LIDS-P-1844

Reliable Communication on Data Links[1]

John M. Spinelli

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

December, 1988

[1]This research was conducted at the MIT Laboratory for Information and Decision Systems, and the Center for Intelligent Control Systems. It was supported in part by a National Science Foundation Graduate Fellowship, by the National Science Foundation under Grant NSF-ECS-8310698, by the Defense Advanced Research Projects Agency under Grant ONR/N00014-84-K-0357, and by the Army Research Office under Grant ARO DAAL03-86-K-0171.

Abstract

The problem of sending a set of data packets from a source to a destination across a single data link is considered. Reliable communication is defined as the delivery of such a set of packets in order, and without any losses or duplicates. Protocols for transmitting and receiving data packets are modeled as automata with outputs. It is shown that when the sending and receiving automata can be "synchronized," reliable communication can be achieved.

The problem of communicating reliably is studied when the sending and receiving nodes may fail and loose their memory. It is shown that when there is no upper bound on the packet transmission delay, reliable communication and synchronization are impossible. Conversely, it is shown that when there is an upper bound on delay, synchronization and reliable communication can be achieved.

Contents

List of Figures

Chapter 1

Introduction

Reliable data communication is the delivery of some set of information packets from a data source to a data sink, in order, and without any lost or duplicated packets. When components of a communication system or network are subject to failures, providing reliable communication can become difficult or impossible. Although there has been a considerable amount of work on data link and network protocols to achieve reliability, the exact situations in which reliable communication is or is not possible are not completely understood. The specific properties of the communications model used, such as transmission delay characteristics and processor failure modes, determine whether reliable communication can be achieved. This work takes a fundamental look at reliable communication on single data links and in computer networks, and studies the impact of model characteristics on the ability to communicate reliably. Several results are presented for single link communication, and future work on reliable network communication is proposed.

Consider the case of reliable communication across a single data link connecting two nodes of a network. In a recent paper [3], Baratz and Segall introduce protocols which can be used to provide reliable communication in this case, provided that node processors have two bits of nonvolatile memory that can survive node processor failures. Using their model, we show that when no such memory is available reliable communication is impossible. A similar result has recently been shown by Lynch, Mansour and Fekete independently from, and concurrently with, this research[11].

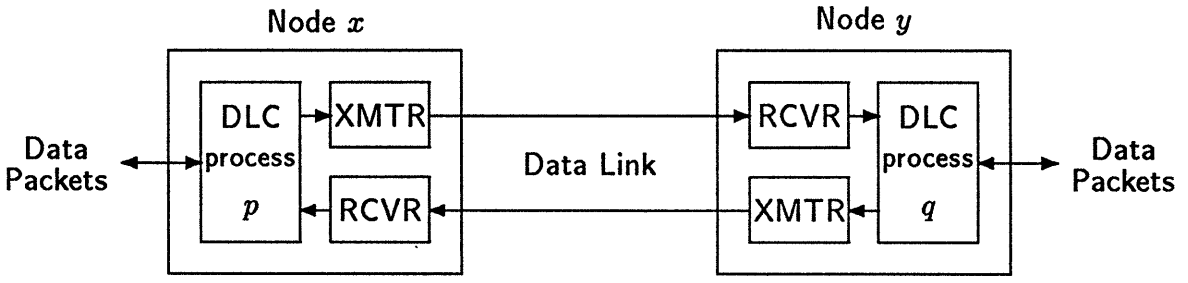

Figure 1.1: Communication Model

Conversely, when a different communication model is used which upper bounds the message transmission delay, we show that reliable communication can be achieved.

1.1 Communication Model

Figure 1.1 shows two node processors connected by a bidirectional link. This could be part of a larger network not shown. Data Link Control (DLC) processes p and q communicate with each other by exchanging *frames*. It is the responsibility of the DLC process to accept data packets from some source, and to transmit them reliably across the link using some protocol. We assume throughout this research that the operation of the DLC processes does not depend on the contents of data packets. This assumption allows us to study the information that a protocol needs in order to transmit data reliably, without allowing protocols to "hide" such information in data packets. It is also consistent with traditional ideas of layering in networks[4].

When process p has a frame to send to q, it gives the frame to the transmitter at x. The transmitter appends error detection information to the frame, and transmits it on the link. Noise may corrupt the transmitted information, but the order of transmitted frames is maintained. There is a nonzero probability that the frame will be received error free. The error detection mechanism at the receiver is such that the probability of an undetected error in a frame is insignificant, and is assumed

henceforth to be zero. When a frame arriving at a receiver contains errors, it is discarded. For convenience, we consider discarded frames not to have been received. Otherwise, the error detection information is removed, and it is given to process q. Communication from node y to node x works in a similar way. This motivates the following models for link operation.

1.1.1 Link Models

All of the communication links considered in the report share the following characteristics.

1. A transmitted frame may be received error free exactly once, or never received.
2. If a frame A is transmitted before a frame B from one end node, and both frames are received, then A is received before B.

This model has left the transmission delay incurred by a received frame unspecified. Two distinct ways of modeling delay will be considered.

Bounded delay link: There is a known upper bound D on the transmission delay of a received frame.

Unbounded delay link: There is no known upper bound for the transmission delay of a received frame.

A major goal of this research is to determine how the type of link delay affects the ability to provide reliable communication.

In practice, most physical data links can be modeled as having bounded delay, but there are several reasons for considering the unbounded case as well. Highly variable message delay may make it difficult to obtain a bound which is tight a large fraction of the time. If a loose bound is used, this can introduce inefficiencies into the DLC protocols. Even when a good bound on delay is known for a particular link, from an engineering prospective it may not be desirable to have the correct operation

of protocols depend on this bound. As the system grows and evolves, it would be necessary to modify the protocol to reflect a changing delay bound. A modern trend has been to integrate the operation of many different networks. In such a situation, what we are viewing as a link may actually be an underlying network whose delay characteristics are unknown. Apart from this there is a theoretical question of how the delay characteristics of a link affect the ability to communicate reliably.

