VICTORIA UNIVERSITY OF WELLINGTON

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Recoverable Sequence Transmission Protocols

Ewan D. Tempero and Richard E. Ladner

Technical Report CS-TR-93/4
September 1993

Abstract

We consider the sequence transmission problem, that is, the problem of transmitting an infinite sequence of messages $x_1x_2x_3\ldots$ over a channel which can both lose and reorder packets. We define performance measures, ideal transmission cost and recovery cost, for protocols which solve the sequence transmission problem. Ideal transmission cost measures the number of packets needed to deliver x_n when the channel is behaving ideally and recovery cost measures how long it takes, in terms of number of messages delivered, for the ideal transmission cost to take hold once the channel begins behaving ideally. We also define lookahead which measures the number of messages the sender can be ahead of the receiver in the protocol. We show that any protocol with constant recovery cost and lookahead requires linear ideal transmission cost. We describe a protocol $\mathcal{P}_{\mathrm{lin}}$ which has ideal transmission cost $2n$, recovery cost 1, and lookahead 0.

Publishing Information

This work was supported by NSF Grants CCR-8714782 and CCR-9108314. It has been submitted for publication. A preliminary version of this paper appeared in the Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, August 22-24, 1990 [TL90].

Author Information

Ewan Tempero is a lecturer in the Department of Computer Science at Victoria University of Wellington.

Richard Ladner is a Professor in the Department of Computer Science and Engineering, University of Washington, Seattle WA 98195, United States of America

1 Introduction

In this paper we examine the behavior of protocols that solve the problem of sending streams of *messages* between processes in a network. In particular, we are interested in the so-called *Sequence Transmission Problem* (STP), which concerns a system with two asynchronous processes that communicate by sending *packets* to each other through an asynchronous channel. One of the processes, the *Sender*, has a infinite sequence of messages, $X = x_1x_2x_3\ldots$, on a one-way read-only input tape. The other process, the *Receiver*, writes a sequence of messages on a one-way write-only output tape. Informally, a solution to STP is a *protocol* followed by the processes that, when started for any input sequence X, results in the Receiver writing X on its output tape.

The design of the protocol will be influenced by the type of behavior expected from the channel. If the channel is guaranteed to deliver each packet unchanged and in the order it was sent, then the protocol's behavior can be quite simple. If the channel can be faulty, then the protocol's behavior may be quite complex, depending on the types of faults the channel exhibits. In this paper we discuss the behavior of protocols used for non-FIFO channels. Such channels can lose and reorder the packets sent over them.

A simple solution to this problem can be developed by using packets of the form (n, x_n). In common terminology, the solution uses *sequence numbers* to detect losses and reordering of packets. This solution is unsatisfactory since it requires arbitrarily long packets. For this work, we will consider only protocols that require bounded length packets. Intuition suggests that STP cannot be solved using only bounded length packets because there are only a finite number of such packets. Because packets can be lost, it is likely that a process must send duplicates to ensure something gets through. Duplicate packets can thus be "captured" by the channel and can later be "played back" to the Receiver causing it to believe it is getting new information when it is not. Surprisingly, Attiya, Fischer, Wang, and Zuck [AFWZ89] overcame this problem and developed a sequence transmission protocol which uses only finitely many packets. This protocol was never published, but a related sequence transmission protocol appeared in the paper [AAF$^+$91]. These protocols are "unbounded" in the sense that there is no bound on the number of packets needed to transmit a given message even if the channel is currently behaving well.

Several notions of *bounded*[1] protocols have been suggested [AAF$^+$91, MS92, WZ89]. Bounded protocols are meant to capture the idea that the performance of an efficient protocol may depend on the position of the message in the input sequence but is otherwise independent of its history. This means we can predict the cost of sending any message when the channel is behaving well. We call this the *ideal behavior* of the protocol. When the protocol is behaving ideally it would be desirable to be able to predict how many packets will be needed to deliver a message. In particular, if the channel has been faulty, but returns to good behavior, the protocol must also return to its ideal behavior, that is, it must recover from the faults. Thus, bounded protocols should be more properly called *recoverable* protocols. Until now, it was not known whether recoverable sequence transmission protocols exist.

1.1 Results

In this paper we introduce the new measures *ideal transmission* and *recovery* costs which measure the performance of a sequence transmission protocol. Informally, a sequence transmission protocol has ideal transmission cost $f(n)$ and recovery cost $g(n)$ if, at any time before the n-th message is written, if the channel begins to behave ideally, then for all $m \geq n + g(n)$ the delivery of at most $f(m)$ packets will cause the m-th message to be written. Put another

[1]Different authors use the term bounded with different, but closely related, meanings.

way, once the channel begins behaving ideally and after a recovery period of $g(n)$ message deliveries, the number of packets needed to deliver a message is bounded by the function f. Thus, if functions f and g exist such that a sequence transmission protocol has ideal transmission cost $f(n)$ and recovery cost $g(n)$, then the protocol is recoverable in the sense we desire. We also introduce the measure of *lookahead* which describes how far ahead in the input sequence the sending process is allowed to read. We relate these new notions to the existing definitions of bounded protocols.

Crucial to our results is what we mean for the channel to behave ideally. We capture this notion in the definition of a run r of a protocol $\mathcal{P}$ being ideal from a time t. The formal definition of a run of a protocol being ideal from a time not only depends on the characteristics of the channel, but on the structure of the protocol.

There are two principal results. First, we show that any sequence transmission protocol with ideal transmission cost $f(n)$ and constant recovery and lookahead costs must have $f(n) = \Omega_\infty(n)$.[2] This lower bound is a very robust result that applies to any definition of ideal run that satisfies a very natural property called the extension property (*cf.* section 3.2).

Second, we present a sequence transmission protocol $\mathcal{P}_{\text{lin}}$ which has ideal transmission cost $2n$, recovery cost 1 and lookahead 0. This result depends on the restricted definition of a run of $\mathcal{P}_{\text{lin}}$ being stop-and-wait ideal (*cf.* section 5.1.3). Thus, this upper bound is not as robust as our lower bound. Nonetheless, the protocol $\mathcal{P}_{\text{lin}}$ is the first recoverable sequence transmission protocol, that is, the first to exhibit any kind of reasonable bounded behavior.

1.2 Related Results

As mentioned earlier Attiya, Fischer, Wang, and Zuck [AFWZ89] showed that sequence transmission protocols exist which use bounded length packets. Later they joined in with Afek, Fekete, Mansour, and Lynch to develop a related sequence transmission protocol [AAF$^+$91]. The unpublished protocol [AFWZ89] requires $\Omega_\infty(2^n)$ packets to deliver the n-th message regardless of the history of the protocol. The protocol of Afek *et al* [AAF$^+$91] uses only a constant number of packets per message if the channel behaved well from the very beginning. However, both protocols are not recoverable in that, depending on the execution history, a given message can require an unbounded number of packets to be successfully delivered.

Lynch, Mansour, and Fekete [LMF88, AAF$^+$91] defined the notion of a $f(n)$-*weakly-bounded* protocol. A protocol is $f(n)$-weakly-bounded for some function f if, immediately after x_{n-1} has been written by the Receiver, it is possible, under ideal conditions, that x_n can be written before not more than $f(n)$ packets are received by the Receiver. That is, such protocols must be able to recover when at most $f(n)$ packets are received after x_{n-1} has been written. They showed that for any constant k, no k-weakly-bounded protocol exists. A sequence transmission protocol with ideal transmission cost $f(n)$, recovery cost 1 and lookahead 0 is a $f(n)$-weakly-bounded protocol. Thus, our protocol $\mathcal{P}_{\text{lin}}$ is $2n$-weakly-bounded. Our proof that any protocol with ideal transmission cost $f(n)$ and constant recovery and lookahead costs must have $f(n) = \Omega_\infty(n)$ is based on the proof technique used in showing there is no k-weakly-bounded protocol [LMF88, AAF$^+$91]. Indeed, our proof can be used to show every sequence transmission protocol which is $f(n)$-weakly-bounded must have $f(n) = \Omega_\infty(n)$.

Wang and Zuck [WZ89] and Mansour and Schieber [MS92] defined a stronger notion of bounded protocols, called $f(n)$-*bounded* protocols.[3] Such protocols differ from weakly-bounded protocols in that the Receiver must be able to write x_n if it receives $f(n)$ packets under ideal circumstances at *any* time after x_{n-1} has been written. In other words, these

[2] We define $f(n) = \Omega_\infty(g(n))$ to mean that for some $c > 0$, $f(n) > c \cdot g(n)$ for infinitely many n.

