Polynomial End-To-End Communication

(Extended Abstract)

Baruch Awerbuch * Yishay Mansour † Nir Shavit ‡

Abstract

A dynamic communication network is one in which links may repeatedly fail and recover. In such a network, though it is impossible to establish a path of unfailed links, reliable communication is possible, if there is no cut of permanently failed links between a sender and receiver.

We consider the basic task of of *end-to-end* communication, that is, delivery in finite time, of data items generated on-line at the sender, to the receiver, in order and without duplication or omission.

The best known previous solutions to this problem had exponential complexity. Moreover, it has been conjectured in [AG88] that a polynomial solution is impossible.

This paper disproves this conjecture, presenting the first polynomial end-to-end protocol. The protocol uses methods adopted from shared memory algorithms, and introduces novel techniques for fast load balancing in communication networks.

1 Introduction

A basic task in any network is that of *end-to-end* communication, that is, delivery in finite time, of data items generated at a designated sender process, to a designated receiver process, without duplication, omission or reordering of the data items. The data items could represent transactions of a stock exchange, speech or video signal, military commands, etc. In almost all cases, data items are generated on-line and are not available at the beginning of the protocol's execution.

In a reliable network, where links never fail, this task is performed easily by establishing a fixed communication path between the sender and the receiver, and sending all data items along this path. Unfortunately, existing communication networks, e.g. the ARPANET [MRR80], DECNET [Wec80], have a *dynamic* topology in the sense that links may repeatedly fail and recover. Since one cannot relay on the use of any single communication path, the problem becomes much more complicated.

The "classical" approach to handle the problem is to construct a new communication path after the previously existing path fails, taking care to purge any messages in transit on the old path. However, this approach is very limited, since its implementations (as in [Fin79, Gal76, AAG87, AS88, AAM89, AGH89]) require strong assumptions regarding the allowable patterns of link failures in the network. In [AAG87, AS88, AAM89] for example, the assumption is that the *whole network* stabilizes for a period of time long enough to allow construction of a path and communication over it. The weakest assumption among

*Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139; ARPANET: baruch@theory.lcs.mit.edu Supported by Air Force Contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract CCR8611442, and a special grant from IBM.

†Laboratory for Computer science, MIT, 545 Tech. sq., Cambridge, MA 02139. partailly supported by NSF 865727-CCR, ARO DALL03-86-K-017 and ISEF fellowship.

‡Hebrew University, Givat-Ram, Jerusalem 91904; Supported by Israeli Communications Ministry Award. Currently visiting the *TDS* group at *MIT*, supported by *NSF* contract no CCR-8611442, by *ONR* contract no N0014-85-K-0168, by *DARPA* contract no N00014-83-K-0125, and a special grant from *IBM*.

those above, presented in the broadcast protocol of [AGH89], still requires that all the edges on some path between the sender and receiver be operational for the entire time period required to construct that path and communicate the data over it.

This assumption is overly optimistic, since for example, if every edge has a constant probability of being operational (or not operational) at a given time, then the probability of the whole path being operational at a given time is exponentially small in the length of the path.

However, one can see that the existence of an operational communication path is *not* a necessary condition for communicating between two nodes. In fact, as stated in [AE86], the necessary condition is merely that there is "eventual connectivity" between the sender and receiver, in the sense that there is no permanent cut (see [Vis83]) of failed edges between them. More precisely, there exists no partition of the network into two sets, one containing the sender and the other the receiver, such that from some time and on, no operating edge connects a node in one set to a node the other set.

Early works [Vis83, AE83, AE86] solving the end-to-end problem under the minimal conditions alone, were based on the use "unbounded sequence numbers", implying that both the message size and amount of memory needed, grow with number of data items transmitted. In other words, the complexity of the protocols was *unbounded* in terms of the true input to the problem, that is, in the size of the network.

The elegant and surprising work of Afek and Gafni [AG88] presented the first "bounded complexity" end-to-end protocol. Unfortunately, this solution required *exponential* message complexity (because the number of times a given data item is sent over a network link is exponential). Moreover, it was believed by many researchers, and conjectured in [AG88], that

> *any solution to the end-to-end problem with bounded space (not a function of number of messages transmitted), has an exponential message complexity,*

implying that if only the necessary condition is assumed, no reasonable solution to the end-to-end problem exists.

In this paper, we disprove this conjecture, presenting an end-to-end protocol that is *polynomial* in both messages and space. The protocol is based on a new technique for sequence numbering, that combines the sequential timestamp schemes used in shared memory algorithms ([Lam86, IL87, DS89]), with a novel and highly fault tolerant load balancing method. The high fault tolerance of the load balancing method results from the ability to preserve global properties based on maintaining local information only (as in Goldberg-Tarjan [GT88] and Ahuja-Orlin [AO87]). We suspect that this technique can be applied to many other problems as well.

