Tight Size Bounds for Packet Headers in Narrow Meshes*

Micah Adler[1], Faith Fich[2], Leslie Ann Goldberg[3], and Mike Paterson[3]

[1] micah@cs.umass.edu, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA.
[2] fich@cs.toronto.edu, Department of Computer Science, University of Toronto, Toronto, Canada, M5S 3G4.†
[3] (leslie,msp)@dcs.warwick.ac.uk, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK.‡

Abstract. Consider the problem of sending a single message from a sender to a receiver through an $m \times n$ mesh with asynchronous links that may stop working, and memoryless intermediate nodes. We prove that for $m \in O(1)$, it is necessary and sufficient to use packet headers that are $\Theta(\log \log n)$ bits long.

1 Introduction

Protocols that send information bundled into packets over a communication network allocate some number of bits in each packet for transmitting control information. We here refer to such bits as *header bits*. These bits might include sequence numbers to ensure that packets are received in the correct order, or they might contain routing information to ensure that a packet is delivered to its destination. When the number of message bits in a packet is small (for example, in acknowledgements), the header bits can make up a significant fraction of the total number of bits contained in the packet. A natural question to ask is the following: how large must packet headers be for reliable communication?

This problem is addressed in [AF99], part of a large body of research on the end-to-end communication problem [AAF+94], [AAG+97], [AG88], [AMS89], [APV96], [DW97], [KOR95], [LLT98]. The *end-to-end communication* problem is to send information from one designated processor (the *sender* S) to another designated processor (the *receiver* R) over an unreliable communication network. This is a fundamental problem in distributed computing, since (a) communication is crucial to distributed computing and (b) as the size of a network increases, the likelihood of a fault occurring somewhere in the network also increases.

* Research Report 366, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
† This work was partly supported by grants from NSERC and CITO.
‡ This work was partly supported by EPSRC grant GR/L60982 and ESPRIT LTR Project ALCOM-FT.

Adler and Fich [AF99] studied the question of how many header bits are required for end-to-end communication in the setting where there is a single message to be sent from S to R. They prove that, for the complete network of n processors or any network that contains it as a minor (such as the n^2-input butterfly or the $n \times n \times 2$ mesh), any memoryless protocol that ensures delivery of a single message using headers with fewer than $\lceil \log_2 n \rceil - 3$ bits, generates an infinite amount of message traffic.

If there is a path of live links from S to R in an n-node network, then there is a simple such path of length at most $n - 1$. Therefore, it suffices to use the simple "hop count" algorithm: use packet headers of length $\lceil \log_2(n-1) \rceil$ to count the number of links that packets have travelled [P81]. Thus, for the complete graph, we have upper and lower bounds that match to within a small additive constant, and, for the n^2-input butterfly and the $n \times n \times 2$ mesh, to within a small multiplicative constant.

However, for several graphs there remains a large gap between the best upper and lower bounds. Planar graphs, including two-dimensional meshes, do not contain a complete graph on more than 4 nodes as a minor [K30], and as a result, no previous work has demonstrated a lower bound larger than a constant for any planar graph. Furthermore, for some networks it is possible to do better than the simple hop count algorithm. For example, Adler and Fich [AF99] observed that, for any feedback vertex set F in a graph G, any simple path visits vertices in F at most $|F|$ times and they obtained a variant of the hop count protocol that uses packet headers of length $\lceil \log_2(|F|+1) \rceil$. However, some graphs have no small feedback vertex sets. In particular, any feedback vertex set for the $m \times n$ mesh has size at least $\lfloor m/2 \rfloor \cdot \lfloor n/2 \rfloor$. In this case, this variant does not offer significant improvement over the hop count algorithm.

Thus, we see that a network that has resisted both lower bound and upper bound improvements is the two-dimensional mesh. Prior to this work, there was no upper bound better than $O(\log mn)$, nor lower bound better than $\Omega(1)$, for any $m \times n$ mesh with $m, n > 2$. Note that for $m = 2$, headers of length one suffice (to indicate which neighbour sent the packet) [AF99]. In [AF99], it is conjectured that $\Omega(\log n)$ header bits are necessary for a protocol to ensure delivery of a single message in an $n \times n$ mesh without generating an infinite amount of message traffic.

Here, we attack this open problem by considering $m \times n$ meshes, for constant $m \geq 3$. We prove the unexpected result that $\Theta(\log \log n)$ bit headers are necessary and sufficient for such graphs.

1.1 Network Model

We model a network by an undirected graph G, with a node corresponding to each processor and an edge corresponding to a link between two processors. Specifically, we consider the graphs $G(m, n)$ with a sender node S and a receiver node R in addition to the mn intermediate nodes, (i, j), for $0 \leq i < m$ and $0 \leq j < n$. There are links between

– node S and node $(i, 0)$, for $0 \leq i < m$,
– node (i, j) and node $(i, j + 1)$, for $0 \leq i < m$ and $0 \leq j < n - 1$,
– node (i, j) and node $(i + 1, j)$, for $0 \leq i < m - 1$ and $0 \leq j < n$, and
– node $(i, n - 1)$ and node R, for $0 \leq i < m$.

The graph $G(3, 6)$ is illustrated in Figure 1.

