)^{m_\pi}$, packet π will not be interrupted when it turns. □

LEMMA 3.4. *An excited packet π will be running successfully to its destination with probability at least $(1-p)^{4n+m_\pi}$.*

PROOF. When a packet becomes excited and then attempts to follow the home run path towards its destination it can only be interrupted in two nodes. The first node is the node where the packet becomes excited and the second node is the node where the packet turns to its destination column. By multiplying the respective probabilities from Lemma 3.2 and Lemma 3.3, the result follows. □

LEMMA 3.5. *Let $m_\pi < 12n$. If an active packet π is deflected, it will become excited and successfully be running to its destination with probability at least $p/2e$.*

PROOF. By Theorem 3.4, after packet π becomes excited it will successfully be running to its destination with probability at least $(1-p)^{4n+m_\pi}$. With $p = 1/16n$ and $m_\pi < 12n$, and by applying Eqn. 1 we get

$$\begin{aligned}(1-p)^{4n+m_\pi} &> \left(1 - \frac{1}{16n}\right)^{16n} \\ &\geq \frac{1}{e}\left(1 - \frac{1}{16n}\right) \\ &\geq \frac{1}{2e}.\end{aligned}$$

Each time the active packet π is deflected, π gets excited with probability p. Therefore the probability for successfully running to its destination after a deflection is at least $p \cdot 1/2e$. □

LEMMA 3.6. *If an active packet π is deflected d times, then it will reach its destination in at most $t := 2d + 2n - 2$ steps. By symmetry, if a packet is active for t time steps, it has been deflected at least $d := t/2 - n + 1$ times.*

PROOF. Initially, the distance from π to its destination is no more than $2n - 2$. Each time π is deflected, the distance increases, and each time it follows a good link, it decreases. □

THEOREM 3.7. *If $m_\pi < 12n$, a non-sleeping packet π is delivered to its destination within $65en$ time steps with probability at least $1 - e^{-1}$.*

PROOF. In the $65en$ time steps, any of the deflections in the first $65en - 2n \geq 64en$ time steps could excite packet π so that it could possibly become running and then reach its destination within the $65en$ time steps.

By Lemma 3.6, in the $64en$ time steps, packet π must have been deflected at least $64en/2 - n + 1 \geq 32en$ times. By Lemma 3.5, each of these deflections fails to excite packet π so that it reaches its destination, with probability at most $1 - p/2e$. Since deflections of the same packet are independent, and by Eqn. 1, all the d deflections fail in the same way with probability at most

$$\left(1 - \frac{p}{2e}\right)^{32en} = \left(1 - \frac{1}{32en}\right)^{32en} \leq e^{-1}.$$

Subsequently, the non-sleeping packet π fails to reach its destination in $65en$ time steps with probability at most e^{-1}, and thus, it reaches its destination with probability at least $1 - e^{-1}$. □

THEOREM 3.8. *If $m_\pi < 12n$, the expected delivery time for a non-sleeping packet π in bounded from above by $O(n)$ time steps.*

PROOF. By Lemma 3.7, the non-sleeping packet π reaches its destination in $t = 65en$ time steps with probability at least $1 - e^{-1}$. Since consecutive time periods of length t are independent, it is expected that packet π reaches its destination within $t/(1 - e^{-1}) = O(n)$ time steps. □

THEOREM 3.9. *If $m_\pi < 12n$, the expected total delivery time for a packet π in bounded from above by $O(n)$ time steps.*

PROOF. When the packet π is injected in the network, it starts out in the sleeping state. After $1/q$ expected time in the network, packet π becomes active. By Lemma 3.8, the active packet π reaches its destination in $O(n)$ expected time steps. Therefore, packet π reaches its destination in $1/q + O(n) = O(n)$ expected total time steps, as needed. □

3.2 The Good, the Bad, and the Basics

In this subsection we give several basic preliminaries for the subsequent subsections.

We use the following standard "Chernoff-type" tail inequality.

