Hot Potato Worm Routing via Store-and-Forward Packet Routing

Ilan Newman † Assaf Schuster ‡

Abstract

The theory of worm routing (rather than packet routing) recently attracts an increased attention as an abstraction of the underlying communication mechanisms in many parallel machines. Routing the worms in the hot potato style is a desired form of communication in high-speed optical interconnection networks. In this work we develop a simple method for the design of parallel hot potato worm routing algorithms. Our basic approach is to simulate known packet routing algorithms, so that in each step worms are moved around instead of packets. By plugging in known results for packet routing, we get the fastest (so far) deterministic batch worm routing algorithms.

Although the results are given for permutation routing on the mesh and the hypercube, the general method can be applied to many other networks and to more general communication patterns as well. Moreover, once better routing algorithms are found for the underlying network, the worm routing algorithms improve as well.


* A short version of this paper was presented in ISTCS, June 1993. This work was supported in part by the French-Israeli grant for cooperation in Computer Science, and by a grant from the Israeli Ministry of Science.

† Department of Mathematics and Computer Science, Haifa University, Haifa, Israel. E-mail: ilan@mathcs2.haifa.ac.il

‡ Department of Computer Science, Technion, Haifa, Israel 32000. E-mail: assaf@cs.technion.ac.il.
1 Introduction

The traditional approach in theoretical distributed communication is to design packet routing algorithms. It is assumed that communication in a massively parallel system will consist of messages which are dropped into fixed-sized packets of information traveling in the system. Extensive literature was developed during the last few years, involving a huge number of parallel routing algorithms for packets. However, in packet routing there is a price-performance tradeoff: On one hand, a goal of massively parallel computer architects is to minimize the buffer capacity at each node of the network. This implies that messages should be split to very small packets. On the other hand, each packet travels in the network independent of the others, hence it needs its own routing mechanism and uses network resources. In other words, when a large message is split into many packets, the overall routing overhead is multiplied accordingly.

In order to bypass this packet routing tradeoff, many recent parallel machines use worms as their communication mechanism. A worm is a contiguous stream of bits, physically occupying a sequence of nodes/edges in the network. There are no queues at intermediate nodes, and a node can only hold a small fraction of a message, called a flit. The header of the worm is the first flit, containing all the routing information. The body of the worm follows the header, i.e., the body is stored along a contiguous path of nodes, each holding the next flit of the message. When the header enters a node, routing operations take place and the header may be directed to one of the outgoing edges. When the header proceeds via an edge, a second flit comes in and is immediately directed to follow its header on the same outgoing edge. Notice that no further routing operation is required until the worm finally exits the node via the assigned outgoing edge. Thus, although the buffers at intermediate processors may be very small, the total routing overhead remains no more than that of corresponding packet mechanisms.

There are difficulties involved in worm routing which do not exist in packet routing. While a worm is using a certain edge, and even if it is currently blocked, no other worm may use it in the same direction (otherwise, larger buffers and more complex routing hardware is required). Thus, a worm of length $k$ may block $k$ other worms, each of them may block several other worms, etc. In this way, there may be sequences of worms blocking each other, so that a single worm may block many worms in distant areas of the network, even those whose paths would never cross its path. It may also happen that two or more worms would block each other, leading to a deadlock.

Consider the body of the worm which is instantaneously directed to the outgoing edge. Such a fast switching mechanism is essential for optical communication networks. Optical communication is becoming more and more popular, with its bandwidth outperforming electronic communication. However, there is a huge obstacle here: no optical storage which is cheap and fast is available. Moreover, electro-optic conversion at each node of the path (in order to use electro-storage) slows down the potential of optical pulse generation techniques in forming optical flits at high data rates. A way to keep the high communication bandwidth is to use hot potato worm routing [19]. With hot potato worm routing a worm that arrives at a node is immediately directed to one of the outbound links leaving that node, unless this node is its destination. Thus each worm, once born (injected
into the network) is never blocked or stored at an intermediate node. Rather, it keeps moving until
it finds its destination. This principle of “ever moving” is called the hot potato principle or the
hot potato paradigm. Indeed, many parallel machines use variants of worm and hot potato routing.
Note, however, that the hot potato principle may disturb the routing when a desired outbound
link is already taken by another worm. In this case a worm is deflected to a distant region in the
network which may not be on the way to its destination. Moreover, worms might deflect each other
in an infinite cycle, which is called: a livelock.

