The Complexity of End-to-End Communication in Memoryless Networks

Micah Adler and Faith Fich
Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 3G4

Abstract

End-to-end communication is the problem of sending a sequence of messages from a sender to a receiver when the network through which they communicate is unreliable. The model considered is an asynchronous network in which intermediate nodes are assumed to have no memory. Dynamic link failures are allowed: links can lose messages, but cannot reorder or duplicate them.

Two problems are studied: sending a single message through a network and sending a stream of messages through a network. We provide lower bounds and upper bounds on the size of the headers needed to transmit information from the sender S to the receiver R.

We 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), headers of length $\Omega(\log n)$ are necessary to ensure delivery of one message, without ever generating an infinite amount of packet traffic. This lower bound holds even if only static link faults are allowed. It also matches the upper bound achieved by the simple "hop count" protocol that uses packet headers to count the number of hops packets have traveled. For other network topologies, we develop new protocols that use substantially smaller headers. For example, if the network has a known feedback vertex set of size f, then headers of length $\lceil \log_2(f+1) \rceil$ suffice.

For the sequence transmission problem, we prove lower bounds as a function of the network topology, implying, for example, that fixed degree meshes with n nodes and the complete graph on n nodes require headers of length $\Omega(n)$ and $\Omega(n \log n)$, respectively. In many cases, including these graphs and all series parallel graphs, our lower bounds match the upper bounds achieved by Dolev and Welch's bounded sequence number protocol.

1 Introduction

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) without communication, it is not possible to perform distributed computing and (b) as the size of a network increases, the likelihood of a fault occurring somewhere in the network also increases. An unreliable network can be viewed as a communication channel in which message losses, duplications and reorderings can occur. Essentially, a good protocol for end-to-end communication allows distributed algorithms to treat an unreliable network as a reliable channel.

Many different protocols for end-to-end communication have been developed when links and/or intermediate processors may fail, either by losing messages, or by ceasing to transmit any information [AAG+97, AG88, AMS89, APV96, DW97, KOR95, LLT98]. Their performance is measured in terms of the amount of communication performed over the links of the network and the amount of storage space used by intermediate nodes of the network.

In this paper, we prove lower bounds for the performance of such protocols. Specifically, we consider the case where intermediate processors are assumed to be memoryless. Thus, when an intermediate processor receives a packet, it decides along which link(s) to send the packet and possibly how to change the packet header, based only on the contents of the packet header and which link the packet arrives on. 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. Moreover, any such algorithm that can handle the crash and recovery of links (or, equivalently, packet losses) can automatically handle the crash and recovery of intermediate processors, since there is no problem with corrupted memory when an intermediate processor recovers.

1.1 The model

A network is represented by an undirected graph G, with a node corresponding to each processor and an edge corresponding to a link between two processors. Throughout this paper, we use n to denote the number of processors in the network. The links of the network are either operational or inoperational. Operational links are asynchronous: the time for a packet to traverse a link is finite, but unbounded. Although this time may differ from the time for another packet to traverse the same link, the packets arrive in the same order as they are sent. Throughout this paper, we assume that links are bidirectional, so that the link from u to v is operational if and only if the link from v to u is operational. However, all of our results also apply to the situation in which reverse links can fail independently.

The *static* fault model allows both operational and inoperational links, but links cannot change their status. Initially, some unknown subset of the links are inoperational. Because links which are inoperational can be thought of as having infinite delay, processors cannot distinguish between an inoperational link and an operational link which is just very slow. With static faults, we assume that there is some path from S to R that is operational, and thus it is possible to deliver packets from S to R.

We also consider an interconnection network where links can experience *dynamic* faults: links can alternate an unlimited number of times between being operational and inoperational. This is equivalent to a model where arbitrary messages can be dropped. Clearly, if there is a permanent S–R cut of inoperational links, then it is not possible to send messages from S to R. Thus, assumptions regarding the behavior of faulty links are necessary. Here, we assume *infinitely frequent path stability* [AGH90, H92]: infinitely often there is some path P from S to R, such that a packet sent from S along path P will arrive at R.

1.2 Our results

In this paper, we consider two problems: sending a single message from S to R and sending an infinite sequence of messages from S to R. We provide lower bounds and upper bounds on the size of the headers (bits of information in addition to actual message information) needed to transmit information from S to R. Note that in the presence of dynamic faults, this is an appropriate measure of the communication requirements of a protocol, since quantities such as the total number of bits required to transmit a message can easily be seen to be unbounded.

For the single message problem, we prove lower bounds for the static fault model. These lower bounds also apply directly to the dynamic fault model, as well as to more difficult problems. We show that if the network G contains the complete graph on r nodes as a minor, then headers of length at least $\lceil \log_2 r \rceil - 3$ are necessary to ensure delivery of the message without ever generating an infinite amount of packet traffic. Thus, if G is a complete graph, headers of size $\lceil \log_2 n \rceil - 3$ are needed and, if G is a hypercubic network such as a butterfly network, $\Omega(\log n)$ bit headers are still needed.

For these networks, the lower bound matches the upper bound achieved by a simple protocol that uses packet headers to keep track of the number of hops packets have traveled [P81]. This protocol applies to both the static failure model, as well as the dynamic failure model. For other network topologies, we develop new protocols that can be adapted to both fault models, and use substantially smaller headers. For example, if the network has a known feedback vertex set of size f, then headers of length $\lceil \log_2(f + 1) \rceil$ suffice.

A second, more difficult problem is to reliably send a long (or possibly infinite) stream of messages from S to R. Here, we require that R outputs the same sequence of messages as is received by S (i.e., the protocol does not lose, duplicate, or reorder messages). We prove lower bounds for this problem in the dynamic fault model as a function of the network topology. For many networks, including the complete graph, series parallel graphs, and finite degree meshes, our resulting lower bounds match an upper bound by Dolev and Welch [DW97], showing that their protocol is optimal.

For the sequence transmission lower bounds, we provide two different proofs that make incomparable assumptions about the computation performed by S and R. The first proof assumes that the protocol is *data oblivious*: no processor can perform any computation on the contents of the messages. The second proof assumes that the protocol uses *finite control*: arbitrary computation on message contents can be performed, but the storage capacity at both S and R is bounded. The details of these lower bound models appear in Section 4.