THEOREM 3.10. *Let $X_1, X_2, \ldots, X_n$ be independent Poisson trials such that, for $i = 1, \ldots, n$, $Pr[X_i] = p_i$, where $0 < p_i < 1$. Then, for $\mu := E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} p_i$, and $\delta > 0$,*

$$Pr[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}.$$

LEMMA 3.11. *For any $t \geq n$, in t time steps, the probability to have more than $8qtn$ new non-sleeping packets with destination in column x is less than $1/4^{qtn}$, for any x.*

PROOF. At any time step there are at most $4n^2$ packets in the mesh. With probability q, a sleeping packet gets active, with probability $1/n$ the random destination of a newly injected (and now active) packet is column x.

In t time steps, we have at most $4tn^2$ chances that a sleeping packet becomes active. The probability P to have not more than $8qtn$ new active packets with destination column x can be bounded with Theorem 3.10. With $\mu = q/n \cdot 4tn^2 = 4qtn$ and $\delta = 1$ we have: $P < (e/4)^{4qtn} < 1/4^{qtn}$. □

LEMMA 3.12. *A deflected packet π enters the running state (in the next steps) with probability at least $1/cn$, with constant $c := 22$.*

PROOF. The deflected packet π becomes running only after it becomes excited, and then successfully follows the first link in the home run path. The deflected packet π becomes excited with probability p. By Lemma 3.2, the excited packet π becomes running with probability at least $(1-p)^{4n}$. Using Eqn. 2, and since $0 < 4pn < 1$, we have,

$$(1-p)^{4n} = \left(1 - \frac{4pn}{4n}\right)^{4n} \geq 1 - 4pn = \frac{3}{4}.$$

Subsequently, the deflected packet π becomes running with probability at least $p \cdot 3/4 \geq 1/cn$. □

LEMMA 3.13. *During its lifetime, a packet σ can interrupt at most $2n$ running packets.*

PROOF. Packet σ can interrupt other running packets only when it is running in the destination column. Let's assume that the packet σ is running. The first location where σ can interrupt other running packets is at the node where it turns to the destination node. Obviously, if packet σ interrupts another running packet when it turns, then σ successfully enters the destination column. After packet σ enters the destination column, then it reaches its destination without interruptions. Therefore, packet σ has only one chance to interrupt other running packets, when it becomes running and successfully enters the destination column for the first and last time.

Since the length of the destination column is n, packet σ can be in its destination column for at most n time steps. In each of the nodes in the destination column, σ can interrupt at most 2 other running packets that try to turn, one arriving from the left and one from the right link. In total, σ interrupts at most $2n$ running packets. □

LEMMA 3.14. *Let t the time period $t = t_0 - t_1$, where $t_0 < t_1$. Let x be a column such that at time t_0 there are at least α non-sleeping packets with destination in column x. Then, by time t_1 at least β of these non-sleeping packets will be absorbed with probability at least $1 - (1 - 1/cn)^r$, where $r := (\alpha - \beta)(t/2 - 2n) - 2\beta n$.*

PROOF. First we compute the probability that at most β of the α non-sleeping packets are absorbed in time period t, so that at least $\alpha - \beta$ packets are not absorbed.

Let π be a packet that was non-sleeping at time t_0 and still not absorbed at time t_1. Any of the deflections of π in the first $t' = (t_1 - 2n) - t_o = t - 2n$ steps could possibly excite π, so that π could be running and reach its destination before the end time period t. With Lemma 3.6, since π is not absorbed, it must have been deflected at least $t'/2 - n + 1 = t/2 - 2n + 1$ times, in the t' time period. Since there are at least $\alpha - \beta$ non-absorbed packets like π, all of them together have been deflected at least $(\alpha - \beta)(t/2 - 2n + 1)$ times.

With Lemma 3.13, the β absorbed packets can together interrupt at most $2\beta n$ running packets. Therefore, in time period t there were at most $2\beta n$ attempts to produce running packets from the $\alpha - \beta$ non-absorbed packets. A packet can become running only after a deflection. Therefore, at most $2\beta n$ deflections of the non-absorbed packets produced running packets.

Combining the deflections, we have at least $(\alpha - \beta)(t/2 - 2n + 1) - 2\beta n \geq r$ deflections of the non-absorbed packets that did not produce running packets.