Formally, a network is a directed graph whose nodes are processors and whose edges are unidirec-
tional links between processors. The network is synchronous so that flits move between neighboring
processors at discrete time steps. At each step, a processor may receive up to one flit from each
incoming edge and submit up to one flit along each outgoing edge. If a received flit belongs to
the body of the worm, it is moved immediately to an outbound link through which the header of
the worm made its exit. Each node has as many outgoing edges as incoming ones. Note that this
ensures that each worm that enters a processor at time step \( t \) will have a free outgoing link available
to leave it at time step \( t + 1 \) (even though it may be in the “wrong direction”). Nodes have no
buffers, a worm keeps moving at every time step until it is eventually absorbed at its destination
node (the hot potato paradigm). The networks that are dealt with are undirected graphs in which
every edge represent two directional links in opposite directions.

The length of a worm is its number of flits, i.e., the number of nodes it occupies at any time.
We assume in the sequel that all the routing information is contained in the first flit of the worm.
We remark here that if this is not the case our algorithms can be modified accordingly in a straight-
forward manner. Worms of length \( \leq k \) are called \( k \)-worms. We say that a routing algorithm is a
hot potato algorithm if the worms move according to the hot potato paradigm.

Studies of worm routing focused on the study of related hardware, simulation studies of network
bandwidth, and on deadlock avoidance [2, 8, 6, 5, 7, 21, 27, 9, 23, 28, 25]. Several works consider
the related (less restrictive and less practical) model of virtual cut-through [14, 18, 20]. Another
related (less restrictive) model is considered by Aiello et. al. [1]. Recently, in a giant step towards
establishing a firm theory for worm routing, Felperin et. al. [10] formalized the theoretical model
and gave probabilistic analysis of algorithms for the mesh and the butterfly. For \( k \)-worms they
achieve an \( O(kn \log n) \) time algorithm for random destinations on the \( n \times n \) mesh. For routing \( 2^n \)
worms on the \( n2^n \) butterfly they get \( O(kn \min\{k, n\}) \) randomized algorithm. Bar-Noy et. al. give
an \( O(kn) \) randomized algorithm and \( O(kn^{1.5}) \) and \( k^{2.5} n2^{\log n \log \log n} \) deterministic algorithms [3].

In this work we develop a simple method for the design of hot potato worm routing algorithms.
In particular it can be used to construct deterministic algorithms. Our basic approach is to
simulate known packet routing algorithms, so that in each step worms are moved around instead of
packets. Implementing this naive idea involves an obvious difficulty. Namely, when a worm takes
a move which corresponds to a move of a packet, it needs a sequence of nodes to store it at its new
place. We solve this problem by packing worms into regions of the network which are dedicated to
this purpose. This should be done while maintaining the hot potato paradigm.
We demonstrate the above general idea by showing how to transform any store-and-forward packet routing algorithm (with certain “good” properties) for the mesh or the hypercube into a hot potato worm routing algorithm. In particular, we have the following results for hot potato permutation routing of \( k \)-worms: For the \( n \times n \) mesh we get an \( O(k^{2.5}n) \) deterministic algorithm, and an \( O(k^{1.5}n) \) algorithm for the corresponding offline problem (in which the permutation is known in advance). For the \( 2^n \)-nodes hypercube we get an \( O(k^3n \log^2 n) \) deterministic algorithm, and an \( O(k^3n) \) randomized algorithm. Our results are essentially ’black box’ simulations of packet routing. Therefore, although the results are given for permutation routing on the mesh and the hypercube, the general method can be applied to more general communication patterns, and to any network which is a product of a network having a “good” packet routing algorithm (see below) and some other network of approximately the size of a worm. Moreover, once better routing algorithms are found for the underlying network, the worm routing algorithm improves, too. For example, let a \( k - k \) routing algorithm be such that up to \( k \) packets originate and up to \( k \) are destined to each node [11, 13]. Motivated by our work, Kaufmann and Sibeyn have reported an \( O(kn) \) steps algorithm for the \( k - k \) routing problem with constant size buffers for the \( n \times n \) mesh [26]. Using our results as explained later, their result yields a hot potato permutation routing algorithm that routes \( k \)-worms in \( O(k^{1.5}n) \) steps. Similarly, once an \( O(n) \) packet routing permutation algorithm for the \( 2^n \)-nodes hypercube is found, it will imply a corresponding \( O(k^3n) \) worm routing algorithm.

2 Routing on the two-dimensional Mesh

We denote by \( M_n \) the \( n \times n \) 2-dimensional mesh, having \( n \) columns and \( n \) rows and a node at each intersection point. We assume that the columns are numbered 0, \( \cdots \), \( n - 1 \) from left to right and that the rows are numbered 0, \( \cdots \), \( n - 1 \) bottom to up. A processor is identified by a pair \((\text{column}, \text{row})\) where \( 0 \leq \text{column}, \text{row} \leq n - 1 \).

