Self-Stabilization Over Unreliable Communication Media

Yehuda Afek* Geoffrey M. Brown†

Abstract

A self-stabilizing system has the property that it will converge to a desirable state when started from any state. Most previous researchers assumed that processes in self-stabilizing systems may communicate through shared variables while those that studied message passing systems allowed messages with unbounded size. This paper discusses the development of self-stabilizing systems which communicate through message passing, and in which messages may be lost in transit. The systems presented all use fixed size message headers.

First, a self-stabilizing version of the *Alternating Bit Protocol*, a fundamental communication protocol for transmitting data across an unreliable communication medium, is presented.

Secondly, the alternating-bit protocol is used to construct a self-stabilizing token ring.

1 Introduction

Since the development of the first self-stabilizing systems by Dijkstra in the early 1970's [Dij73, Dij82] most researchers have considered systems in which the processes communicate through shared variables [Kru79, Lam84, BGW87, BGW89, BP89]. In this paper we present three self-stabilizing systems, two data-link protocols and a token ring, which communicate over unreliable message channels.

The Alternating Bit protocol is a fundamental data-link protocol for reliably transferring data between a transmitter and a receiver across an unreliable transmission medium [BSW69,

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

†School of Electrical Engineering, Cornell University

This work supported in part by NSF grant CCR-9058180

Ste76]. 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.

The Alternating Bit protocol is a special case of the sliding-window protocol [Tan81, BS83]. Gouda and Multari have developed a self-stabilizing sliding window protocol (and hence an Alternating Bit protocol) [?]. Their protocol requires the use of unbounded sequence numbers which would result in an unbounded message size. Furthermore, 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. Katz and Perry have also developed self-stabilizing systems which communicate through message passing and use unbounded sequence numbers [KP89].

Gouda and Multari essentially proved, although did not directly argue so, that any self-stabilizing sliding window protocol requires some kind of an "infinite resource". We present two self-stabilizing variations of the Alternating Bit protocol. Both use bounded sequence numbers (message headers) and the "unbounded resource" in each is confined to a particular oracle at the transmitter. In the first protocol the oracle generates an aperiodic string of sequence numbers taken from a bounded size alphabet and in the second it generates a string of random sequence numbers over the same alphabet. In both cases each number in the string generated by the oracle is different from its predecessor. Thus, in the first protocol the transmitter is an infinite state machine and in the second it is a probabilistic finite state machine.

An interesting and desirable property of our randomized algorithm is the speed at which it converges to a legal state. Specifically, the expected time for our randomized alternating bit protocol to stabilize as a function of h, the message header size in bits, is $O(2^{-h})$. The expected stabilization time is also dependent on the number of messages in transit in the initial state, in which case the dependence is linear. ¿From a practical point of view we note in Section 6 that the unbounded resource in our protocols can be trivially replaced by a bounded resource whenever a bound on the link capacity is given.

Finally, a self-stabilizing token ring is presented by augmenting the Alternating Bit protocols. This demonstrates the broader applicability of the techniques used to make the Alternating Bit protocols self-stabilizing.

The paper is organized as follows. The system model and problem statement for data-link protocols are presented in Section 2. In Section 3 the Alternating Bit protocols are presented and, in Section 4, a proof that they are self-stabilizing together with their analysis is given. Finally the token ring protocol is described in Section 5. Practical applications of the protocols and possible extensions are discussed in Section 6.

2 Self-stabilizing Protocols

The basic definition of a self-stabilizing system is one which will converge to a legal state when started from any state. The first step in the design of such a system is to identify the "legal" states. At the specification level, the legal states are identified by the properties which must hold when the system is in a legal state. It is then the designer's duty to prove that an implementation stabilizes to a set of system states that satisfy the properties required by the specification. In the remainder of this section a self-stabilizing data link protocol is specified.

The purpose of a data-link protocol is to reliably transmit messages from one end of an error prone communication medium to the other end [BSW69, AUWY82, BS83, LMF88]. By reliably we mean that messages arrive error free, without duplication or loss, and in the order sent.

The system consists of two processes, the *transmitter* **T** and the *receiver* **R**, connected by two directed communication links, **T-R** and **R-T**, 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.

In a data-link protocol the transmitter has an input tape with an infinite sequence $I = (D_0, D_1, \ldots)$ of data elements. The transmitter reads these data elements and tries to transmit them to the receiver. The receiver must write these data elements onto an output tape O. The goal is to design a protocol such that, *safety:* O is always a prefix of I and *liveness:* under reasonable *fairness* conditions, every data element D_i is eventually written to O.