With Lemma 3.12, a deflected packet will fail to become running with probability at most $1 - 1/cn$. All the r deflections of the non-absorbed packets will fail to produce running packets with probability at most $(1 - 1/cn)^r$. Therefore, with at most this probability, at least $a - b$ packets were not absorbed, or equivalently at most b packets were absorbed. Subsequently, with probability at least $1 - (1 - 1/cn)^r$, at least b packets are absorbed. □

3.3 Good 2 Good

In this subsection we show that any column remains in good condition for a long time (Corollary 3.19).

For the analysis, we divide the time in consecutive time periods, each of length $6n$, and we examine the condition of a column at the beginning of each time period.

By Lemma 3.11 we have:

COROLLARY 3.15. *In $6n$ time steps we will have less than $2n$ new non-sleeping packets with destination in a specific column x with probability at least $1 - 1/4^{n/4}$, for any x.*

By Corollary 3.15, we have:

COROLLARY 3.16. *Consider a time period of $6n$ time steps. If at the beginning of the time period column x is in a good condition, then, during the time period, the number of non-sleeping packets with destination in column x never exceeds $12n$ with probability at least $1 - 1/4^{n/4}$.*

LEMMA 3.17. *Consider a time period of $6n$ time steps. If at the beginning of the time period column x is in a good condition and at least $8n$ non-sleeping packets have destinations in this column, then at least $2n$ of these packets will reach their destinations within this time period with probability at least*

$$1 - \frac{1}{e^{2n/c}}.$$

PROOF. By applying Lemma 3.14, with $t = 6n$, $\alpha = 8n$, $\beta = 2n$, we have that the desired probability is at least

$$\begin{aligned} & 1 - \left(1 - \frac{1}{cn}\right)^{(8n-2n)(6n/2-2n)-4n^2} \\ = \; & 1 - \left(1 - \frac{1}{cn}\right)^{2n^2} \\ = \; & 1 - \left(1 - \frac{1}{cn}\right)^{cn \cdot 2n/c} \\ \geq \; & 1 - \frac{1}{e^{2n/c}}. \end{aligned}$$

(By applying equation 1)

□

THEOREM 3.18. *If at time t_0, column x is in a good condition (where less than $10n$ non-sleeping packets have destinations in column x) then after time period $6n$ it will remain in the good condition with probability at least*

$$1 - \frac{2}{e^{2n/c}}.$$

PROOF. Consider the time $t_1 = t_0 + 6n$. By Corollary 3.15, by that time the newly introduced non-sleeping packets are at most $2n$, with probability at least

$$1 - \frac{1}{4^{n/4}}.$$

Consider first the case where at time t_0 column x is in a good condition and the number of non-sleeping packets with destinations in x are less than $8n$. At time t_1, the number of non-sleeping packets with destinations in x will be less than $8n + 2n = 10n$, and column x remains in a good condition, with probability at least

$$1 - \frac{1}{4^{n/4}} \geq 1 - \frac{2}{e^{2n/c}}.$$

Consider now the case where at time t_0 column x is in a good condition and at least $8n$ non-sleeping packets have destinations in x. By Lemma 3.17, the number of packets that are absorbed by time t_1 is at least $2n$ with probability at least

$$1 - \frac{1}{e^{2n/c}}.$$

Subsequently, at time t_1 the number of non-sleeping packets with destinations in x will be less than $10n + 2n - 2n = 10n$, and column x remains in a good condition, with probability at least

$$\begin{aligned} & \left(1 - \frac{1}{4^{n/4}}\right) \cdot \left(1 - \frac{1}{e^{2n/c}}\right) \\ \geq \; & \left(1 - \frac{1}{e^{2n/c}}\right)^2 \\ \geq \; & 1 - \frac{2}{e^{2n/c}}. \end{aligned}$$

(By applying equation 2)

□

COROLLARY 3.19. *If a column is in good condition, it will remain in good condition for expected time at least $\Omega(ne^n)$.*

PROOF. At time t_0 we are in good condition. The probability for not being in good condition anymore at time $t_1 = t_0 + 6n$ is according to Theorem 3.18 less than $P = \frac{2}{e^{2n/c}}$. Since this probability does not depend on the history of the network, future probabilities are independent. Thus the expected time of being constantly in good condition is $6n \cdot 1/P = 3ne^{2n/c} = \Omega(ne^n)$. □

3.4 Bad 2 Good

In this subsection we show that if a column departs from the good condition, then it returns back to the good condition in very short expected time (Corollary 3.23).

