END-TO-END COMMUNICATION

Faith E. Fich
Department of Computer Science, University of Toronto, CANADA
Laboratoire de Recherche en Informatique, Université de Paris-Sud, FRANCE

Abstract

End-to-end communication is the problem of sending a sequence of messages from one processor, the sender, to another processor, the receiver, through an unreliable communication network. This paper surveys positive and negative results concerning deterministic protocols for end-to-end communication in asynchronous networks and presents open questions.

1 Introduction

Sending information from one designated processor to another designated processor is a fundamental activity in network based systems. However, networks, especially large networks, are subject to faults, for example, message loss, link failures, and processor crashes. Because of their complexity, communication networks are usually built in a sequence of layers [25]. At each layer (above the physical layer), there is a protocol that uses the services provided by the layer below to provide better or more complicated services to the layer above. This may introduce other sorts of faults, for example, message duplication or reordering.

End-to-end communication is the problem of sending a sequence of messages from a sender S to a receiver R, through an unreliable communication network. The network can be modelled by a graph, where the nodes represent processors and the edges represent links between them. Packets are used to transmit messages and control information. Each packet contains at most one message and zero or more header bits. Processors receive packets along their in-coming links and transmit packets along their out-going links. When a packet arrives, an a intermediate processor can forward copies of the packet along any subset of its out-going links, possibly with different headers. In some cases, intermediate processors can also store the information contained in a packet and send it later. The sequence transmission problem and the reliable message transmission problem are other names for the end-to-end communication problem.

In an asynchronous network, the time for a packet to travel along a link is finite but unbounded. Two packets travelling along the same link may take different amounts of time. Depending on the link, it may even be possible for messages to arrive in a different order than which they were sent. A link which has failed (i.e. stops transmitting any packets and never recovers) can be viewed as a link on which the travel time for packets is infinite. This means that a processor cannot distinguish between a packet which has not yet arrived because it is travelling slowly and a packet which will never arrive because the link on which it is supposed to be travelling has failed. Links may be bidirectional (i.e. the link from one processor to another processor is operational if and only if the link in the opposite direction is operational) or unidirectional (i.e. opposite direction links may fail separately from one another). Packet losses may be viewed as dynamic link failures: the link fails, thereby losing the packet, and then starts working again.

End-to-end communication has been studied in other settings, such as synchronous models (where the time for a packet to traverse a link is fixed and the same for all links) [7], semi-synchronous models (where there is a known upper bound on the time for a packet to traverse a link) [29, 24], and models where faults and/or packet delivery times are probabilistic [20, 21].

Randomized protocols that allow a small probability of error have also been considered [17]. These are all beyond the scope of this survey.

2 Networks with No Intermediate Nodes

The simplest interconnection network of interest consists of two processors with one link going from each processor to the other processor. The end-to-end communication problem for this simple network is part of the problem of implementing the data link layer of a communication network on top of its physical layer.

When packet loss, duplication, and reordering cannot occur, it suffices for S to simply send its messages in packets without headers. They are received by R as long as the link from S to R remains operational.

Another simple protocol is for S to wait for an acknowledgement from R before sending each subsequent message in the sequence. This variant eliminates the problem of packet reordering by not putting more than one packet on a link at a time.

Packet loss, duplication, and reordering can all be tolerated using sequence numbers [23]. Essentially, S sends the i'th message in a packet with header i and keeps sending it until receiving an acknowledgement packet with header i from R. Then S proceeds to the next message. Similarly, R continues to send the same acknowledgement until it receives a packet containing the next message. Unfortunately, in this protocol, the length of packet headers grows without bound. However, Wang and Zuck [28] have shown that any protocol tolerating both packet reordering and duplication requires unbounded sequence numbers.

The alternating bit protocol [12, 20] uses single bit headers: the least significant bit of the sequence number. It tolerates both packet loss and duplication. With the possibility of packet loss, Fekete and Lynch [14] have proved that there are no correct headerless protocols; thus the alternating bit protocol uses the shortest possible headers.