An obvious lower bound for the \( k \)-worm permutation routing is \( \Omega(kn) \), which can be seen from both bisection or edge counting considerations. An obvious upper bound is \( O(kn^2) \) which is obtained by sending the worms along a Hamiltonian cycle.

We construct efficient worm routing algorithms by ’black-box’ simulation of packet routing algorithms of a given class. By plugging in a specific \( O(n) \) packet routing we get \( O(k^2.5n) \) worm permutation routing.

Let \( A(n) \) be a packet routing algorithm for \( M_n \). We need the following assumptions on \( A(n) \).

Assumption 2.1 \( A(n) \) uses buffer of size at most four.

Assumption 2.2 \( A(n) \) can route any partial permutation in \( t_A(n) \) steps.

Assumption 2.3 At every step of Algorithm \( A(n) \), the decisions when, where and if to send a packet is taken locally by the node in which it resides without considering the contents of any other node.
**Fact 2.1** [24, 15] There is a packet routing algorithm $A(n)$ for $M_n$ that meets assumptions 2.1, 2.2, and 2.3, of complexity $t_A(n) = O(n)$ for any partial permutation.

We note that any hot potato packet routing algorithm automatically satisfies the assumptions above. Most of the store-and-forward packet routing algorithms that appear in the literature do not meet Assumption 2.3 literally. In particular, most of the algorithms for routing that use sorting assume that edges act like comparators. However, they can be transformed to meet the assumptions, slowing them down by at most a factor of two in an obvious way.

Let $N = n/(2\sqrt{k} + 3)$. For simplicity we assume that the parameters are such that $N$ and $\sqrt{k}$ are integers, and that $\sqrt{k}$ is even.

**Theorem 2.1** Let $A(N)$ be a packet routing algorithm for $M_N$ which terminates in $t_A(N)$ steps, and which meets assumptions 2.1, 2.2 and 2.3. Then there is a worm permutation routing algorithm $\hat{A}(n)$ for $M_n$, routing $k$-worms, $1 \leq k \leq n^2/5$, in time $O(k^3t_A(N))$. Moreover, $\hat{A}$ can be made to work in the hot potato paradigm in the same time bound.

We conclude the following corollary.

**Corollary 2.1** There is a hot potato worm routing algorithm that can route any (partial) permutation of $k$-worms, $1 \leq k \leq n^2/5$, on $M_n$ in $O(k^{2.5}n)$ steps.

**Proof of Theorem 2.1:** We split $M_n$ into submeshes of size $(2\sqrt{k}+3) \times (2\sqrt{k}+3)$, called windows. The rows and columns inside each such window are numbered from 0 to $2\sqrt{k} + 2$. Each window will emulate the operation of a single node from $A(N)$. We refer to a step of $A(N)$ (or to its simulation) as 'step' to distinguish it from a real step of $\hat{A}$ which is referred simply as a step. At the end of each 'step', every worm in each window is packed in a snakelike path in one of the four $\sqrt{k} \times \sqrt{k}$ submeshes, leaving the three middle rows and the three middle columns of the window empty, as is depicted in Figure 1. We call these middle columns and rows highways (either column highway or row highway). Each highway is composed of three columns (rows), a middle one, and two others which are called lanes. We call the $(\sqrt{k}+1, \sqrt{k}+1)$ point the center of the window. The worms are packed symmetrically around the center. We distinguish between black windows in which the center’s coordinates sum is odd and white windows in which this sum is even. Note that the black and white windows are interleaved in a way reminiscent of squares on a chess board. In black windows the worms are packed so that their headers are at distance four from the center, while in white windows this distance is five, as can be seen in Figure 1. This distinction between white and black windows is done in order to ensure that the distance between worms in neighboring windows is even. This will play an important role in achieving the hot potato paradigm.

The middle row (the $\sqrt{k}+1$ row) and its parallel lanes are divided into two parts - the left part and the right part. Similarly the middle column and its parallel lanes are divided into two parts - the lower half and the upper half. We assign these four segments and their parallel lanes to the four $\sqrt{k} \times \sqrt{k}$ submeshes for communication purposes, a segment for each. Each submesh gets
the segment (lane) that is adjacent to it in the clockwise direction. In other words, the bottom right submesh gets the lower middle column and the bottom left gets the left half row, etc. The assignments of the lanes is the same as that of the segments.

![Diagram of segments and lanes](image)

Figure 1: Windows storing four packed worms at the end of a 'step' during the routing algorithm \( \hat{A} \). The left window is white and the right one is black.