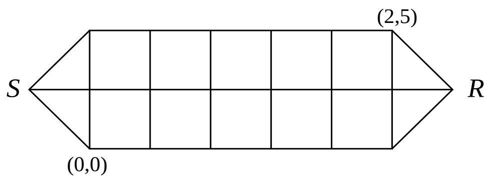

Fig. 1. The graph $G(3, 6)$

Processors communicate by sending packets along links in the network. Each packet consists of data (i.e. the message) and a header. The processor at an intermediate node may use information in the header to determine what packets to send to its neighbours, but they cannot use the data for this purpose. Furthermore, headers may be modified arbitrarily; however, data must be treated as a "black box". This *data-oblivious* assumption is appropriate when one views end-to-end communication protocols as providing a reliable communication layer that will be used by many different distributed algorithms. We also assume that a processor cannot detect from which of its neighbours a packet was sent. Since the degree of the graphs is constant, this is not important: when each processor sends a packet, it can indicate the direction of travel using a constant number of header bits.

Intermediate processors are assumed to be *memoryless*, and thus processors can only send packets as a result of receiving a packet and must decide along which link(s) to forward the message and how to change the packet header, based only on the contents of the header. This is an appropriate model for a network with simultaneous traffic between many different pairs of processors, for example, the Internet, where no information concerning past traffic is stored.

The links of the network are either *alive* or *dead*. At any time, a live link may become dead. However, once a link becomes dead, it remains so. Processors do not know which subset of the links are alive.

To simplify our protocols, we will adopt the convention that packets "sent" to non-existent nodes are ignored.

Live links deliver packets in a first in, first out manner. However, the time for a packet to traverse a link may differ at different times or for different links. We assume that the time for a packet to traverse a link is finite, but unbounded. Edges which are dead can be thought of as having infinite delay. In this asynchronous model, a processor cannot distinguish between an dead link and a link which is just very slow.

1.2 Summary of Results

In this paper, we consider the problem of sending a single message from S to R. Our goal is to ensure that

– as long as there is some simple S–R path of live links, at least one copy of the message gets sent from S to R, and
– even if all links are alive, only a finite number of packets are generated.

We say that a protocol which satisfies these requirements *delivers a message from S to R with finite traffic.* In this paper, we provide an algorithm that does this using $O(m \log\log n)$-bit headers for any network $G(m,n)$. For the case of $G(3,n)$, this is improved to $\log_2 \log_2 n + O(1)$. Furthermore, we demonstrate that for $G(3,n)$, $\log_2 \log_2 n - O(\log\log\log n)$ bits are required. Thus, for any constant $m \geq 3$, we have optimal bounds to within a constant factor on the number of header bits that are necessary and sufficient to deliver a message from S to R with finite traffic in $G(m,n)$. For the case of $G(3,n)$, our bounds are within an additive term of $O(\log\log\log n)$ from optimal.

Our upper bounds use a new technique to obtain an approximate count of how many nodes a message has visited, which is sufficient to guarantee that only a finite number of packets are generated. This technique may have applications to other networks.

Our upper bounds also provide upper bounds for any graphs that are minors of $G(m,n)$, for any constant m. Similarly, we get lower bounds for any graphs that contain $G(3,n)$ as a minor. These are consequences of the following observation from Adler and Fich [AF99].

Proposition 1. *Suppose G' is a minor of G and S' and R' are the supernodes of G' containing S and R, respectively. Then any protocol for G that delivers a message from S to R with finite traffic gives a protocol for G' with the same packet headers that delivers a message from S' to R' with finite traffic.*

The protocol for $G(3,n)$ is given in the next section. Section 3 extends this result to $G(m,n)$ for any constant $m \geq 3$. This is followed in Section 4 by our lower bound for $G(3,n)$ and, hence, for $G(m,n)$ with $m > 3$, which contains $G(3,n)$ as a minor.

2 A Protocol for $G(3,n)$

In this section, we present a protocol using $O(\log\log n)$ header bits that delivers a message from S to R with finite traffic in $G(3,n)$. Throughout the protocol, each packet will contain the message as its data. Consequently, we mention only the header bits in what follows.

We describe the protocol in pieces. We first consider some simple paths in the graph, and focus on paths that move "right", that is, on paths from (r_1, c_1) to (r_2, c_2) with $c_2 \geq c_1$. There are four types of paths: U, D, S, and Z. For each of these paths, there is a simple communication protocol, such that when a packet with header "Ready" enters the first node (r_1, c_1) of the path, and all of the edges in the path are alive, the protocol sends a packet with header "Ready" from the last node of the path to the node $(r_2, c_2 + 1)$. The protocol uses a finite number of header bits.

A U-*path* (D-*path*) consists of zero, one or two upward (downward respectively) edges. The protocol for a U-path is as follows.

If (r, c) receives a packet with header "Ready" or "up", it sends a packet with header "Ready"to $(r, c + 1)$, and a packet to $(r + 1, c)$ with header "up".

Note that the U-path protocol only generates a finite number of packets, and results in a "Ready" packet at the end of the path. The protocol for D-paths is analogous, substituting "down" and $r - 1$ for "up" and $r + 1$.

An *S-path of extent* $j \geq 1$ is a path from $(0, c)$ to $(2, c + j - 1)$. It consists of

- A left-to-right path of length $j - 1$ along the bottom row from $(0, c)$ to $(0, c + j - 1)$, followed by
- the vertical edge from $(0, c + j - 1)$ to $(1, c + j - 1)$, followed by
- a right-to-left path of length $j - 1$ along the middle row from $(1, c + j - 1)$ to $(1, c)$, followed by
- the vertical edge from $(1, c)$ to $(2, c)$, followed by
- a left-to-right path of length $j-1$ along the top row from $(2, c)$ to $(2, c+j-1)$.