The next section presents a brief summary of some of the most closely related work. Our results for the problem of sending a single message are given in Section 3, with lower bounds appearing in Section 3.1, followed by our new upper bounds in Section 3.2. Section 4 contains our lower bounds for sending a stream of messages in the presence of packet loss. Open problems are discussed in Section 5.

2 Related Work

For arbitrary networks, the simplest algorithm to send a single message from S to R is to flood the network with packets containing the message. It suffices to store a single bit at each intermediate processor to indicate whether a packet containing the message has passed through. When a processor receives a packet from one of its neighbors, it forwards that packet to all of its other neighbors, if and only if the message has not previously passed through. With no storage available at intermediate processors, it is still possible to ensure that the network does not produce an infinite stream of messages by using the following observation: if there is a path of operational links from S to R in an n node network, then there is a simple path of operational links of length at most $n - 1$. Thus, it suffices to use packet headers that count the number of intermediate processors the packet has passed through (including duplicate visits) [P81]. To send a message to R, processor S sends a packet with header 0 and the message to each of its neighbors. When an intermediate node receives a packet from a neighbor and the packet header is less than $n-2$, the processor increments the header and forwards the packet to all its other neighbors. The number of header bits used by such a protocol is $\lceil \log_2(n-1) \rceil$.

In the case where S must send a sequence of messages, it is worth first considering the simplest interconnection network: two processors connected by a single link. For static faults, it suffices for S to send its messages in packets without headers. They are received if and only if the link is operational. If S wants to know that R has received a message, S can wait for an acknowledgment. This is the best possible, since no information can be sent from S to R unless there is a path connecting them. For this network, the straightforward scheme in which S uses sequence numbers as packet headers and R sends the sequence numbers back as acknowledgment is sufficient to ensure reliable delivery of a sequence of messages, while tolerating packet loss, duplication and reordering on the link [S76, V83, AE86]. Unfortunately, the length of packet headers grows without bound. The alternating bit protocol [BSW69, LLT98] uses single bit headers, but only tolerates packet loss and duplication, not reordering of packets. With message loss, there is no correct protocol without headers [FL90]. Moreover, any protocol that tolerates arbitrary packet reordering and duplication requires unbounded sequence numbers [WZ89]. Similarly, any protocol that tolerates both packet reordering and loss requires either unbounded sequence numbers or requires that R receive an unbounded number of packets per message [AAF+94].

For arbitrary networks, it is possible to send a stream of messages by attaching sequence numbers to the packets sent by S [V83, AE86, AAG87]. Each intermediate processor remembers the largest sequence number it has seen and only forwards a packet containing the next higher number. Unfortunately, the number of bits of storage needed at each intermediate node and the number of header bits per packet increases without bound as the number of messages in the stream increases. Instead of using sequence numbers, there are algorithms in which S sends a rapidly increasing number of packets for each successive message it wants to send to R [AAF+94]. Processor R determines when it has a new message by keeping track of the number of copies of each different packet it has received.

If one assumes that each individual link delivers packets in order and without duplication, then this restricts the possible reorderings and duplications that the network as a whole (if viewed as a virtual link) can produce. Protocols for arbitrary networks can take advantage of this.

There are sophisticated algorithms [AG91, AAG+97, KOR95] that use no header bits and perform only a polynomial (in n) amount of communication for each message. However, processors must store packets and auxiliary information (using space bounded by a logarithmic or polynomial function of n) for each of their incident links. Although these protocols do not handle packet losses, they can be adapted to the dynamic model by performing the alternating bit protocol along each link and using buffers at each node to hold any packet that might not yet have been successfully transmitted along some incident link [AG88].

If no packet losses, duplications or reorderings occur on any link of the network (but operational links can become inoperational), then a stream of messages can be sent without using memory at internal processors by treating each simple S–R path separately. More specifically, for each message, S sends a packet containing the message along each simple S–R path, using the index of the path as a packet header. (Thus, headers are of length $\lceil \log_2 p \rceil$, where p is the number of simple paths in the network from S to R.) Processor R counts the number of packets received on each path to determine when a new message has arrived. One problem with this protocol is that R uses an unbounded amount of memory. Dolev and Welch [DW97] combine this approach with bounded sequence numbers to obtain a protocol with $O(\log p)$ bit headers that uses $O(\log p)$ bits of storage at R and $O(p \log p)$ bits of storage at S. By incorporating a version of the alternating bit protocol along each path, the protocol can also tolerate packet losses.

More detail concerning related work can be found in the survey article [F98].

3 Reliable Delivery of One Message

In this section we provide lower and upper bounds for the delivery of a single message. Our lower bounds are proven in the static fault model and, thus, also apply to the dynamic fault model. The protocols for our upper bounds are also described for the static fault model, but they can easily be adapted to the dynamic fault model by using acknowledgments.

We say that a protocol *ensures delivery* of a message from node S to node R in a network with underlying graph G, subject to static faults, provided that a message sent by S will be received by R, no matter what links are operational, as long as there is always a path of operational links between S and R. If a protocol ensures delivery of a message from S to R and never generates an infinite number of packets, we say the protocol *ensures correct delivery* from S to R.

3.1 Lower Bounds

This section begins by proving there is no protocol for sending a single message from one processor to another that works for all n node networks and uses $o(\log n)$ bit headers. This result is also true for bounded degree networks.

Theorem 1 *Consider any protocol that ensures delivery of a single message from S to R in the n node complete graph under the static fault model. If at most $\lfloor n/2e \rfloor$ different headers are used, then there is an execution in which an infinite number of packets are received by R.*

Proof: Let $\{v_1, \ldots, v_{n-2}\}$ denote the set of possible intermediate nodes in the network and let $k = \lfloor n/2 \rfloor - 1$. We consider the set of all simple S–R paths of length $2k+1$ that pass through each of the nodes $v_1, \ldots, v_{2k}$

exactly once and where node v_{2i} immediately follows node v_{2i-1}, for $i = 1, \ldots, k$. There are $k!$ such paths.

Let each possible packet header be represented by a different color. Let the *transcript* of such a path π be the sequence of colors $c_1, \ldots, c_k$, where c_i is the color of the last packet sent from node v_{2i-1} to node v_{2i} when only the links on path π are operational. If there are at most $\lfloor n/2e \rfloor$ colors, then there are at most $\lfloor n/2e \rfloor^k < k!$ different possible transcripts. Hence, there are two paths, π and π', with the same transcript.