The algorithm is composed of \((2\sqrt{k} + 3)^4 = O(k^2)\) rounds. During the \((i, j, k, l)\)-th round \(0 \leq i, j, k, l \leq 2\sqrt{k} + 2\), we route the worms that originate from the \((i, j)\)-th point of a window, and are destined to the \((k, l)\)-th point of a window. All the worms that belong to previous rounds already arrived at their destinations and are not in the system anymore. All the worms which belong to future rounds are not yet born. Note that for permutation routing this ensures that during each round at most one worm is born at each window, and at most one is destined to each window. In other words, in each round we route a (partial) permutation of worms between the windows. Consider now a single round, and view \( M_n \) as a mesh of \( N \times N \) windows \( \hat{M}_N \). In order to route the permutation of worms on \( \hat{M}_N \) we apply \( A(N) \), the packet routing algorithm on \( M_N \). That is, each round of \( \hat{A} \) simulates the full algorithm \( A(N) \) on the (partial) permutation of worms.

The simulation of \( A(N) \) (a single round) is done 'step' by 'step'. As we have mentioned before, we want that prior to the beginning of each 'step' the worms should be packed in the windows as described. We will make sure that this holds at the end of each simulated 'step'. However, we need to ensure for each round that this is also the case prior to the beginning of the first 'step'. This requires an initialization at the beginning of each round. Namely, in each window if there is a worm that is supposed to take part in the round that is supposed to begin, it is born and packs itself in one of the submeshes. To do that, note that the four directed snakelike paths in the submeshes can
be extended to a directed cycle that passes through all points of the window and passes through each edge at most once in each direction. This is done by connecting the head of each snakelike path to the tail of the path that is in the next anticlockwise direction, through the segment and the lanes. The details are left for the reader.

The initialization process is as follows: A worm that is born will start moving along this cycle until it finds itself packed in the first new submesh that it enters. Observe that in a given round, the relative location inside the window is the same for all newborn worms. Hence white worms have the same path length while black worms have one additional step to go. White worms start this initialization one step after the black worms in order to finish this initialization at the same time. The initialization phase takes at most $2k + o(k)$ steps.

The simulation of a 'step' of $A(N)$:
Each 'step' from the viewpoint of some arbitrary node $v$ in $M_N$ is as follows. At the beginning of the 'step' $v$ contains up to four packets $c_1, \ldots, c_k$. Some of these packets leave $v$ in different directions, and $d_1, \ldots, d_l$ stay. During the 'step' more packets may enter $v$ from $r$ different directions, say $a_1, \ldots, a_r$ where $r + l \leq 4$. A packet that arrives to its destination $v$, is 'absorbed' in the output buffer (unless the algorithm $A$ forbids this until some later 'step'). We denote by $\hat{v}$ the window that corresponds to a node $v \in M_N$, similarly we denote the corresponding worms in $\hat{A}(n)$ by $\hat{c}$’s $\hat{a}$’s and $\hat{d}$’s.

The simulation is done in three phases.

Phase 1. The worms $\hat{c}_1, \ldots, \hat{c}_k$ switch places in order to prepare for a conflict free exit.

Phase 2. $\hat{c}_1, \ldots, \hat{c}_k$ exit to neighboring windows, while $\hat{a}_1, \ldots, \hat{a}_r$ enter from neighboring windows.

Phase 3. Worms move around in the window so that a worm that is supposed to be absorbed in its destination can reach its real destination and be absorbed there.

We next describe these phases in details.

Phase 1: Rearranging. At the beginning of this phase the worms are packed on the snakelike path as in Figure 1 in some arbitrary order. We want to arrange them so that each worm will exit along the segment that is assigned to the submesh in which it is packed. Thus, in order for the algorithm $\hat{A}$ to send the worms out of $\hat{v}$ in the directions that $A$ sends the packets out of $v$, they should be permuted among the submeshes. The $3 \times 3$ empty space around the center of the window guarantees enough space for all $4!$ permutations to be performed without conflicts. This is done as follows.

First, all worms in the window $\hat{v}$ move towards the center so that all the headers hit it at the same time. The center then emulates the decision procedure that is taken by $v$ in $A$: Which worms stay, which exit, and in what directions. Once this is decided, the center sends each worm to the submesh from which it can exit in the desired direction during the next phase. Approaching the center is done via the assigned segments, i.e., each worm goes towards its assigned segment and then along this segment to the center. Exiting the center is done along the segments assigned to
the target submesh. There are no conflicts since different worms aim towards different submeshes. After leaving the center, a worm moves along the segment of its target submesh to its end and packs itself through the tail of the snakelike path, see Figure 2.