This paper considers a system consisting of a transmitter, a receiver, and communication links that are subject to transient errors at arbitrary times (e.g. host crashes). The transmitter and receiver are deterministic finite state machines; however, the transmitter has access to a *choose* oracle. After a transient error the transmitter, receiver, oracle, and links are in arbitrary states. Note that losing a message on a link is not considered an error, it is a possible behavior of the system that the protocol should tolerate in normal operation. 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 more faults aside from message loss) that is infinitely long [AAG87]. The last operational interval consists of two periods, a convergence period, and an infinitely long stable period, called the *final interval*. In the convergence period the protocol automatically moves the system from the arbitrary global state at which it began to a legal global state which guarantees its correct operation thereafter, i.e. it is

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

guaranteed to remain in a legal state in the future unless additional faults occur.

The goal is to design a data-link protocol that satisfies the *safety* and *liveness* conditions above and which is self-stabilizing, i.e. that following a sequence of faults will converge into a legal state in finite time. The *safety* and *liveness* conditions above were specified for a system that does not fail. When the system is subject to transient errors at arbitrary times it is required to behave correctly only during the final interval, i.e. (a) *safety:* the sequence of data elements written to O during the final interval is always a prefix of the sequence of data elements read from I during the final interval, and (b) *liveness:* in the final interval it is always true that within a finite time one more data element will be written to O.

3 A Generic Data-Link Protocol

The two Alternating Bit protocols are variants of a single "generic" data-link protocol which is much like the original protocol of [BSW69]. In the original protocol message sequence numbers are either 0 or 1, that is the size of the sequence numbers alphabet, Σ, is 2. In the two variants the size of the sequence number alphabet is at least 3, e.g. $\{0, 1, 2\}$. Furthermore, the series of sequence numbers selected by these protocols is aperiodic in one and random in the other variant.

The generic data-link protocol, called Protocol **G**, consists of two processes, a transmitter **T** and a receiver **R**. The code for each process consists of a set of *actions* (see Figure 1). Each action may be *enabled* or *disabled* depending on the state of the system. An enabled action may be executed at any time with the restriction that the actions of the system (both **T** and **R**) are executed one at a time, i.e. atomically.

Each action has the form *guard* $\rightarrow$ *command*. The guard may be either a boolean expression or a receive statement of the form `rcv(_)`. The guard `timeout` refers to a boolean variable set by a timeout mechanism. This mechanism is assumed to consist of a free running clock which periodically sets `timeout` to true. The timeout mechanism is assumed to be self-stabilizing. The commands are constructed from primitive commands using sequencing and alternative constructs. The primitive commands consist of: assignment statements, send commands of the form `send(_)`, input commands of the form `read(_)`, output commands of the form `write(_)`, and the function `choose(`Σ`/x)` which selects an element other than `x` from Σ – the set of sequence numbers used by the protocol. In addition the command `reset_timeout` reinitializes the timeout clock and resets the `timeout` boolean variable.

An action of process **T**(**R**) is enabled if its guard is a boolean expression which evaluates to *true* or its guard is a receive statement and there is a message in the **R-T** (**T-R**) link.

Execution of an action of **T**(**R**) consists of receiving a message from the **R-T** (**T-R**) link if the guard is a receive statement, and then executing the command. Execution of a

```
Transmitter T::

seq int init 0;
i   int;
msg data; read_next(msg); /*initialize */

timeout → send(seq,msg);
          reset_timeout
rcv(i)  → if (seq = i)
          then seq := choose(Σ/seq);
               read_next(msg);
               send(seq,msg)
          fi ;
          reset_timeout
```

```
Receiver R::

ack  int init 1 ;
j    int ;
rmsg data ;

rcv(j,rmsg) → if ack ≠ j
              then ack := j;
                   write_next(rmsg)
              fi ;
              send(ack)
```

Figure 1: The Generic Alternating Bit Protocol

`send` statement by **T**(**R**) causes a message to be added to the **T-R** (**R-T**) link. Execution of a `read` command reads the next input from I. Execution of a `write` command writes to output O.

The protocol works as follows. The transmitter and the receiver each hold a sequence number – initially different. To transmit the current message from the input to the output, the transmitter tags the message with its current sequence number, `seq`, and sends it periodically (using timeouts) to the receiver until it receives an acknowledgment from the receiver with its current sequence number. The receiver writes to its output any message which arrives with a sequence number different from its own (`ack`). The transmitter may, at any time, retransmit its current message (e.g. due to a timeout).

The variable `seq` is used to hold the current sequence number (initially 0), `msg` holds the current data message, and `i` is used to receive acknowledgment messages. The initial values of `seq`, `ack` and `msg` are not necessary since the protocol is self-stabilizing; however, we include them so that the system will start in a legal state under normal use.