1.1.2 Failure Model

In order to examine whether reliable communication is possible in a network, it is necessary to have a model which specifies how the nodes and links fail, as well as how they behave in normal operation. Failures can be divided into two general types: simple and Byzantine. A simple failure means that a node or link stops operating. In Byzantine failures, nodes may act in an arbitrary manner perhaps specifically designed to cause problems. This report is concerned solely with simple failures. The problem of making network layer protocols robust with respect to Byzantine failures is currently being addressed by Perlman [12].

The failure of a link results in transmitted frames not being received in one or both directions for some time interval. We distinguish between an actual physical failure of the link and the method by which the DLC process tries to detect that such a failure has occurred. A failure detected by one end node may indicate an actual physical failure, or may be a decision by the end node that the link is performing so poorly that data communication should not be attempted. In our model, the only method that the DLC uses to detect failures is to run a protocol which attempts to transmit frames, and observes any frames received from the other end node. For example, if no frames were received for some specified time period, the protocol at an end node would conclude that the link or the other end node has failed. This implies that failures of short duration may not be detected, and that there may not be a one to one correspondence between failures detected at each end node of

a link. Other methods of detecting failures (such as receiving a lost carrier signal from a modem) could easily be incorporated into the model, but are ommitted for notational simplicity. In Chapter 2 we will show that developing protocols which properly detect link failures is an important and difficult part of achieving reliable communication.

When a node fails, any process operating at it is halted, and any frames being received or transmitted by the node are lost. After a failure, if the node has sufficient nonvolatile data memory which can survive a node failure, it can restart its DLC process in the state that the process was in before the failure occurred. Otherwise, the process must be restarted in some initial state. The node is assumed to have some fixed storage which contains the code that constitutes the DLC process itself. A major goal of this thesis is to determine how the presense or lack of nonvolatile data storage affects the ability to provide reliable communication.

When a node has nonvolatile data memory, a node failure can often be modeled, with respect to either of the previous link models, as a failure of all the links incident on a given node. Let a node fail during some time interval T. If the node restarts in the same state that it was in before the failure, then this situation is equivalent to one in which the node does not fail, but frames received or transmitted by it during T are discarded. Therefore, if the DLC process is capable of achieving reliable communication in the presence of link failures, it can also deal with node failures of the above type. Conversely, when a node failure results in the loss of a DLC process' state information, it is a fundamentally different event from a link failure. For this reason, we will be concerned with situations when nodes do not have nonvolatile data memory. Henceforth, the term *node failure* is used to imply a loss of state information.

1.2 Definition of Reliable Communication

Consider the problem of communicating reliably across a single data link. In defining reliable communication it is usually assumed that there is a single infinitely long sequence of data packets which must be sent from the source to the destination without losses or duplicates [1,7]. This definition might be appropriate when the source and destination nodes are not subject to failures. If node failures are permitted, the definition should be modified for two principle reasons.

1. The data packets may be buffered within the node itself. They may not be able to survive a node failure since the node has no nonvolatile data memory.

2. When nodes fail it is often desirable to inform higher layers of the failure, and to cancel or reroute sessions as appropriate. When the node becomes operational, it is unrealistic to assume that it would have the same set of data packets to transmit as it had before the failure.

In order to present a definition of reliable communication on data links which is appropriate for node failures, the following notation is introduced. A time interval during which a node is operational is called a *node up period* (NUP). During a *node down period* (NDP), the node is not operational. Throughout a NUP, the DLC process at a node is responsible for defining intervals of time during which it believes the link and the other end node are operational, and during which an attempt is made to transmit or receive data packets. Each such interval is called a *link up period* (LUP). A *link down period* is defined as an interval between successive LUPs. Each NUP may contain zero or more LUPs. The DLC process determines the start and end of a LUP by using one or more protocols which will be described more formally in Chapter 2. There is not necessarily a one to one correspondence between the LUP's defined by the two end nodes of a link. Establishing some type of correspondence between LUPs is one of the goals of DLC protocols.

In order to describe the sequence of data packets accepted from the data source, and released to the data sink during a LUP, we use the concept of a string. A string is a finite ordered sequence of zero or more elements. The empty string is denoted ϵ. A prefix of a string s is any initial sequence of s, including s itself. The empty string is a prefix of every string.

Let x and y be two nodes connected by a bidirectional data link, and consider the problem of sending data packets reliably from x to y. For convenience in the following definition, assume that the LUPs at x and y can be numbered, and that each data packet can be uniquely identified. Let $S_i(t)$ be the string of packets that node x has accepted from its data source by time t of LUP i at x. Similarly, let $R_j(t)$ be the string of packets that node y has released to its data sink by time t of LUP j at y. For convenience, we define $S_i(t)$ to be equal to the empty string for all t prior to the start of LUP i at x. For all t following the end of LUP i at x, we define $S_i(t)$ to be equal to the string of all packets accepted during LUP i. A similar convention applies to $R_j(t)$.

Definition 1 (Reliable Communication) Communication of messages from x to y is defined as reliable if and only if the following properties hold.

1. If $R_j(t')$ is nonempty for some t' in LUP j at node y, then there exists a LUP i at node x such that $R_j(t)$ is a prefix of $S_i(t)$ for all t in LUP j.
2. Let $R_i(t)$ be a nonempty prefix of $S_k(t)$, and let $R_j(t')$ be a nonempty prefix of $S_l(t')$. Then, $i < j$ if and only if $k < l$.

Together, the two properties assert that if node y releases some string of data packets to the data sink during a LUP, then there exists exactly one corresponding LUP at x during which the data packets were accepted from the data source. Furthermore, if two LUPs at y release data packets, then the two corresponding LUPs at x are in the same relative order as those at y.

The above definition does not contain a liveness property. The issue of liveness will be addressed when specific protocols are discussed in Section 2.1.