Since π and π' are distinct, there exist two odd numbers i, j such that v_i occurs before v_j in π but v_i occurs after v_j in π'. In any protocol, the packets sent by a node of the network depend solely on the color of the incoming packet that caused that node to act, and on the edge the incoming packet arrived on. This and the assumption that π and π' have the same transcript mean that the action node v_{i+1} performs on receiving its last packet from v_i when only the links on path π are operational is the same as it performs on receiving its last packet from node v_i when only the links on path π' are operational.

We say that a path is *active* if all of its links are operational. When path π is active, the protocol eventually sends a packet that is forwarded from node v_i to node v_{i+1}, and this packet has the same header as the last packet sent from node v_i to node v_{i+1} when path π' is active. Thus, if paths π and π' are both active, we have the following situation: a packet is forwarded along path π, through node v_{i+1} until it reaches node v_{j+1}. It is then forwarded along path π', until it reaches node v_{i+1}. It is then forwarded again along path π, until it reaches node v_{i+1}, 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. ■

A graph G' is a *minor* of a graph G, if G' can be obtained from G by a finite sequence of edge deletions and contractions. The contraction of an edge $\{u, v\}$ causes the nodes u and v to be replaced by a single node whose neighbors are the union of the neighbors of u and v. Neither self-loops nor parallel edges are introduced by this operation. Thus the set of nodes in G' (called supernodes) can be viewed as a collection of disjoint subsets of nodes in G. A protocol for end-to-end communication on a particular graph can also be used on any minor of that graph.

Proposition 2 *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 ensures correct delivery from S to R is also a protocol for G' that ensures correct delivery from S' to R'.*

Proof: The proof is by structural induction using the following observations. An edge deletion can be simulated by making the edge inoperational (or equivalently, by losing all packets sent along the edge). An edge contraction can be simulated by making the contracted edge operate with no delay and deleting any edges that introduce self-loops or parallel edges. ■

This proposition can be combined with Theorem 1 to give the following corollary.

Corollary 3 *Consider any graph G that contains the complete graph on r nodes as a minor where S and R are in different supernodes. Any protocol for G in the static fault model that ensures correct delivery of a single message from S to R requires headers of length at least $\lceil \log_2 r \rceil - 3$ bits.*

Many commonly used communication networks contain a large complete graph as a minor. Hence Corollary 3 can be used to get good lower bounds on header length for such networks.

Lemma 4 *The complete graph on r nodes is a minor of the $r \times r \times 2$ mesh.*

Proof: Let $M = \{(i, j, k) \mid i, j \in \{1, \ldots, r\}, k \in \{1, 2\}\}$ be the set of nodes in the $r \times r \times 2$ mesh. For $i = 1, \ldots, r$, let P_i denote the set of all nodes of M in row i row of the bottom layer or in column i of the top layer, i.e. $P_i = \{(i, j, 1) \mid j \in \{1, \ldots, r\}\} \cup \{(j, i, 2) \mid j \in \{1, \ldots, r\}\}$.

The subgraph induced by P_i is connected, since row i of the bottom layer and column i of the top layer are joined by the edge $\{(i, i, 1), (i, i, 2)\}$. Furthermore, the edge $\{(i, j, 1), (i, j, 2)\}$ connects P_i and P_j, so the corresponding supernodes are connected. ■

Thus, the $r \times r \times 2$ mesh requires headers of size $\lceil \log_2 r \rceil - 3$ to ensure correct delivery. This result also applies to the d-dimensional mesh with side length r, for all $d \geq 3$.

Lemma 5 *The complete graph on $r = 2^k$ nodes is a minor of the r^2 input butterfly.*

Proof: The r^2 input butterfly consists of the nodes $\{(w, i) \mid w \in \{0, 1\}^{2k}, i \in \{0, \ldots, 2k\}\}$, with edges between $(w, i-1)$ and (w', i) if and only if w and w' agree on all bits, except possibly the ith.

For $u \in \{0, 1\}^k$, let $P_u = P'_u \cup P''_u$, where
$P'_u = \{(uv, i) \mid v \in \{0, 1\}^k, i \in \{0, \ldots, k\}\}$ and
$P''_u = \{(vu, i) \mid v \in \{0, 1\}^k, i \in \{k+1, \ldots, 2k\}\}$.

The subgraph induced by P_u is connected, since P'_u is an r-input butterfly and, for each $v \in \{0, 1\}^k$, there is a path from (uu, k) to $(uv, 2k)$ (and, hence, (uv, i) for all $i \in \{k+1, \ldots, 2k\}$) obtained by correcting the $k+1$st through $2k$th bits, one at a time.

Furthermore, for each $u, v \in \{0, 1\}^k$, there is an edge between $(uv, k) \in P_u$ and $(uv, k+1) \in P_v$, so the corresponding supernodes are connected. ■

Thus, the $2^{2k}(2k+1)$-node butterfly requires headers of size $k - 3$ to ensure correct delivery. We can also demonstrate comparable results for other hypercubic networks, such as the hypercube, the shuffle-exchange graph, the de Bruijn graph, and the cube connected cycles graph. See [L92] for descriptions of these networks.

Note that, for the complete graph, we are within an additive constant of 3 of the upper bound obtained by counting the number of steps the packet has taken. Also, for the butterfly and the mesh graphs mentioned above, we are within a small constant multiplicative factor of the "hop count" algorithm.

However, any planar graph, including the 2-dimensional mesh, does not contain the complete graph on 5 nodes as a minor [K30]. Hence, other techniques are needed to prove lower bounds for such networks.

3.2 Upper Bounds

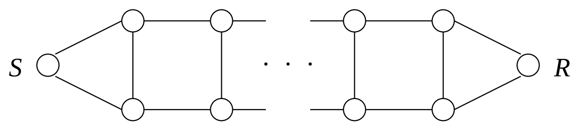
There is a simple improvement to the "hop count" algorithm based on the observation that, for any feedback vertex set F, any simple path visits vertices in F at most $|F|$ times. A set of nodes $F \subseteq V$ is a *feedback vertex set* for a graph $G = (V, E)$ if every cycle in G contains at least one node in F. Thus, it is sufficient for a packet header to count its visits to nodes in F.