The behavior of Protocol **G** depends on the specification of the `choose` function. In the deterministic Alternating Bit protocol, `choose` produces an aperiodic sequence (e.g. the sequence of digits representing an irrational number). The probabilistic protocol is a special case in which the sequence produced by `choose` is random. Note that if $\Sigma = \{0, 1\}$ then Protocol **G** is the standard Alternating Bit protocol [BSW69].

The safety and liveness properties of Protocol **G** with $\Sigma = \{0, 1\}$, and no failures are proved in [AUWY82, Hai85, Gou85]. In the following section we prove that Protocol **G** is a self-stabilizing data-link protocol if $|\Sigma| > 2$.

4 Properties of Protocol G

In this section we demonstrate that protocol **G** is self-stabilizing provided that the sequence of numbers selected by `choose` is aperiodic. (In Section 6, we demonstrate that the choose oracle can be implemented as a deterministic finite state process if the communication links have finite capacity.) The expected cost of stabilization for the probabilistic version is analyzed.

We begin with some definitions and notation necessary for the analysis of the protocol. Since our interest is not with individual states but with classes of states (in particular the legal states), we develop a classification scheme for states. In subsection 4.1 the self-stabilization proof is given and in subsection 4.2 the performance of the probabilistic protocol is evaluated.

Since protocol **G** is data-oblivious, the state of the processes and the links will be represented by the sequence numbers on them leaving out the data elements.[2] The state of the transmitter is represented by the value of its current sequence number, `seq`, and the state of the receiver by the value of its current acknowledgment sequence number, `ack`. Thus the number of states of each is $|\Sigma|$.

The state of the **T-R** link is denoted by the (possibly empty) sequence $\alpha_1\alpha_2\ldots\alpha_n$, $\alpha_i \in \Sigma$, and the state of the **R-T** link is denoted by the (possibly empty) sequence $\beta_1\beta_2\ldots\beta_m$, $\beta_i \in \Sigma$, for some $m,\ n\ \geq 0$. The system state s is the concatenation of the states of the processors and the links:

$$s \;=\; \texttt{seq}|\alpha_1\alpha_2\ldots\alpha_n|\texttt{ack}|\beta_1\beta_2\ldots\beta_m$$

Given this characterization of the system states, we represent all allowable actions of the system by a transition relation $\delta : S \times S$, where S is the state space. This transition relation has six types of transitions ($\delta_1, \delta_2, \delta_3, \delta_4, \delta_5$, and δ_6). If $s_1\delta_i s_2$, for some state s_2 then we say that δ_i is applicable to state s_1. These transitions represent the following six possible events:

δ_1: The transmitter retransmits a message.

δ_2: The transmitter receives an acknowledgment which equals `seq` and it sends a new message with a new sequence number.

δ_3: The transmitter receives an acknowledgment which does not equal `seq`.

δ_4: The receiver receives a message and it sends an acknowledgment.

$\delta_5(i)$: Losing the i-th message on the **T-R** link.

[2] In Section 4.1, when considering the safety requirement of our protocol, we will need to consider the data elements; however, it simplifies the discussion to ignore them for now.

$\delta_6(j)$: Losing the j-th message on the **R-T** link.

More precisely:

$$\delta_1: \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}|\texttt{seq}\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m$$

$$\delta_2: \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}'|\texttt{seq}'\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_{m-1}$$
$$\text{where } \beta_m = \texttt{seq} \text{ and } \texttt{seq}' = \mathit{choose}(\Sigma/\texttt{seq})$$

$$\delta_3: \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_{m-1}$$
$$\text{where } \beta_m \neq \texttt{seq}$$

$$\delta_4: \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}|\alpha_1\ldots\alpha_{n-1}|\alpha_n|\alpha_n\beta_1\ldots\beta_m$$

$$\delta_5(i): \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}|\alpha_1\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m$$

$$\delta_6(j): \quad \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_m \rightarrow \texttt{seq}|\alpha_1\ldots\alpha_n|\texttt{ack}|\beta_1\ldots\beta_{j-1}\beta_{j+1}\ldots\beta_m$$

Where $m, n \geq 0$, $1 \leq i \leq n$, and $1 \leq j \leq m$.

A *schedule* is a finite or infinite sequence of transitions. A schedule σ is *applicable* to a configuration s if every transition in σ is applicable in turn to the resulting intermediate configurations starting with s. The configuration resulting from the application of a finite schedule σ to a configuration s is denoted $s\sigma$.