[3] The terminology we use here is from Wang and Zuck's work.

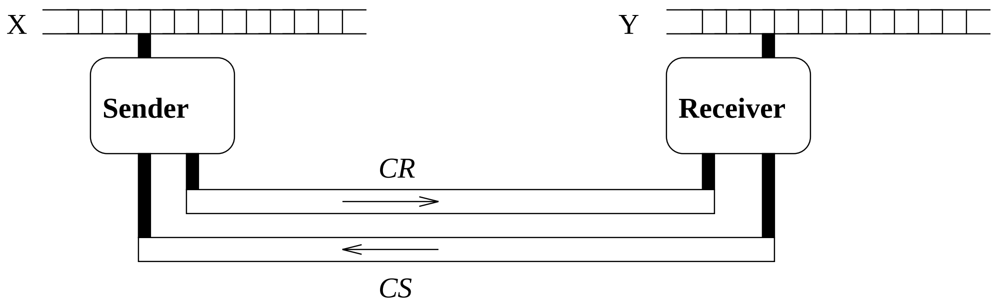

Figure 1: Model for the sequence transmission problem. The Environment is everything depicted except the Sender and Receiver.

protocols must be able to recover at any time, not just at those times at which a message is written. Wang and Zuck showed that, even if the sending process knows the entire input sequence in advance, a behavior they call *non-uniform*, there is no $f(n)$-bounded protocol for any f. Mansour and Schieber also showed that $f(n)$-bounded protocols do not exist. An $f(n)$-bounded protocol is equivalent, in our terminology, to a protocol with ideal transmission cost $f(n)$ and recovery cost 0.

In most of the previous work [AFWZ89, AAF$^+$91, LMF88, MS92] it is assumed that the sending process does not have control over the arrival inputs, so that their protocols are required to be *on-line*, that is, the sending process will not read x_n until the receiving process has written x_{n-1}. This on-line requirement is expressed in our terminology as lookahead 0. We consider protocols which may have lookahead greater than zero. Infinite lookahead corresponds to the notion of non-uniform defined by Wang and Zuck [WZ89].

1.3 Organization

The remainder of the paper is organized as follows. The next section summarizes our formal model. Section 3 presents our measures of protocol behavior and section 4 gives our lower bound for recoverable protocols. In section 5 we present and prove the correctness of the sequence transmission protocol $\mathcal{P}_{\text{lin}}$ which has ideal transmission cost $2n$, recovery cost 1, and lookahead 0. In section 6, we present our conclusions.

2 A Model for Sequence Transmission Protocols

For our purposes, a sequence transmission protocol consists of three major parts, a Sender denoted by **S**, a Receiver denoted by **R**, and an Environment denoted by **E**, which are described informally in Figure 1. The Sender and Receiver are deterministic programs which interact with the Environment, but do not interact with each other except indirectly via the communication channels. The Environment contains global time, the input tape, output tape, the Sender-to-Receiver channel, $\mathcal{CR}$, and the Receiver-to-Sender channel, $\mathcal{CS}$. The Environment is adversarial in that its actions are not deterministic. Informally, a sequence transmission protocol is a pair of programs for the Sender and Receiver which satisfy some given properties no matter what the Environment does. Naturally, there may be some restrictions on the Environment to ensure it is "fair" to the Sender and Receiver. We formally define these properties later.

The input tape, which is read by the Sender, contains the *input sequence*, $X = x_1x_2x_3...$, which is an infinite sequence of *messages*. An *output sequence*, $Y = y_1y_2y_3...$, will be written by the Receiver on the output tape. Messages are taken from a finite message alphabet, Σ, that is, the set of possible input sequences is Σ^ω. We assume that Σ has at least two members,

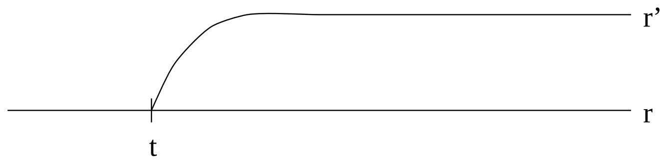

Figure 2: The run r' extends the point (r, t).

otherwise the sequence transmission problem is trivial. The input tape is read-only and the output tape is write-only.

Communication between the Sender and Receiver is done over the two channels, $\mathcal{CR}$ and $\mathcal{CS}$. There is a finite set of *packets*, Σ_P, which may be sent by the Sender on the $\mathcal{CR}$ channel or by the Receiver on the $\mathcal{CS}$ channel. There is no strict relationship between the packet alphabet Σ_P and the message alphabet Σ, although it is helpful to think of Σ_P as pairs (h, x) where h is header and x is a message. The Sender can receive a packet from the $\mathcal{CS}$ channel and the Receiver from the $\mathcal{CR}$ channel. Being adversarial, the Environment may choose to cause a channel to deliver any undelivered packet. In addition, there is a special packet ϕ, not in Σ_P, which the channel can deliver at any time. This "non-packet" can be used in the case where a process is trying to receive a packet but either there are no packets in the channel or the Environment does not want to deliver a packet. Rather than force the process to wait until a packet is ready, the Environment can instead deliver ϕ. This is similar to the use of a "time-out" mechanism. Thus ϕ is used to model the variable delay between when a packet is sent and the time when it is received. In our model packets are never duplicated and lost packets are simply packets that remain in a channel forever.

2.1 The Runs Model

As a formalism, we adopt the runs model of Halpern and Zuck [HZ93]. We briefly review the runs model as it relates to the sequence transmission problem. We will begin with an abstract specification, filling in details as needed.

There is a set of *global states*, $\mathcal{G}$, which consists of triples of the form $(s_{\mathbf{S}}, s_{\mathbf{R}}, s_{\mathbf{E}})$, where $s_{\mathbf{S}}$ is a local state of the Sender, $s_{\mathbf{R}}$ is a local state of the Receiver, and $s_{\mathbf{E}}$ is a local state of the Environment. There is a set of initial states $\mathcal{G}_0 \subseteq \mathcal{G}$. A *run* r is a sequence of global states $r(0), r(1), r(2), ...$ where the t-th global state is $r(t)$. It is helpful to think of t as "global time" which is generally not visible to either the Sender or Receiver. In its most abstract form, a protocol $\mathcal{P}$ is simply a relation on global states, that is, a subset of $\mathcal{G} \times \mathcal{G}$. A run is *consistent* with a protocol $\mathcal{P}$ if $r(0) \in \mathcal{G}_0$ and, for any $t > 0$, $(r(t-1), r(t)) \in \mathcal{P}$. We will denote the set of runs consistent with $\mathcal{P}$ by $\mathcal{R}_{\mathcal{P}}$.

For a run r and a time t, we call the pair, (r, t), a *point*. The point (r, t) describes $r(0), r(1), ..., r(t)$, the run r from time 0 through time t. We say a run $r' = r'(0), r'(1), r'(2), \ldots$ *extends* a point (r, t) if $r(i) = r'(i)$, $0 \leq i \leq t$. This can be visually depicted in figure 2. Note in particular that the input sequence is the same for both r and r' since $r(0) = r'(0)$ and the initial states include the input sequence.

A consistent run of $\mathcal{P}$ describes the global history of an execution of the protocol. Also important are the local histories as viewed individually by the Sender and Receiver. For a run r and time t, the history of the Sender in r until time t, denoted by $H(\mathbf{S}, r, t)$, is defined inductively on t as

$$\begin{array}{rcll} H(\mathbf{S}, r, 0) & = & s_0 & \text{if } r(0) = (s_0, s_1, s_2) \\ H(\mathbf{S}, r, t) & = & H(\mathbf{S}, r, t-1) & \text{if } r(t-1) = (s_0, s_1, s_2) \text{ and } r(t) = (s_0, s_1', s_2') \\ H(\mathbf{S}, r, t) & = & H(\mathbf{S}, r, t-1), s_0' & \text{if } r(t-1) = (s_0, s_1, s_2)\ , r(t) = (s_0', s_1', s_2') \\ & & & \text{and } s_0 \neq s_0' \end{array}$$

In a similar way, the history of the Receiver $H(\mathbf{R}, r, t)$ can be defined. Thus, the history of the Sender in r until time t is just the sequence of local states that the Sender goes through until time t. We say that two points (r, t) and (r', t') are *similar to* P for $P \in \{\mathbf{S}, \mathbf{R}\}$, denoted $(r, t) \stackrel{P}{\sim} (r', t')$, if $H(P, r, t) = H(P, r', t')$. The fact that it is possible that $(r, t) \stackrel{S}{\sim} (r', t')$ and $t \neq t'$ demonstrates that the Sender may not "know" the global time.