Theorem 6 *If the underlying graph of a network has a feedback vertex set of size f, then there is a protocol that ensures correct delivery of a single message from S to R in that network using at most $f+1$ different headers.*

Proof: To send a message to R, processor S sends a packet containing the message and the header 0 to each of its neighbors. Each intermediate node not in F just forwards a packet it receives from one neighbor to all of its other neighbors. Upon receiving a packet, an intermediate node in F increments the header and forwards the packet to all of its other neighbors, provided the header was less than f. If the header has value f, the packet is ignored, since it has already visited some node in F at least twice. Hence, if there is a simple path of operational edges from S to R, some other packet would have taken it. ■

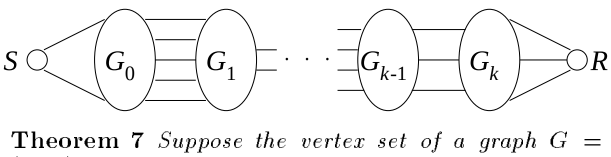
Unfortunately, finding a minimum size feedback vertex set in a graph is an NP-hard problem. Also, some graphs have no small feedback vertex sets. For example, any feedback vertex set for the complete graph on n nodes has size at least $n-2$ and any feedback vertex set in an $r \times c$ node mesh (with $r, c \geq 2$) has size at least $\lfloor r/2 \rfloor \cdot \lfloor c/2 \rfloor$.

Moreover, there are networks with only large feedback vertex sets for which it is possible to get a much better protocol. For example, the following graph requires no header bits at all.



It suffices for each intermediate node to forward the packets it receives from its left neighbor to both its right neighbor and its neighbor in the same column, but packets received from the neighbor in the same column are only forwarded to its right neighbor.

This example motivates another protocol.



Theorem 7 *Suppose the vertex set of a graph $G = (V, E)$ can be partitioned into a sequence of vertex sets $V_0, V_1, \ldots, V_k$, where $S \in V_0$, and $R \in V_k$, with the following properties:*

- *there are no edges in E connecting V_i and V_j for $j \neq i-1, i, i+1$*
- *b_i is the size of a maximum matching in the bipartite graph $(V_{i-1} \cup V_i, E \cap (V_{i-1} \times V_i))$, for $i = 1, \ldots, k$, and*
- *the subgraph $G_i = (V_i, E_i)$ of G induced by V_i has a feedback vertex set F_i of size at most f.*

Then there is a protocol that ensures correct delivery of a message from S to R in a network with this underlying graph using

$$\left\lceil \log_2(f+1) + \log_2\left(1 + \sum_{i=1}^{k} \lfloor (b_i - 1)/2 \rfloor\right)\right\rceil$$

bit headers.

Proof: Any simple path π from S to R contains at most b_i edges connecting nodes in V_{i-1} and nodes in V_i, since these edges form a matching in the bipartite graph $(V_{i-1} \cup V_i, E \cap (V_{i-1} \times V_i))$. In π, edges from V_{i-1} to V_i alternate with edges from V_i to V_{i-1} beginning and ending with the former. Hence, π contains at most $\lfloor (b_i - 1)/2 \rfloor$ edges from V_i to V_{i-1}. In total, any simple path contains at most $\sum_{i=1}^{k} \lfloor (b_i - 1)/2 \rfloor$ edges that go from a node in one set to a node in the preceding set.

Similarly, any segment of a simple path that stays within one set of nodes V_i visits nodes in F_i at most f times. Therefore, it suffices for a packet header to contain two counters, both initialized to 0. The first counter is incremented whenever the packet is sent along an edge that goes from a node in one set to a node in the preceding set. If doing so would cause this counter to exceed the value $\sum_{i=1}^{k} \lfloor (b_i - 1)/2 \rfloor$, the packet is ignored.

The second counter is re-initialized to 0 whenever the packet is sent along an edge connecting nodes in different sets and is incremented whenever the packet visits a node in F_i, for $1 \leq i \leq k$. If doing so would cause this counter to exceed the value f, the packet is ignored. The total number of bits needed for these two counters is the logarithm of the number of pairs of values these two counters can assume. ■

Note that the minimum feedback vertex set of an $r \times r$ mesh has size $\Omega(r^2)$ and any partition of the $r \times r$ mesh into two roughly equal sets has $\Omega(r)$ disjoint edges between its two parts. Thus, for two dimensional square meshes, the algorithms of Theorems 6 and 7 do not offer any significant improvement over the simple "hop count" algorithm.

4 Lower Bounds for Sequence Transmission

In this section, we consider the problem where S has an infinite sequence of messages, $d_1, d_2, \ldots$ to transmit to R, who must output the same sequence of messages, without omissions, duplications, or reordering. We assume that the messages are given to S via a one-way read-only input tape, and R must write the messages onto a one-way write-only output tape. We consider two different lower bound models: a *data oblivious* model and a *finite control* model.

In the data oblivious model, messages are treated as black boxes that are read from the input tape by S and must be written to the output tape by R. These black boxes can be sent as part of a packet. However, packets are not required to contain a black box: they are also allowed to contain only header bits. Our lower bounds for this model are on the number of header bits that packets must contain. This is in addition to any bits used to describe a black box in the packet. Black boxes are not allowed to be compared or modified, nor can they be stored at the receiver R. Processors S and R make all decisions based on their current state, the edge on which they receive a packet, and the contents of the packet header. In this model, there is no limit placed on the number of states of either R or S. In fact, they can each have an infinite number of states.

In the finite control model, on the other hand, both S and R are required to have some fixed number of states. The number of states is allowed to depend on the size and topology of the network, but not the number of messages read. However, in this model, messages are treated as a sequence of bits that can be manipulated and stored in any manner. For example, the bits of a single message could be sent sequentially, one per packet. Our lower bounds are on the size of the packets that are required.

Recall that, in both models, the intermediate processors of the network have no state. Thus, when an intermediate processor receives a packet, it decides on the basis of the header and the incoming edge along which the packet arrives, where to forward that packet and what header to use. The intermediate processors behave in the same manner no matter when they receive a packet with a particular header along a particular edge.

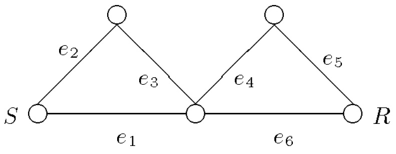
In the finite control model, we also assume that packets are routed *non-reflectively*: when an intermediate processor of the network receives a packet, it does not send a packet back on the incoming edge. A non-reflective protocol does allow R to send acknowledgments back to S. Note that protocols based on routing packets along simple paths are non-reflective. We also point out (without providing the details) that if we add the non-reflective assumption to the data oblivious model, then our lower bounds for that model can be adapted to apply even if we allow R to store (but not manipulate) an unlimited number of black boxes containing messages.