In data-link protocols it is perfectly legal to have states in which sequences of identical messages are in transit. In order to further characterize the system state, we will ignore the length of such sequences and consider only their pattern. The system state s is compressed into a "compact state" $CS(s)$ by first treating it as one long string of sequence numbers:

$$\texttt{seq}\alpha_1\alpha_2\ldots\alpha_n\texttt{ack}\beta_1\beta_2\ldots\beta_m$$

and second by replacing each maximal contiguous segment of equal sequence numbers by one instance of that sequence number. For example, if

$$s = 2|2\ 2\ 2\ 0\ 0\ 0\ 1|1|1\ 1\ 1$$

then

$$CS(s) = 2\ 0\ 1$$

The compact state of state s and of $s\delta_i$ may be the same in many cases, e.g. $CS(s) = CS(s\delta_1)$. A transition δ_i that does change the compact state of the system is denoted as $\hat{\delta}_i$, e.g. $CS(s) \neq CS(s\hat{\delta}_2)$. Note for example that δ_1 never effects the compact state of the system while δ_2 always will and thus will be denoted $\hat{\delta}_2$.

A length function $l(s) = |CS(s)|$ is associated with each state s, i.e. the number of different blocks of identical sequence numbers in the system. A *Rank* function is similarly defined:

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

The legal states of the system are those states whose rank is 2, in which all the messages with the same sequence number have the same data field, and the messages with sequence number `seq` from the transmitter code have `msg` in their data field. To prove that the protocol self-stabilizes we first show that the rank function is decreasing with time until $Rank(s) = 2$. This proof depends upon the following fairness assumptions.

Fairness assumption on the transmitter and the receiver: If an action of the transmitter or receiver is continuously enabled, then eventually it will be executed.

Fairness assumption on communication links: If a process repeatedly sends the same message over a link, eventually a copy of the message will reach the other end of the link without being lost.

Recall our assumption that all components operate without error in the last operational interval.

The following observation about a liveness property of the protocol can be easily proved from the above assumptions and the assumption that `read_next(msg)` is continuously enabled. Similar properties with a proof-outline are given in Theorem 2.

Observation 1 *In an infinite schedule of the protocol* $\hat{\delta}_2$ *and* δ_4 *appear infinitely often.*

4.1 Self-Stabilization with Infinite Aperiodic Sequences

In this section we show that a deterministic version of the protocol converges to a legal state if a sequence of calls to *choose* returns an aperiodic series of sequence numbers, *ap*. The sequence *ap* is aperiodic iff:

$$\neg(\exists\alpha,\ \beta\ : \alpha,\beta \in \Sigma^+ : ap = \alpha\beta^\infty)$$

Theorem 1 *If* $Rank(s) > 2$ *and* `choose` *returns an aperiodic series of sequence numbers, then* $Rank(s)$ *never increases and is eventually reduced.*

Proof: Assume the contrary, i.e. there exists an infinite schedule σ such that for any prefix σ' of σ, $Rank(s\sigma') \geq Rank(s) > 2$ and the sequence of numbers returned by the calls to *choose* in σ is aperiodic. Since none of the six transitions $\delta_1 \cdots \delta_6$ increases $Rank$, it must be that $Rank(s\sigma') = Rank(s) > 2$.

By Observation 1, $\hat{\delta_2}$ appears infinitely often in σ. Thus, the execution of the protocol under schedule σ has the form: $\mu_0 s_1 \hat{\delta_2}^1 \mu_1 s_2 \hat{\delta_2}^2 \mu_2 s_3 \hat{\delta_2}^3 \mu_3 \ldots \mu_{i-1} s_i \hat{\delta_2}^i \mu_i \ldots$, where $\hat{\delta_2}^i$ is the ith occurrence of $\hat{\delta_2}$ in σ and the μ's are $\hat{\delta_2}$-free alternating sequences of states and transitions. We claim that *for any i, the sequence number returned by choose in $\hat{\delta_2}^i$ equals the next to the last sequence number in $CS(s_i)$*. Proving this claim completes the proof of the theorem because, $Rank(s_i)$ is the same for all i while, by the claim, the system behaves like a cyclic shift register whose next to last number is fed into the front. Thus contradicting the assumption that *choose* returns an aperiodic sequence of numbers.