1.3 Summary of Report

Using the previous definitions, Chapter 2 examines the problem of reliable communication on a single data link. It is shown that communicating reliably depends upon being able to synchronize protocols at the source and the receiver. Two types of synchronization are studied and are shown to be equivalent. It is shown that for a given communication model, if there exists a protocol which achieves synchronization, then reliable communication can be achieved as well.

The major results of Chapter 2 show that, using the unbounded delay link model, synchronization and reliable communication are impossible when node failures may occur. Conversely, it is shown that using the bounded delay link model, both synchronization and reliable communication are possible.

Chapter 2

Single Link Communication

The problem of reliable communication on data links has been studied for many years, but there still exists a lack of understanding concerning the specific capabilities of DLC protocols. For example, a recently published paper [3] has shown that HDLC (a common bit oriented DLC process) can allow inadvertent loss of data packets in situations involving link failures. In this chapter, it is shown that when node failures may occur, reliable communication is possible for bounded delay links, and impossible for unbounded delay links.

2.1 Protocol Model

A model of the DLC process operating at each end node of a link is needed to make precise statements about the capability for reliable communication. The model used here is similar to those in [1] and [3]. The DLC process operating at a node x is modeled by one or more automata which have inputs and outputs associated with their state transitions. In this report, we assume that the automata are finite. This restriction is not essential, and will be removed in later work. Formally, one such automaton at node x is defined by a Mealy machine [15, p. 42], with some additional capabilities which will be described. The Mealy machine can be expressed as a seven-tuple.

$$A_x = (Q_x, \Sigma_x, \Delta_x, \delta_x, \lambda_x, i_x, f_x) ,$$

where Q_x is a set of states,

Σ_x is the input alphabet

Δ_x is the output alphabet,

δ_x is a transition function[1] mapping $Q_x \times \Sigma_x$ into Q_x,

λ_x is a transition function[2] mapping $Q_x \times \Sigma_x$ into Δ_x ,

i_x $i_x \in Q_x$ is the initial state,

f_x $f_x \in Q_x$ is the final state.

A similar definition is made for node y and defines an automaton A_y. A particular execution of an automaton during some time interval is called an *instance* of the automaton. Together, A_x and A_y define a *protocol.* The DLC processes at x and y may consist of several protocols, each defined by a pair of communicating automata. Typical tasks of individual protocols will be addressed in the next section.

The input alphabet Σ_x may be subdivided into two classes, R_x and T_x. R_x is the set of message inputs which can be received from y, and T_x is a set of *timeout* transitions. Therefore, $\Sigma_x = R_x \cup T_x$. A timeout transition from a given state has a particular value of time associated with it. If the automaton has been in the given state for the specified amount of time, then the transition occurs. Thus, an automaton has a relative sense of time in terms of how long it spends in each state, but has no absolute time reference. This facility allows protocols to react to delay conditions on the link, and to repetitively transmit messages. Some of the message inputs which are received from y may also contain a piggy-backed data packet. The automaton can store a finite number of received data packets, and can release a previously received packet to the data sink when making a transition. The behavior of the automaton may depend upon receiving a message transmitted together with a data packet, but cannot depend upon the contents of data packets

[1] $\delta_x(q, a)$ is the next state if the current state is q, and a is the input.

[2] $\lambda_x(q, a)$ gives the output associated with a transition from state q, with input a.

themselves.

The output alphabet Δ_x can be divided into three classes, M_x, S_x, and ϵ. Therefore, $\Delta_x = M_x \cup S_x \cup \{\epsilon\}$. M_x is the set of messages which the automaton may try to send across the link to node y. S_x is a set of start signals for the other automata which comprise the DLC process at node x. When a member s of S_x is output, A_x terminates, and the automata indicated by s is started in its initial state. The empty output ϵ indicates that no message is sent for a particular transition. When sending a message $m \in M_x$, A_x may piggy-back a data packet which it has previously accepted from the data source. The automaton can store a finite number of accepted data packets, and can append a previously accepted data packet to a transmitted message.

Let A_x and A_y be a pair of automata defining some protocol. The automata are communicating, so each should accept the other's messages. Thus, $M_x \subset R_y$ and $M_y \subset R_x$. Since each DLC process may be composed of more than one automaton, when A_x is operating at node x, it may receive input messages from any of the automata in the DLC process of node y.

Although A_x and A_y are deterministic automata, a particular sequence of received messages may result in several different state transition sequences depending upon the relative timing of the received messages. Thus both a message sequence and a particular timing are needed to uniquely determine a particular path in an automaton. The appropriate timing is indicated by having members of T_x interspersed in the input to A_x.

At the start of a NUP, one of the automata (called the *initial automaton*) in a node's DLC process is started in its initial state. A particular execution of an automaton at a node is called an *instance* of the automaton. When the node crashes, any running automaton is halted, and all state information is lost. An automaton being *halted* is different from it terminating and starting another automaton as described above.

For convenience, the DLC process has been modeled as a collection of automata which can call one another, and each of which presumably performs a different task. However, one can also view this situation as one large automaton performing the entire DLC process. This approach will be taken in Section 2.5 when proving impossibility results.

This protocol model is general enough to express the types of operation usually found in practical DLC processes such as HDLC, but several issues relating to the practical operation of such a protocol model have been omitted since they are unimportant to the theoretical issues under study.

2.2 Protocol Types

The DLC process at a node is typically composed of at least two protocols which deal with different aspects of reliable communication. A *link initialization* (LI) protocol is often used at the start of a NUP and between successive LUPs at a node. This protocol is responsible for determining when to begin the next LUP, and for establishing a set of conditions on the link which will enable the data transmission protocol which follows it to work correctly. When the LI protocol decides that a LUP should begin, it terminates and starts a data transmission protocol called an ARQ (automatic repeat request) protocol. The ARQ protocol attempts to transmit and receive data packets and determines if and when the LUP should end. To end a LUP, the ARQ protocol terminates and starts the LI protocol.

