Self-Stabilization of the Alternating-Bit Protocol

Yehuda Afek* Geoffrey M. Brown†

Abstract

The Alternating-Bit protocol is a fundamental protocol for transmitting data across an unreliable transmission medium; however, the reliability of the protocol depends upon its initial state. We present a ***self-stabilizing*** version of the Alternating-Bit protocol, i.e., the system will converge to a state that guarantees reliable data transmission regardless of its initial state.

1 Introduction

The Alternating-Bit protocol is a fundamental protocol for reliably transferring data between a transmitter and a receiver across an unreliable transmission medium [3,11]. The reliable transmission of data by the Alternating Bit protocol requires that the transmitter, the receiver, and the communication medium remain tightly synchronized. In the event that synchronization is lost, it might never be recovered. We present a variant of the Alternating-Bit protocol that is self-stabilizing [4,5], i.e. the system is guaranteed to eventually resynchronize regardless of its initial state.

The Alternating Bit protocol is a special case of the sliding-window protocol [12,2]. Gouda and Multari have developed a self-stabilizing sliding window protocol (and hence an alternating-bit protocol) [6]. Their protocol requires the use of unbounded sequence-numbers which would result in an unbounded message size. Yet, they have proved that it is impossible to develop a self-stabilizing version of the sliding window protocol that uses bounded sequence-numbers and deterministic finite state transmitter and receiver. Our protocol is self-stabilizing, deterministic, and uses bounded sequence-numbers; however, it requires the transmitter to generate an infinite aperiodic sequence of numbers.

*AT&T Bell Laboratories, & Computer Science Department, Tel-Aviv University.

†School of Electrical Engineering, Cornell University

The design of data-link protocols under weaker failure model, but stronger safety conditions were considered by Baratz, and Segall, and by Lynch, Mansour and Fekete. In [2] Baratz and Segall conjectured and in [10] Lynch, Mansour and Fekete proved that it is impossible to design a reliable data-link protocol that tolerates host crashes without losing even one message. During the recovery period of our protocols, a small number of messages might be lost. However the system is guaranteed to eventually converge to a reliable operation.

This paper is organized as follows. The system model and problem statement are presented in Section 2. In Section 3 we present our protocol and, in Section 4, demonstrate that it is self-stabilizing. We discuss practical applications of our protocol and possible extensions In Section 5.

2 Model

The purpose of a data link protocol is to reliably transmit messages from one end of an error prone communication media to the other end [3,1,2]. By reliably we mean that messages arrive error free, without duplication or loss, and in the order sent.

The system consists of two processors, the *transmitter* T and the *receiver* R, connected by two directed links oriented in opposing directions. Messages transmitted over the links may be lost[1]; however, those that are not lost arrive error free and in the order sent (FIFO). Although timeouts may be used by the protocol, no bound on the transmission delay is assumed.

The transmitter and the receiver are deterministic finite state machines with access to a *choose* oracle. The transmitter T is connected to a read-only input tape with an infinite sequence $I = (D_0, D_1, \ldots)$ of data elements. T reads these data elements one at a time and tries to transmit them to the receiver R over the transmission media. R must write

[1]Message corruption is treated as message loss by assuming the use of error detecting codes.

these data elements onto a write-only output tape O.

The system is subject to transient errors at arbitrary times (e.g. host crashes). After a transient error the transmitter, receiver, oracle, and links are in arbitrary states. To model long periods of time during which all components operate without errors, we assume that eventually, after an arbitrary number of errors the transmitter, receiver and links enter a last operational interval (in which there are no errors) that is infinitely long. The last operational interval consists of two periods, a convergence period, and an infinitely long stable period, called the *final interval.*

The goal is to design a protocol such that, (a) *stabilization:* the convergence period is finite, (b) *safety:* the sequence of data elements written in O during the final interval is always a prefix of the sequence of data elements read from I during the final interval, and (c) *liveness:* In the final interval it is always true that within a finite time one more data element will be written to the output tape.

3 A Generic Data-Link Protocol

Our protocol alternates between at least three sequence-numbers, instead of two sequence-numbers as in the standard Alternating Bit protocol [3]. Furthermore, the series of sequence-numbers selected by the protocol is aperiodic.

The generic data link protocol, called Protocol G, is given in Figure 1. The function tape-read returns the value of the input tape at the current head location. Σ is the set of sequence-numbers used by the protocol and $choose(\Sigma/B)$ is a function that selects one element from the set $\Sigma - B$. The protocol works as follows: The transmitter and the receiver, each hold a sequence number. Initially the sequence numbers held by the transmitter and the receiver are different. To transmit the current message from the input tape to the output tape the transmitter tags the message with its current sequence number and sends it periodically to the receiver until it receives an acknowledgment from the receiver with its current sequence number. The receiver writes to its output tape any message with a sequence number different than its own. The transmitter may, at any time, retransmit its current message (e.g. due to a time-out).