The claim is also proved by contradiction. Assume that the claim does not hold for $i = k$. Let $CS(s_k) = \alpha_1 \ldots \alpha_{n-2}\alpha_{n-1}\alpha_1$, then α_0, the number returned by *choose* in $\hat{\delta_2}^k$, is different from both α_{n-1} and α_1. Since μ_k is $\hat{\delta_2}$-free and $\hat{\delta_2}$ is the only transition that changes `seq`, α_0 is the first element in $CS(s_{k+1})$. Since $\hat{\delta_2}$ is applicable to s_{k+1} the last number in $CS(s_{k+1})$ equals α_0 and thus is neither α_{n-1} nor α_1, i.e. α_1 and α_{n-1} must have been deleted from the end of $CS(s_k)$ by the transitions in μ_k. Because no transition aside from $\hat{\delta_2}$ adds a number to the system state, $l(s_k) > l(s_{k+1})$ and thus $Rank(s_k) = l(s_k) + 1 > l(s_{k+1}) + 1 \geq Rank(s_{k+1})$, which contradicts our assumption that $Rank(s\sigma') = Rank(s)$. ∎

Theorem 2 *In a state s such that $Rank(s) = 2$, the protocol satisfies its liveness requirement. Furthermore, the system is guaranteed to reach a legal state from which point on the protocol satisfies its safety requirement.*

Proof outline: There are three types of compact states $CS(s)$ with $Rank(s) = 2$. We will show that in a correct operation and under the fairness assumptions above the system must cycle between the first and third type (possibly going through the second one). The three types are: (1) $\alpha\beta$ where `ack` $= \beta$, (2) $\alpha\beta$ where `ack` $= \alpha$, and (3) α, where $\alpha,\ \beta \in \Sigma,\ \alpha \neq \beta$, and `ack` is the receiver `ack` variable. We consider each possibility separately.

1. $CS(s) = \alpha\beta \wedge$ `ack` $= \beta$. By the fairness assumptions, since the sender repeatedly sends a message with sequence number α, a message with sequence number α will eventually be received by the receiver. Thus the system eventually reaches either state s' such that $CS(s') = \alpha\beta \wedge$ `ack` $= \alpha$, or a state s'' such that $CS(s'') = \alpha$. ¿From the code, this also implies that the receiver writes the message that corresponds to α to output O.

2. $CS(s) = \alpha\beta \wedge$ `ack` $= \alpha$. By the fairness assumption on transmitter and receiver, every message sent is either lost or received. Thus, the system must eventually reach a state s' where $CS(s') = \alpha$.

3. $CS(s) = \alpha$. By the fairness assumptions, repeated transmissions of a message will eventually result in a copy of the message being received. Thus, the system must

eventually reach a state s' where $CS(s) = \alpha'\alpha \wedge \texttt{ack} = \alpha$. The required liveness property follows from 1. Furthermore, from this point forward, every data message with sequence number **seq** (from the sender's code) has data **msg**.

¿From the preceding argument, the system must eventually reach a state s in which $CS(s) = \alpha$ and every message has data **msg**. The required safety property, that the data sequence written to O is a subsequence of the data sequence read from I, holds from this point on. ∎

Suppose that $\Sigma = \{a,\ b,\ c\}$, then the sequence $abcababcabababc\ \ldots$ has the required property. Although this string certainly satisfies the requirements, it may not produce the most desirable convergence rates. The quiescence time complexity (the time it takes the system to converge into a legal state) of the deterministic protocol highly depends on the characteristics of the aperiodic sequence. Hence, in the next subsection a randomized *choose* is used to achieve faster convergence rates.

4.2 Probabilistic Self-Stabilization

We are interested in establishing bounds on the length of the convergence period and on the number of erroneous messages received or the number of messages lost during the convergence period. These bounds depend upon the state of the system at the beginning of the convergence period. In particular upon the length of the compact state ($|CS|$).

First the expected number of times the transmitter receives a "correct" acknowledgment and hence transmits a new message during the convergence period is calculated. The number of new messages transmitted during the convergence period is the number of times $\hat{\delta_2}$ is applied to the system. The protocol guarantees that the transmitter will not have to wait to transmit a new message for longer than the round-trip delay between the transmitter and receiver. (A longer wait would imply that all messages with sequence numbers different from `seq` had been lost or received and hence the system had stabilized.) Thus, this gives a bound on the expected convergence time. In the probabilistic version of Protocol G *choose* selects a sequence number at random, i.e.,

$$Prob(choose(S) = a) = \frac{1}{|S|}\ \ \forall\, a \in S$$

We assume that sequence numbers appear in the compact state with equal probability at the beginning of the convergence period.