For convenience, all frames transmitted by an instance of one of the two automata which define an ARQ protocol are called *ARQ frames.* Similarly, frames transmitted by an instance of one of the automata which define some protocol P are called *protocol P frames.*

2.2.1 Correct ARQ Initialization

A major reason for breaking the reliable communication problem up into two pieces is that there are well understood ARQ protocols which are known to transmit data packets reliably provided that they are properly initialized by an LI protocol. Examples of such protocols include Stop and Wait, Go back-n, and Selective Repeat[4]. The following definitions specify what is required to properly initialize an ARQ protocol, and describe the sense in which it operates correctly. Assume that A_x and A_y define an ARQ protocol for transmitting data from node x to node y.

Definition 2 (Corresponding Automata Instances) An instance A_x^i of A_x and an instance A_y^j of A_y are said to correspond if:

1. All of the ARQ frames received by A_x^i were sent by A_y^j, and
2. all of the ARQ frames received by A_y^j were sent by A_x^i.

Definition 3 (ARQ Correctness) An ARQ protocol is correct if given that an instance A_x^i of A_x and an instance A_y^j of A_y correspond, then the string of packets released to the data sink by A_y^j is a prefix of the string of packets accepted from the data source by A_x^i.

A proof that standard ARQ methods such as Go back-n are correct in the above sense can be found in [4].

2.2.2 Synchronization

Due to the success of ARQ protocols, much of the research on reliable communication has focused on the link initialization (LI) protocols which are used to recover from link and node failures. The idea is to design an LI protocol which brings the two end nodes of a link to a state such that an ARQ protocol can begin transmitting packets reliably in the above sense. Such an LI protocol is said to accomplish synchronization of the two end nodes.

Weak Synchronization

Weak synchronization is a requirement on an LI protocol which will be shown to enable correct ARQ initialization. It is *weak* in the sense that it is a less stringent requirement than *strong* synchronization which will be discussed below. In the definition which follows, we assume that there is a protocol defined by A_x and A_y, and state the conditions it must satisfy to achieve weak synchronization.

Definition 4 (Weak Synchronization) Let an instance of A_x start in its initial state at time t_1, and enter its final state at time t_2. The protocol achieves weak synchronization at node x if in the interval $[t_1, t_2]$, node x must receive a protocol frame that was sent by an instance of A_y during $[t_1, t_2]$. A protocol achieves weak synchronization if it achieves weak synchronization at both nodes x and y.

Since y must send a protocol message to x during $[t_1, t_2]$, the definition implies that the protocol must have been running at y at some time during this interval. This required time overlap between instances of A_x and A_y is the important property that will allow an ARQ protocol to be correctly initialized.

Strong Synchronization

Strong synchronization is the somewhat standard type of synchronization which is used in [3]. Let nodes x and y be connected by a link. An LI protocol provides strong synchronization if it achieves the *clear property* defined below.

Definition 5 (Clear Property) During an LDP at one end node, there is a time when the link is down at the other end node, and no ARQ frames are in transmission on the link. Such a point in time is called a *clear point.*

At this point we can observe the difference between strong and weak synchronization. The definition of weak synchronization implies that there must be a time during an LDP at an end node when the link is down at the other end node, but does not address the issue of ARQ frames being present on the link.

2.2.3 Protocol Liveness Conditions

DLC protocols are designed to accomplish a particular task provided that the link and the end nodes operate properly, and the frame delay is acceptably small. For LI protocols, the task to be performed is to terminate under the proper conditions, and to start an ARQ protocol. A *terminable* protocol is defined as follows.

Definition 6 (Terminable Protocol) Assume that both end nodes are operational for a sufficiently long time interval following the start of a protocol at one end node, and that during this interval all transmitted packets are received with sufficiently small delay. The protocol is terminable if both automata must have entered a final state within a finite time after starting, and if the arrival of any protocol frames at an end node after the protocol terminates does not cause the protocol to restart.

What constitutes a "sufficiently long" time interval, and a "sufficiently small" frame delay depends on the definition of a particular protocol. It can be seen from the last part of this definition that an LI protocol being terminable depends upon the behavior of the ARQ protocol which follows it.

Liveness conditions for ARQ protocols are stated in a similar way, except that the required task is releasing one or more packets to a data sink.

2.3 Achieving Reliable Communication

In this section it is shown that if a protocol exists for achieving weak synchronization, then that protocol achieves strong synchronization as well. This shows that although their definitions are different, the two types of synchronization are in a sense equivalent. Given that a correct synchronization protocol exists, we show that it can be used in conjunction with an ARQ protocol to achieve reliable communication. We assume that the required liveness conditions for all protocols are met.

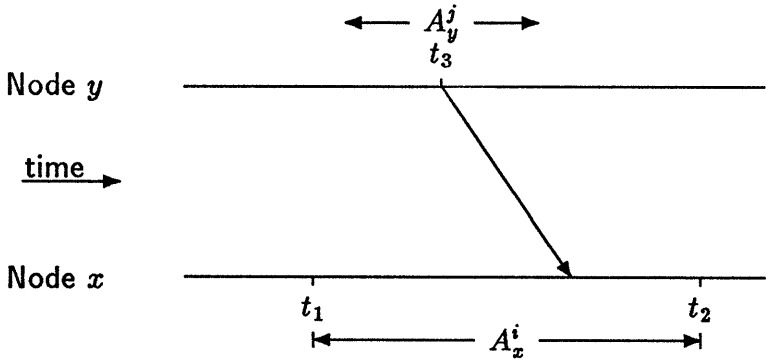

Figure 2.1: Timing Diagram for Lemma 1

Lemma 1 Let P be a protocol defined by A_x and A_y which achieves weak synchronization. Assume that an instance A_x^i of A_x starts at time t_1 and terminates at time t_2. This is shown in Figure 2.1. From the definition of weak synchronization, A_x^i must receive at least one protocol P frame which was transmitted by y in $[t_1, t_2]$. Let the last such frame be transmitted by an instance A_y^j of A_y at time t_3. Then, A_y^j must receive a protocol P frame from A_x^i before time t_3.