4.1 Edge order consistent paths

Our lower bounds are based on a function $\rho(G)$ of the network topology G, which we now describe. Let $G = (V, E)$ be an undirected graph and let $\prec$ be any total order on E. Say $E = \{e_1, \ldots, e_m\}$, where $e_1 \prec e_2 \prec \cdots \prec e_m$. A simple path $\pi = v_0, v_1, \ldots, v_l$ is *consistent with* $\prec$ if $\{v_{i-1}, v_i\} \prec \{v_i, v_{i+1}\}$ for $i = 1, \ldots, l-1$. Such a path can be described by a characteristic vector $c(\pi) = (c_1, \ldots, c_m) \in \{0,1\}^m$, where $c_j = 1$ if and only if edge e_j lies on path π.

Let P be a set of simple S–R paths in G that are consistent with $\prec$. We extend $\prec$ to a total order on P by defining $\pi \prec \pi'$ if the largest edge (according to $\prec$) in the symmetric difference of π and π' belongs to π'. More formally, $\pi \prec \pi'$ if and only if there exists a positive integer $k \le m$ such that $c(\pi)_k = 0$, $c(\pi')_k = 1$, and $c(\pi)_j = c(\pi')_j$ for $k < j \le m$ (or, in other words, the string $c(\pi)_m \cdots c(\pi)_1$ is lexicographically less than the string $c(\pi')_m \cdots c(\pi')_1$

For example, the series parallel graph



has all four of its simple S–R paths consistent with the order $e_1 \prec e_2 \prec e_3 \prec e_4 \prec e_5 \prec e_6$. This order is extended to the set of simple S–R paths as follows: $e_1e_4e_5 \prec e_2e_3e_4e_5 \prec e_1e_6 \prec e_2e_3e_6$.

We use the following observation about the total order $\prec$ on P.

Lemma 8 *Suppose $\pi \prec \pi' \prec \pi''$. If $c(\pi)_j = 1$, $c(\pi')_j = 0$, and $c(\pi'')_j = 1$, then there exists $k > j$ such that $c(\pi)_k = 0$ and $c(\pi'')_k = 1$.*

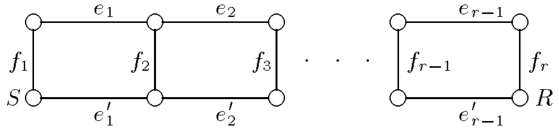
Proof: Since $\pi \prec \pi''$, there exists $k \le m$ such that such that $c(\pi)_k = 0$, $c(\pi'')_k = 1$, and $c(\pi)_l = c(\pi'')_l$ for $k < l \le m$. Similarly, $\pi \prec \pi'$ implies that there exists $h \le m$ such that $c(\pi)_h = 0$, $c(\pi')_h = 1$, and $c(\pi)_l = c(\pi')_l$ for $h < l \le m$, Since $c(\pi)_j = 1$ and $c(\pi')_j = 0$, it follows that $j < h$.

Suppose $k \le j$. Then $k < h$, so $c(\pi'')_h = c(\pi)_h = 0$. But $c(\pi')_h = 1$ and $c(\pi'')_l = c(\pi)_l = c(\pi')_l$ for $h = \max\{h, k\} < l \le m$. Hence $\pi'' \prec \pi'$, contradicting the fact that $\pi' \prec \pi''$. Therefore $k > j$. ■

Let $\rho_\prec(G)$ denote the number of simple S–R paths in G that are consistent with the total order $\prec$. Let $\rho(G)$ denote the maximum of $\rho_\prec(G)$ over all total orders $\prec$. Next, we bound the value of $\rho(G)$ for a number of different graphs.

Lemma 9 *Any $2 \times r$ mesh, $M_{2\times r}$, with its source in column 1 and its sink in column r, has $\rho(M_{2\times r}) = 2^{r-1}$.*

Proof: Let $V = \{(i,j) \mid i \in \{1,2\}, j \in \{1,\ldots,r\}\}$ denote the set of nodes of $M_{2\times r}$ and let $E = \{e_1, \ldots, e_{r-1}, e'_1, \ldots, e'_{r-1}, f_1, \ldots, f_r\}$, where $e_j = \{(1,j),(1,j+1)\}$, $e'_j = \{(2,j),(2,j+1)\}$, and $f_j = \{(1,j),(2,j)\}$, denote the set of edges of $M_{2\times r}$.

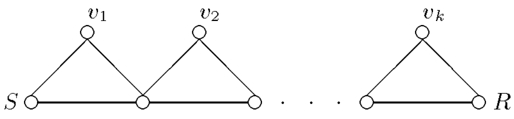


Consider the edge order $f_1 \prec e_1 \prec e'_1 \prec f_2 \prec \cdots \prec f_{r-1} \prec e_{r-1} \prec e'_{r-1} \prec f_r$. There are 2^{r-1} simple S–R paths, each specified by the subset of vertical edges it includes. All simple S–R paths are consistent with this edge order. Thus, $\rho(M_{2\times r}) = 2^{r-1}$. ■

Lemma 10 *Any series parallel graph G with source S and sink R has $\rho(G)$ equal to the number of simple S–R paths in G.*

Proof: Let $G = (V, E)$ be a series parallel graph with source S and sink R. Suppose G is the series or parallel composition of $G' = (V', E')$ and $G'' = (V'', E'')$, where S is the source of V' and R is the sink of V''. We define an order $\prec$ on E as follows: Recursively define an order $\prec'$ on G' and an order $\prec''$ on G''. Define $\prec$ so that it agrees with $\prec'$ when restricted to E', it agrees with $\prec''$ when restricted to E'', and $e' \prec e''$ for all $e' \in E'$ and $e'' \in E''$. By definition of a series parallel graph, all simple S–R paths in G are consistent with $\prec$. ∎

In particular, for the series parallel graph SAW_k depicted below, $\rho(SAW_k) = 2^k$.



Lemma 11 *The complete graph K_n on n nodes has $\rho(K_n) \geq \lfloor n/3 \rfloor !$.*

Proof: Suppose the set of nodes in the graph is $\{1, \ldots, n\}$, where $S = 1$ and $R = 2\lfloor n/3 \rfloor$. Order the edges in order of the smaller endpoints so, for example, all edges incident to node 1 precede all edges not incident to node 1. Among all edges with i as their smaller endpoint, order edge $\{i, i+1\}$ last; otherwise order these edges arbitrarily.