Similar to Subsection 3.3, we examine the condition of a column at the beginning of every time period of length $6n$.

LEMMA 3.20. *Consider a time period with $6n$ time steps. If at the beginning of the time period the column x is in bad condition (where at least $10n$ non-sleeping packets have destinations in column x) then at least $3n$ of these packets will be absorbed in this time period with probability at least*

$$1 - \frac{1}{e^{n/c}}.$$

PROOF. By applying Lemma 3.14, with $t = 6n$, $\alpha = 10n$,

$\beta = 3n$, we have that the desired probability is at least

$$\begin{aligned} & 1 - \left(1 - \frac{1}{cn}\right)^{(10n-3n)(6n/2-2n)-6n^2} \\ = \; & 1 - \left(1 - \frac{1}{cn}\right)^{n^2} \\ = \; & 1 - \left(1 - \frac{1}{cn}\right)^{cn \cdot n/c} \\ \geq \; & 1 - \frac{1}{e^{n/c}}. \end{aligned}$$

(By applying equation 1)

□

LEMMA 3.21. *If at time t_0 column x is in bad condition and α ($\alpha \geq 10n$) non-sleeping packets have destinations in this column, then at time $t_1 = t_0 + 6n$ the number of these non-sleeping packets will be reduced to $\alpha - n$ with probability at least*

$$1 - \frac{2}{e^{n/c}}.$$

PROOF. Consider the time period $t = t_1 - t_0 = 6n$. By Corollary 3.15, in this time period, the newly introduced non-sleeping packets with destinations in column x are at most $2n$, with probability at least

$$1 - \frac{1}{4^{n/4}}.$$

By Lemma 3.20, the number of these non-sleeping packets that are absorbed in time period t is at least $3n$, with probability at least

$$1 - \frac{1}{e^{n/c}}.$$

Subsequently, at time t_1 the number of non-sleeping packets will be at most $\alpha + 2n - 3n = \alpha - n$, with probability at least

$$\begin{aligned} & \left(1 - \frac{1}{4^{n/4}}\right) \cdot \left(1 - \frac{1}{e^{n/c}}\right) \\ \geq \; & \left(1 - \frac{1}{e^{n/c}}\right)^2 \\ \geq \; & 1 - \frac{2}{e^{n/c}}. \end{aligned}$$

(By applying equation 2)

□

THEOREM 3.22. *If at time t_0 column x is in bad condition then in at most $24n^2$ time steps the column will reach a good condition (where less than $10n$ non-sleeping packets have destinations in column x) with probability at least*

$$1 - \frac{8n}{e^{n/c}}.$$

PROOF. At the worst case, at time t_0 there are at most $4n^2$ non-sleeping packets with destinations in column x. By Lemma 3.21, at time $t_0 + 6n$ the number of these non-sleeping packets will be at most $4n^2 - n$, with probability at least

$$1 - \frac{2}{e^{n/c}}.$$

By applying Lemma 3.21 at most k times, we have that at time $t_0 + 6kn$ the number of non-sleeping packets will be at most $4n^2 - kn$, with probability at least

$$\left(1 - \frac{2}{e^{n/c}}\right)^k.$$

For $k = 4n \geq 4n - 10$, by time $t_0 + 24n^2$, the non-sleeping packets are less than $10n$, with probability at least

$$\left(1 - \frac{2}{e^{n/c}}\right)^{4n} \geq 1 - \frac{8n}{e^{n/c}}.$$

(By applying equation 2)

□

COROLLARY 3.23. *If a column is in bad condition, it will enter the good condition in expected time at most $O(n^2)$.*

3.5 And now all together

In this subsection we combine all the results from the previous subsections and we show that any packet is delivered to its destination in expected $O(n)$ time under any conditions.

COROLLARY 3.24. *Let the network be in a steady state. Then, at a random time t, the probability that a specific column x is in good condition is $1 - O(n/e^n)$, for any column x.*

PROOF. Corollary 3.19 says that a column remains in good condition for expected time at least $E_{good} = \Omega(ne^n)$. Corollary 3.23 says that the expected time to remain in bad condition is at most $E_{bad} = O(n^2)$.