Proof. By contradiction. Assume that A_y^j does not receive any protocol P frames from A_x^i before time t_3. We construct a modified scenario at x which will cause the protocol to fail. Let the instance A_x^i be delayed by Δ time units, where $\Delta > t_3 - t_1$. Similarly, let all messages received by A_x^i be delayed by Δ. A_x^i receives the same messages with the same relative timing as before, but does not achieve weak synchronization since $t_3 < t_1 + \Delta$. □

Theorem 2 If a terminable protocol P achieves weak synchronization, then it also achieves strong synchronization.

Proof. Refer to Figure 2.2. Let protocol P be defined by A_x and A_y. Assume that an instance A_x^i of A_x executes at node x starting at t_1 and terminating at t_2. A_x^i must receive at least one protocol P frame which was transmitted by y in $[t_1, t_2]$.

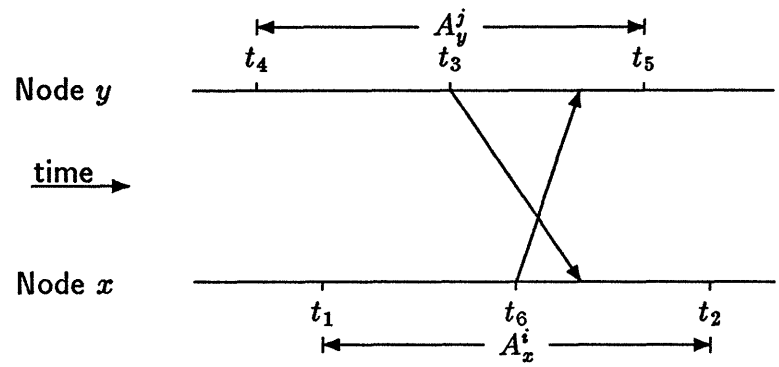

Figure 2.2: Timing Diagram for Proof of Theorem 2

Let the last such frame be transmitted by an instance A_y^j of A_y at time t_3. Similarly, A_y^j must receive a least one protocol P frame that was transmitted by x in $[t_4, t_5]$. Let the last such frame be transmitted by A_x^i at time t_6. From Lemma 1, A_y^j must receive a frame from A_x^i before t_3. Therefore, the link is clear of ARQ frames from x to y in $[t_3, t_2]$. Using Lemma 1 in the other direction shows that the link is clear of ARQ frames from y to x in $[t_6, t_5]$. This implies that the greater of t_3 and t_6 is a clear point. Thus, strong synchronization is achieved. □

To show that the ability to perform synchronization implies that reliable communication is possible we rely on the correctness of well known ARQ strategies mentioned in Section 2.2.1. Correctness proofs for such strategies may be found in [3] and [4]. The use of synchronization in conjunction with ARQ strategies to provide reliable communication is not new. An alternate development of it can be found in [3].

Theorem 3 A link which can be synchronized can provide reliable communication.

Proof. To prove the theorem, we must show that synchronization allows nodes to initialize an ARQ protocol in such a way as to satisfy the conditions for correctness in Section 2.2.1. Assume that nodes x and y execute an LI protocol

which achieves synchronization before starting any instance of an ARQ protocol. Let a correct ARQ protocol be defined by A_x and A_y. Assume that some instance A_y^j of A_y executes at node y and releases some nonempty string of data packets to the data source. Since each instance of A_x or A_y is both preceded and followed by a clear point, all ARQ frames which are received by A_y^j must have been sent by a single instance of A_x, say A_x^i. Similarly, any ARQ frames received by A_x^i must have been sent by A_y^j. Therefore, the conditions for correct initialization of the ARQ protocol are satisfied, and reliable communication is achieved. □

2.4 Unbounded Delay Links

In this section it is shown that reliable communication and both types of synchronization are impossible on unbounded delay links with node failures, when protocols are modeled as stated in Section 2.1. It is common in papers on network protocols [14,8] to use the unbounded link model with node failures, and yet to assume that synchronization is possible. The results of this section show that such an assumption is inconsistent with the model.

To obtain the desired impossibility results, we model a candidate DLC protocol for achieving synchronization or reliable communication as a pair of automata with output, communicating via the unbounded delay link model. We then construct a scenario of node failures and message timing that will cause the protocol to fail to meet its objective.

2.4.1 Automata Paths

Let A_x and A_y be two automata defining a protocol on the link connecting x and y. Assume that A_x and A_y communicate with each other via the unbounded delay link model for some time, and that any protocol liveness conditions are met. We assume that during this communication no frames are lost. A_x and A_y will exchange some

set of messages which depends upon the delay incurred by the transmitted frames. Let W_x be the set of inputs to A_x (including both message and timeout inputs) which results from the communication between A_x and A_y. String W_x takes A_x from its initial state i_x to some state j_x. Similarly, let W_y be the set of inputs to A_y that results from this communication. W_x and W_y are two sample paths that the automata might traverse in accomplishing some task. Using only prefixes of these paths, we will design a failure scenario which will cause the protocol to fail.

The following definitions are used in specifying the behavior of the automata. Let s be a string, and let T be a set. Then

$s\backslash T$ is the string which results from removing the elements of T from s,

$I_x(s)$ is the largest prefix of W_x such that[3] $(I_x(s)\backslash T_x) = s$, and

$O_x(s)$ is the output string of A_x if it accepts input string s, starting from its initial state.