Thus, an S-path of extent j contains $3(j - 1)$ horizontal edges and 2 vertical edges, for a total length of $3j - 1$. Similarly, a Z-*path of extent* j is a simple path of total length $3j - 1$ from $(2, c)$ to $(2, c + j - 1)$, to $(1, c + j - 1)$, to $(1, c)$, to $(0, c)$, and finally to $(0, c + j - 1)$.

Our communication protocol for an S-path of extent j has the property that when a packet enters the first node $(0, c)$ with header "Ready", and all of the edges are alive, the protocol sends a packet with header "Ready" from the last node to $(2, c + j)$. The protocol will use $O(\log \log n)$ header bits. Furthermore, it will only generate a finite number of packets. It may result in more than one new packet with header "Ready", but each of these new packets will arrive at a node which is *to the right of* column c. Thus, only a finite number of packets will be generated overall, so we will be able to combine the protocol with our other protocols to achieve our two goals.

For any nonnegative integer k, we say that column c is a k-*counter* if and only if $c = 0 \bmod 2^k$. In particular, every column is a 0-counter and column 0 is a k-counter for all $k \geq 0$.

The protocol for S-paths of extent greater than one is as follows.

- If $(0, c)$ receives a packet with header "Ready", then for each value $k \in \{1, \ldots, \lceil \log_2 n \rceil\}$, it sends a packet to $(0, c+1)$ with header (S, k, "unmarked").
- If $(0, c)$ receives a packet with header (S, k, "unmarked"), it sends a packet to $(0, c+1)$ with header (S, k, x), where $x =$ "marked" if c is a k-counter and $x =$ "unmarked" otherwise. In addition, if c is a k-counter, then $(0, c)$ also sends a packet to $(1, c)$ with header (S, k, "up").
- If $(0, c)$ receives a packet with header (S, k, "marked") and c is not a k-counter, it sends a packet to $(1, c)$ with header (S, k, "up"), and a packet to $(0, c+1)$ with header (S, k, "marked").
- If $(1, c)$ receives a packet with header (S, k, "up"), it sends a packet to $(1, c-1)$ with header (S, k, "unmarked") if c is not a k-counter and with header (S, k, "marked") if c is a k-counter.
- If $(1, c)$ receives a packet with header (S, k, "unmarked"), it sends a packet to $(1, c-1)$ with header (S, k, x), where $x =$ "marked" if c is a k-counter and $x =$ "unmarked" otherwise. In addition, if c is a k-counter then $(1, c)$ also sends a packet to $(2, c)$ with header (S, k, "up").
- If $(1, c)$ receives a packet with header (S, k, "marked"), it sends a packet to $(2, c)$ with header (S, k, "up"). If c is not a k-counter, it also sends a packet to $(1, c-1)$ with header (S, k, "marked").
- If $(2, c)$ receives a packet with header (S, k, "up"), it sends a packet to $(2, c+1)$ with header (S, k, "unmarked").
- If $(2, c)$ receives a packet with header (S, k, "unmarked"), it sends a packet to $(2, c+1)$ with header (S, k, x), where $x =$ "marked" if c is a k-counter and $x =$ "unmarked" otherwise. If c is a k-counter, $(2, c)$ it also sends a packet to $(2, c+1)$ with header "Ready".
- If $(2, c)$ receives a packet with header (S, k, "marked"), it sends a packet to $(2, c+1)$ with header "Ready". If c is not a k-counter, it also sends a packet to $(2, c+1)$ with header (S, k, "marked").

Lemma 1. *Suppose that the S-path communication protocol is run as a result of a packet with header "Ready" arriving at* $(0, c)$*. Then*

(A) For $1 < j \leq n - c + 1$*, if all of the edges in the S-path of extent* j *from* $(0, c)$ *to* $(2, c + j - 1)$ *are alive, then a packet with header "Ready" is sent to* $(2, c + j)$*.*

(B) The only new packets which are generated with header "Ready" have destinations in columns which are to the right of column c*.*

Proof. We first prove (A). Suppose that there is a $k \in \{1, \ldots, \lceil \log_2 n \rceil\}$ such that exactly *one* column c' in $\{c+1, \ldots, c+j-1\}$ is a k-counter. Then it is straightforward to verify that packets travel from $(0, c)$ to $(0, c')$ with header (S, k, "unmarked"), from there to $(0, c+j-1)$ with header (S, k, "marked"), up to $(1, c+j-1)$ with header (S, k, "up"), from there to $(1, c')$ with header (S, k, "unmarked"), from there to $(1, c)$ with header (S, k, "marked"), up to $(2, c)$ with header (S, k, "up"), from there to $(2, c')$ with header (S, k, "unmarked"), from there to $(2, c+j-1)$ with header (S, k, "marked"), and from there to $(2, c+j)$

with header "Ready". We will now verify that such a k exists. Let $k' = \lfloor \log_2 j \rfloor$. Then there are either one or two k'-counters in $\{c+1, \ldots, c+j-1\}$. Suppose that there are two, in columns $m2^{k'}$ and $(m+1)2^{k'}$. Then one of m and $m+1$ is even, so there is exactly one $(k'+1)$-counter in $\{c+1, \ldots, c+j-1\}$.