2 Problem Statement

2.1 The network model

Consider a communication network in the form of an undirected graph $G(V, E)$, where the nodes are the processors and the edges are the links of communication. Each *undirected* link consists of two *directed* links, delivering messages in the opposite directions. Below we describe the properties of a directed link.

Each link has a *finite* capacity, in the sense that only constant number of messages is allowed to be in transit on a given link at a given time (I.e. we consider only protocols that obey this property.) The communication over links obeys the FIFO rule, that the sequence of messages *received* over the link is a prefix of the sequence of messages sent over the link. Also, the communication is completely *asynchronous*, namely, there is no a priori bound on link delays.

A link is *non-viable* if starting from some message, and on, it will not deliver any further messages to the other end-point; for those messages the delay in ∞. In this case, the sequence of messages received is a *proper* prefix of the sequence of messages sent. Otherwise, the link is *viable*. An undirected link is *viable* if both of the directed links that it consists of are viable.

We say that the sender is *eventually connected* to the receiver if there exists a (simple) path from the sender to the receiver, consisting entirely of viable links. Clearly, if non-viable links create a *cut* of the network, disconnecting the sender from the receiver, then, eventually, the sender will not be capable of delivering messages to the receiver.

2.2 The end-to-end problem

The purpose of the end-to-end protocol is to establish a (directed) "virtual link" connecting sender with the receiver, to be used for delivery of data items from the sender to the receiver. It is required that this virtual link be viable if and only if the sender is eventually connected to receiver. This virtual link should have the same properties as a "regular" network link, namely:

Safety: The sequence of data items output by receiver is a prefix of the sequence of data items input to the sender.

Liveness: If the sender is eventually connected to the receiver, then each data item input to the sender is eventually output by the receiver.

2.3 The complexity measures

We consider the following complexity measures.

Communication: The number of bits transferred in the network, per data item. That is, this the total number of bits sent in the period of time between two successive data item deliveries at the receiver.

Space: The maximal amount of space required by a node's program throughout the protocol.

Time: The maximal length of a time interval between two successive data item deliveries at the receiver, under the assumption that delays of viable links are upper-bounded by 1 time unit.

Computation time: The maximal number of local computation steps a node performs in the interval of time between two successive data item deliveries, provided that this data item is not the last data item.

Definition 2.1 A protocol is *bounded* if its communication, space, time, and computation time complexities are <u>independent</u> of the number of data items, and only depend on the size of the network.

Definition 2.2 A protocol is *polynomial* if its communication, space, time and computation time complexities are upper-bounded by <u>polynomials</u> of the size of the network.

We would like to stress the fact here that being able to send (receive) infinite number of messages does not require either sender (receiver) to have infinite space. A single buffer at the sender (receiver) is sufficient in order to store next data item to be transmitted. The precise formulation of this "interactive" statement of the problem can be found in [LMF88].

2.4 Relations to other models

The model described above is called the "∞-delay model" in [AG88], and the "fail-stop model" in [AM88]. As mentioned in the introduction, our motivation here is to deal with networks frequently changing topology. In such *dynamic* networks, links may fail and recover many times (yet nodes never fail). Each failure or recovery of network link is eventually reported at both end-points by some underlying link protocol. As pointed out by Afek and Gafni [AG88], this dynamic model is easily reducible to the model described above. The simulation of the dynamic model by the fail-stop model is as follows. A message to be forwarded on a link is stored in a buffer, until the link recovers, and all the previous messages have been delivered. A protocol similar to the data-link initialization protocol of Baratz and Segall [BS88] is used to guarantee that no messages are lost or duplicated. Each link in the dynamic network that does (does not) fail forever is represented by a viable (non-viable) link in the fail-stop model. Any two nodes eventually connected in the dynamic network are eventually connected in the fail-stop network.

3 Informal Description

In this section an informal outline of the protocol and the main ideas leading to it are presented.

The presentation begins with a description of a very simple protocol using *unbounded sequence numbers*, i.e, messages of unbounded size. This algorithm is then refined through a series of steps, to derive a *bounded* protocol with polynomial complexity.

The simple unbounded protocol involves two types of messages. The "data" message contains some data item which is to be delivered to the receiver; the "acknowledgment" (in short, "ack") message serves to acknowledge the receipt of the data item at the receiver. Both the data and the ack messages carry the (unbounded) sequence number of the data item in question; the data message also carries the data item itself.

The protocol works as follows. Once the sender inputs data item of sequence number ℓ, it sends the data message (indexed with ℓ) to all its neighbors. Every node receiving this message forwards it to all its neighbors, unless it already received a data message with higher sequence number. The receiver, upon receiving the data message of sequence number ℓ, sends back an ack message with sequence number ℓ. The sender will input next data item (with sequence number $\ell + 1$) only after it received the ack for the data item ℓ. Note that the protocol creates a situation where many messages may be in transit on the same edge at the same time.