We have the following:

$$Pr_{good_{t \to \infty}} = \frac{E_{good}}{E_{good} + E_{bad}} = 1 - O(n/e^n).$$

□

We get our main result immediately:

THEOREM 3.25. *Let the network be in a steady state. If a packet is injected into the network at a random time, the expected delivery time is $O(n)$, which is asymptotically optimal.*

PROOF. If the packet is injected when its destination column is in good condition, with Theorem 3.9 the expected delivery time is $O(n)$, as long as the column stays in good condition for long enough time, which it does by Corollary 3.19. If the packet is injected when its destination column is in bad condition or when the column is not in good condition for long enough time, by Corollary 3.23 we have to

wait $O(n^2)$ time until the network is in good condition for a long enough time again.

With Corollary 3.24 we know that with very high probability, the first is true. The Theorem follows. □

4. PACKET INJECTION

In our algorithm, the packets can be injected into the network whenever there are free links, and we have proven in the previous section that even with this very liberal injection policy, the expected delivery time of a packet is asymptotically optimum.

In this section, we show that there actually *are* free links every now and then, at *any* node. In other words, will show that a specific node u can inject a new packet every $O(n)$ expected time in the worst case, which is also asymptotically optimum.

If *all* nodes inject packets whenever they have free links, the desired result follows naturally: there are $O(n^2)$ packets in the network and from Theorem 3.25 we immediately know that in $O(n)$ time an expected number of $O(n^2)$ of these packets are delivered. Since destinations are random, $O(1)$ of the delivered packets are routed to node u. So every $O(n)$ expected time slot, node u absorbs a packet from the network. In the following time step node u has a guaranteed free link and can itself inject a new packet into the network.

However, we don't want to make any assumptions on the injection behavior of other nodes in the network. Paradoxically, it is more difficult to show that there is a free link every now and then, when other nodes do *not* use every free link to inject a packet. One could say that from a worst case point of view the network behaves best if it is fully loaded, a paradox called *non-monotonicity* which is known for routing and thoroughly studied in [11].

Indeed, it can be shown to be impossible to give a worst-case injection guarantee without further precautions. Let u be a node in the mesh and let V be the set of (up to four) direct neighbors of u. Assume that at each step the nodes in V inject as many new packets into the network as they have free links, which is up to 16. All other nodes in the network are completely passive and do not inject any packets at all. Then node u will be completely jammed, only by the traffic generated at the neighbor nodes V. Since the nodes in V only inject $O(1)$ packets per time step (and destinations are random), node u must wait an expected number of $O(n^2)$ time steps to receive one of these packets with destination itself. Therefore, node u's time to wait for a free link is worse than in the case described above, where all (other) nodes inject a new packet into the network whenever they have free links.

In order to have both, the liberal injection policy (nodes can inject a new packet whenever there is a free link), and the worst-case upper bound (the time between two free links for a specific node is upper bounded by $O(n)$) we use the following simple trick:

> When the network is initialized, at step 0, all the links are free and all nodes can inject up to four packets. We demand the nodes to mark one of their packets with a "replace" flag. If at step 0 a node does not want to inject a packet, the node must inject an empty (but flagged) packet into the network. An empty packet behaves exactly like a standard packet (also with a random destination), but doesn't carry any information.
>
> Later, upon receiving a flagged packet, nodes must inject a flagged packet again (if no real packet is available, an empty packet must be injected instead). Therefore we always have exactly n^2 flagged packets in the network. With the same argumentation as above these packets will make sure that each node receives a free link every $O(n)$ time step. And we still have about $3n^2$ free links in the network that can be used by any node if there is not so much traffic.

With the above trick, a constant fraction of the bandwidth of the network may be used by empty flagged packets. However, we have (many) empty packets only when the overall traffic in the network is low. As soon as traffic in the network picks up, the empty packets will be replaced by real ones quickly and do not affect the overall performance. In the extreme case where all nodes have packets to inject there will be no empty packets that affect the bandwidth of the network.