We now prove (B). Let c' be the first k-counter to the right of c and let c'' be the first k-counter to the left of c'. Then column c' is the leftmost column in which a packet can enter row 1 as a result of a packet with header "Ready" arriving at $(0, c)$. By the time that a packet gets to column c'' in row 1, it is marked. Thus, the leftmost column in row 1 which is reached is column c''. These packets travel right in row 2, but they are unmarked as they enter column c'. Thus, the leftmost column to which a header "Ready" packet is sent is column $c'+1$. □

The communication protocol that we use for Z-paths is analogous to the one that we use for S-paths. We now make the following observation.

Lemma 2. *Every simple path from S to R can be formed by concatenating paths of types U, D, S, and Z, using left-to-right edges.*

Proof. Consider a simple path P from S to R. We first observe that any right-to-left edge in P can only be in row 1. For contradiction, suppose that $((1, c+1), (1, c))$ is a right-to-left edge along row 1. It is easy to see that any path from S to $(1, c+1)$ must intersect any path from $(1, c)$ to R. Since P is simple, we have a contradiction.

Hence every right-to-left edge occurs in row 1 with left-to-right edges immediately above and below it in rows 0 and 2. As a consequence, every consecutive sequence of right-to-left edges occurs as the central section of an S-path or a Z-path. Since any subpath which contains no right-to-left edges is formed by concatenating zero or more U-paths and D-paths with left-to-right edges, the proof is complete. □

Thus, using Lemma 1 (and the corresponding observation for Z-paths), we can prove Theorem 1.

Theorem 1. *There is a protocol which delivers a message from S to R in $G(3, n)$ with finite traffic, using headers of length $\log_2 \log_2 n + O(1)$.*

Proof. Consider the communication protocol that starts by sending a packet with header "Ready" from S to each of its neighbours $(0, 0)$, $(1, 0)$, and $(2, 0)$, that performs the U, D, S, and Z protocols at all intermediate nodes, and that sends a packet to R from its neighbours $(0, n-1)$, $(1, n-1)$, and $(2, n-1)$, whenever they receive a packet.

Since each of the four types of protocol ends by sending a packet with header "Ready" to the right, a packet path can be regarded as a sequence of "basic" paths (of type U, D, S or Z) concatenated by horizontal edges along which these packets are sent. Formally, we can prove that if there is a simple path of live edges from S to R, then R will receive a packet. This is done by induction on the number of basic paths which get concatenated to form the simple path.

Now suppose that all of the edges in the graph are alive. When a node (r, c) receives a packet of type "Ready", the result is a bounded number of new packets of type "Ready" all of which are sent to vertices in columns to the right of column c, and a bounded number of packets of other types. Thus, only a finite number of packets are generated. □

3 A Protocol for $G(m, n)$

In this section, we provide an upper bound on the header size required for sending a single message from S to R in $G(m, n)$. Since $G(m, n)$ is a minor of $G(m, n')$ for all $n \leq n'$, by Proposition 1, it suffices to assume that $n = 2^h + 1$ for some positive integer h.

We begin by giving a characterization of certain simple paths.

Definition 1. *For $r_1 \leq r_2$ and $c_1 \neq c_2$, a (c_1, c_2, r_1, r_2)-*bounded path *is a simple path that starts in column c_1, ends in column c_2, and does not go through any node in a column less than* $\min\{c_1, c_2\}$*, a column greater than* $\max\{c_1, c_2\}$*, a row less than r_1, or a row greater than r_2.*

Note that every simple path from the first column of $G(m, n)$ to the last column of $G(m, n)$ is a $(0, n-1, 0, m-1)$-bounded path. A (c_1, c_2, r, r)-bounded path is a simple path of horizontal edges.

Definition 2. *For $r_1 < r_2$ and $c_1 \neq c_2$, a (c_1, c_2, r_1, r_2)-*bounded loop *is a simple path that starts and ends in column c_1, and does not go through any node in a column less than* $\min\{c_1, c_2\}$*, a column greater than* $\max\{c_1, c_2\}$*, a row less than r_1, or a row greater than r_2.*

We focus attention on bounded paths between columns which are consecutive multiples of some power of 2, i.e. from column $c2^k$ to column $c'2^k$, where $c' = c \pm 1$.

Lemma 3. *Let c_1, c_2, and c_3 be consecutive nonnegative integers, with c_2 odd, and let k be a nonnegative integer. Then every $(c_1 2^k, c_3 2^k, r_1, r_2)$-bounded path can be decomposed into a $(c_1 2^k, c_2 2^k, r_1, r_2)$-bounded path, followed by a series of $r_2 - r_1$ or fewer $(c_2 2^k, c_1 2^k, r_1, r_2)$- and $(c_2 2^k, c_3 2^k, r_1, r_2)$-bounded loops, followed by a $(c_2 2^k, c_3 2^k, r_1, r_2)$-bounded path.*

Proof. Consider any $(c_1 2^k, c_3 2^k, r_1, r_2)$-bounded path. The portion of the path until a node in column $c_2 2^k$ is first encountered is the first subpath, the portion of the path after a node in column $c_2 2^k$ is last encountered is the last subpath, and the remainder of the path is the series of loops. The bound on the number of loops follows from the fact that the path is simple, so the first subpath and each of the loops end on different nodes in column $c_2 2^k$. □

This gives us a recursive decomposition of any simple path from the first column to the last column of $G(m,n)$, where n is one more than a power of 2. Specifically, such a $(0, n-1, 0, m-1)$-bounded path consists of a $(0,(n-1)/2, 0, m-1)$-bounded path, followed by a series of at most $m-1$ different $((n-1)/2, n-1, 0, m-1)$ and $((n-1)/2, 0, 0, m-1)$-bounded loops, followed by a $((n-1)/2, n-1, 0, m-1)$-bounded path. Each of the bounded paths can then be similarly decomposed. Furthermore, we can also decompose the bounded loops.