The behavior of Protocol G depends on the specification of the *choose*(X) function. In Section 4 we show that if the sequence of numbers returned by *choose* during the execution of the protocol is aperiodic (e.g. the sequence of digits representing an irrational number), then the protocol is self-stabilizing. Note that if $\Sigma = \{0; 1\}$ then Protocol G is the standard Alternating Bit protocol.

The safety and liveness properties of Protocol G with $\Sigma = \{0; 1\}$ and no failures are essentially proved in [9,1,8,7].

```
                         Transmitter

Initially:  $B$ = '0' ;
            Tape head at beginning of tape;

TRANSITIONS:
1: Time-out: $C$ := tape-read;
            Send $(C, B)$;

2: Receive ACK$(D)$:
            if $D = B$
            then $B$ := choose$(\Sigma/B)$;
                   Move tape head one location;
            $C$ := tape-read;
            Send $(C, B)$;

                         Receiver

Initially:   $A$ = '1' ;

TRANSITIONS:
3: Receive$(M, E)$:
             if $A \neq E$
             then $A$ := $E$;
                    Write $M$ to
                    output tape;
             Send ACK$(A)$;
```

Figure 1: Data Link Protocol G

In the following two sections we will sketch the proof that Protocol G is a self-stabilizing data link protocol if $|\Sigma| > 2$.

4 Properties of Protocol G

Since protocol G does not use any properties of the data elements (in contrast to [1]) the state of the processors and the channels will be represented by the sequence-numbers on them (leaving out the data elements). The state of the transmitter is represented by the value of its current sequence number, B, and the state of the receiver by the value of its current sequence number, A. Thus the number of states of each is $= |\Sigma|$.

The state of the $T - R$ link is denoted by $\beta_1\beta_2\ldots\beta_n$, $\beta_i \in \Sigma; \epsilon$, and the state of the $R - T$ link: $\alpha_1\alpha_2\ldots\alpha_m$, $\alpha_i \in \Sigma; \epsilon$. The system state s is the concatenation of the

states of the processors and the links:
$s = B|\beta_1\beta_2\ldots\beta_n|A|\alpha_1\alpha_2\ldots\alpha_m$. We condense the system state s into system state $CS(s)$ by first treating it as one long string of sequence-numbers: $B\beta_1\beta_2\ldots\beta_n A\alpha_1\alpha_2\ldots\alpha_m$ and second by replacing each maximal contiguous segment of equal sequence-numbers by one instance of that sequence-number (e.g., if $s = 2|2\ 2\ 2\ 0\ 0\ 0\ 1|1|1\ 1\ 1$ then $CS(s) = 2\ 0\ 1$).

The transition function $\delta : S \rightarrow S$, where S is the state space, has five types of transitions ($\delta_1, \delta_2, \delta_3, \delta_4$, and δ_5) representing the following five possible actions: (1) retransmitting a message by the transmitter, (2) receiving an acknowledgment by the transmitter followed by sending a message by the transmitter, (3) receiving a message by the receiver followed by sending an acknowledgment by the receiver, (4) losing a message on the $T - R$ channel, and (5) losing an acknowledgment on the $R - T$ channel. More precisely:

$\delta_1 : B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m \rightarrow$
$\quad B|B\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m$

$\delta_2 : B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m \rightarrow$
$\quad B'|B'\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_{m-1}$ where $B = B'$ iff $\alpha_m \neq B$

$\delta_3 : B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m \rightarrow$
$\quad B|\beta_1\ldots\beta_{n-1}|\beta_n|\beta_n\alpha_1\ldots\alpha_m$

$\delta_4 : B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m \rightarrow$
$\quad B|\beta_1\ldots\beta_{i-1}\beta_{i+1}\ldots\beta_n|A|\alpha_1\ldots\alpha_m$

$\delta_5 : B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_m \rightarrow$
$\quad B|\beta_1\ldots\beta_n|A|\alpha_1\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_m$

With each state s of the system we associate a length function $l(s) = |CS(s)|$, i.e., the number of different blocks of identical sequence-numbers in the system. The *Rank* function is defined similarly:

$$Rank(s) = \begin{cases} l(s)+1 & \text{if } first(s) = last(s) \\ l(s) & \text{otherwise} \end{cases}$$

Note that under normal conditions, without failures or erroneous transitions, the system should be all the times at a state s, such that $Rank(s) = 2$. To prove that the protocol self-stabilizes we will show that the rank function is decreasing with time until $Rank(s) = 2$.

4.1 Self-Stabilization with Infinite Aperiodic Sequences

In this section we show that a deterministic version of the protocol will stabilize to correct operation if repeated calls to *choose* return an aperiodic sequence of sequence-numbers, *ap*. The sequence *ap* is aperiodic iff:

$$\neg(\exists N,\ \beta\ : N > 0\ \wedge\ \beta \in \Sigma^+ : ap = a_1a_2\cdots a_N\beta^\infty)$$

Theorem 1 *If Rank(s) > 2 and* choose *returns an aperiodic sequence of sequence-numbers, then Rank(s) is eventually reduced.*

Proof: Consider the mapping between compact states, $CS(s)$ to $CS(\delta_i(s))$, which is defined by applying transitions δ_i, $i = 1,\ 2,\ 3,\ 4,\ 5$ on state s.