Claim 1 *Let $e(n)$ be the expected number of new messages transmitted by the transmitter (through $\hat{\delta_2}$ transitions) during the convergence period of a system which starts at a state s*

such that $|CS(s)| = n$. *Then*

$$e(n) \;\leq\; \frac{(n-2)}{(|\Sigma|-2)} \tag{1}$$

Proof of Claim: We are interested in deriving a bound for $e(n)$. Thus, we do not consider the effects of transitions which fortuitously reduce the rank of the system (e.g. through message loss). In particular, we consider schedules in which the only transitions which alter the compact state are $\hat{\delta_2}$ and $\hat{\delta_3}$. By Observation 1, $\hat{\delta_2}$ occurs infinitely often. Thus computing $e(n)$ is equivalent to solving the following combinatorial problem:

Given an initial string of length n, such that each two successive elements in the string are different, add a new element to the front of the string whenever the two ends of the string are equal and delete an element from the tail of the string whenever the two ends are different. The problem is to determine $e(n)$, the expected number of times an element is added to the front of the string before the string length becomes two. All additions are chosen from alphabet Σ using a uniform distribution (preserving the property that no two adjacent elements are equal). Furthermore, we assume that the elements in the string are initially chosen from this alphabet using a uniform distribution.[3]

Let Q_i be the probability that the two ends of a random string of length i are equal and let $t = |\Sigma|$. Clearly, $Q_2 = 0$ and $Q_3 = 1/(t-1)$. In general Q_i depends on i. If we construct a string of length i from a string of length $i-1$ whose ends are equal, then the ends of the new string are equal with probability 0. However, if we start with a string of length $i-1$ whose ends are different, we will get a string whose ends are equal with probability $\frac{1}{t-1}$. Thus,

$$Q_i = (1 - Q_{i-1})\frac{1}{(t-1)}$$

Since $Q_2 = 0$:

$$\begin{aligned} Q_i &= -\sum_{j=0}^{i-2}\left(\frac{1}{1-t}\right)^j \\ &= \frac{1-(\frac{1}{1-t})^{i-2}}{t} \end{aligned} \tag{2}$$

Let $eq(i)$ be the expected number of additions performed on a random string of length i whose ends are equal. In order to prove the claim we first derive an expression for $eq(i)$. In this case we start with a string of the form

$$\overbrace{a_1 ... a_1}^{i}$$

[3]There are seemingly trival ways to solve this combinatorial problem; however, as one of the anonymous referees made clear, the probability distribution of the strings involved depends upon string length and alphabet size in subtle ways.

and add a number to yield a string of the form

$$a_0\overbrace{a_1...a_1}^{i}$$

We can compute $eq(i)$ by calculating the probability that, once we have performed the addition, we will get to delete numbers until we end up with a string of length k whose ends are again equal. $eq(i)$ then involves a sum of these terms.

The probability for a random string to have the form:

$$\overbrace{a_0a_1...a_0}^{k}\underbrace{\overbrace{a_j...}^{i-k}}_{\neq a_0}a_1$$

is:

$$Q(k)(\frac{1}{t-1})(\frac{t-2}{t-1})^{i-k}$$

for an arbitrary a_1. Where $\frac{1}{t-1}$ is due to the fact that we chose a_0 out of the set $\Sigma - a_1$.

The conditional probability for a string with the above form, given a random string of length i whose ends are equal is:

$$\frac{Q(k)(\frac{1}{t-1})(\frac{t-2}{t-1})^{i-k}}{Q(i)}$$

We can now calculate $eq(i)$, by considering all the cases and adding 1 because we have to perform at least the first addition.

$$\begin{aligned} eq(i) &= \frac{\sum_{k=3}^{i}(\frac{1}{t-1})Q(k)(\frac{t-2}{t-1})^{i-k}eq(k)}{Q(i)} + 1 \\ &= \frac{1}{t-2}\frac{\sum_{k=3}^{i-1}Q(k)(\frac{t-2}{t-1})^{i-k}eq(k)}{Q(i)} + \frac{t-1}{t-2} \\ &= \frac{1}{t-2}\frac{\sum_{k=3}^{i-1}Q(k)}{Q(i)} + \frac{t-1}{t-2} \end{aligned}$$

To prove the last equality it suffices to demonstrate that:

$$\sum_{k=3}^{i-1}Q(k) = \sum_{k=3}^{i-1}Q(k)(\frac{t-2}{t-1})^{i-k}eq(k)$$

which follows from induction on i.