Let P be the set of all paths of the form $1v_1 2 3 v_2 4 \cdots (2i-1) v_i (2i) \cdots (2\lfloor n/3 \rfloor - 1) v_{\lfloor n/3 \rfloor} (2\lfloor n/3 \rfloor)$, where $v_1, \ldots, v_{\lfloor n/3 \rfloor} \in \{2\lfloor n/3 \rfloor + 1, \ldots, n\}$ are all distinct. Then P contains at least $\lfloor n/3 \rfloor !$ paths, all of which are consistent with the given order on the edge set. ∎

4.2 Data Oblivious Lower Bound

Theorem 12 *Consider any data oblivious protocol that enables processor S to transmit a stream of messages to processor R in a network with underlying graph G. The protocol must use headers of length at least $\left\lceil \log_2 \frac{\rho(G)}{\deg(R)} \right\rceil$ bits, where $\deg(R)$ is the degree of processor R in the network.*

Proof: Suppose the protocol uses headers of length less than $\left\lceil \log_2 \frac{\rho(G)}{\deg(R)} \right\rceil$ bits. Then we will construct an execution in which the protocol causes processor R to output an incorrect message. To do this, we use execution fragments in which all packets traveling along the links of a particular simple S–R path arrive safely, but all other packets are lost. Each time S sends a packet with a particular header along the first edge of this path, it causes various packets (with possibly different headers) to be sent along various links on this path. We can represent this using a tree, with one node for each packet sent along a link. When a packet is received by a processor, it may cause a packet to be sent along the path towards R and it may cause a packet to be sent along the path towards S. If so, these packets are its children in the tree. A leaf of this tree corresponds to a packet sent to S or R or which does not cause other packets to be sent along the path.

By definition, there is a total order $\prec$ on the edges of G consistent with exactly $\rho(G)$ simple S–R paths in G. Each of these paths ends with one of the $\deg(R)$ edges incident to R. Thus, there is some edge e incident to R which is the last node in $r \geq \rho(G)/\deg(R)$ of these paths. Say $e_1 \prec e_2 \prec \cdots \prec e_m$ are the edges of G and $\pi_r \prec \cdots \prec \pi_1$ are the simple S–R paths ending in e and consistent with $\prec$. Let $d_1, \ldots, d_r$ be a sequence of distinct messages. We use this sequence as input to S.

Consider the following execution. First all the packets on path π_1 arrive safely, but all other packets are lost. Eventually, there must be a packet (g_1, d_1) sent from S that would cause a packet (h_1, d_1) to be received by R, causing R to output message d_1. We will first suppose the packet (g_1, d_1) is lost. Later, we show that some ancestor of (h_1, d_1) in the tree of packets generated by (g_1, d_1) may actually be in the network. In effect, message d_1 is "sneaking its way" to R. We refer to this packet as a *ghost* of (g_1, d_1).

Next, packets sent on path π_2 arrive safely, but all other packets are lost. Eventually, message d_1 will be output and there will be a packet (g_2, d_2) sent from S that would cause a packet (h_2, d_2) to be received by R, causing R to output message d_2. Again, we will suppose that this message is lost. This continues until all p paths have been considered.

Since there are less than $\lceil \log_2 r \rceil$ header bits, there are less than r different headers, so $h_i = h_{i'}$ for some $1 \leq i < i' \leq r$. We construct an execution in which the sequence of messages $d_1, \ldots, d_{i'-1}, d_i$ is output by R. This is incorrect, since $d_1, \ldots, d_{i'-1}, d_{i'}$ is input to S.

Consider the first edge e_j on path π_i that is not on path π_{i+1}. Such an edge must exist, since $\pi_{i+1} \prec \pi_i$. When the packet (g_i, d_i) is sent from S, instead of being lost, we allow its ghost to proceed to edge e_j and hide there. More precisely, in the tree generated by (g_i, d_i) for path π_i, consider the closest ancestor of packet (h_i, d_i) that travels along edge e_j. Allow the execution to proceed along path π_i until this ancestor packet is traveling on edge e_j. All other packets that are generated by this execution fragment are lost. Note that the ghost (i.e. the ancestor packet hiding on edge e_j) does not interfere with the subsequent execution proceeding along path π_{i+1}, since e_j does not lie on π_{i+1}.

Now consider the point in our original execution just before each subsequent path π_k, $k > i+1$ is used. If the edge currently containing the ghost is not on path π_k the ghost will not interfere with the execution along path π_k and so the ghost can remain hidden there. Otherwise, by Lemma 8 applied to paths $\pi_k \prec \pi_{k-1} \prec \pi_i$, there is some edge further along path π_i on which the ghost can hide from the execution that will proceed along path π_k.

This hiding continues until the point in the execution just before packet $(h_{i'}, d_{i'})$ is delivered to R. If it were to be delivered, the message $d_{i'}$ would be output. (Note that R has already output $d_1, \ldots, d_{i'-1}$.) Instead, suppose this message is lost and the ghost (h_i, d_i) finally arrives at R. Processor R is not allowed to compare black boxes, $h_i = h_{i'}$, and all packets arrive at R along edge e_m. Therefore, R is not able to distinguish the packet (h_i, d_i) from the packet $(h_{i'}, d_{i'})$ and will incorrectly output message d_i. ■

4.3 Finite Control Lower Bound

Theorem 13 *Consider any non-reflective finite control protocol that enables processor S to transmit a stream of bits (or messages) to processor R in a network with underlying graph G. The protocol must use packets of length at least $\left\lceil \frac{1}{2}\log_2 \frac{\rho(G)}{\deg(S)\deg(R)} \right\rceil$ bits.*

Proof: By definition, there is a total order $\prec$ on the edges of G consistent with exactly $\rho(G)$ simple S–R paths in G. Each of these paths begins with one of the $\deg(S)$ edges incident to S and ends with one of the $\deg(R)$ edges incident to R. Thus, there is some edge e incident to S and some edge e' incident to R such that $r \geq \rho(G)/\deg(S)\deg(R)$ of these paths begin with e and end with e'. Let P denote the set of these r paths and let $e_1 \prec e_2 \prec \cdots \prec e_m$ denote the edges of G.