2.2 Model Details

To be a sequence transmission protocol a protocol $\mathcal{P}$ must be more than simply a relation on global states. The relation $\mathcal{P}$ is determined by the states and allowable actions of the Sender, Receiver, and Environment. A complete description of the model can be found in Tempero's dissertation [Tem90]. We give here the salient aspects of the model details needed for the proofs in the paper.

A local state of the Sender (or Receiver) is its complete state as a program. A state of the Environment is the global time, the contents and head positions of the the input and output tapes and the contents of the two channels. The initial states are those where the Sender and Receiver are on their initial instructions, and the Environment has global time zero, empty channels, blank output tape, the input tape loaded with an input and the input and output heads are on the first position on their respective tapes. Since the Sender and Receiver do not directly interact, the relation $\mathcal{P}$ is determined by the *joint actions* of the Sender and Environment or Receiver and Environment. A joint action in a global state s yields a state s' so that the pair (s, s') is in the relation $\mathcal{P}$. Basically, s' is a result of either the Sender or the Receiver, but not both, executing one instruction in the state s. If r is a run and a joint action causes the transition $(r(t-1), r(t)) \in \mathcal{P}$ then we say the action happened at time t or at point (r, t), since the result of the action is observable in the state $r(t)$. The Sender/Environment joint actions include *read*, *send*, and *receive*, and the Receiver/Environment joint actions include *write*, *send*, and *receive*. Both Sender and Receiver may take other actions as well. The only effect on the Environment of one of these other actions is that global time is incremented. If s is a global state we specify the state of any entity Z by Z^s. For example, the contents of the two channels $\mathcal{CR}$ and $\mathcal{CS}$ in state s are $\mathcal{CR}^s$ and $\mathcal{CS}^s$, respectively.

To be more specific about the send and receive actions, if at a point (r, t) the Sender sends a packet $p \in \Sigma_P$, then $\mathcal{CR}^{r(t)} = \mathcal{CR}^{r(t-1)} \cup \{(p, t)\}$, that is a "time-stamped" copy of the packet p is put into the $\mathcal{CR}$ channel. If the Receiver receives a packet at a point (r, t) then one of two things could have happened at point (r, t), as determined adversarily by the Environment. Either the Receiver receives a packet $p \in \Sigma_P$ and $\mathcal{CR}^{r(t)} = \mathcal{CR}^{r(t-1)} - \{(p, t')\}$, for some $(p, t') \in \mathcal{CR}^{r(t-1)}$, or the Receiver receives ϕ and $\mathcal{CR}^{r(t)} = \mathcal{CR}^{r(t-1)}$. In the former case the Receiver actually receives a packet sent previously by the Sender and in the latter case ϕ is received and the channel $\mathcal{CR}$ does not change. In a similar way the Receiver can send and the Sender can receive.

A read is a joint action by the Sender/Environment and a write is a joint action by the Receiver/Environment. A read at point (r, t) means that the message under the read head in state $r(t-1)$ is incorporated into the local state of the Sender in state $r(t)$ and the read head is advanced. A write at point (r, t) means that a message is written in state $r(t)$ by the

Receiver in the write head position of state $r(t-1)$ and the write head position is advanced. The value of X never changes during a run, that is, for a run r, $n \geq 1$ and $t \geq 0$, $x_n^{r(t)} = x_n^{r(0)}$. With Y, it is possible that y_n is blank for some values of t. However, once y_n is written (set to a non-blank value), it will always have that value, that is: For any run r, $n \geq 1$, and $t \geq 0$, if $y_n^{r(t)}$ is not blank then for every $t' \geq t$, $y_n^{r(t')} = y_n^{r(t)}$. We will use y_n^r to denote the nth message written on the output tape and can unambiguously refer to x_n^r.

If y_n^r is written at point (r,t) then we define $t_n^r = t$. If y_n is never written in run r then we define $t_n^r = \infty$, otherwise t_n^r is finite. For convenience we define $t_0^r = 0$. Thus, the sequence of times $t_0^r, t_1^r, t_2^r, t_3^r \ldots$ represents the protocol's progress in run r.

In order to properly define a correct solution to the STP we need to define the notion of a fair run of a protocol. Informally, a run r is fair if the processes are scheduled fairly and the channels behave fairly. We say that the processes are *scheduled fairly in* r if for infinitely many t, the Sender/Environment have a joint action at point (r,t) and for infinitely many t the Receiver/Environment have a joint action at point (r,t). For, channel fairness we choose the notion of copy fairness defined by: the $\mathcal{CR}$ channel is *copy fair in* r if for every $p \in \Sigma_P$, if there are infinitely many t such that the Sender sends p at point (r,t) and there are infinitely many t such that the Receiver receives at point (r,t), then for infinitely many t the Receiver receives p at point (r,t). Copy fairness for the $\mathcal{CS}$ channel is defined similarly. Let $\mathcal{P}$ be a protocol:

> A run $r \in \mathcal{R}_{\mathcal{P}}$ is *fair* if the processes are scheduled fairly in r and the channels are copy fair in r.

We let $\mathcal{F}_{\mathcal{P}}$ denote the set of fair runs of the protocol $\mathcal{P}$.

A weaker notion of channel fairness, called *packet fairness*, can be defined. A channel is packet fair for a run if infinitely many packets are sent on the channel by one process and infinitely many receives are attempted by the other process then infinitely many packets are received by the other process. It is possible for a channel to be packet fair in a run but not copy fair. For example, if in a run both p_1 and p_2 are sent infinitely often, but instances of p_2 are never delivered, then the channel is packet fair but not copy fair in the run. We chose the notion of copy fairness because the sequence transmission protocol we present is correct under copy fairness, but not under packet fairness. Both notions of fairness are much weaker than any fairness condition which could be expected from a real channel.

2.3 Sequence Transmission Protocol

We can now formally state what it means to solve the sequence transmission problem.

> A protocol $\mathcal{P}$ is a *sequence transmission protocol* if the following conditions hold:
>
> **Safety:** For any $r \in \mathcal{R}_{\mathcal{P}}$, $t \geq 0$, and $n \geq 1$, if $t_n^r \leq t$ then $y_n^{r(t)} = x_n^{r(t)}$. That is, at any point, Y is a prefix of X.
>
> **Liveness:** For any fair run, $r \in \mathcal{F}_{\mathcal{P}}$, and $n \geq 1$, $t_n^r < \infty$. That is, in a fair run, an infinite number of messages are written.

3 Protocol Measures

3.1 Motivation

The purpose of our work is to find the "best" protocol for solving the sequence transmission problem when the channels are non-FIFO. To do so, we must say what "best" means.

For most of this work we will be assuming an adversarial Environment, that is, it will be allowed to behave in any manner that does not violate the fairness requirements of the channels. In particular, it is allowed to behave in a manner that appears to be a deliberate attempt to cause a protocol to fail. For this reason, measures of protocol cost that reflect total time taken, or total packets sent are not very meaningful since such channels can force these costs to be arbitrarily large. However a "good" protocol should have "nice" properties at least when the channels *are* cooperating. In particular, we expect a protocol to be efficient when the channels are good and at least make progress when the channels are bad. For these reasons, the measures we use describe the protocol's cost on a per-message basis only when the channels are cooperating. Our notion of ideal transmission cost is meant to capture the efficiency of the protocol when the channels are cooperating.

In a real network, a channel can behave nicely for long periods of time but have short periods when it is faulty. If the protocol is efficient during the good periods then it must, in some sense, recover from the channel faults, that is, when the channel goes from bad behavior to good behavior, the protocol must return from just making progress to sending messages efficiently. Our notion of recovery cost is meant to measure how long this transition takes.

Finally it is possible that the Sender may "read ahead" of what it is currently sending to the Receiver. An example of this is the sliding-window class of protocols [Ste76, Tan88, Sta88] which have the property that the Sender is allowed to send the next message without receiving any acknowledgements for the previous message provided it is not too far ahead of the Receiver. In the traditional use of sliding-window protocols the specific contents of the messages that are read does not affect the behavior of the protocol. By contrast, the protocols we consider have the property that the actions taken by the Sender and Receiver may depend on the contents of the messages. Thus, in full generality, the ability to look ahead may affect the performance of sequence transmission protocols. Our notion of lookahead is meant to measure how far the Sender can be ahead of the Receiver in their message sequences.