The first very simple modification guarantees that at a given time, there is at most one message in transit on an edge. This is achieved by letting every process send the ack of ℓ on every edge the data message with ℓ was received. A process does not send a future data messages on an edge, before it receives the ack for the last data message sent on that edge. Though this only means that messages are stored in the process rather than on the channel, the number of messages stored per edge can be reduced to one, by noticing that it suffices to maintain only the message with highest sequence number. Observe, however, that the complexity is still unbounded, since the size of the sequences numbers ℓ is not bounded.

At this point, the following important observation is made. Although the protocol uses an unbounded number of different label values, at each point in time, the number of different label values in the system is linear in the number of edges. Let it first be shown how, assuming that the sender has a *label oracle* telling it whether a given ℓ value exists in the system, a polynomial end-to-end protocol using only bounded size labels can be designed (this protocol is called the *main protocol*). The oracle enables the sender to compute the set of values that exists in the system, denoted by $\bar{\ell}$.

The idea is that the sender, in order to send a new data item, generates a label that is "greater" than all the labels in $\bar{\ell}$. A well known mechanism that achieves this goal is a bounded sequential time stamp system (see [IL87, DS87].) The time stamp system to be used will be of size N, where N is polynomial in n, and each label has size of $O(N)$ bits. Such a system guarantees that for any set of values $\bar{\ell}$, that has less than N values, a label can be found, such that it is not in $\bar{\ell}$ and is "greater" than any value that is in $\bar{\ell}$. Note that the system also defines the operation "greater" associated with the labels. Using such a time stamp system and the oracle, the sender can always find a new label, that is greater than all the labels in the system, and is of size polynomial in n.

Most of the effort in the protocol is implementing a mechanism similar to the above oracle. One would like a node to "know" locally that it is clean of a given label ℓ, that is, all references to it can be eliminated. To achieve this two modifications are introduced. First, the sending of replies to a message with label ℓ is restricted to edges that were traversed by the message ℓ. This implies that a node can receive a reply to message ℓ only on edges on which message ℓ has been sent. Second, a node does not send a new data message on an edge until it receives a reply to the previous message sent along it.

Assume that node v receives a message with label ℓ and forwards it. After some time it received a reply for ℓ on all the edges that it forwarded the message (and also received a later message in the meantime). Node v is thus ready to inform the sender that it received a reply to all the messages it had forwarded. At this point none of the local variables in v contain label ℓ, and so node v creates a *token* that includes its name v, the label ℓ and the list of edges that the message was sent on.

The token is an indication that node v is clear of references to label ℓ.

The remaining unresolved problem is that of getting all the tokens with label ℓ to the sender, so it can deduce that ℓ is not in the system, since all nodes were clean when the tokens were created, and all tokens are in the sender. If the sender collects the tokens from some of the nodes, but not all of them, it can check locally if it has all the tokens in the following way. Using the list of edges in the tokens of label ℓ, the sender can create a set S of nodes that certainly received the message ℓ. If from every node in S, the sender has received a token, then there is no node in S that sent message ℓ to a node not in S. Since the sender is in S, nodes not in S never received message ℓ.

The problem in getting all the tokens to the sender, is that on one hand, some of the edges that the tokens are sent on may fail. This problem may be solved by duplicating tokens and sending them on different paths. However, duplicating tokens disables the sender from checking locally if it has all the tokens of ℓ that were created.

Assume one could bound the token capacity of the network so that each process apart from the sender (whose capacity is unbounded) could contain at most some fixed number of tokens. After the network's token capacity is reached, the creation of a new token in any process would imply that some token was received by the sender. By simple pigeon hole arguments it can be shown that after a bounded number of such token creations, there would be a message ℓ for which all the tokens were received by the sender.

The solution to the token collection problem is thus a fault tolerant load balancing protocol to assure that tokens are evenly distributed among processes in the system, maintaining the property that every process apart from the sender has bounded capacity. The protocol assures that no matter which communication path becomes eventually connected, there are sufficiently many tokens on this path, and one will eventually be forwarded to the sender.

The basic idea behind the load balancing protocol is the following. Each node has some quantity of tokens. Assume for a moment that the network is synchronous and static. Consider the following protocol. At every even clock tick a node sends a token to each neighbor if it had Δ or less tokens than it, and sends nothing otherwise. At every odd clock tick, each node updates its neighbors about the number of tokens it has. After a polynomial number of iterations the protocol will, given that $\Delta = \Omega(n)$, converge to a steady state (i.e. no tokens are sent). The surprising fact is that a very similar protocol will converge, and with polynomial communication, in an asynchronous network where links may fail.