To prove the claim, consider a random string of length n. The probability that $(n-k)$ sequence numbers are removed before the first addition is

$$Q(k)(\frac{t-2}{t-1})^{n-k-1}$$

for $k < n$ and $Q(n)$ for $k = n$. As with $eq(i)$, we consider all cases in computing $e(n)$:

$$\begin{aligned} e(n) &= \sum_{k=3}^{n-1} Q(k)(\frac{t-2}{t-1})^{n-k-1} eq(k) + Q(n) eq(n) \\ &= \frac{t-1}{t-2}\sum_{k=3}^{n-1} Q(k) + \frac{t-1}{t-2} Q(n) + \frac{1}{t-2}\sum_{k=3}^{n-1} Q(k) \\ &= \frac{t}{t-2}\sum_{k=3}^{n} Q(k) - \frac{Q(n)}{t-2} \end{aligned}$$

We now prove that $(t-2)e(n) = (n-2)$.

$$\begin{aligned} t\sum_{k=3}^{n} Q(k) - Q(n) &= n-2 \\ t\sum_{k=3}^{n} Q(k) &= n-2+Q(n) \\ \sum_{k=3}^{n}(1-(\frac{1}{1-t})^{k-2}) &= n-2+Q(n) \\ -\sum_{k=1}^{n-2}(\frac{1}{1-t})^{k} &= Q(n) \end{aligned}$$

Which follows (2). ∎

The fact that the time to stabilize is a function both of the initial state and of the size of the sequence number domain is not surprising. Notice that for reasonable values of n and $|\Sigma|$ the system stabilizes very rapidly. Even if started with a large number k of erroneous messages on the links, the system will converge to a legal state in expected $O(\frac{k}{|\Sigma|})$ round trip delays.

Note that $(n-2+e(n))$ bounds the expected number of messages that may be lost during the convergence period, where $(n-2)$ of those were on the links at the beginning of the convergence period and $e(n)$ where transmitted during the convergence period.

Unlike the expected number of messages transmitted during the convergence period the expected number of duplicate receptions is $O(n)$, since all the messages on the links could be received by the receiver at this time. To avoid this problem of large number of duplicates the *Probe* technique that is suggested in [AG88] could be employed. Essentially, in this modified protocol a process posts each message that it should send on a bulletin board instead of sending it (i.e., the transmitter posts the messages while the receiver posts its acknowledgments). Each processor is then responsible to probe (to read) the messages for it from the bulletin board of the other processor. The sequence of probe operations of each processor is implemented in the same way that the alternating bit protocols of this paper are. That is, each processor repeatedly sends a request to read the other's bulletin board, changing the sequence number on the requests each time it starts a new probe.

5 A Self-Stabilizing Token Ring

The alternating bit protocols of the previous sections can be extended to develop self-stabilizing token rings in which processes communicate asynchronously through FIFO links. A token ring consists of a cycle of processes which circulate a single privilege or token. Possession of the token is required to perform some action, e.g. accessing a shared bus. Although numerous previous papers have dealt with the problem of developing self-stabilizing token rings, these systems have assumed that the processes can read the states of their neighbors [Dij73, Dij82, Kru79, BGW87, BGW89, BP89].

First it is shown that the alternating-bit protocols presented can be viewed as degenerate token rings consisting of two processes. Then it is shown that a token ring can be created by introducing additional links and receiver processes. Consider the following "stripped down" version of protocol **G** in which the input and output have been removed and the statement `critical_section` has been added to both processes:

The statement "critical_section" in each process's code denotes the commands to be executed when the process holds the token. Notice that the transmitter may enter its critical section only when it receives a message with a sequence number containing its current state and the receiver may enter its critical section only when it receives a message with a sequence number different than its current state. Further, when the transmitter is in the critical section then its timeout is disabled, so it does not send any messages while it is in the critical section. In those system states where $Rank = 2$, at most one of these two may be in the critical section at a time. Furthermore, the system is guaranteed to converge to such a state.

A general token ring is created by adding receiver processes to the ring. The stabilization proof that has been previously presented is still valid: Consider a ring form from one transmitter **T'** and n processes $\mathbf{R'}_i$, $0 < i < n$. The state of this system is given by an expression

```
Transmitter T'::

seq int init 0;
i   int;

timeout → send(seq);
          reset_timeout
rcv(i)  → if (seq = i)
          then critical_section;
               seq := choose(Σ/seq);
               send(seq);
          fi ;
          reset_timeout
```

```
Receiver R'::

ack  int init 1 ;
j    int ;

rcv(j) → if ack ≠ j
         then critical_section;
              ack := j
         fi ;
         send(ack)
```

Figure 2: The protocol for the token ring processors, T' is the leader and R' are all the other processors

of the form:

$$\mathbf{T}'.\mathtt{seq}|t_0...t_s|\mathbf{R}'_1.\mathtt{ack}|r_0^1...r_u^1|...|\mathbf{R}'_n.\mathtt{ack}|r_0^n...r_v^n$$

The transitions of the system can then be modeled by $\delta_1...\delta_6$ given previously where ack in the definitions of $\delta_1...\delta_6$ is the ack variable of any of the n receivers. By using the same ranking function and proof for the alternating-bit protocols, we can prove the following corollary to Theorem 1.