![Diagram of worm movement](image)

**Figure 2**: The paths used during Phase 1 by a worm which originates at the upper left submesh.

Observe that all the worms in the above maneuver have to travel the same distance, $k + \sqrt{k} + 8$. This forces all the worms in a window to arrive at their new places at the same time. Finally, observe that the length of the paths is the same for black windows and for white windows, which makes all worms in all windows finish Phase 1 at the same time.

**Phase 2: Exit Window.** A worm $e_{out}$ which needs to exit the window goes (shortest path) to its assigned highway lane and then moves on it towards its destination window. Moving on the lane takes each worm to the direction it needs to go. Once a worm enters the neighboring window it moves from the lane to the parallel segment on which it travels through the center and is directed to a submesh as in phase 1. This movement is depicted in Figure 3.

At most four worms reach the center at this stage via the segments. The center directs the worms to empty submeshes.

Observe that worms that switch windows exchange the color of their window. Worms originating in white windows have to go for $k + 3\sqrt{k} + 10$ steps until they are packed in a black window. Worms that originate in black windows have to travel for only $k + 3\sqrt{k} + 8$ steps. Worms that have to stay in their previous window have 0 steps to travel. In order to conform with the hot potato paradigm, we slow down the latter ones, by making them move along extra edges. The traversal of the unnecessary edge is followed by an immediate return. In this way worms are always slowed
Figure 3: The paths followed by worms that exit and enter a window during Phase 2. A worm which exits the window does not move via the center. Worms that stay in place slow themselves down by using delay loops.

down by an even number of steps. This maneuver is called a delay loop, see Figure 3. We force worms which stay in place to move for \( k + \sqrt{k} \) steps by directing them along a cycle which uses the nearest highway lane. \( 2\sqrt{k} + 8 \) additional steps can be spent on delay loops. We conclude that Phase 2 is completed by all the live worms at the same time.

**Phase 3: The Death of a Worm.** This phase guarantees that if a packet is supposed to be absorbed in the current 'step' of the algorithm \( A(N) \) then its corresponding worm will be absorbed too (die). Recall that there is a directed cycle that passes through all the vertices in a window. The worms move along this cycle until the one that has to be absorbed reaches its destination. Once two full cycles have been completed, a worm that has to die has been absorbed while other worms are properly packed.

Clearly each of the three phases in the simulation of a 'step' of \( A \) takes \( O(k) \) steps. There are \( t_A(N) \) such 'steps' thus a round takes \( O(k t_A(N)) \) steps and the whole algorithm takes \( O(k^3 t_A(N)) \)
2.1 Comments:

Many-to-One Routing
Observe that in the 'step' by 'step' simulation, the only requirement from algorithm $A(N)$ is that it uses buffers of size at most four, and the "locality of decisions". We have restricted ourselves to permutation routing only to get the performance asserted in corollary 2.1. However, the simulation result is in fact stronger and we get the following corollary.

**Corollary 2.2** Let $A(N)$ be a packet routing algorithm that meets assumptions 2.1 and 2.3. Let $t_A(N)$ be the complexity of $A(N)$ for a given class of mappings, then there is a hot potato worm routing algorithm $\hat{A}$ that can route any mapping of $k$-worms from the same class, on $M_n$, in time $O(k^3 t_A(\frac{n}{k}))$.

Off-Line routing
We present here an algorithm based on the same ideas as before that can route each permutation in $O(k^{1.5} n)$ provided that there is some global knowledge of the permutation. In other words, if offline precomputation is allowed before the routing starts.

**Theorem 2.2** Routing a permutation of $k$-worms can be completed in $O(k^{1.5} n)$ steps on $M_n$ if 'global' offline precomputation is allowed.

**Proof:** Consider the mesh $M_n$ as a mesh $M_N$ of windows as before. Each window is about to produce up to $h = O(k)$ worms and about to receive up to $h = O(k)$ worms. Consider the bipartite graph in which each side contains a copy of the vertices of $M_N$ and there is an edge between a vertex $v$ of the first part to $w$ in the second part if there is a worm to be routed from $v$ to $w$. This bipartite graph has degree bounded by $h$ thus the set of edges can be partitioned into up to $2h$ matchings [12], see also [17]. In fact even a 'greedy coloring' can ensure a partition of at most $2h$ matchings. Now each such matching defines a partial permutation which can be routed by one round of the algorithm $\hat{A}$ in $O(\sqrt{k} n)$ steps by using an $O(N)$ store-and-forward packet routing algorithm. There are $h = O(k)$ partial permutations to complete the routing which gives $O(k^{1.5} n)$ for the whole problem.