δ_1: CS is left unaltered.

δ_2: If the first and last elements of CS are different, then δ_2 leaves CS unaltered, otherwise δ_2 adds a new element to CS and, possibly, deletes the original first or last element.

$\delta_{3,4,5}$: Either leave CS unaltered, or remove an element other than the first.

It is straightforward to show that if *Rank*$(s) > 2$, then CS must eventually be altered. The effect of each transition that modifies CS can be modelled by one or more of the following actions.

a_1: Delete the last element of CS.

a_2: Delete an element from the middle of CS.

a_3: If the first and last elements of CS are the same, add a new first element.

None of these three actions can increase *Rank* and a_2 is guaranteed to reduce *Rank*. Thus, we consider only executions of actions a_1 and a_3.

Suppose that the system has reached state s such that:

$$CS(s) = m_0m_1\cdots m_nm_0$$

If the next sequence-number m_j added via a_3 is different than m_n, then the system will reach state s' where

$$CS(s') = m_jm_0m_1\cdots m_{n-1} \qquad m_j \neq m_n$$

via the following sequence of actions

$$m_0m_1\cdots m_nm_0 \xrightarrow{a_3} m_jm_0m_1\cdots m_n \xrightarrow{a_1} m_jm_0m_1\cdots m_{n-1}$$

hence reducing Rank ($Rank(s) = n+3 \wedge Rank(s') \leq n+2$). Thus, the only way in which Rank will not eventually be reduced is if each new sequence number that is added to the head of $CS(s)$ is equal to the next to last sequence number in $CS(s)$ (i.e. $m_j = m_n$). But if each new sequence number that is added equals the next to last sequence number then the sequence of sequence-numbers generated by *Choose* has the form

$$(m_n m_{n-1} \cdots m_0)^\infty$$

which violates our assumption that ap is aperiodic. ∎

5 Discussion

Our protocol requires that the transmitter generate an infinite aperiodic sequence of numbers. Such a sequence is easy to generate, for example, the sequence *cacabcabacababc*... has the required property. However, generating this sequence requires an infinite state sender. In practice, an infinite sequence will not be required. Whenever a bound γ is given on the number of messages that can simultaneously be in transit, the protocol can be used with a sequence of numbers ap whose period is larger than γ i.e.:

$$\neg(\exists N,\ \beta \in \Sigma^{\leq\gamma} : N > 0 : ap = a_1 a_2 \cdots a_N \beta^\infty)$$

Although the sequence given above satisfies the aperiodicity requirement for stabilization, it may not have the best convergence properties. An alternative implementation of our protocol uses a random sequence. Under the assumption that the system starts in an initial state s where $Rank(s) = i$, it is possible to show that the system will have the following expected stabilization time (normalized to the maximum roundtrip delay in the system).

$$\frac{(i-2)(|\Sigma|-1)}{|\Sigma|-2}$$

We believe that the randomized version of Protocol G is a good practical solution to the link initialization problem in many applications. For example: in voice, video, or real time applications the crash and recovery would result in the loss of one data point, which in most cases is insignificant in the overall behavior of the system. Clearly, Protocol G can be extended to a sliding window protocol with similar properties by running W copies of Protocol G with W distinct sets of sequence-numbers.

References

[1] A. V. Aho, J. D. Ullman, A. D. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Comp. & Maths. with Apls.*, 8:3:205–214, 1982.

[2] A. E. Baratz and A. Segall. Reliable link initialization procedures. In H. Rudin and C.H. West, editors, *IFIP 3rd Workshop on Protocol Specification, Testing and Verification, III*, May 1983.

[3] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Comm. of the ACM*, 12:260–261, 1969.

[4] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *CACM*, 17:643–644, November 1974.

[5] E. W. Dijkstra. Ewd 391 self-stabilization in spite of distributed control. *Lecture Notes in Computer Science 92*, 41–46, 1982.

[6] M. Gouda and N. Multari. Self-stabilization communication protocols. 1988. In preparation.

[7] M. G. Gouda. On "a simple protocol whose proof isn't": the state machine approach. *IEEE Trans. on Communication*, COM-33(4):380–383, April 1985.

[8] B. Hailpern. A simple protocol whose proof isn't. *IEEE Trans. on Communication*, COM-33(4):330–337, April 1985.

[9] J. Y. Halpern and L. D. Zuck. A little knowledge goes a long way: simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proceedings of the ACM Symp. on Principles of Distributed Computing*, August 1987.

[10] N. Lynch, Y. Mansour, and A. Fekete. The data link layer: two impossibility results. In *Proc. of the ACM Symp. on Principles of Distributed Computing*, August 1988.

[11] M. V. Stenning. A data transfer protocol. *Comput. Networks*, 1:99–110, 1976.

[12] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, 1981.