3.2 Ideal Runs and the Extension Property

In the real world, a protocol developer has a particular view of how networks behave, that is, what its properties are. For example, packets arrive in the order sent and take a bounded amount of time to arrive, where the bound can be determined in advance. The developer then proceeds to develop a protocol that takes as much advantage as possible of those properties. The more information about the network that is available, the more advantage the protocol can take and different properties result in different protocols.

In the situation we are interested in, the network's behavior is such that some of the time it behaves in such a way that efficient protocols are possible (the "ideal channel behavior") but some of the times it behaves in such a way that efficient protocols are not possible. The protocol developer tunes the protocol to take advantage of efficient behavior while at the same time reducing as much as possible the effect of the inefficient behavior.

In this way, the notion of "ideal channel behavior" can be discussed independently of the actual protocol, while still explaining why our definitions of ideal behavior specifically mention the protocol. Because we are using the runs model we can only describe what happens to the protocol as it interacts with the channel. Thus, it is impossible to describe the ideal behavior of the channel without also talking about the protocol that accesses the channel. As a consequence, we model the notion of "ideal channel behavior" with the concept that a run $r \in \mathcal{R}_{\mathcal{P}}$ *is ideal from time* t.

Since different protocols may have different expectations of an ideal channel it seems counterproductive to give a single definition which would cover all cases. For example, the Sender in a stop-and-wait protocol does alternating sends and receives while in a sliding-window protocol the Sender may execute a number of send before doing a receive. In section 5.1.3 we give the formal definition of *stop-and-wait ideal runs* which applies to our protocol $\mathcal{P}_{\text{lin}}$ and similar stop-and-wait protocols.

For the time being we will just state the *extension property* which must be true for any definition of ideal runs, not only stop-and-wait ideal runs. The extension property is the only property of ideal runs required to prove our lower bound. In addition, the extension property guarantees that the definition of ideal transmission and recovery costs is non-trivial.

Property 3.1 (Extension Property) *Let $\mathcal{P}$ be a sequence transmission protocol. For every $r \in \mathcal{R}_{\mathcal{P}}$ and $t \geq 0$, there is a run $r' \in \mathcal{F}_{\mathcal{P}}$ extending (r,t) that is ideal from time t and for any $t' \geq t$, $\mathcal{CR}^{r(t)} \subseteq \mathcal{CR}^{r'(t')}$ and $\mathcal{CS}^{r(t)} \subseteq \mathcal{CS}^{r'(t')}$.*

That is, every run can be extended to an ideal run in which no old packets are ever delivered. The concept of being ideal from a time t is meant to capture the notion that the channel does what is expected from time t on, but any behavior from the channel is possible before time t including the capturing of all previously sent packets.

3.3 Ideal Transmission and Recovery Costs

Before describing these measures formally, we introduce the concept of transmission cost of a run from a time. For this cost, we choose the number of packets that the Receiver receives between writing messages. We consider number of packets to be more appropriate than time taken because this is something the protocol has some control over whereas, in a real system, the time taken depends on such things as bandwidth, which the protocol has no control over.

If $r \in \mathcal{R}_{\mathcal{P}}$, $n > 0$, and $t^r_{n-1} \leq t < t^r_n$ then we call the sequence of packets (including ϕ's) received by the receiver from time t until time t^r_n a *delivery sequence* and denote it by $\sigma(r,t)$. Thus, if $\sigma(r,t) = p_1p_2p_3\ldots p_k$, then there is a sequence $t \leq t^1 < t^2 < t^3 < \ldots < t^k < t^r_n$ such that, for $1 \leq i \leq k$ packet p_i was received by the Receiver at point (r,t_i). Furthermore, if $t \leq t' < t^r_n$ and $t' \neq t_i$ for any i, then the Receiver does not receive any packet (including ϕ's) at point (r,t'). If y^r_n never gets written then $\sigma(r,t)$ may have infinite length. Thus, in the run r, if the Receiver receives the sequence $\sigma(r,t)$ after time t, then it will write the next message, y^r_n. If σ is a delivery sequence we let $|\sigma|$ denote the length of σ.

Let r be any run in $\mathcal{R}_{\mathcal{P}}$ and let $f : N^+ \rightarrow N$. A $r \in \mathcal{R}_{\mathcal{P}}$ has *transmission cost $f(n)$ from time t*, if $|\sigma(r,t)| \leq f(n)$ and also $|\sigma(r,t^r_{m-1})| \leq f(m)$ for all $m > n$, where n is an integer such that $t^r_{n-1} \leq t < t^r_n$. That is, a run has transmission cost $f(n)$ from time t if every message, y_m written after time t in r requires not more than $f(m)$ packets to be delivered.

Ideal transmission and recovery costs are defined jointly because there is an interaction between the two concepts. Ideal transmission cost measures the number of packets needed to deliver the n-th message when the channel is behaving ideally while recovery cost measures how long it takes for a protocol to achieve its ideal behavior after a run becomes ideal. The results of Mansour and Schieber [MS92] indicate that the number of packets needed to deliver the very next message is unbounded. Thus, we measure the recovery time as the number of messages which need to be delivered before the ideal transmission cost is achieved. One way to think of recovery cost is that it is the time required to "resynchronize" after the channel returns to ideal behavior, that is, the time needed by both processes to realize that the channel is indeed no longer faulty. Since the writing of a message by the receiver signifies that progress has been made, it seems to be the best time for a protocol to evaluate the channel's current behavior. Thus, the number of messages that have to be written before the protocol recovers seems an appropriate choice.

Since recovery cost represents how long it takes for a protocol to achieve its ideal behavior, it cannot be defined without also saying what the ideal transmission cost is. Let N be the set of non-negative integers and N^+ be the set of positive integers. Let $f : N^+ \rightarrow N$ and $g : N^+ \rightarrow N^+$. Formally:

> A sequence transmission protocol $\mathcal{P}$ has *ideal transmission cost* $f(n)$ and *recovery cost* $g(n)$ if, for any run $r \in \mathcal{R}_\mathcal{P}$, $n \geq 1$, and t such that $t^r_{n-1} < t \leq t^r_n$, any run $r' \in \mathcal{F}_\mathcal{P}$ extending (r,t) that is ideal from time t has transmission cost $f(n)$ from time $t^{r'}_{n-1+g(n)}$.

Intuitively, ideal transmission cost $f(n)$ and recovery cost $g(n)$ means that if the protocol is in the process of delivering the n-th message and the protocol begins behaving ideally, then after $g(n)$ messages are delivered, all subsequent messages, x_m, are to be delivered after the Receiver receives not more than $f(m)$ packets. We call the period between t and $t^{r'}_{n-1+g(n)}$ the *recovery period*. At the end of the recovery period, the protocol must behave within the ideal transmission cost bound $f(n)$. The extension property (property 3.1) guarantees that the definition of ideal transmission and recovery costs is non-trivial, that is, it guarantees the existence of runs r' extending (r,t) which are ideal from time t.

For a given protocol $\mathcal{P}$ there may not be a unique minimal pair (f,g) such that $\mathcal{P}$ has ideal transmission cost $f(n)$ and recovery cost $g(n)$. If g is made higher, then potentially f could be made lower.

3.4 Lookahead

Informally, the lookahead of a sequence transmission protocol is the number of messages the Sender can be ahead of the Receiver. Formally, let $h : N^+ \rightarrow N$.

> A sequence transmission protocol $\mathcal{P}$ has *lookahead* $h(n)$ when, for any $n \geq 0$, run $r \in \mathcal{R}_\mathcal{P}$ and time $t \leq t^r_n$, if the Sender has read x_m at (r,t) then $m \leq n + h(n)$.

That is, if the Sender reads x_m at some time on or before y_n is written, then m cannot exceed n plus the lookahead bound $h(n)$. Lookahead 0 means that reads and writes must alternate which is similar to the on-line property of Lynch, Mansour, and Fekete [LMF88] and Mansour and Schieber [MS92].

3.5 Recoverable Sequence Transmission Protocols

Let $f : N^+ \rightarrow N$, $g : N^+ \rightarrow N^+$, and $h : N^+ \rightarrow N$. We define recoverable protocols as follows:

> A sequence transmission protocol $\mathcal{P}$ is $(f(n), g(n), h(n))$*-recoverable* if it has ideal transmission cost $f(n)$, recovery cost $g(n)$, and lookahead $h(n)$.