Two remarks are due here: First note that the proper partition into $h$ parts can be efficiently done (a reduction to matching), the partition to $2h$ parts is even simpler as it is 'greedy'. Thus if it is the case that such offline situation is relevant, the algorithm can be implemented efficiently. We also remark that randomized algorithms for worm routing that route in expected time of $O(k n)$ are known [3]. This proves the existence (by standard arguments) of an offline algorithm of the same complexity, however this is not constructive.

$k - l$ Routing with constant size buffers:
A $k - l$ routing problem is such that up to $k$ packets originate and up to $l$ packets are destined to
A \(k-l\) routing algorithm is said to use constant size working buffers, if each node has constant size buffer in which newly entering packets can be stored. The packets originating from a node can only be fetched out of its input buffer, but no packet can be put there.

**Corollary 2.3** (1) Let \(1 \leq k \leq n^2/5\) and \(A(N)\) be a \(1-k\) packet routing algorithm of complexity \(t_A(N)\), that meets Assumptions 2.1 and 2.3, then there is a hot potato worm routing algorithm \(\hat{A}\) for \(M_n\), that routes \(k\)-worms in time \(O(k^d t_A(\frac{r}{k}))\).

(2) Let \(1 \leq k \leq n^2/5\) and \(A(N)\) be a \(k-k\) packet routing algorithm of complexity \(t_A(N)\), using a working buffer of size four, that meets Assumption 2.3, then there is a hot potato worm routing algorithm \(\hat{A}\) for \(M_n\), that routes \(k\)-worms in time \(O(k t_A(\frac{n}{k}))\).

**Proof:** The first claim follows by the same simulation as before, except that we use only \((2\sqrt{k} + 3)^2 = O(k)\) rounds: In the \((i,j)\)-th round we route the worms that originate in the \((i,j)\) position of the windows.

For the second claim a single round suffices. An initialization phase is required, in which every window center collects the destinations of all worms in the window and acts as the decision maker thereafter.

Motivated by this work, recently Kaufmann and Sibeyn have developed \(1-k\) and \(k-k\) packet routing algorithms using working buffers of size three and time bounds of \(O(\sqrt{k} n + k^{5/8} n^{3/4})\) and \(O(k n)\) respectively [26]. Plugging these results into our simulation one gets \(O(k^{1.5} n)\) a hot potato \(k\)-worm routing algorithm.

## 3 Routing on the Hypercube

We denote by \(C^n\) the \(n\)-dimensional \(2^n\)-nodes Boolean hypercube. Each node is identified with an element of \(\{0,1\}^n\) and two vertices are connected if their Hamming distance is one.

We are not aware of any lower bound for permutation worm routing other than the trivial \(\Omega(k+n)\) steps for routing \(k\)-worms on the hypercube. If we bound the power of the processors, so that each can handle only a constant number of worms in each step, then a lower bound of \(\Omega(k^{\sqrt{n}})\) steps follows by a bisection argument.

As for upper bounds, we are not aware of any result except the randomized algorithm of Felperin et. al. [10] on the butterfly. This algorithm does not comply with the hot potato paradigm.

As in the mesh, we present deterministic worm routing algorithms for the \(n\)-dimensional hypercube \(C^n\), by simulating an algorithm for the packet routing problem. Here too we require that the simulated packet routing algorithm is restricted. Let \(A(n)\) be a packet routing algorithm for the \(n\)-dimensional Boolean hypercube \(C^n\). We will need the following assumptions on \(A(n)\).

**Assumption 3.1** \(A(n)\) uses at each node a buffer of size at most \(b\), where \(b\) is some constant independent of \(n\).
Assumption 3.2 $A(n)$ can route any partial permutation in $t_A(n)$ steps.

Assumption 3.3 At every step of Algorithm $A(n)$, the decisions when, where and if to send a packet is taken locally by the node in which it resides without considering the contents of any other node.

Fact 3.1 For every $n$ there is a packet routing algorithm $A(n)$ for $C^n$ that meets the above assumptions with $b$ and $t_A(n)$ as follows:

1) Deterministic, with $b = 2$, and whose running time $t_A(n) = O(n \log^2 n)$.

2) Randomized, with $b = O(1)$, and whose running time $t_A(n) = O(n)$ with probability of success $\geq 1 - \frac{1}{2^n}$ for any constant $c$.