Counting protocols [2, 26] are another way to handle packet reordering and loss. They are based on the following observation: Suppose R knows an upper bound b on the number of old packets in the network at a particular time. Then R can ensure that it has a new packet by waiting for $b+1$ copies of the same packet to arrive. In these protocols, the value of b depends on the number of packet losses that have occurred. Hence, b and the number of packets successfully transmitted per message can be arbitrarily large. However, Afek et al. [2] have proved that any protocol tolerating both packet reordering and loss either requires unbounded sequence numbers or can cause R to receive an unbounded number of packets per message.

3 General Networks

For general networks, end-to-end communication is not possible if there is a permanent cut of failed links separating S and R. Therefore, there must exist a path from S to R (and a path from R to S) such that any packet sent along an edge of the path infinitely many times will be successfully transmitted along that edge infinitely many times. This condition is known as eventual connectivity [5, 6, 13].

When links are subject to packet duplication, reordering, and loss, end-to-end communication can be achieved by protocols that use sequence numbers. Each intermediate processor passes on the packets it receives to all of its other neighbours, flooding the network. To avoid having one message generate an infinite number of packets (in networks with cycles), each intermediate processor also remembers the largest sequence number it has seen and only forwards packets with that sequence number [8, 27].

An alternative method to limit the number of packets in the network is to add a hop count to the packet headers [22]. Each time a packet is forwarded along an edge, its hop count is incremented. A packet is destroyed whenever its hop count exceeds the number of processors in the network or the length of the longest simple path between S and R.

Instead of counting the number of hops a packet has travelled, it suffices to identify a feedback vertex set (a set of vertices, at least one of which must appear in every cycle) and count until the number of visits to processors in this set exceeds the set's cardinality [16]. For many networks, this quantity may be substantially smaller.

The slide protocol [4] was designed for networks with bidirectional links that can fail (but not recover). It behaves very differently from sequence number based protocols: No headers are used and intermediate processors do not duplicate or destroy packets. Stacks of at most n packets are stored at each intermediate processor, one for each edge. (Here n is the number of processors in the network.) Packets repeatedly move from one stack to a lower position on a stack in a neighbouring processor. Only one processor sends packets to any particular stack. That processor is notified when packets are removed, so it can maintain a sufficiently good upper bound on the stack height. Packets that reach R are immediately removed from the network, so R's stacks are essentially always empty. Because the number of packets in the network is always bounded above by a polynomial in n, R can use the counting protocol approach to determine the next message to output.

Kushilevitz, Ostrovsky, and Rosén [19] improve this protocol by having different packets cancel one another, producing empty packets. Only one packet is stored at each intermediate processor. Two counters replace each stack: one to keep track of the number of empty packets in the stack and the other to keep track of the number of occurrences of the packet stored at the processor. This protocol uses space logarithmic in n. Both this protocol and the original slide protocol communicate only a polynomial (in n) number of packets for each message in the sequence.

Any protocol that tolerates only link failures (such as the previous two protocols) can be extended to handle packet loss and duplication by using the alternating bit protocol on every link [5]. At each intermediate processor, one bit of storage is required for each in-coming and out-going link, and storage for one packet is needed for each out-going link.

Protocols that do not store any information at intermediate nodes can automatically tolerate the crash and recovery of intermediate processors, since there is no problem with corrupted memory when an intermediate processor recovers.

Without packet duplications or reorderings, bounded packet headers suffice for end-to-end communication. The idea is to treat each simple S–R path separately. First, suppose that only link failures can occur. Then, 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. Dolev and Welch [13] combine this approach with bounded sequence numbers to obtain a protocol with $O(\log p)$ bit headers, where p is the number of simple paths in the network from S to R. storage at S. The protocol can be extended to tolerate packet loss and duplication, without storage at intermediate processors, by incorporating a version of the alternating bit protocol along each path, rather than along each link.

For end-to-end protocols that do not use any memory at intermediate nodes, Adler and Fich [1] have proved lower bounds on the size of packet headers necessary to tolerate packet loss. These lower bounds are expressed as a function of the network topology. For many networks, including the complete graph, series-parallel graphs, and fixed degree meshes, the lower bounds are $\Omega(\log p)$, demonstrating the optimality of Dolev and Welch's protocol.