A $(f(n), g(n), h(n))$-recoverable sequence transmission protocol is one which recovers from bad channel behavior in bounded "time," uses a bounded number of packets per message once the recovery period is over, and allows the Sender only bounded extra information about the input sequence.

We will be mainly interested in protocols with constant recovery cost and lookahead. Hence, we define:

> A sequence transmission protocol $\mathcal{P}$ is $f(n)$*-recoverable* if for some constants $b \geq 1$ and $c \geq 0$, $\mathcal{P}$ is $(f(n), b, c)$-recoverable.

3.6 Relationship to Other Measures

At this point we would like to relate our notion of ideal transmission and recovery cost to the notions of weakly-bounded and bounded protocols mentioned in our description of previous work. The notion of a $f(n)$-weakly-bounded protocol as used by Lynch, Mansour, and Fekete [LMF88] means that recovery is immediate from a time t_n^r. More precisely,

> A sequence transmission protocol $\mathcal{P}$ is $f(n)$-*weakly-bounded* if it has lookahead 0 and for any run $r \in \mathcal{R}_\mathcal{P}$, $n \geq 0$, any $r' \in \mathcal{F}_\mathcal{P}$ extending (r, t_n^r) that is ideal from time t_n^r has transmission cost $f(n)$ from time $t_n^{r'}$.

Thus, a sequence transmission protocol with ideal transmission cost $f(n)$, recovery cost 1, and lookahead 0 is $f(n)$-weakly-bounded.

The notion of a $f(n)$-bounded protocol as used by Wang and Zuck [WZ89] and Mansour and Schieber [MS92] means that recovery is immediate from any time. More precisely,

> A sequence transmission protocol $\mathcal{P}$ is $f(n)$-*bounded* if it has lookahead 0 and for any run $r \in \mathcal{R}_\mathcal{P}$, $n \geq 1$, and t such that $t_{n-1}^r < t \leq t_n^r$, any run $r' \in \mathcal{F}_\mathcal{P}$ extending (r, t) that is ideal from time t has transmission cost $f(n)$ from time t.

This definition is tantamount to saying that the recovery cost is zero in our definition of ideal transmission and recovery cost. Because it has been shown that $f(n)$-bounded protocols do not exist for any f [WZ89, MS92] we do not consider recovery cost zero. Our recovery cost bound, $g(n)$, is assumed to be positive.

4 Lower Bound for Recoverable Protocols

In this section we present results relating to lower bounds on the costs of sequence transmission protocols according to the measures introduced in the previous section. The main result of the section, stated below, says that recoverable protocols, that is, sequence transmission protocols with constant recovery cost and lookahead, must have at least linear ideal transmission cost.

Theorem 4.1 (Lower Bound) *If $\mathcal{P}$ is a $f(n)$-recoverable sequence transmission protocol then $f(n) = \Omega_\infty(n)$.*

The proof of the theorem is in section 4.4 and relies on two lemmas, the Playback lemma and Capture lemma.

4.1 Properties of the Model

In order to prove the Lower Bound theorem we rely on several properties of the model. Let $r \in \mathcal{R}_\mathcal{P}$, $t \geq 0$, and $p \in \Sigma_P$. Define $\mathcal{CS}^{r(t)}[p]$ to be the cardinality of the set $\{(p, t') | (p, t') \in \mathcal{CS}^{r(t)}\}$, that is, $\mathcal{CS}^{r(t)}[p]$ is the number of "copies" of the packet p in the $\mathcal{CS}$ channel at the point (r, t). We similarly define $\mathcal{CR}^{r(t)}[p]$. The following properties state the basic adversarial properties of the Environment.

The first property simply states that a channel is free to deliver any packet it has or can deliver a ϕ.

Property 4.1 *Let $\mathcal{P}$ be a sequence transmission protocol. For every $r \in \mathcal{R}_\mathcal{P}$ and $t \geq 0$, if the Sender (respectively, the Receiver) receives in state $r(t+1)$ then the following conditions hold:*

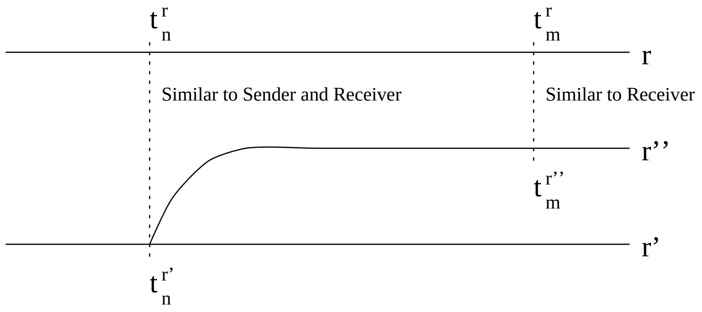

Figure 3: Description of runs r, r' and r'' in the proof of the Playback lemma.

(a). For every packet $p \in \Sigma_P$ such that $\mathcal{CS}^{r(t)}[p] > 0$ ($\mathcal{CR}^{r(t)}[p] > 0$) there is a run r' extending (r,t) such that the Sender (Receiver) receives packet p at point $(r', t+1)$.

(b). There is a run r' that extends (r,t) such that the Sender (Receiver) receives packet ϕ at point $(r', t+1)$.

The second property simply states that the Environment may select an input to be any value as long as it has not yet been read by the Sender.

Property 4.2 *Let $\mathcal{P}$ be a sequence transmission protocol. For any run $r \in \mathcal{R}_\mathcal{P}$, $t \geq 0$ and $n \geq 0$ such that the Sender has not read x_n^r at (r,t), and message $m \in \Sigma$, there is a run r' such that $x_n^{r'} = m$, $(r',t)\overset{S}{\sim}(r,t)$ and $(r',t)\overset{R}{\sim}(r,t)$.*

Because we must deal with lookahead greater than 0, we extend the notion of delivery sequence defined earlier. For a run $r \in \mathcal{R}_\mathcal{P}$, time t, and positive integer m, define $\sigma(r,t,m)$ to be the sequence of packets (including ϕ's) between time t and time t^r_{n-1+m}, where $t^r_{n-1} \leq t < t^r_n$. This is the sequence of packets which would cause the Receiver to write the next m messages.

Let $\texttt{packets}(\Upsilon)$ denote the multi-set of packets from Σ_P that make up any structure Υ, such as a delivery sequence, or a channel set ($\mathcal{CR}$ or $\mathcal{CS}$). Consider any sequence transmission protocol $\mathcal{P}$. For any run $r \in \mathcal{R}_\mathcal{P}$, time t, and $m \geq 1$, we say $\alpha \in (\Sigma_P \cup \{\phi\})^*$ can be *created from* $\mathcal{CR}^{r(t)}$ if $\texttt{packets}(\alpha) \subseteq \texttt{packets}(\mathcal{CR}^{r(t)})$.

4.2 Playback Lemma

The Playback lemma states that there is a limit to the information that captured packets can represent during the execution of a sequence transmission protocol with bounded lookahead. If the transmission cost and lookahead are known then for some type of packet the number captured by the channel must be bounded. Suppose that the sequence transmission protocol $\mathcal{P}$ has lookahead $h(n)$ and run $r \in \mathcal{R}_\mathcal{P}$ has transmission cost $f(n)$ from time t^r_n. If every member of Σ_P has $\sum_{i=n+1}^{n+h(n)+1} f(i)$ copies in the $\mathcal{CR}$ channel at time t^r_n then there are enough packets in the channel needed to create any sequence of packets that will cause the Receiver to write $h(n)+1$ messages. But the Sender has no knowledge of the $(h(n)+1)$-st message, thus an adversarial Environment can play back these packets, causing the Receiver to write the wrong message.