Lemma 4. *Let k, r_1, r_2, c_1 and c_2 be nonnegative integers, where c_1 and c_2 are consecutive, c_1 is odd, and $r_1 < r_2$. Then every $(c_1 2^k, c_2 2^k, r_1, r_2)$-bounded loop can be decomposed into the prefix of a $(c_1 2^k, c_2 2^k, r_1+1, r_2)$-bounded path, followed by a downward edge, followed by the suffix of a $(c_2 2^k, c_1 2^k, r_1, r_2-1)$-bounded path, or the prefix of a $(c_1 2^k, c_2 2^k, r_1, r_2-1)$-bounded path, followed by an upward edge, followed by the suffix of a $(c_2 2^k, c_1 2^k, r_1+1, r_2)$-bounded path.*

Proof. Consider any $(c_1 2^k, c_2 2^k, r_1, r_2)$ bounded loop. Let c be the column farthest from $c_1 2^k$ that this path reaches and let (r,c) be the first node in this path in column c. Let p_1 be the prefix of this path up to and including node (r,c). The next edge is vertical. Let p_2 be the remainder of the bounded loop following that edge.

Since the loop is a simple path, paths p_1 and p_2 do not intersect. Thus, either p_1 is completely above p_2, so p_1 never uses row r_1 and p_2 never uses row r_2, or p_1 is completely below p_2, so p_1 never uses row r_2 and p_2 never uses row r_1. □

We use this recursive decomposition of simple paths in our protocol. Instead of trying just the simple S–R paths in $G(m,n)$, our protocol tries **all** S–R paths that can be recursively decomposed in this way.

Our basic building block is a protocol that sends a packet from column c_1 to column c_2, where c_1 and c_2 are consecutive multiples of some power of 2, using some set of r adjacent rows. The protocol does this by first sending the packet from column c_1 to the middle column $(c_1+c_2)/2$, recursively. Then it sends the packet looping around the middle column at most $r-1$ times. Each loop consists of a first half and a second half, each of which uses at most $r-1$ rows. Both of these subproblems are solved recursively. Finally, the protocol recursively sends the packet from the middle column to column c_2.

It follows by Lemmas 3 and 4 that, if there is a simple path of live edges from S to R, then our protocol finds it. Note that, at the lowest level of the recursion, a packet is always travelling in what is considered the forward direction (when the bounded path is from right to left, this will be in the backwards direction of the original problem, but still in the forward direction of the lowest level subproblem). Thus, the difficult part of this protocol is performing the bounded loops in such a way that the packet does not travel in an infinite loop.

Let $\#_2(0) = \infty$ and for every positive integer c, let $\#_2(c)$ denote the largest power of two that divides c. Thus, if c can be expressed as $c_1 2^k$ for an odd number c_1, then $\#_2(c) = k$. In our protocol, the packet header is used to keep track of the column in which the current loop started and the distance to the other column boundary. If we naively stored these numbers, then $\Omega(\log n)$ header bits would be required. However, because our decomposition only uses bounded loops of the form $(c_1 2^k, (c_1 \pm 1)2^k, r_1, r_2)$, where c_1 is odd, it is sufficient to keep track of k (i.e., $\#_2(c_1 2^k)$). Note that k can be represented using only $\lceil \log_2 \log_2 (n-1) \rceil$ bits. Using the quantity k, a packet can tell when it reaches its boundary columns. In particular, while its current column c is *between* the boundaries, $\#_2(c) < k$ but when c is at the boundaries $\#_2(c) \geq k$.

When the algorithm is doing a bounded loop from column $c_1 2^k$ the following quantities are stored.

- *power* $= \#_2(c_1 2^k)$ (which is equal to k),
- *minRow*, the smallest row that can be used,
- *maxRow*, the largest row that can be used,
- *loopCounter*, the number of loops that have already been done around column $c_1 2^k$ in the current path,
- *loopHalf* (0 if the current packet is in the first bounded path that forms this loop and +1 if it is in the second),
- *forward*, the direction in which the packet is travelling on the current path (+1 if the packet is going from left to right and −1 it is going from right to left).

Although our path decomposition has $\log_2(n-1)$ levels of recursion, at most m loops can be active at any one time. This follows from Lemma 4, since the number of allowed rows decreases by 1 for each active loop. We shall think of the bits in the packet header as a stack and, for each active loop, the above mentioned variables will be pushed onto the stack. Finally, we use two additional bits with each transmission to ensure that any node receiving a packet knows where that packet came from. In total, our protocol uses headers with at most $O(m(\log \log n + \log m))$ bits.

At the start, S sends a packet to each node in column 0. The header of each packet contains the following information in its only stack entry: *power* $= \log_2(n-1)$, *minRow* $= 0$, *maxRow* $= m-1$, *forward* $= 1$, *loopHalf* $= 1$, and *loopCounter* $= 0$. (To be consistent with other levels of recursion, we are thinking of the path from column 0 to column $n-1$ as being the second half of a $(n-1, 0, 0, m-1)$-bounded loop.)

