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], [F98]. 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.

* A full version appears at http://www.dcs.warwick.ac.uk/~leslie/papers/endtoend.ps.

† 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 links may fail. 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 path of live links of length at most $n-1$. Therefore, it suffices to use the simple "hop count" algorithm [P81] which discards messages that have been forwarded $n-1$ times. Since this can be done with headers of size $\lceil \log n \rceil$, 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 graphs it is possible to do better than the simple hop count algorithm. For example, Adler and Fich [AF99] observed that if F is a *feedback vertex set* of the underlying graph G (that is, if every cycle of G contains at least one vertex of F), then one can use a variant of the hop count protocol which discards messages that have visited F more than $|F|$ times. The discarding does no harm, since a simple path visits F at most $|F|$ times. But it ensures that the amount of traffic generated is finite. Note that in this variant of the hop count protocol, the length of packet headers is at most $\lceil \log_2(|F|+1) \rceil$.

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 $m, n > 2$. For $m = 2$, headers of length one suffice in our network (since no backward move is needed and we need only distinguish vertical and horizontal arrivals) [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 $0 \le i < m$ and $0 \le j < n$. There are links between

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

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

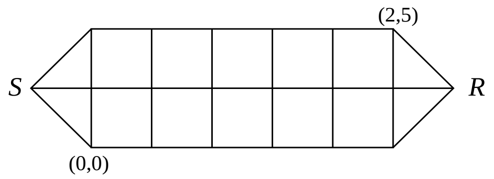

Fig. 1. The graph $G(3,6)$

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. Headers may be modified arbitrarily; however, data must be treated as a "black box". That is, processors may make copies of the data, but they may not modify it. 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. Typically in end-to-end communication, one assumes that when a processor receives a packet, it cannot detect which of its neighbours sent the packet. This assumption is not relevant to our problem since the degree of the underlying network is bounded, and so the identity of the sender can be encoded in a constant number of header bits.

Intermediate processors are assumed to be *memoryless*. 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. Once a link becomes dead, it remains so. Processors do not know which subset of the links are alive. For simplicity, it is assumed that processors never fail. However, a dead processor can be simulated by considering each of its incident links to be dead.

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 a 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 $O(m(\log\log n + \log m))$-bit headers for any network $G(m,n)$. For the case of $G(3,n)$, this is improved in the full version of our paper to $\log_2 \log_2 n + O(1)$. In Section 3 we demonstrate that for $G(3,n)$, $\log_2 \log_2 n - O(\log\log\log n)$ bits are required. Using the following observation of Adler and Fich [AF99], this lower bound can be extended to $G(m,n)$.

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.*

In particular, since for $m \geq 3$, $G(m,n)$ has $G(3,n)$ as a minor, $\log_2 \log_2 n - O(\log\log\log n)$ bits are required for $G(m,n)$. 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. By Proposition 1, our upper bounds also provide upper bounds for any graphs that are minors of $G(m,n)$, for any constant m.

The next section describes our protocol for $G(m,n)$ for any constant $m \geq 3$. This is followed in Section 3 by our lower bound for $G(3,n)$ and, hence, for $G(m,n)$ with $m > 3$.

2 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. The characterization will be used in Lemma 1 to parse simple paths. We will be considering sub-paths that go from left-to-right (from small c_1 to big c_2) and also sub-paths that go from right-to-left (from big c_1 to small c_2), but we will always work within a bounded region of rows consisting of row r_1 up to row r_2.

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 1. *Let c_1, c_2, 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 starting and ending in column $c_2 2^k$. 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 $c_2 2^k$. $\square$

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 2. *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. $\square$

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 1 and 2 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 2, 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 $d = m - \mathit{maxRow} + \mathit{minRow}$, 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 $r < \mathit{maxRow}$ and the packet was not received from node $(r+1, c)$, send the packet to node $(r+1, c)$.
- If $r > \mathit{minRow}$ and the packet was not received from node $(r-1, c)$, send the packet to node $(r-1, c)$.
- If $\mathit{power} > \#_2(c)$, then send the packet to node $(r, c + \mathit{forward})$.