As remarked for the mesh, most packet routing algorithms that appear in the literature can be made to follow Assumption 3.3, slowing them down by at most a factor of two. Thus, (1) follows from reducing routing to sorting, and using the algorithm of Plaxton [22]. There is a subtlety here, as Plaxton’s sorting algorithm requires replication of keys. The extra copies are eventually being “deleted”. We assume that replication can be performed on headers of worms at individual nodes, and furthermore that a duplicate header can be recognized as such. Duplicate headers travel in the network with no body behind, which is sufficient for sorting purposes. If this assumption is unacceptable, then we have to use the Batcher sorting algorithm instead [4], with a running time of $t_A(n) = O(n^3)$. (2) follows by the results of Leighton et. al. [16].

Theorem 3.1 Let $A(n)$ be a packet routing algorithm for $C^n$ which meets the assumptions and let $r$ be the largest integer such that $2^{n-r-1} \geq k$, where $b$ is the buffer size of the algorithm $A(r)$. Then for any $1 \leq k \leq 2^{n-r-1}$ there is a hot potato $k$-worm routing algorithm $B(n,k)$ that can route any (partial) permutation in $O(k^3t_A(n))$ steps.

We conclude the following corollary.

Corollary 3.1 There is a hot potato algorithm for routing $k$-worms on $C^n$, $1 \leq k \leq 2^{n-r-1}$, which routes any (partial) permutation in the following time bounds:

1) Deterministically, in $O(k^3n \log^2 n)$.

2) Randomized, in $O(k^3n)$ steps with probability of success $\geq 1 - \frac{1}{2^n}$, for any constant $c$.

Proof of Theorem 3.1: Let $A(r)$ be a packet routing algorithm that meets the assumptions and let $r$ be the largest integer such that $2^{n-r-1} \geq k$, where $b$ is the buffer size of the algorithm $A(r)$. Intuitively, we view the $n$-dimensional hypercube as being an $r$-dimensional hypercube so that every ’point’ in $C^r$ will be able to “store” $b$ $k$-worms. This allows us to pack $k$-worms into one virtual node of $C^r$ and thus reduce the problem to the traditional packet routing problem on $C^r$. 
Formally, we identify the points of $C^n$ with the set of all $n$ binary strings. We think of $C^n$ as $C^r \times C^{n-r}$. This defines an “embedding” of $C^r$ in $C^n$ in the following way: A point $w \in C^r$ is represented by a 'point' $\hat{w} = \{wu | u \in \{0,1\}^{n-r}\}$, which is a subcube of $C^n$. We identify the edges of $C^r$ by: the edge $(a,b) \in C^r$ is mapped to $(aw^v, bw^v) \in C^n$. Thus the point $w0^{n-r}$ in each 'point' $\hat{w}$ will handle the communication to and from $\hat{w}$. In this way we view $C^n$ as an $r$-dimensional cube $\hat{C}^r$ of 'points' $\hat{w}$.

We further refine a 'point' $\hat{w} \in \hat{C}^r$ as follows. We want such a point to contain $b$ worms of size $k$. We do that by partitioning the set $\hat{w} = \{wu | u \in \{0,1\}^{n-r}\}$ into $b+1$ sets. The first $b$ sets are $W_i = \{w1e_iu | v \in \{0,1\}^{n-r-k-1}\}$, $1 \leq i \leq b$, where $e_i$ is the $i$th unit vector of dimension $b$ (i.e., $e_i = 0^{i-1}10^{b-i}$). The $(b+1)$th set is the rest and plays no role in storing worms, it contains however the point $w0^{n-r}$ which channels the communication to and from the 'point' $\hat{w}$. Each $W_i$ is of size $2^{n-r-k-1}$ and is capable to store a whole worm. Every subcube $W_i$ contains a Hamiltonian cycle. The $i$-th worm, $i \leq b$ is stored contiguously along the Hamiltonian cycle of $W_i$. Here too, we distinguish between 'black' 'points' and 'white' 'points' according to the color classes of $C^r$ as a bipartite graph. The worms are stored in $W_i$ with their heads at $w1e_i0^{n-r-k-1}$ if $w$ is black, and at $w0e_i0^{n-r-k-1}$ if $w$ is white.

We now describe the worm routing algorithm $B(n,k)$. The algorithm is divided into $2^{2(n-r)} = O(k^2)$ rounds. In the round $(u,v)$, $u \in C^{n-r}$, $v \in C^{n-r}$ we route only these worms that have as their source a point of the form $wu$ for some $r$ string $w$ and whose destination is of the form $xv$ for some $r$ string $x$. All worms that belong to previous rounds already arrived at their destinations and are not in the system anymore. All the worms which belong to future rounds are not yet born. Note that for permutation routing, as in the mesh, this ensures that during each round, at most one worm is born at each 'point' $\hat{w} \in \hat{C}^r$, and at most one worm is destined to each 'point'. In other words, in each round we route a (partial) permutation of worms between the 'points'. We do that by simulating the algorithm $A(r)$ on $\hat{C}^r$. Again we refer to 'steps' for the steps of the algorithm $A(r)$ while real steps are just steps. The Algorithm $A(r)$ is simulated 'step' by 'step'.