Lemma 4.1 (Playback) *Let $\mathcal{P}$ be any sequence transmission protocol with lookahead $h(n)$. Then, for any run $r \in \mathcal{R}_\mathcal{P}$ and $n \geq 1$ such that r has transmission cost $f(n)$ from t^r_n, it must be the case that for some packet $p \in \sigma(r, t^r_n, h(n)+1)$, $\mathcal{CR}^{r(t^r_n)}[p] < \sum_{i=n+1}^{n+h(n)+1} f(i)$.*

Proof: Assume for purposes of coming to a contradiction that the hypothesis of the lemma holds, but for all $p \in \sigma(r, t_n^r, h(n)+1)$, $\mathcal{CR}^{r(t_n^r)}[p] \geq \sum_{i=n+1}^{n+h(n)+1} f(i)$. Since $\mathcal{P}$ has lookahead $h(n)$, the Sender has not read $x_{n+h(n)+1}^r$ at time t_n^r and so, by property 4.2, there is a run r' and a time t' such that $(r, t_n^r) \overset{S}{\sim} (r', t')$, $(r, t_n^r) \overset{R}{\sim} (r', t')$ and $x_{n+h(n)+1}^r \neq x_{n+h(n)+1}^{r'}$. Since $(r, t_n^r) \overset{S}{\sim} (r', t')$ and $(r, t_n^r) \overset{R}{\sim} (r', t')$, the Sender and Receiver have sent and received the same packets, so $\mathcal{CR}^{r(t_n^r)} = \mathcal{CR}^{r'(t')}$. Furthermore, since $(r, t_n^r) \overset{R}{\sim} (r', t')$, the Receiver has written exactly the same outputs in both runs up to times t_n^r and t' respectively. Thus, $t' = t_n^{r'}$. Since r has transmission cost $f(n)$ from t_n^r, $|\sigma(r, t_n^r, h(n)+1)| \leq \sum_{i=n+1}^{n+h(n)+1} f(i)$ and so $\sigma(r, t_n^r, h(n)+1)$ can be created from $\mathcal{CR}^{r(t_n^r)}$. Since $\mathcal{CR}^{r(t_n^r)} = \mathcal{CR}^{r'(t_n^{r'})}$, $\sigma(r, t_n^r, h(n)+1)$ can also be created from $\mathcal{CR}^{r'(t_n^{r'})}$.

Let $\sigma(r, t_n^r, h(n)+1) = p_1 p_2 p_3 \ldots p_k$. Let r'' be a run which extends the point $(r', t_n^{r'})$ in which only the Receiver acts after time $t_n^{r'}$ and for $1 \leq i \leq k$, if the Receiver does an i-th receive after time $t_n^{r'}$ at point (r'', t_i) then it receives p_i at that point. Since $\sigma(r, t_n^r, h(n)+1)$ can be created from $\mathcal{CR}^{r'(t_n^{r'})}$ then by property 4.1 such a run r'' exists. Because $(r, t_n^r) \overset{R}{\sim} (r', t_n^{r'})$ and r'' extends $(r', t_n^{r'})$ then $(r, t_n^r) \overset{R}{\sim} (r'', t_n^{r''})$. Since, the Receiver makes at least k receives after time t_n^r in the run r, then it can do so after time $t_n^{r''}$ in the run r'' as well. Thus, we must have $(r, t_m^r) \overset{R}{\sim} (r'', t_m^{r''})$ where $m = n + h(n) + 1$. That is, from the Receiver's point of view the runs r and r'' are identical up to the time y_m is written. We have:

$$\begin{array}{rll} x_m^{r'} = x_m^{r''} & \text{since } r'' \text{ extends } (r', t_n^{r'}) \\ x_m^{r''} = y_m^{r''} & \text{by safety} \\ y_m^{r''} = y_m^r & \text{since } (r, t_m^r) \overset{R}{\sim} (r'', t_m^{r''}) \\ y_m^r = x_m^r & \text{by safety.} \end{array}$$

Thus, $x_m^{r'} = x_m^r$, which contradicts the choice of r', which was chosen so that $x_m^{r'} \neq x_m^r$. Figure 3 shows how the proof of the lemma works. ■

4.3 Capture Lemma

From the Playback lemma we know that the $\mathcal{CR}$ channel cannot have too many packets of each type. However, the Environment, being adversarial, can capture packets and keep them in the $\mathcal{CR}$ channel for a possible future playback attack. The Capture lemma shows that the Environment has the ability to capture useful packets in a timely fashion. A useful packet is one with not enough copies captured so far and in a timely fashion means that it can be captured while still potentially useful in a playback attack.

Lemma 4.2 (Capture) *Let $\mathcal{P}$ be an $(f(n), g(n), h(n))$-recoverable sequence transmission protocol. For any $n \geq 0$ and $r \in \mathcal{F_P}$, there is a run $r' \in \mathcal{F_P}$ extending (r, t_n^r) and a packet $p \in \Sigma_P$ such that $\mathcal{CR}^{r(t_n^r)}[p] < \sum_{i=n+1}^{N} f(i)$, $\mathcal{CR}^{r'(t_N^{r'})}[p] \geq \mathcal{CR}^{r(t_n^r)}[p] + 1$ and, for all $q \in \Sigma_P$ $\mathcal{CR}^{r'(t_N^{r'})}[q] \geq \mathcal{CR}^{r(t_n^r)}[q]$, where $N = n + g(n) + h(n-1+g(n))$.*

Proof: Let $\mathcal{P}$ be $(f(n), g(n), h(n))$-recoverable and let r be a run in $\mathcal{F_P}$. Let $M = n-1+g(n)$ and $N = M + h(M) + 1 = n + g(n) + h(n-1+g(n))$. The extension property guarantees the existence of $r'' \in \mathcal{F_P}$ which extends (r, t_n^r), is ideal from time t_n^r, and no packets sent on or before time t_n^r are ever delivered in run r'' after time t_n^r. Since $\mathcal{P}$ has ideal transmission cost $f(n)$ and recovery cost $g(n)$, r'' has transmission cost $f(n)$ from time $t_M^{r''}$.

By the Playback lemma, there is $p \in \sigma(r'', t_M^{r''}, h(M)+1)$ such that $\mathcal{CR}^{r''(t_M^{r''})}[p] < \sum_{i=M+1}^{N} f(i)$. Choose $t' \geq t_M^{r''}$ to be the smallest time such that for some t'', $(p, t'') \in \mathcal{CR}^{r''(t')}$

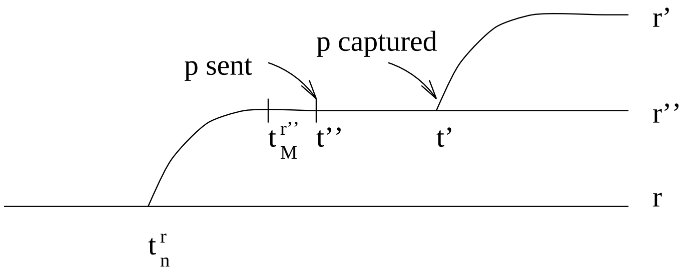

Figure 4: Description of runs r, r', and r'' in the proof of the Capture lemma.

and $(p, t'') \notin \mathcal{CR}^{r''(t'+1)}$. Thus, p is received by the Receiver at the point $(r'', t'+1)$. By the extension property there exists a run r' extending (r'', t') which is ideal from time t' and no packets sent on or before time t' are ever delivered in run r' after time t'. In particular, in the run r' the packet $(p, t'') \in \mathcal{CR}^{r'(t)}$ for all $t \geq t'$.

Since $\mathcal{CR}^{r''(t^{r''}_M)}[p] < \sum_{i=M+1}^{N} f(i)$, $n \leq M$, and no packets in $\mathcal{CR}^{r(t^r_n)}$ are delivered in run r'' after time t^r_n, then $\mathcal{CR}^{r(t^r_n)}[p] < \sum_{i=n+1}^{N} f(i)$.

We must have $t' < t^{r'}_N$ because $(r', t') \overset{R}{\sim} (r'', t')$ and $t' < t^{r''}_N$. Since r'' and r' were chosen to satisfy the extension property for times t^r_n and t' respectively and runs r and r'' respectively, then $r' \in \mathcal{F}_{\mathcal{P}}$, r' extends (r, t^r_n), and for all $q \in \Sigma_P$, $\mathcal{CR}^{r'(t^{r'}_N)}[q] \geq \mathcal{CR}^{r(t^r_n)}[q]$. In addition, we must have $t'' > t^r_n$ since any packet sent before time t^r_n is never delivered in the run r''. The run r' was chosen so as not to allow p to be received and (p, t'') to be removed from the $\mathcal{CR}$ channel. Thus, $\mathcal{CR}^{r'(t^{r'}_N)}[p] \geq \mathcal{CR}^{r(t^r_n)}[p] + 1$.

Figure 4 describes the proof of the Capture lemma. ■

4.4 Proof of Lower Bound for Recoverable Protocols

We now apply the Playback and Capture lemmas to prove the Lower Bound theorem.

Lower Bound Theorem(Theorem 4.1) *If $\mathcal{P}$ is a $f(n)$-recoverable sequence transmission protocol then $f(n) = \Omega_\infty(n)$.*

Proof: The Capture lemma guarantees that a useful packet can be captured within a period that depends on the recovery cost and the lookahead. If the recovery cost and lookahead are constant, then a useful packet can be captured every time a constant number of messages are written. That is, useful packets can be captured at a linear rate. This rate cannot allow the channel to capture enough useful packets so that the the Playback lemma would be violated. This forces $f(n)$ to be at least linear for infinitely many n.