We shall refer to the variable $d = m - maxRow + minRow$, which is equal to the recursion depth. We describe the actions of any node (r, c) that does not appear in the first or last column of $G(m, n)$. The actions of the nodes in the first (or last) column are identical, except that they do not perform the specified forwarding of packets to the left (or right, respectively). In addition, if a node in the last column of $G(m, n)$ ever receives a packet, it forwards that packet to R.

Protocol DELIVER

On receipt of a packet at node (r, c) with (*power*, *minRow*, *maxRow*, *loopCounter*, *loopHalf*, *forward*) at the top of its stack

/* The default move is to forward a packet up, down, and in the current direction of travel. */

- If $r < maxRow$ and the packet was not received from node $(r + 1, c)$, send the packet to node $(r + 1, c)$.
- If $r > minRow$ and the packet was not received from node $(r - 1, c)$, send the packet to node $(r - 1, c)$.
- If $power > \#_2(c)$, then send the packet to node $(r, c + forward)$.

/* In addition, we may choose to start a set of loops starting at the current column. */

- If $power > \#_2(c)$, $d < m$, and $r > minRow$, then, for $f = \pm 1$, send the packet to node $(r, c + f)$ with $(\#_2(c), minRow + 1, maxRow, 0, 0, f)$ pushed onto its stack.
- If $power > \#_2(c)$, $d < m$, and $r < maxRow$, then, for $f = \pm 1$, send the packet to node $(r, c + f)$ with $(\#_2(c), minRow, maxRow - 1, 0, 0, f)$ pushed onto its stack.

/* If a loop is in its first half, it can switch to the second half at any step. */

- If $loopHalf = 0$, let $minRow'$ denote the value of $minRow$ at the previous level of recursion (i.e. in the record second from the top of the stack).
  If $minRow = minRow'$
  - then send the packet to node $(r+1, c)$ with (*power*, *minRow*+1,*maxRow*+1,*loopCounter*,1,−*forward*) replacing the top record on its stack.
  - else send the packet to node $(r-1, c)$ with (*power*, *minRow*−1,*maxRow*−1,*loopCounter*,1,−*forward*) replacing the top record on its stack.

/* If a packet has returned to the column where it started its current set of loops, it has two options. */

- If $\#_2(c) \geq power$ and $loopHalf = 1$ then

  /* Option 1: start the next loop in the set. Note that if the second half of the previous loop allows the use of rows r_1 to r_2, then the previous level of the recursion allows the use of either rows r_1 to $r_2 + 1$ or rows $r_1 - 1$ to r_2. In the first case, the first half of the next loop can use either rows r_1 to r_2 or rows $r_1 + 1$ to $r_2 + 1$. In the second case, the first half of the next loop can use either rows r_1 to r_2 or rows $r_1 - 1$ to $r_2 - 1$. */
  - If $loopCounter < maxRow - minRow - 1$, then
    * For $f = \pm 1$, send the packet to node $(r, c + f)$ with (*power*, *minRow*, *maxRow*, *loopCounter*+1, 0, f) replacing the top record on its stack.

* Let $minRow'$ and $maxRow'$ denote the value of $minRow$ and $maxRow$ at the previous level of recursion (i.e. in the record second from the top of the stack).
* If $minRow = minRow'$ and $r > minRow$ then for $f = \pm 1$, send the packet to node $(r, c+f)$ with ($power$, $minRow + 1$, $maxRow + 1$, $loopCounter + 1$, 0, f) replacing the top record on its stack.
* If $maxRow = maxRow'$ and $r < maxRow$ then for $f = \pm 1$, send the packet to node $(r, c+f)$ with ($power$, $minRow - 1$, $maxRow - 1$, $loopCounter + 1$, 0, f) replacing the top record on its stack.

/* Option 2: stop the current set of loops and return to the previous level of the recursion. */

- If $d > 1$, pop one record off the stack. Let $forward'$ denote the value of $forward$ at the new top level of the stack. Send the resulting packet to node $(r, c + forward')$.

End of protocol.

Lemma 5. *The header of any packet produced by the Protocol* **DELIVER** *has a length of at most* $m(\lceil \log_2 \log_2 (n-1) \rceil + 3\lceil \log_2 m \rceil + 2) + 2$ *bits.*

Proof. It is easily verified that the maximum depth of the recursion produced by Protocol **DELIVER** is m. For each such level, the variable *power* can be represented using $\lceil \log_2 \log_2 (n-1) \rceil$ bits, the variables *maxRow*, *minRow*, and *loopCounter* can be represented using $\lceil \log_2 m \rceil$ bits, and *forward* and *loopHalf* can each be represented using a single bit. The final two bits come from the fact that each transmission informs the recipient of the direction from which the packet came. □

Lemma 6. *Protocol* **DELIVER** *transmits only a finite number of packets.*

Proof. We provide a potential function Φ for any packet in the system, such that there is a maximum value that Φ can attain and, every time a packet is forwarded, the corresponding value of Φ is increased by at least 1. (That is, each packet P has a potential exceeding the potential of the packet whose arrival caused P to be sent.) For each level of recursion i, $1 \leq i \leq m$, we define three variables: lc_i, lh_i, and $dist_i$. All of these variables are defined to be 0 if $i > d$. For $i \leq d$, lc_i and lh_i are the *loopCounter* and *loopHalf* variables, respectively, for level i in the recursion. For $i \leq d$, the variable $dist_i$ is the number of horizontal steps taken by the packet starting from the time that the *forward* variable at the i'th level of recursion was last set, counting only those steps that occurred when $d = i$. Note that a packet can only move horizontally in the direction specified by the *forward* variable, and thus all of these steps will be in the same direction. This means that $dist_i \leq n$. We also define the variable *vert* to be the number of steps taken in a vertical direction on the current column since last moving there from another column.