/* In addition, we may choose to start a set of loops starting at the current column. This can happen only if $r > \mathit{minRow}$ or $r < \mathit{maxRow}$, either of which implies that $d < m$. */

- If $power > \#_2(c)$ 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 $power > \#_2(c)$ 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 3. *The header of any packet produced by the Protocol* **DELIVER** *has a length of at most* $m(\lfloor \log_2 \log_2 (n-1) \rfloor + 3\lceil \log_2 m \rceil + 3) + 2$ *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 $\lfloor \log_2 \log_2 (n-1) \rfloor + 1$ 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 4. *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, $1 \le i \le m$, we define three variables: lc_i, lh_i, and $dist_i$. All of these variables are defined to be 0 if $i > d$, the current recursion depth. For $i \le d$, lc_i and lh_i are the *loopCounter* and *loopHalf* variables, respectively, for level i in the recursion. For $i \le d$, 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 $dist_i \le n$. 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, $\Phi \ge 0$. 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 $vert \le m-1$ and, for any i, $lc_i \le m$, $dist_i \le n$, and $lh_i \le 1$. 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 1 and 2 that, if there is a simple path of live edges from S to R, then Protocol **DELIVER** finds it. We combine Lemmas 3 and 4 to get our main result.

Theorem 1. *Protocol* **DELIVER** *delivers a message from* S *to* R *with finite traffic using* $O(m(\log \log n + \log m))$*-bit headers .*

3 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 uses the following definitions. An *S-path of extent* $j \geq 1$ is a path from $(0, c)$ to $(2, c + j - 1)$, for some column c, where $0 \leq c \leq n - j$. 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 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 $h = 3$.

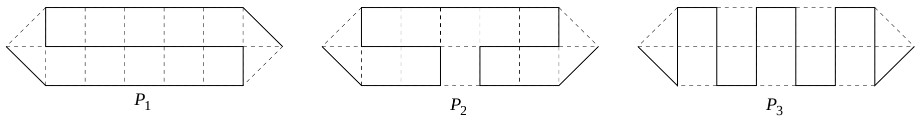

Fig. 2. Paths P_1, P_2, and P_3 for $h = 3$

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 $i_1 = 0$, $i_2 = 1$, $i_3 = 2$, and $i_4 = 0$.

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 same 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 $d = 1, \ldots, k$. $\square$

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 5. *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. $\square$

Lemma 6. *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 5. $\square$

Theorem 2. *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 6, $L \geq \log_2 h = \log_2 \log_2 n - O(\log\log\log n)$. □

References

[AF99] M. Adler and F.E. Fich, *The Complexity of End-to-End Communication in Memoryless Networks*, Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, (1999), 239–248.

[AAF+94] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck, *Reliable Communication Over Unreliable Channels*, Journal of the ACM, **41(6)**, (1994), 1267–1297.

[AAG+97] Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosén, and N. Shavit, *Slide–The Key to Polynomial End-to-End Communication*, Journal of Algorithms, **22(1)**, (1997), 158–186.

[AG88] Y. Afek and E. Gafni, *End-to End Communication in Unreliable Networks*, Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing, (1988), 131–148.

[AMS89] B. Awerbuch, Y. Mansour, and N. Shavit, *Polynomial End to End Communication*, Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, (1989), 358–363.

[APV96] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilizing End-to-End Communication*, Journal of High Speed Networks, **5(4)**, (1996), 365–381.

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

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

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

[KOR95] E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Log-Space Polynomial End-to-End Communication*, Proceedings of the 28th ACM Symposium on the Theory of Computing, (1995), 559–568.

[LLT98] R. Ladner, A. LaMarca, and E. Tempero, *Counting Protocols for Reliable End-to-End Transmission*, Journal of Computer and Systems Sciences, **56(1)**, (1998), 96–111.

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