If s is a message string, $I_x(s)$ is the largest prefix of W_x that can be constructed by appropriately interspersing timeout inputs in s. Since W_x and W_y result from some communication between A_x and A_y we have

$$W_x = I_x\left[O_y\left(W_y\right)\right] \tag{2.1}$$

and

$$W_y = I_y\left[O_x\left(W_x\right)\right] \tag{2.2}$$

The proofs of impossibility which follow will make use of a particular sequence of prefixes of W_x and W_y which are now defined. Since A_x and A_y traverse paths defined as W_x and W_y by exchanging messages, at least one automaton must send a message upon receiving one or more initial timeout inputs, and no message inputs. This

[3]For example, if $T_x = \{t_1, t_2\}$, and $W_x = t_1 m_1 m_2 t_2 m_3$, then $I_x(\epsilon) = t_1$, $I_x(m_1) = t_1 m_1$, $I_x(m_1 m_2) = t_1 m_1 m_2 t_2$, and $I_x(m_1 m_2 m_3) = W_x$.

implies that $O_x(I_x(\epsilon))$ or $O_y(I_y(\epsilon))$ must be nonempty. Without loss of generality, it is assumed that $O_x(I_x(\epsilon))$ is nonempty. A sequence of prefixes X_n of W_x and a sequence of prefixes Y_n of W_y are defined as follows.

$$X_0 \equiv I_x(\epsilon), \tag{2.3}$$

$$Y_n \equiv I_y[O_x(X_n)] \qquad \text{for } n \geq 0, \tag{2.4}$$

$$X_n \equiv I_x[O_y(Y_{n-1})] \qquad \text{for } n \geq 1. \tag{2.5}$$

String X_0 is the largest prefix of W_x which A_x accepts before receiving any messages from Y. String Y_n is the largest prefix of W_y which A_y accepts after receiving only those messages produced by A_x when A_x accepted X_n. Similarly, X_n for $n \geq 1$ is the largest prefix of W_x which A_x accepts after receiving only those messages produced by A_y when A_y accepted Y_{n-1}.

Lemma 4 There exists an n^* such that for all $n > n^*$, $X_n = W_x$ and $Y_n = W_y$.

Proof. Assume that $X_n \neq W_x$ for some n. Let m be the member of W_x which follows the last member of X_n. By the definition of X_n, m must be a message input. Automaton A_x accepts W_x by definition, so A_x must receive m from A_y. When A_y has received all of the messages in $O_x(X_n)$, it must send m in order to eventually accept W_y. This implies that $O_y(Y_n)$ must contain m, and by equation 2.5, so must X_{n+1}. Therefore, X_{n+1} is larger than X_n, and there must be some value of n, say n_x^* such that $X_{n_x^*} = W_x$. By equations 2.4 and 2.5, if $X_n = W_x$ for some $n = n_x^*$, this will be true for all $n > n_x^*$ as well. A similar argument can be used to show the existence of an n_y^*, such that $Y_n = W_y$ for all $n \geq n_y^*$. To complete the proof we let $n^* = \max(n_x^*, n_y^*)$. □

This lemma will be used to construct a set of general failure scenarios which will prove the desired impossibility results.

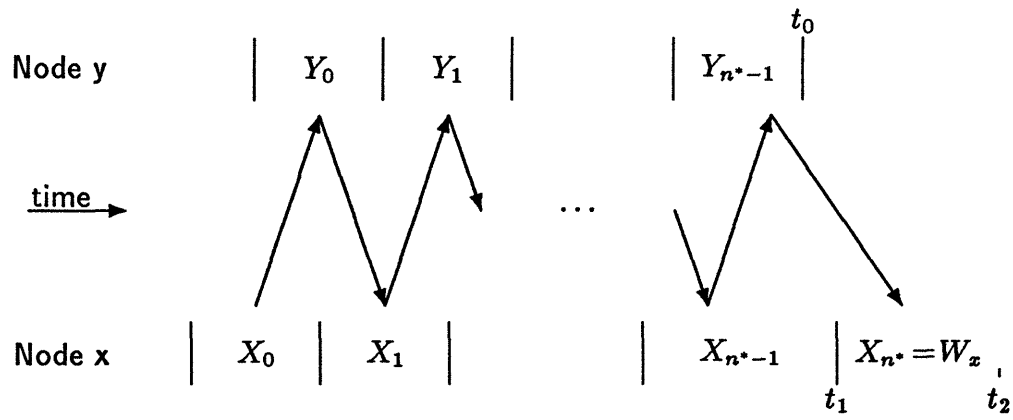

Figure 2.3: Failure Scenario for Weak Synchronization

2.4.2 Impossibility Results

Theorem 5 Using the unbounded delay link model with node failures, there does not exist a protocol which achieves weak synchronization.

Proof. Let A_x and A_y define a candidate terminable protocol for x to initiate weak synchronization. We define a scenario of frame timing and node failures which will cause the protocol to fail. Let W_x be a path in A_x taking it from an initial state to a final state. When it enters its final state, A_x asserts that weak synchronization has been completed. Let W_y be any path in A_y such that equations 2.1 and 2.2 are satisfied. If the protocol works, then such a W_x and W_y must exist. Prefix sequences X_n and Y_n are defined as before.

Consider the following node failure scenario. Automaton A_x accepts X_0 and then node x fails. Automaton A_y receives the messages sent by A_x; A_y accepts Y_0, and then node y fails. Each node begins operating immediately after failing. This process continues until for some n^* we have $X_{n^*} = W_x$. By lemma 4, such an n^* must exist. This is illustrated in Figure 2.3. The vertical lines indicate node failures. The arrows indicate the string of output messages produced when the indicated string is

accepted as input. Let t_1 be the starting time of the indicated NUP, and time t_2 be the time at which A_x first reaches its final state during that NUP. The node failure at x indicated by t_1 is delayed so that it occurs after the node failure at y indicated by t_0. At time t_2, A_x decides that a weak synchronization has been achieved, but none of the messages received by x in $[t_1, t_2]$ were transmitted by y in $[t_1, t_2]$. Thus the protocol fails. □

Theorem 6 Using the unbounded delay link model with node failures, there does not exist a protocol which achieves strong synchronization.

Proof. The same method is used as in the proof of Theorem 5. Automata A_x and A_y define a terminable protocol for achieving synchronization. Let W_x define a path in A_x from its initial state to a final state where strong synchronization is achieved, and the link is declared up. Similarly, W_y takes A_y from its initial to final states. Strings W_x and W_y are such that equations 2.1 and 2.2 are satisfied. If the protocol achieves strong synchronization, such a pair must exist. Prefix strings X_n and Y_n are defined as in the proof of Theorem 5, and a similar failure scenario is used. Let n^* be an integer such that $X_{n^*} = W_x$, and $Y_{n^*} = W_y$. By Lemma 4, such an n^* must exist.