Corollary 2 *A token ring formed from a transmitter process* **T'** *and* n *receiver processes* **R'** *is guaranteed to converge to a state* s *where* $Rank(s) = 2$ *provided that* **choose** *returns an aperiodic series of sequence numbers.*

In any state where $Rank = 2$, at most one process is enabled to enter its critical section. Furthermore, provided that each process continues to execute enabled actions, eventually each process will enter its critical section.

6 Discussion

Although the presented protocols require that the transmitter generate an infinite aperiodic sequence of numbers, in practice, this may 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.:

$$\neg(\exists \alpha, \beta : \alpha \in \Sigma^{+}, \beta \in \Sigma^{\leq \gamma} : ap = \alpha\beta^{\infty})$$

The probabilistic version of protocol **G** is a good practical solution to the link initialization problem in several 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 [PS88].

As a final note, Burns, Gouda, and Miller have recently studied "pseudo self-stabilizing" protocols [BGM90]. Although the original Alternating Bit protocol is not self-stabilizing, it is *pseudo self-stabilizing* in the following sense. From an illegal state with rank $K+2$ the system may lose K messages at arbitrary times, but otherwise it behaves legally. One of the anonymous referees has pointed out that use of the original Alternating Bit protocol for a token ring results in a protocol which is not even pseudo self-stabilizing.

Acknowledgments We would like to thank the referees for helpful comments. In particular the comments of referee C led us to some improvements in the analysis, and saved us from some embarrassing errors.

References

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. of the 28th IEEE Annual Symp. on Foundation of Computer Science*, pages 358–370, October 1987.

[AFWZ88] H. Attiya, M. J. Fischer, D. Wang, and L. D. Zuck. Reliable communication using unreliable channels. Manuscript, 1988.

[AG88] Y. Afek and E. Gafni. End-to-end communication in unreliable networks. In *Proc. of the Seventh ACM Symp. on Principles of Distributed Computing*, pages 131–148, August 1988.

[AGR90] Y. Afek, E. Gafni, and A. Rosen. Optimal fault tolerant file transfer protocol. Extended Abstract, April 1990.

[AUWY82] 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 Appls.*, 8(3):205–214, 1982.

[BGM90] J. E. Burns, M. G. Gouda, and R. E. Miller. Stabilization and pseudo stabilization. Technical Report TR-90-13, The University of Texas at Austin, May 1990.

[BGW87] G. M. Brown, M. G. Gouda, and C-l Wu. A self-stabilizing token system. In *Hawaii International Conference on System Sciences*, 1987.

[BGW89] G. M. Brown, M. G. Gouda, and C-l Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, c38(6):845–852, 1989.

[BP89] J. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11:330–344, 1989.

[BS83] 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*, 1983.

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12:260–261, 1969.

[Dij73] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1973.

[Dij82] E. W. Dijkstra. EWD 391 self-stabilization in spite of distributed control. In *Lecture Notes in Computer Science 92*, pages 41–46, 1982.

[GM88] M. G. Gouda and N. Multari. Self-stabilization of the window protocol. in preparation, 1988.

[Gou85] M. G. Gouda. On a 'simple protocol whose proof isn't': The state machine approach. *IEEE Transactions on Communications*, com-33(4):380–383, 1985.

[Hai85] B. Hailpern. A simple protocol whose proof isn't. *IEEE Transactions on Communications*, com-33(4):330–337, 1985.

[KP89] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. In M. Evangelist and S. Katz, editors, *MCC Technical Report Number STP-379-89, Proceedings of the MCC Workshop on Self-Stabilizing Systems*, 1989. (Also in PODC-90.).

[Kru79] H.S.M. Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 8(2):91–95, 1979.

[Lam84] L. Lamport. The mutual exclusion problem: Part II – statement and solutions. *Journal of the ACM*, 33(2):327–348, 1984.

[LMF88] N. Lynch, Y. Mansour, and A. Fekete. The data link layer: Two impossibility results. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 1988.

[Mul89] N. Multari. *Self-Stabilizing Protocols*. PhD thesis, The University of Texas at Austin, 1989.

[PS88] K. Paliwoda and J. W. Sanders. The sliding-window protocol in CSP. Technical Report PRG-66, Oxford University Computing Laboratory, 1988.

[Ste76] M. V. Stenning. A data transfer protocol. *Computer Networks*, 1:99–110, 1976.

[Tan81] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, 1981.