Let $\mathcal{P}$ be a $(f(n), b, c)$-recoverable sequence transmission protocol where $b > 0$ and $c \geq 0$. Our goal is to show that $f(n) = \Omega_\infty(n)$.

Consider the run $r = \lim_{j \to \infty} r^j$ where $r^0 \in \mathcal{R}_{\mathcal{P}}$ is ideal from time 0 and for any run r^j, $j \geq 0$, r^{j+1} is a run which extends (r^j, u^j), where u_j is defined by $u^j = t^{r^j}_{(b+c)j}$, and which "captures" another packet in the manner guaranteed by the Capture lemma. In our application of the Capture lemma, $N = n+b+c$. Thus, r^{j+1} has the property that there is a packet $p \in \Sigma_P$ such that $\mathcal{CR}^{r^j(u^j)}[p] < \sum_{i=j(b+c)+1}^{(j+1)(b+c)} f(i)$, $\mathcal{CR}^{r^{j+1}(u^{j+1})}[p] \geq \mathcal{CR}^{r^j(u^j)}[p] + 1$ and, for every packet $q \in \Sigma_P$, $\mathcal{CR}^{r^{j+1}(u^{j+1})}[q] \geq \mathcal{CR}^{r^j(u^j)}[q]$ (see figure 5).

Now assume that $f(n) \neq \Omega_\infty(n)$, that is, for every constant k, there is an n_k such that

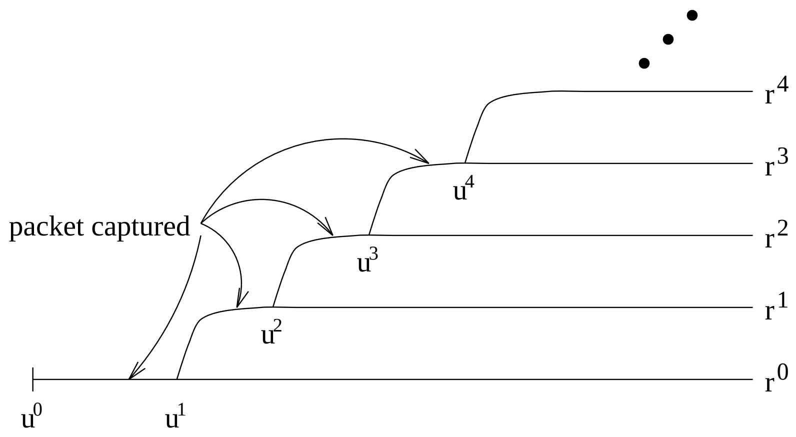

Figure 5: Description of the construction of the run r in the proof of the Lower Bound theorem.

for every $n \geq n_k$, $f(n) \leq kn$ and consider the case where

$$k = \frac{1}{3(b+c)^2|\Sigma_P|}.$$

Choose I to be the smallest such that $I(b+c) \geq n_k$ and let $J = 2I + 1$. Define:

$$K = \max\{\sum_{i=j(b+c)+1}^{(j+1)(b+c)} f(i) \ : \ I \leq j < J\}.$$

Since $f(n) \leq kn$ for all $n \geq I(b+c)$ we have:

$$\begin{aligned} K &\leq \sum_{i=(J-1)(b+c)+1}^{J(b+c)} ki \\ &\leq kJ(b+c)^2 \\ &\leq k3I(b+c)^2 \\ &\leq \frac{I}{|\Sigma_P|}. \end{aligned} \tag{1}$$

From the construction of r, for every j such that $I \leq j < J$, there is a time, t^j, such that $u^j \leq t^j < u^{j+1}$ and for some $p^j \in \Sigma_P$, the pair $(p^j, t^j) \in \mathcal{CR}^{r(u^{j+1})} - \mathcal{CR}^{r(u^j)}$ and

$$\mathcal{CR}^{r(u^j)}[p^j] < \sum_{i=j(b+c)+1}^{(j+1)(b+c)} f(i) \leq K.$$

Thus, for each $p \in \Sigma_P$ there are less than K distinct j's such that $p = p^j$. There are exactly $J - I$ pairs (p^j, t^j) such that $I \leq j < J$. Thus, we have $J - I \leq |\Sigma_P|K$. By inequality 1, $|\Sigma_P|K \leq I$. Hence, $J - I \leq I$ which contradicts our choice of $J = 2I + 1$. ■

4.5 Lower Bound for Weakly-Bounded Protocols

A modification of the proof of the Lower Bound theorem gives a lower bound for weakly-bounded protocols. The proof then becomes very similar to the proof that there is no k-weakly-bounded protocol.

Theorem 4.2 *If $\mathcal{P}$ is a $f(n)$-weakly-bounded sequence transmission protocol then $f(n) = \Omega_\infty(n)$.*

Proof: We just point out the differences in this proof from the proof of the Lower Bound theorem 4.1. First, the Playback lemma can be simplified to state that for any run $r \in \mathcal{R}_\mathcal{P}$ and $n \geq 1$ such that r has transmission cost $f(n)$ from t_n^r, it must be the case that for some packet $p \in \sigma(r, t_n^r)$, $\mathcal{CR}^{r(t_n^r)}[p] < f(n+1)$. Thus, there cannot be too many copies of some packet in the $\mathcal{CR}$ channel at times when messages are written. Second, the Capture lemma can be simplified to state that for any $r \in \mathcal{F}_\mathcal{P}$, there exists a run r' extending (r, t_n^r) and a packet $p \in \Sigma_P$ such that $\mathcal{CR}^{r(t_n^r)}[p] < f(n+1)$, $\mathcal{CR}^{r'(t_{n+1}^{r'})}[p] \geq \mathcal{CR}^{r(t_n^r)}[p] + 1$ and, for every packet $q \in \Sigma_P$ such that $q \neq p$, $\mathcal{CR}^{r'(t_{n+1}^{r'})}[q] \geq \mathcal{CR}^{r(t_n^r)}[q]$. Thus, for every message written a "useful" packet can be captured. Finally, in the proof in section 4.4, k can be chosen to be $\frac{1}{3|\Sigma_P|}$. ■

5 A $2n$-Recoverable Protocol

In this section we present the sequence transmission protocol $\mathcal{P}_{\text{lin}}$ which is $2n$-recoverable. This protocol demonstrates the tightness of our lower bound in section 4, that is, a protocol with linear ideal transmission cost and constant recovery cost and lookahead. We begin with an informal description of the protocol. We then give the complete description of $\mathcal{P}_{\text{lin}}$, prove its correctness, and that it has the behavior we claim. Because $\mathcal{P}_{\text{lin}}$ is a stop-and-wait protocol we define the ideal runs of $\mathcal{P}_{\text{lin}}$ as stop-and-wait ideal runs. We show that with this definition of ideal runs $\mathcal{P}_{\text{lin}}$ has the desired recoverable behavior. In addition we prove that the necessary extension property holds for $\mathcal{P}_{\text{lin}}$ and its stop-and-wait ideal runs.

As mentioned in the introduction, the behavior of recoverable protocols must be bounded when conditions are ideal in some sense, but when conditions are less than ideal, it is sufficient that such protocols just make progress. The protocol we present reflects this behavior in that it has two modes of operation, *normal mode* and *recover mode*. Normal mode is designed so that, when conditions are ideal, the cost of sending a message depends only on the message's position in the input sequence. Recover mode is not concerned with cost, its sole purpose is to make sure that progress is made. To distinguish packets sent in the two modes, the set of packets is partitioned into two *classes* of packets: *normal* packets and *recovery* packets.

The protocol is *receiver-driven*, that is, everything the Sender does is dictated by what it receives from the Receiver and it is the Receiver that decides when to move to the next message in the input, or when to switch modes. The decision as to when to switch modes is based on what the Receiver perceives the channel behavior to be. It will always request delivery of a message via normal mode, but if it decides it is not making progress, it will then switch to recover mode.

The operation of both modes are based on the observation used in the unbounded protocol [AAF$^+$91, AFWZ89], that is, if an upper bound on the number of undelivered packets is known by a process, then the process can avoid the confusion of out-of-order packets by requiring the delivery of more copies of a packet than there are total undelivered packets. The problem is reduced to somehow keeping the number of undelivered normal packets bounded.