In the protocol, the number of tokens stored in a node is bounded by a polynomial in n. In order to enforce this bound, a node that has more than a certain amount of tokens is *blocked*, and does not respond to messages of the main protocol. This guarantees that a node will not generate tokens locally. Furthermore, it can be proved that this rule does not cause deadlock. The bound on the number of tokens in each node implies that the number of tokens that can be in the network in any point of time is polynomial is n (which is less than N). Any label that is sent, for which not all the tokens have been collected, is assumed to exist in the system. The value of N will be chosen to be more than the "label capacity" of the network, and thus the sender can always generate a new label.

The formal proof of the load balancing protocol uses amortized analysis to show that though it could be that some given token cycles forever in the network, the total number of tokens sent in the period of time between two successive message receipts at the receiver, is bounded by a polynomial in n.

4 Acknowledgements

We wish to thank Yehuda Afek and Adi Rosen for their helpful comments, especially in pointing out an error in the counting arguments of the complexity proof.

References

[AAG87] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In 28^{th} *Annual Symposium on Foundations of Computer Science*, IEEE, October 1987.

[AAM89] Yehuda Afek, Baruch Awerbuch, and Hezi Moriel. Overhead of resetting a communication protocol is independent of the size of the network. May 1989. Unpublished manuscript.

[AE83] Baruch Awerbuch and Shimon Even. *A Formal Approach to a Communication-Network Protocol; Broadcast as a Case Study.* Technical Report TR-459, Electrical Engineering Department, Technion-I.I.T., Haifa, December 1983.

[AE86] Baruch Awerbuch and Shimon Even. Reliable broadcast protocols in unreliable networks. *Networks*, 16(4):381–396, Winter 1986. Previous version was "A Rigorous Treatment of a Communication Protocol: Broadcast as a Case Study." That version contained proofs.

[AG88] Yehuda Afek and Eli Gafni. End-to-end communication in unreliable networks. In *Proceedings of the 7^{th} Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada*, pages 131–148, ACM SIGACT and SIGOPS, ACM, 1988.

[AGH89] Baruch Awerbuch, Oded Goldreich, and Amir Herzberg. A quantitative approach to dynamic networks. May 1989. Unpublished manuscript.

[AM88] Baruch Awerbuch and Yishay Mansour. Efficient topology update algorithms. 1988. Unpublished manuscript.

[AO87] Ravindra K. Ahuja and James B. Orlin. *A Fast and Simple Algorithm for the Maximum Flow Problem.* Working Paper 1905-87, MIT Sloan School of Management, June 1987.

[AS88] Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks. In 29^{th} *Annual Symposium on Foundations of Computer Science*, pages 206–220, IEEE, October 1988.

[BS88] Baratz and Segall. Reliable link initialization procedures. *IEEE Trans. Comm.*, February 1988.

[DS87] D. Dolev and N. Shavit. A note on bounded time-stamp systems. July 1987. Unpublished manuscript.

[DS89] D. Dolev and N. Shavit. Bounded concurrent time-stamps are constructible! In *Proceedings of the 21^{st} Annual ACM Symposium on Theory of Computing, Seattle, Washington*, ACM SIGACT, ACM, 1989.

[Fin79] Steven G. Finn. Resynch procedures and a fail-safe network protocol. *IEEE Trans. Comm.*, COM-27(6):840–845, June 1979.

[Gal76] Robert G. Gallager. *A Shortest path Routing Algorithm with Automatic Resynch.* Technical Report , MIT Lab. for Information and Decision Systems, March 1976.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. *J. ACM*, 35(4):921–940, October 1988. Also appeared in 18th stoc (1986).

[IL87] A. Israeli and M. Li. Bounded time stamps. In 28^{th} *Annual Symposium on Foundations of Computer Science, White Plains, New York*, pages 371–362, Comp. Soc. of the IEEE, IEEE, 1987.

[Lam86] Leslie Lamport. On interprocess communication. part ii: algorithms. *Distributed Computing*, (12):86–101, 1986.

[LMF88] Nancy A. Lynch, Yishay Mansour, and Alan Fekete. The data link layer: two impossibility results. In *Proceedings of the 7^{th} Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada*, pages 149–170, ACM SIGACT and SIGOPS, ACM, Toronto, Canada, 1988. Also, Technical Memo MIT/LCS/TM-355, May 1988.

[MRR80] John McQuillan, Ira Richer, and Eric Rosen. The new routing algorithm for the arpanet. *IEEE Trans. Comm.*, 28(5):711–719, May 1980.

[Vis83] U. Vishkin. A distributed orientation algorithm. *IEEE Trans. Info. Theory*, June 1983.

[Wec80] S. Wecker. DNA: the digital network architecture. *IEEE Transactions on Communication*, COM-28:510–526, April 1980.