We consider execution fragments in which packets traveling along the links of some path $\pi \in P$ arrive safely, but all other packets are lost. Since the protocol is non-reflective, each packet q sent by S (along edge e) will cause at most one packet q' to be received by R (along edge e'). Let $Q(\pi)$ denote the set of all such pairs (q, q') that can occur using only path π. Note that, since the protocol is valid, $Q(\pi) \neq \phi$.

We say that path π is *covered* if $Q(\pi) \subseteq \bigcup_{\pi' \neq \pi} Q(\pi')$. If no path in P is covered, then for every path $\pi \in P$, there is a pair (q, q') that appears only in $Q(\pi)$. This implies that there are at least r different pairs.

Suppose the protocol uses packets of length $L < \frac{1}{2}\log_2 r$. Then there are at most $2^{2L} < r$ different pairs. Hence, some path π is covered. We will show that, in this case, the protocol can be made to output an incorrect result. It follows that the protocol must use packets at least $\left\lceil \frac{1}{2}\log_2 \frac{\rho(G)}{\deg(S)\deg(R)} \right\rceil$ bits long.

Assume that the input tape begins with a large number (to be chosen below) of copies of the same message, d (for example, 0). Suppose path π is covered.

Consider the following execution of the protocol, which repeatedly performs phases initiated by the sender S interleaved with phases initiated by the receiver R. A phase initiated by S begins with S performing some action, which may include reading from the input tape, changing internal state, or sending a packet. Any packets sent on edges in the path π are transmitted successfully by intermediate processors, but all other packets are lost. By the non-reflective assumption, this causes at most one packet to be received at R. This phase concludes when a packet is received at R or no packets remain in the network. Next, R is allowed to perform an action, which may include writing to the output tape, changing internal state or sending a packet. Again, any packets sent on the path π are transmitted successfully by intermediate processors, any other packets are lost, and the phase ends when a packet is received at S or no packets remain in the network. Throughout this execution, we can describe the system state simply by the internal state of S, the internal state of R, and any packet on the path π.

Both S and R have finite controls, there is at most one packet in the system at any one time, and the size of any packet is finite. Thus, there are a finite number of possible system states. This means that if the sequence of copies of the message d on the input tape is long enough, the same system state will be repeated: the system will be in a loop that will continue until a different message is read from the input tape. In the process of going around this loop once, the message d must be written to the output tape at least once. We are going to force R to go through this loop one more time than S, and thus the message d will be written to the output tape too many times.

In this loop, starting from some arbitrary system state σ, there is some sequence of packets $q'_1, \ldots, q'_\ell$ that are received by R. For each packet q'_i, let q_i be the packet sent by S that causes q'_i to be received by R, so $(q_i, q'_i) \in Q(\pi)$. Since the path π is covered, there is some other path π_i such that $(q_i, q'_i) \in Q(\pi_i)$. Consider the point where paths π and π_i first diverge. Since all paths in P begin with the same edge e, this occurs at some intermediate node A_i. The packet q_i sent by S will cause a packet to be received at node A_i, which sends a packet along path π, causing q'_i to be received by R. Since the processor at node A_i has no state, it will also send a packet along path π_i, which can cause q'_i to be received by R. Thus, it is possible for the packet q'_i to be successfully delivered, while at the same time causing a ghost to be created that may eventually cause a duplicate of packet q'_i to be delivered to R. We will store the ghost on the last edge e_{j_i} of π_i that does not occur on path π. Note that $\pi_i = \pi_k$ can occur for $i \neq k$. Also, it is possible that $e_{j_i} = e_{j_k}$, even if $\pi_i \neq \pi_k$ (because π_i and π_k can merge before rejoining path π).

The ghosts stay put until all of them are stored. When S and R are next both in the same states as they were in σ and there are no other packets in the system, we let the ghosts cause copies of q'_i to be delivered to R, one at a time, in the proper order. During this period of time, all packets sent by R are lost, so S does not receive any packets and, thus, remains in the same state. After the entire sequence of duplicate packets has arrived at R, R has gone around the loop exactly once and, so, has returned to its previous state. Thus, the system is now in state σ.

The computation then proceeds as before, neither party having any knowledge that R has gone around the loop an extra time. There will eventually be some message $d' \neq d$ on the input tape, where d' is not read until after the sequence of duplicate packets has been delivered. When R writes d' to the output tape, it will have output too many copies of d before d' and, thus, the protocol is incorrect.

It remains to show that we can deliver the ghosts in the proper order. There will be one ghost created and stored each time the system goes around the loop once. The order they are created is as follows: ghosts stored on different edges are created and stored from largest edge to smallest edge (according to $\prec$), and ghosts on a given edge are created and stored in the order that they are eventually used to deliver duplicate packets to R. If a ghost traveling along a path π_i produces any packets on other paths, these packets are lost.

By storing the ghosts in the described order, the ghost caused by q_i is able to travel from S to e_{j_i} without interfering with previously stored ghosts. In fact, except possibly for edge e_{j_i}, no edge already storing a ghost is used. Storing the ghosts from largest edge to smallest edge ensures this, since the paths are edge order consistent and the protocol is non-reflective.

The order the ghosts are stored also ensures that a ghost stored on edge e_{j_i} can cause packet q_i' to be delivered to R without interfering with ghosts that are stored for later delivery. Because e_{j_i} is the last edge of path π_i that is different from the path π, the delivery proceeds entirely on the path π once it leaves edge e_{j_i}. Furthermore, the ghosts stored on edge e_{j_i} are in the order they are to be delivered, so they don't interfere with one another during delivery. ∎

4.4 Lower Bounds for Specific Networks

Theorems 12 and 13 can be combined with lemmas from Section 4.1 to obtain lower bounds on header lengths for transmitting a sequence of messages through certain networks.

Corollary 14 *For sequence transmission in the $2 \times n$ mesh, with S and R at opposite ends of the mesh, any data-oblivious protocol requires headers of length $\Omega(n)$ and any non-reflective finite control protocol requires packets of length $\Omega(n)$.*

Corollary 15 *For sequence transmission in the n-node complete graph, any data-oblivious protocol requires headers of length $\Omega(n \log n)$ and any non-reflective finite control protocol requires packets of length $\Omega(n \log n)$.*

Corollary 16 *For sequence transmission in a series parallel graph with p simple S–R paths and constant degree at S and R, any data-oblivious protocol requires headers of length $\Omega(\log p)$ and any non-reflective finite control protocol requires packets of length $\Omega(\log p)$.*

These lower bounds match, to within a constant factor, the $O(\log(\text{number of simple } S\text{–}R \text{ paths}))$ upper bounds shown by Dolev and Welch [DW97]. Also, as in the single message case, lower bounds for a particular graph extend to all graphs that contain it as a minor.