From the above, we have the desired result:

THEOREM 4.1. *In the worst case, still any node can inject a new packet in $O(n)$ expected time.*

5. CONCLUSION

We presented the first dynamic hot-potato routing algorithm which doesn't use any explicit flow control mechanism. Without flow control packets can be injected whenever there are free links, and therefore the network capacity is exploited to its maximum. Although our algorithm doesn't use flow control, the analysis shows that the expected delivery time for a packet is $O(n)$, and in the worst case a node obtains a free link, and can inject a packet, every $O(n)$ time steps. Both measures are optimal up to a constant factor.

The network we studied in this paper is the $n \times n$ mesh. Our analysis can be trivially extended to the two-dimensional $n \times n$ torus (actually, in the torus the expression constants are improved because the maximum distance between any pair of nodes is $n-1$, which is smaller than the maximum distance $2n-2$ in the mesh). An interesting open problem is to extend the analysis of our algorithms to other kinds of network topologies.

Acknowledgments

Most of all we would like to thank Eli Upfal from Brown University for pointing us to the problem of dynamic routing analysis, and giving us excellent guidance for how to prove stability in dynamic systems. We would also like to thank the referees for their useful comments.

6. REFERENCES

[1] A. S. Acampora and S. I. A. Shah. Multihop lightwave networks: a comparison of store-and-forward and hot-potato routing. In *Proc. IEEE INFOCOM*, pages 10–19, 1991.

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

[3] P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, pages 1–9, 1964.

[4] Bartzis, Caragiannis, Kaklamanis, and Vergados. Experimental evaluation of hot-potato routing algorithms on 2-dimensional processor arrays. In *EUROPAR: Parallel Processing, 6th International EURO-PAR Conference*. LNCS, 2000.

[5] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory of Computing Systems*, 31(1):41–61, Jan./Feb. 1998.

[6] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 376–385, Philadelphia, Pennsylvania, 22–24 May 1996.

[7] A. Borodin, Y. Rabani, and B. Schieber. Deterministic many-to-many hot potato routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):587–596, June 1997.

[8] A. Broder and E. Upfal. Dynamic deflection routing on arrays. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 348–358, May 1996.

[9] C. Busch, M. Herlihy, and R. Wattenhofer. Hard-potato routing. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC'00)*, pages 278–285, May 2000.

[10] C. Busch, M. Herlihy, and R. Wattenhofer. Randomized greedy hot-potato routing. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 458–466, Jan. 2000.

[11] U. Feige. Nonmonotonic phenomena in packet routing. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 583–591, New York, May 1–4 1999. ACM Press.

[12] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 553–562, Pittsburgh, PN, Oct. 1992. IEEE Computer Society Press.

[13] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley and Sons, 1998.

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

[15] W. D. Hillis. *The Connection Machine*. MIT press, 1985.

[16] N. Kahale and T. Leighton. Greedy dynamic routing on arrays. *Journal of Algorithms*, 29(2):390–410, Nov. 1998.

[17] C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 273–282, Velen, Germany, June 30–July 2, 1993. SIGACT and SIGARCH.

[18] S. Keshav. *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Addison-Wesley, Reading, MA, USA, 1997.

[19] F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In A.-S. ACM-SIGARCH, editor, *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10, Island of Crete, Greece, July 1990. ACM Press.

[20] N. F. Maxemchuk. Comparison of deflection and store and forward techniuques in the Manhattan street and shuffle exchange networks. In *Proc. IEEE INFOCOM*, pages 800–809, 1989.

[21] M. Mitzenmacher. Bounds on the greedy routing algorithm for array networks. *Journal of Computer and System Sciences*, 53(3):317–327, Dec. 1996.

[22] C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 215–224, New York, May 1–4 1999. ACM Press.

[23] C. L. Seitz. The caltech mosaic C: An experimental, fine-grain multicomputer. In *4th symp. on Parallel Algorithms and Architectures*, June 1992. Keynote Speech.

[24] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. Fourth Symp. Real Time Signal Processing IV*, pages 241–248. SPIE, 1981.

[25] Stamoulis and Tsitsiklis. The efficiency of greedy routing in hypercubes and butterflies. *IEEETCOMM: IEEE Transactions on Communications*, 42, 1994.