The potential function Φ that we define can be thought of as a $(3m+1)$-digit mixed radix number, where for $t \in \{1, \ldots, m\}$, digit $3(t-1)+1$ is lc_t, digit $3(t-1)+2$ is lh_t, and digit $3(t-1)+3$ is $dist_t$. Digit $3m+1$ is $vert$. It is easily verified that when a packet is first sent, $\Phi \geq 0$. Also, by checking each of the possible actions of a node on the receipt of a packet, we can verify that every time a packet is forwarded, Φ increases by at least 1. We also see that Φ is bounded, since $vert \leq m-1$ and, for any i, $lc_i \leq m$, $dist_i \leq n$, and $lh_i \leq 1$. Since each packet receipt causes at most a constant number of new packets to be sent out, it follows that the total number of packets sent as a result of Protocol **DELIVER** is finite. □

It follows from the decomposition of simple S–R paths given by Lemmas 3 and 4 that, if there is a simple path of live edges from S to R, then Protocol **DELIVER** finds it. We combine Lemmas 5 and 6 to get our main result.

Theorem 2. *Protocol* **DELIVER** *delivers a message from S to R with finite traffic using $O(m(\log\log n + \log m))$-bit headers .*

4 A Lower Bound

In this section, we prove that $\Omega(\log\log n)$ header bits are necessary for communicating a single message in a $3 \times n$ grid. First, we consider the graph $G(3, n)$ with $n = h!$. The proof is similar in flavour to the lower bound for communicating a single message in a complete graph [AF99].

Our proof focusses attention on h particular simple S–R paths, defined as follows. For $k = 1, \ldots, h$, let P_k consist of $k!$ alternating S-paths and Z-paths, each of extent $h!/k!$, concatenated using single horizontal edges. Figure 2 shows paths P_1, P_2, and P_3 for the case $h = 3$.

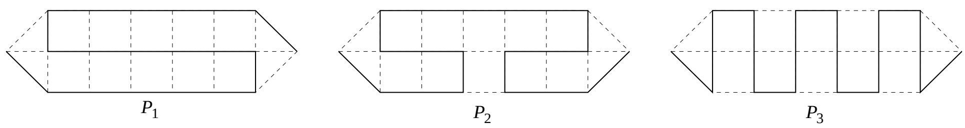

Fig. 2. Paths P_1, P_2, and P_3 for $h = 3$

For $0 \leq i < n$, let $i_1, \ldots, i_h$ be such that $i = \sum_{k=1}^{h} i_k n/k!$ where $0 \leq i_k < k$. In other words, $(i_1, \cdots, i_h)$ is the mixed radix representation of i, where the k'th most significant digit is in base k. Note that i_1 always has value 0. For example, if $n = 24 = 4!$ and $i = 20$, then $i_1 = 0$, $i_2 = 1$, $i_2 = 2$, and $i_3 = 0$.

Proposition 2. *Let $0 \leq i < j < n$. Node $(1, j)$ appears before node $(1, i)$ in path P_k if and only if $i_d = j_d$ for $d = 1, \ldots, k$.*

Proof. In every S-path or Z-path, the nodes in row 1 appear in order from largest numbered column to smallest numbered column. Since path P_k is the concatenation of S-paths and Z-paths, node $(1, j)$ appears before node $(1, i)$ if and only if columns i and j are in the same S-path or Z-path. Since each S-path and Z-path comprising P_k has extent $n/k!$, it follows that i and j are in the S-path or Z-path if and only if $\lfloor i/(n/k!) \rfloor = \lfloor j/(n/k!) \rfloor$, which is true if and only if $i_d = j_d$ for $d = 1, \ldots, k$. □

Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Since node $(1, c)$ is on path P_k, it receives at least one packet when only the links on the simple S–R path P_k are alive. Let $H_k(c)$ denote the header of the last packet received by node $(1, c)$ in this situation that causes a packet to be received by R.

Lemma 7. *Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Then, for all path indices $1 \leq j < k \leq h$ and all columns $0 \leq c < c' < h!$ such that $(c_1, c_2, \ldots, c_j) = (c'_1, c'_2, \ldots, c'_j)$ and $(c_1, c_2, \ldots, c_k) \neq (c'_1, c'_2, \ldots, c'_k)$, either $H_j(c) \neq H_k(c)$ or $H_j(c') \neq H_k(c')$.*

Proof. To obtain a contradiction, suppose that $H_j(c) = H_k(c)$ and $H_j(c') = H_k(c')$, for some path indices $1 \leq j < k \leq h$ and some columns $0 \leq c < c' < h!$ such that $(c_1, c_2, \ldots, c_j) = (c'_1, c'_2, \ldots, c'_j)$ and $(c_1, c_2, \ldots, c_k) \neq (c'_1, c'_2, \ldots, c'_k)$. Then, by Proposition 2, node $(1, c')$ appears before node $(1, c)$ in path P_j but after node $(1, c)$ in path P_k.