Node failures occur as before, except that after A_y accepts Y_{n^*}, node y does not fail, and thus declares the link to be up at time t_0. The scenario is illustrated in Figure 2.4. The failure of node x after accepting X_{n^*} is delayed until time $t_1 > t_0$. Node x calls the link up at time t_2 after accepting X_{n^*}. The interval $[t_1, t_2]$ defines an LDP at x, but the link is called up at y during this interval. Such a situation violates the clear property of strong synchronization. □

Theorem 7 Using the unbounded delay link model with node failures, there does not exist a protocol which achieves reliable communication, if two or more data packets may be released in a LUP.

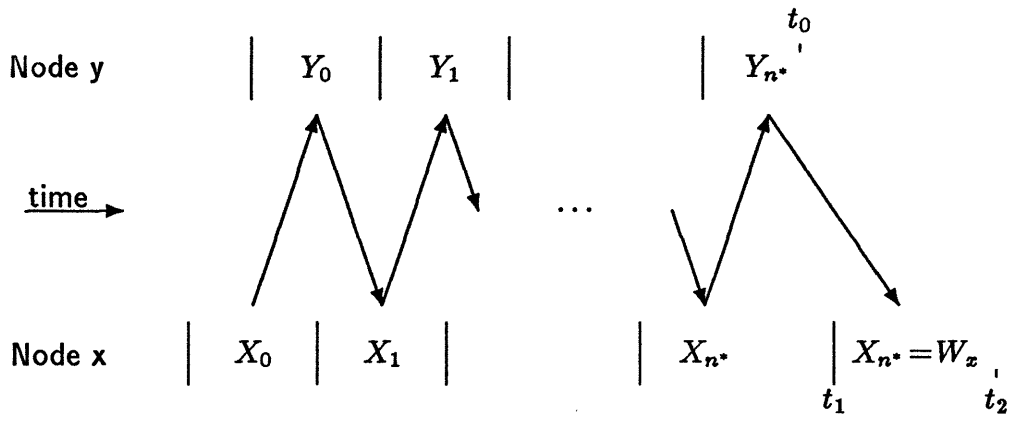

Figure 2.4: Failure Scenario for Strong Synchronization

Proof. The same method as in the proofs of Theorems 5 and 6 is used, except that since A_x and A_y may not be terminable, W_x and W_y are defined differently. Automata A_x and A_y define a protocol intended to provide reliable communication from x to y. Let W_x define a path in A_x, starting from its initial state, which results in the transmission of the first $n \geq 1$ data packets of a NUP. Let W_y define a path in A_y, starting from its initial state, which results in the release of the first $n \geq 1$ data packets of the NUP to a data sink. Similarly, let $\widetilde{W}_x$ and $\widetilde{W}_y$ be additional input strings for A_x and A_y such that $W_x\widetilde{W}_x$ and $W_y\widetilde{W}_y$ result in the transmission and release of the first $m > n$ messages of a NUP.

Consider a node failure scenario shown in Figure 2.5. The index n^* is large enough such that $X_{n^*} = W_x$ and $Y_{n^*} = W_y$. The NUPs at x and y have been numbered for convenience. Let the string of data packets accepted from the data sink by x during NUP i be $P_x^i = p_1p_2p_3\ldots$, and for NUP j, let the string be $P_x^j = p'_1p'_2p'_3\ldots$. Node y accepts W_y and releases data packets $p_1p_2\ldots p_n$ at the start of NUP k. Similarly, node x accepts W_x at the beginning of NUP j, but all of the frames it sends, $O_x(W_x)$, are lost. At this point, A_x and A_y begin error free transmission of data packets $p'_{n+1}p'_{n+2}\ldots p_m$ by accepting $\widetilde{W}_x$ and $\widetilde{W}_y$, respectively. The result is that

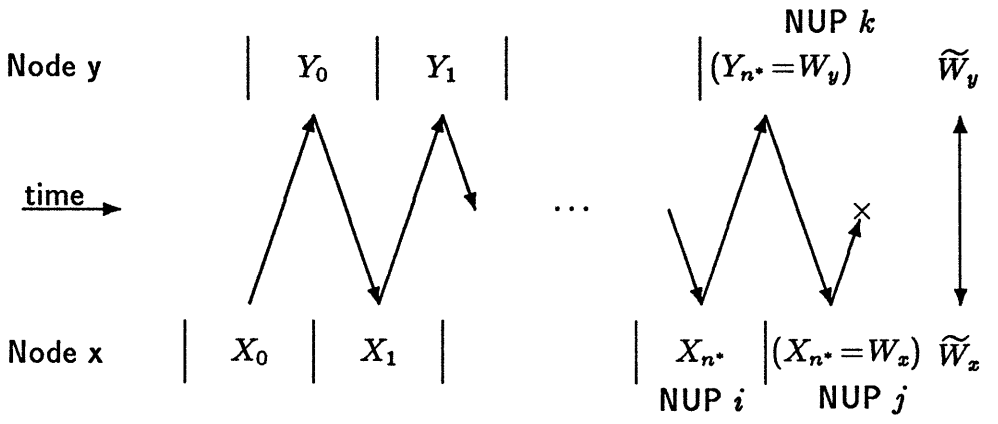

Figure 2.5: Failure Scenario for Reliable Communication

during NUP k, node y releases the string $P_y^k = p_1 p_2 \ldots p_n p'_{n+1} \ldots p'_m$.