4 Open Questions

For general networks in which only packet loss and duplication can occur, the difficulty of end-to-end communication is not well understood.

- Among protocols that do not use storage at intermediate processors, is it possible to use shorter headers if only packet duplication or only link failures occur?
- Are there protocols that do not use packet headers and only use a constant amount of storage at each intermediate processor? If not, is space that is linear in the degree of the intermediate processor enough? Is it even possible to use $o(\log n)$ space per link, where n is the number of processors in the network?
- More generally, is there a tradeoff between the packet header length and the amount of storage at intermediate processors?
- How much communication is necessary when only link failures occur? Does this depend on the amount of storage available at intermediate nodes or the length of packet headers?
- Is end-to-end communication easier for bidirectional links than for unidirectional links?

End-to-end communication is just one part of the problem of implementing the network and transport layers of a communication network.

- How do issues such as congestion and flow control affect end-to-end communication?
- Does the ability to detect packet loss (for example, a corrupted packet in an error detecting code) or notification of link failure make end-to-end communication easier?
- For links which are known to have very long transmission times, for example in satellite communication, what is the best way to perform pipelining? Are better protocols possible?

There are also a number of interesting related problems:

- simultaneously sending streams of messages between different pairs of processors at the same time, and
- reliable broadcast (and multicast) of message streams.

What are the complexities of these problems in terms of packet header length, space, and amount of communication? What are the tradeoffs?

Acknowledgements

I am grateful to Adi Rosén for introducing me to this topic and to both him and Micah Adler for many helpful discussions. This work was supported, in part, by grants from the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.

References

[1] M. Adler and F. Fich, *The Complexity of End-to-End Communication in Memoryless Networks*, manuscript, 1998.

[2] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck, *Reliable Communication Over Unreliable Channels*, JACM, vol. 41, no. 6, 1994, pages 1267–1297.

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

[4] Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosèn, and N. Shavit, *Slide–The Key to Polynomial End-to-End Communication*, Journal of Algorithms, vol. 22, no. 1, 1997, pages 158–186.

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

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

[7] A. Aho, J. Ullman, A. Wyler, and M. Yannakakis, Comput. Math. Appl., vol. 8, 1982, pages 205–214.

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

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

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

[11] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilizing End-to-End Communication*, Journal of High Speed Networks, vol. 5, no. 4, 1996, pages 365–381.

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

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

[14] A. Fekete and N. 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.

[15] A. Fekete, N. Lynch, Y. Mansour, and J. Spinelli, *The Impossibility of Implementing Reliable Communication in the Face of Crashes*, JACM, vol. 40, no. 5, 1993, pages 1087–1107.

[16] F. Fich and A. Rosén, manuscript, 1997.

[17] O. Goldreich, A. Herzberg, and Y. Mansour, *Source to Destination Communication in the Presence of Faults*, 8th PODC, 1989, pages 85–101.

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

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

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

[21] Y. Mansour and B. Schieber, *The Intractability of Bounded Protocols for On-Line Sequence Transmission Over Unreliable Channels*, JACM, vol. 39, no. 4, 1992, pages 783–799.

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

[23] M. Stenning, *A Data Transfer Protocol*, Computer Net., vol. 1, 1976, pages 99–110.

[24] C. Sunshine and Y. Dalal, *Connection Management in Transport Protocols*, Computer Net., vol. 2, 1978, pages 454–473.

[25] A. Tanenbaum, *Computer Networks, Third Edition*, Prentice Hall, 1996.

[26] E. Tempero and R. Ladner, *Recoverable Sequence Transmission Protocols*, JACM, vol. 42, no. 5, 1995, pages 1059–1090.

[27] U. Vishkin, *A Distributed Orientation Algorithm*, IEEE Trans. on Information Theory, vol. 29, 1983, pages 624–629.

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

[29] D. Wang and L. Zuck, *Real-Time Sequence Transmission Problem* 10th PODC, 1991, pages 111–124.