Proposition 17 *Suppose G' is a minor of G and S' and R' are the supernodes of G' containing S and R, respectively. Then any sequence transmission protocol for G from S to R is also a sequence transmission protocol for G' from S' to R'.*

Corollary 18 *For sequence transmission in a fixed degree mesh M with n nodes and dimension at least 2, any data-oblivious protocol requires headers of length $\Omega(n)$ and any non-reflective finite control protocol requires packets of length $\Omega(n)$.*

Proof: Here, we describe the case of a $(4r+2) \times k$ mesh with S appearing in row 1 and column 1 and R at the opposite corner of the mesh. The technique used extends easily to any mesh of dimension at least 2. For the $(4r+2) \times k$ mesh, we see that we have the $2 \times m$ mesh as a minor, for $m = (2r+1)(k-2)+1$. This mesh appears in a "snakelike" ordering (e.g., see [L92], page 144) in the original mesh. To construct this minor, we proceed as follows, for $j = 0, \ldots, r-1$: We delete all but the last two edges between rows $4j+2$ and $4j+3$, and all but the first two edges between rows $4j+4$ and $4j+5$. We also combine nodes $(4j+1, k-1)$, $(4j+1, k)$ and $(4j+2, k)$ into a supernode and nodes $(4j+3, k)$, $(4j+4, k)$, and $(4j+4, k-1)$ into another supernode. This creates a "turn of the snake" for one side of the mesh. Turns for the other side are created by combining nodes $(4j+3, 2)$, $(4j+3, 1)$ and $(4j+4, 1)$ into a supernode and nodes $(4j+5, 1)$, $(4j+6, 1)$, and $(4j+6, 2)$ into another supernode. ∎

Corollary 19 *For sequence transmission in a hypercube of dimension $d \geq 2$, any data-oblivious protocol requires headers of length $\Omega(d)$ and any non-reflective finite control protocol requires packets of length $\Omega(d)$.*

Proof: Since the hypercube of dimension $d-1$ is Hamiltonian (for example, using a Gray code), the $2 \times 2^{d-1}$ mesh is a minor of the hypercube of dimension d. Any pair of nodes S and R in the original hypercube can always be forced to have distance at least 2^{d-2} in the resulting mesh by positioning the start and end of the mesh appropriately within the Hamiltonian cycle that appears in the hypercube of dimension $d-1$. ∎

5 Open Problems

A number of interesting questions remain. For the problem of sending a single message through a network, the two-dimensional mesh seems likely to require a large amount of header information, since it has no small vertex feedback set. However, since it is planar, our techniques do not provide interesting lower bounds for this network. We conjecture $\Omega(\log n)$ header bits are required to ensure delivery of one message in the two-dimensional mesh. Ideally, we would like a general characterization of the number of header bits needed to send a single message through a network in terms of graph theoretic properties of the underlying graph.

For the problem of sending a sequence of messages through a network, it would be interesting to extend our lower bounds to other graph families. In particular, we conjecture that all graphs require $\Omega(\log p)$ header bits for sequence transmission, where p is the number of simple S–R paths. This would match, within a constant factor, the size of the headers used by Dolev and Welch's protocol. Furthermore, it would be worthwhile to either extend our lower bounds to models that are more general than the data oblivious model and the non-reflective finite control model, or to demonstrate that better protocols exist for more general models.

It would also be interesting to examine the effect of adding memory to the intermediate nodes of the network. In particular, we would like to quantify the tradeoff between the amount of memory at intermediate nodes and the packet header lengths for sending a sequence of messages, assuming a bounded amount of communication per message. We conjecture that, in general, it is impossible to solve the sequence transmission problem using only a constant number of header bits and a constant number of states at each processor.

Acknowledgments

We are grateful to Adi Rosén for introducing us to this topic and for many helpful discussions. This research was supported in part by grants from the Natural Sciences and Engineering Research Council, the Information Technology Research Centre of Ontario, and the Fields Institute. Part of this research was conducted while the second author was a visitor at Laboratoire de Recherche en Informatique, Université de Paris-Sud, Orsay, France.

References

[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.

[AAG87] Afek, Awerbuch, and Gafni, *Applying Static Network Protocols to Dynamic Networks*, 28th FOCS, 1987, pages 358–370.

[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.

[AG91] Afek and Gafni, *Bootstrap Network Resynchronization*, 10th PODC, 1991, pages 295–307.

[AE86] Awerbuch and Even, *Reliable Broadcast Protocols in Unreliable Networks*, Networks, 16, 1986, pages 381–396.

[AGH90] Awerbuch, Goldreich, and Herzberg, *A Quantitative Approach to Dynamic Networks*, 9th PODC, 1990, 189–203.

[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*, Journal of High Speed Networks, vol. 5, no. 4, 1996, pages 365–381.

[BSW69] Bartlett, Scantleburg, and Wilkinson, *A Note on Reliable, Full-Duplex Transmission over Half-Duplex Links*, CACM, vol. 12, 1969, pages 260–261.

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

[FL90] Fekete and Lynch, *The Need for Headers: An Impossibility Result for Communication over Unreliable Channels*, CONCUR '90, Lecture Notes in Computer Science, vol. 458, Springer-Verlag, 1990, pages 199–215.

[F98] Fich, *End-to-end Communication*, Proceedings of 2nd OPODIS, Amiens, France, 1998, pages 37–43.

[H92] Herzberg, *Connection-Based Communication in Dynamic Networks*, 11th PODC, 1992, pages 13-24.

[K30] 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.

[L92] Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaugmann, 1992.

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

[S76] Stenning, *A Data Transfer Protocol*, Computer Net., vol. 1, 1976, pages 99-110.

[V83] Vishkin, *A Distributed Orientation Algorithm*, IEEE Trans. on Information Theory, vol. 29, 1983.

[WZ89] Wang and Zuck, *Tight Bounds for the Sequence Transmission Problem*, 8th PODC, 1989, pages 73–83.