Consider the situation when the links on both paths P_j and P_k are alive. The protocol forwards a packet along path P_k until a packet with header $H_k(c')$ reaches node $(1, c')$. This causes a packet to be received by R. Since $H_k(c') = H_j(c')$ and node $(1, c')$ occurs before node $(1, c)$ on path P_j, it also causes a packet with header $H_j(c)$ to be received at node $(1, c)$. Likewise, since $H_j(c) = H_k(c)$ and node $(1, c)$ occurs before node $(1, c')$ on path P_k, this causes a packet with header $H_k(c')$ to be received at node $(1, c')$, and we have an infinite loop. Each time such a packet goes through the loop, it produces a new packet that is sent to the destination R. This contradicts the finite traffic assumption. □

Lemma 8. *Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Then, for $1 \leq k \leq h$, there exist nonnegative digits $i_1 < 1, i_2 < 2, \ldots, i_k < k$ such that the k headers $H_1(c), \ldots, H_k(c)$ are distinct for each column c with $(c_1, c_2, \ldots, c_k) = (i_1, i_2, \ldots, i_k)$.*

Proof. To obtain a contradiction, suppose the lemma is false. Consider the smallest value of $k \leq h$ for which the lemma is false. Since there are no repetitions in a sequence of length one, $k > 1$. Let $i_1 < 1, i_2 < 2, \ldots, i_{k-1} < k - 1$ be such that the $k - 1$ headers $H_1(c), \ldots, H_{k-1}(c)$ are distinct for each column c with $(c_1, c_2, \ldots, c_{k-1}) = (i_1, i_2, \ldots, i_{k-1})$. Then, for each digit $i_k \in \{0, \ldots, k - 1\}$, there exists a path index $j \in \{1, \ldots, k - 1\}$ and a column c such that $(c_1, c_2, \ldots, c_{k-1}, c_k) = (i_1, i_2, \ldots, i_{k-1}, i_k)$ and $H_k(c) = H_j(c)$.

Since there are k choices for i_k and only $k-1$ choices for j, the pigeonhole principle implies that there exist distinct $i_k, i'_k \in \{0, \ldots, k-1\}$ which give rise to the same value of j and there exist columns c and c' such that $(c_1, c_2, \ldots, c_{k-1}) = (c'_1, c'_2, \ldots, c'_{k-1})$, $c_k = i_k \neq i'_k = c'_k$, $H_k(c) = H_j(c)$, and $H_k(c') = H_j(c')$. But this contradicts Lemma 7. □

Theorem 3. *Any protocol for* $G(3,n)$ *that delivers a message from* S *to* R *with finite traffic uses headers of length at least* $\log_2 \log_2 n - O(\log\log\log n)$.

Proof. Let h be the largest integer such that $n \geq h!$. Then $n < (h+1)! < (h+1)^h$, so $h \log_2(h+1) > \log_2 n$ and $h \in \Omega(\log n / \log\log n)$.

Consider any protocol for $G(3,n)$ that uses headers of length L. Since $G(3,h!)$ is a minor of $G(3,n)$, it follows from Proposition 1 that there is a protocol for $G(3,h!)$ using headers of length L. Hence, by Lemma 8, $L \geq \log_2 h = \log_2 \log_2 n - O(\log\log\log n)$. □

References

[AF99] Micah Adler and Faith E. Fich, *The Complexity of End-to-End Communication in Memoryless Networks*, with Micah Adler, 8th Annual ACM Symposium on Principles of Distributed Computing, May 1999, pages 239–248.

[AAF+94] Afek, Attiya, Fekete, Fischer, Lynch, Mansour, Wang, and Zuck, *Reliable Communication Over Unreliable Channels*, JACM, vol. 41, no. 6, 1994, pages 1267–1297.

[AAG+97] Afek, Awerbuch, Gafni, Mansour, Rosén, and Shavit, *Slide–The Key to Polynomial End-to-End Communication*, Journal of Algorithms, vol. 22, no. 1, 1997, pages 158–186.

[AG88] Afek and Gafni, *End-to End Communication in Unreliable Networks*, 7th PODC, 1988, pages 131–148.

[AMS89] Awerbuch, Mansour, and Shavit, *Polynomial End to End Communication*, 30th FOCS, 1989, pages 358–363.

[APV96] Awerbuch, Patt-Shamir, and Varghese, *Self-stabilizing End-to-End Communication*,

[DW97] Dolev and Welch, *Crash Relient Communication in Dynamic Networks*, IEEE Transactions of Computers, vol. 46, 1997, pages 14–26.

[F98] Faith E. Fich, *End-to-end Communication*, Proceedings of the 2nd International Conference on Principles of Distributed Systems, Amiens, France, 1998, pages 37–43.

[K30] K. Kuratowski, *Sur le Problème des Courbes Gauches en Topologie*, Fund. Math., vol. 15, 1930, pages 271–283.

[KOR95] Kushilevitz, Ostrovsky, and Rosén, *Log-Space Polynomial End-to-End Communication*, 28th STOC, 1995, pages 559–568.

[LLT98] Ladner, LaMarca, and Tempero, *Counting Protocols for Reliable End-to-End Transmission*, JCSS, vol. 56, no. 1, 1998, pages 96-111.

[P81] Postel, *Internet Protocol*, Network Working Group Request for Comments 791, September 1981.