Since p_n and p'_{n+1} were transmitted during different NUPs at x, they must also have been transmitted during different LUPs. Assume that NUP k contains a LUP which releases two or more data packets to the data sink. If we choose n such that p_n and p'_{n+1} are released during this LUP, then node y violates condition one of reliable communication. □

2.5 Bounded Delay Links

Given the impossibility results of the previous section, it is interesting to examine whether synchronization and reliable communication are possible when an upper bound is known on the frame transmission delay. The following theorem shows the not very surprising result that these tasks are indeed possible for the bounded delay case. The method used to avoid the scenarios of the previous section is to have an automata ignore received messages, and not send any messages, for some time interval after it is started. This has the effect of sweeping the link clear of old messages, and is effective because an upper bound on the frame transmission delay

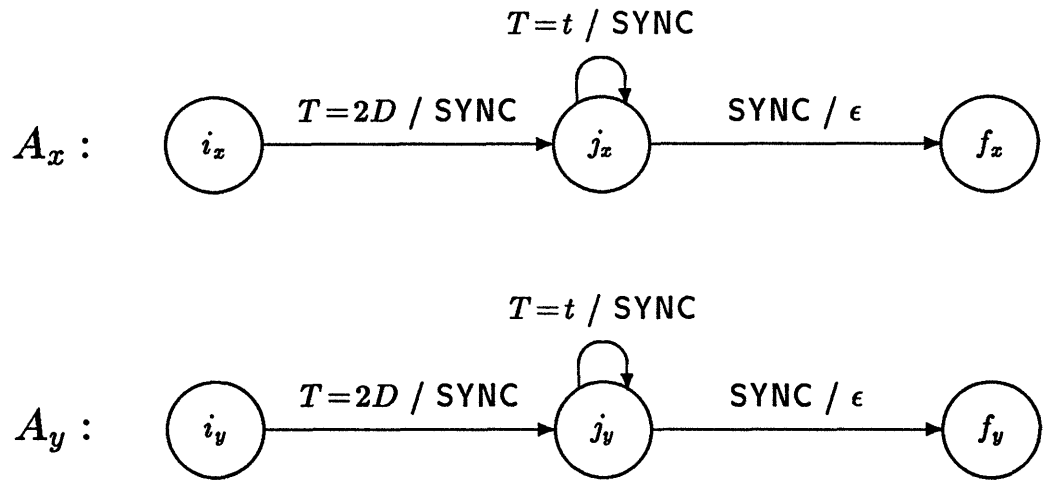

Figure 2.6: Protocol for Weak Synchronization

is known.

Theorem 8 Using the bounded delay model with node failures, there exist protocols for achieving weak synchronization, strong synchronization, and reliable communication.

Proof. By the results of Section 2.3, it suffices to show the theorem for weak synchronization, and by symmetry, it suffices to show weak synchronization at node x. Let the maximum frame transmission delay be D time units. Consider the protocol defined in Figure 2.6. On each arc the input message or timeout is indicated, followed by the output which results from the transition, if any. Node x starts A_x in state i_x whenever it wishes to achieve weak synchronization at x. We assume that if node y does not receive any frames other than SYNC for some time interval, then any automaton which is running at y must start A_y in state i_y. Let node x start A_x at time t_1. While it is in its initial state i_x, A_x ignores all received frames. If both nodes and the link are operational for a sufficient time following t_1, then A_x must eventually enter state f_x. Call this time t_2. Automaton A_x must

enter state j_x at time $t_1 + 2D$. Any SYNC frames which A_x receives while in state j_x must have been sent by y after time $t_1 + D$. This implies that a SYNC received while A_x is in state j_x was sent in $[t_1, t_2]$, and proves weak synchronization at x. A similar argument shows weak synchronization at y. $\square$

Bibliography

[1] A. V. Aho, D. Wyner, and M. Yannakakis, "Bounds on the Size and Transmission Rate of Communications Protocols," *Comp. & Maths. with Appls.*, Vol. 8, No. 3, pp. 205–214, 1982.

[2] B. Awerbuch, "Fail-Safe Compilation of Protocols on Dynamic Communication Networks," MIT Laboratory for Computer Science, 1987.

[3] A. E. Baratz and A. Segall, "Reliable Link Initialization Procedures," *IEEE Trans. Commun.*, Vol. 36, No. 2, pp. 144–152, February, 1988.

[4] D. Bertsekas and R. Gallager, *Data Networks*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

[5] S. G. Finn, "Resynch. Procedures and a Fail-Safe Network Protocol," *IEEE Trans. Commun.*, Vol. 27, No. 6, pp. 840–845, June, 1979.

[6] E. Gafni, "Topology Resynchronization: A New Paradigm for Fault Tolerance in Distributed Algorithms (Extended Abstract)," UCLA Technical Report, 1987.

[7] E. Gafni and Y. Afek, "End-to-End Communication in Unreliable Networks (Extended Abstract)," UCLA, Dept. of Computer Science, 1988.

[8] P. Humblet and S. Soloway, "A Fail-Safe Layer for Distributed Network Algorithms and Changing Topologies," MIT Laboratory for Information and Decision Systems Report No. LIDS–P–1702, May, 1987.

[9] P. Humblet and S. Soloway, "Distributed Network Protocols for Changing Topologies: A Counterexample," MIT Laboratory for Information and Decision Systems Report No. LIDS–P–1700, May, 1987.

[10] G. Le Lann and H. Le Goff, "Verification and Evaluation of Communication Protocols," Computer Networks, Vol. 2, pp. 50-69, 1978.

[11] N. Lynch, Y. Mansour, and A. Fekete, "The Data Link Layer: Two Impossibility Results," MIT, Laboratory for Computer Science, (in preperation).

[12] R. Pearlman "Network Layer Protocols with Byzantine Robustness," MIT Dept. of Electrical Engineering and Computer Science, Ph. D. Thesis Proposal, October, 1987.

[13] A. Segall, "Distributed Network Protocols," *IEEE Transactions on Information Theory*, Vol. 29, No. 1, January 1983.

[14] J. Spinelli, "Broadcasting Topology Information in Computer Networks," MIT Center for Intelligent Control Systems Report No. CICS-P-9, July, 1987.

[15] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Reading, MA:Addison-Wesley Publishing Co., 1979.