Prior to the beginning of each 'step' the worms should be packed in the subcubes as described. We will make sure that this holds at the end of each simulated 'step'. We need to ensure that this holds prior to the first 'step' too. Observe that the Hamiltonian cycles in the subcubes $W_i$, $1 \leq i \leq b$, of a 'point' $\hat{w}$ can be augmented into a cycle that passes through all the vertices of $\hat{w}$ using each edge (of $C^n$) at most once in each direction. We call this cycle $C(\hat{w})$ for future reference. Each worm travels along $C(\hat{w})$ until it 'packs' itself properly in the first $W_i$ it enters along this cycle. Since all the worms are born in the same relative position in the beginning of a round, black worms start this movement one step after the white worms (i.e., black worms are born with one step of delay). This ensures that all the worms arrive to their proper place at the same time.

**The simulation of a 'step' of $A(r)$:**
A typical 'step' of $A(r)$ is the following: Each site $w$ of $C^r$ receives $p$ packets from its neighbors (at most one packet from each). It also sends $q$ of its current packets, each to a different neighbor and stores $t$ of its current packets where $t + p \leq b$. Observe that the neighbors of a white 'point'
are black and visa versa. Let $\hat{w} \in \hat{C}_r$ be a black 'point'. The movement of the $i$th worm from $\hat{w}$ into the $j$th subcube $U_j$ of a neighbor 'point' $\hat{n}$ will be as follows (we describe only the movement of the header as the worm follows its header): It starts at $w1e_i0^{n-r-b-1}$ and goes

$$w1e_i0^{n-r-b-1} \rightarrow w0e_i0^{n-r-b-1} \rightarrow w0^{n-r} \rightarrow u0^{n-r} \rightarrow u0e_j0^{n-r-b-1}$$

in four steps. Movement from a white point to a black one is done using the same path in the opposite direction. A worm that has to stay at a black site moves through

$$w1e_i0^{n-r-b-1} \rightarrow w0e_i0^{n-r-b-1} \rightarrow w0^{n-r} \rightarrow w0e_j0^{n-r-b-1} \rightarrow w1e_i0^{n-r-b-1}$$

Similarly a white worm that has to stay in its site moves in two steps to $u10^{n-r-b-1}$ and in two more steps back to its place. Note that after four such steps, disregarding the origin, each worm header is in its proper place. The worm then moves along the Hamiltonian cycle in its subcube $W_i$ for $2^{n-r-b-1}$ steps (a complete cycle) in order for the body of the worm to get into $W_i$ and pack itself as needed.

In order to complete the simulation of a 'step' we have to take care of the situation where a worm that has just arrived at a point $\hat{w} \in \hat{C}_r$ might correspond to a packet that in the algorithm $A(r)$, arrives to its final destination at $w$, and is absorbed there. Such a worm has to reach its true final destination in $\hat{w}$, and be absorbed there. To allow this recall the cycle $C(\hat{w})$ that passes through all the vertices of $\hat{w}$. The $b$ worms in each subcube $\hat{w}$ move along $C(\hat{w})$ for $2^{n-r+1}$ steps (two cycles). A worm that has to be absorbed reaches is destination along this tour and is absorbed. Other worms eventually return to their original place.

The simulation of a step of $A(r)$ takes at most $2^{n-b-r} + 2^{n-r+1} + 4 = O(k)$. Thus a round takes at most $O(k \cdot t_A(r))$, where $t_A(r)$ is the running time of $A(r)$ and the whole algorithm takes $O(k^3 t_A(r)) = O(k^3 t_A(n))$ for constant $b$.

**Remark:** The same comments as in Section 2.1 apply here too.

## 4 Concluding Remarks and Open Problems:

We have shown how to simulate packet routing algorithms in order to give hot potato worm routing algorithms for the mesh and the boolean hypercube. In the description we focused on permutation routing. However, the simulation method is fairly general, and may be applied to more general communication patterns as discussed at the end of Section 2. It can also be applied to other types of networks.

The algorithms introduced in this paper are structured in nature, thus suitable for synchronous batch routing. A major open problem for worm routing (as in the hot potato packet routing model) is to devise efficient unstructured algorithms. In particular, this goal is important for worm routing, since there is a gap of at least $\sqrt{k}$ between the lower bound and the best possible algorithm that can be obtained using our methods. For the Hypercube the situation is even worse.
References