Consider the following behavior for normal mode. The Receiver sends a normal packet which the Sender interprets as a request for the current message. The Sender replies to this request with a normal packet containing the current message. If the Receiver gets the reply it is expecting, it will send another request. The Receiver will continue with the request-reply exchanges until it receives "enough" correct replies. If, at some time, the Receiver does not get the reply it is expecting, it goes to recover mode.

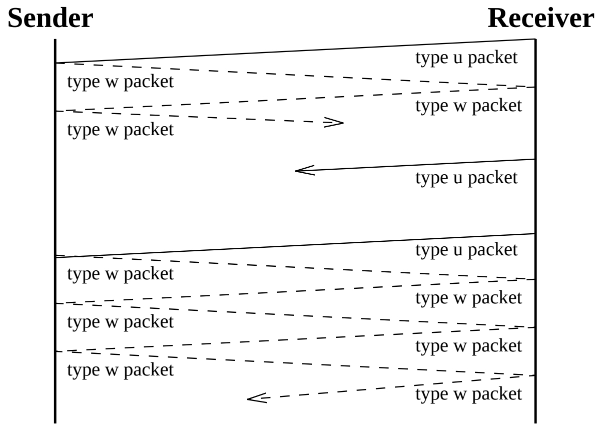

Figure 6: Pattern of packet exchange for type u and w packets.

Notice that this behavior is only possible if no packets get lost. Furthermore, if the Receiver goes into recover mode, then, either the packet the Receiver just sent is lost, or still in transit, or the Sender's reply to that packet is lost or in transit. This means that the Receiver goes into recover mode as a result of one normal packet not being delivered. Since the Receiver goes into recover mode every time a normal packet is not delivered, and the Receiver is in normal mode exactly once for every message, there can not be more undelivered normal packets than there are messages sent. So, if the Receiver has just gone into normal mode for the message x_n, then there can not be more than $n-1$ undelivered normal packets in the channel and so, if the Receiver achieves n successful request-reply exchanges, at least one of the replies it received is not an old packet, that is, n is "enough" replies.

For a protocol to have lookahead 0, the Sender should not read the next message until the Receiver has written the previous one. This synchronization is done by further partitioning the packets into those used for messages whose position in the input sequence is even and those whose position is odd. This can be represented as including a bit in the packet header, which we call the *sync count*, corresponding to the message number modulo 2, in the same manner as in the Alternating Bit Protocol [BSW69].

When the Receiver has written a message (whether in normal or recover mode), it begins requesting the next message. The Sender detects the next message is being requested by the change in the sync count of the packets it is receiving. However, the Sender cannot just immediately read the next message when it gets a packet with the sync count of the next message, since that packet may be an old request that has only just been delivered. Instead the Sender must wait until it receives "enough" requests with the sync count of the next message. By the same argument as for the Receiver, if the Sender is currently on message x_{n-1}, there cannot be more than $n-1$ undelivered normal requests and so n requests is "enough."

Because the Receiver expects n replies for x_n, but the Sender requires n requests for x_n before it reads it (and so starts sending), the Receiver will not write x_n until it receives $2n$ packets, that is, the ideal transmission cost is $2n$. Furthermore, since the Receiver goes into normal mode every time it writes a message, it is always possible for the next message to be written after receiving a bounded number of packets. Thus, the recovery cost is 1.

Normal mode bounds the number of undelivered normal packets by simply not sending more normal packets when the Receiver does not get a reply. As a side-effect, normal mode also provides enough information to determine an upper bound on the number of undelivered normal packets. This kind of behavior is not possible for recover mode. Since progress must be made, the Receiver cannot simply give up when it does not get the reply it expects.

In order for the Sender and Receiver to maintain a bound on the number of undelivered recovery packets two types of recovery packets are used in the protocol, type u and w recovery packets. The Sender maintains a count which bounds the number of undelivered type w recovery packets and the Receiver maintains a count which bounds the number of undelivered recovery packets regardless of type. It may be helpful for the reader to think of type w packets as *wanted* packets because it it the receipt of type w packets that causes the Sender and Receiver to make progress. Type u packets are *unwanted* because when they are sent or received, counts which bound the number of undelivered packets are increased. As the counts grow so do the number of packets needed for delivery of messages in recover mode.

To explain further, since our protocol is receiver-driven each packet sent by the Receiver is a request. A type u recovery packet is sent by the Receiver when its previous request is not met, that is, when the Receiver receives a ϕ instead of a packet sent by the Sender. Thus, when the Receiver sends a type u recovery packet it should increase its count which bounds the number of undelivered packets by one, since its previous request may have been captured in the $\mathcal{CS}$ channel or its previous request may have been delivered and the Sender's response been captured by the $\mathcal{CR}$ channel.

The Sender only sends type w recovery packets. In a symmetric way, a type w recovery packet, if received by the Receiver, should trigger a type w recovery packet from the Receiver. If instead, the Sender receives a type u recovery packet, then either its previous type w recovery packet may have been captured by the $\mathcal{CR}$ channel or its previous type w recovery packet was delivered and the Receiver's response of a type w recovery packet was captured by the $\mathcal{CS}$ channel. Thus, the Sender should increase its count which bounds the number of undelivered type w packets by one when it receives a type u recovery packet. There is no way for the Sender to detect undelivered type u recovery packets. Thus, the Sender's count of undelivered packets only bounds the number of undelivered type w recovery packets. As a consequence, the Sender should only consider type w recovery packets as requests to move on to the next input message. This mechanism is described in figure 6.

Normal mode of the protocol also uses type u and w packets. The initial request by the Receiver for a new message is always a type u normal packet, since it is not generally triggered by the receipt of type w normal packet. Otherwise, each type w normal packet sent by the Receiver is triggered by the receipt of a normal packet from the Sender. If the Receiver receives a ϕ instead of a normal packet then it immediately switches to recover mode. Thus, the relationship between the number of undelivered normal packets and the number of type u normal packets sent is similar to the same relationship for recovery packets. This relationship is described in lemma 5.1 on page 20.

5.1 The Protocol $\mathcal{P}_{\text{lin}}$

The protocol $\mathcal{P}_{\text{lin}}$ is shown in figures 7 and 8. The variable i is the index of the current message being sent by the Sender and j is the index of the message being requested by the Receiver. The Sender's count of undelivered type w recovery packets is u_S and the Receiver's count of undelivered recovery packets is u_R . The variables d_S and d_R are increments in these counts, which are eventually added into u_S and u_R, respectively. The Sender uses *nReq* (respectively *rReq*) to record the number of normal (recovery) request packets it has received. The Receiver uses the same identifiers to record how many replies it has received, except that, because it must distinguish between replies containing different messages, the identifiers refer to arrays.

Packets are represented by a tuple (C, v, m, s) where C is the packet class (N for normal or R for recovery), v is the packet type (type w or type u), m is a message, and s is the sync count (0 or 1) . A "$*$" indicates that the value is unspecified. The Receiver's protocol

SENDER

Send

```
  i, u_S, d_S ← 0
  REPEAT
    inc(i)
    X[i] ← read
    [d_S] ← SendMsg(i, u_S)
    u_S ← u_S + d_S
  UNTIL forever
```

SendMsg(i, u_S) RETURNS $[d_S]$

```
  nReq, rReq, d_S ← 0
  REPEAT
    pkt ← receive
    CASE
      ∘ pkt = (N, u, ∗, sc)
        send(N, w, X[i], i mod 2)
      ∘ pkt = (R, u, ∗, sc)
        send(R, w, X[i], i mod 2)
        inc(d_S)
      ∘ pkt = (N, w, ∗, sc)
        IF sc = i + 1 mod 2 THEN
          inc(nReq)
        ENDIF
        send(N, w, X[i], i mod 2)
      ∘ pkt = (R, w, ∗, sc)
        IF sc = i + 1 mod 2 THEN
          inc(rReq)
        ENDIF
        send(R, w, X[i], i mod 2)
    END CASE {ignore ϕ}
  UNTIL nReq ≥ i ∨ rReq > u_S
```

Figure 7: The Sender's protocol for $\mathcal{P}_{\text{lin}}$

RECEIVER

```
Receive
  j, u_R, d_R ← 0
  REPEAT
    inc(j)
    IF NormalMode(j) THEN
      [d_R] ← RecoverMode(j, u_R)
      u_R ← u_R + d_R
    ENDIF
    write(Y[j])
  UNTIL forever

RecoverMode(j, u_R) RETURNS [d_R]
  ∀x ∈ Σ, rReq[x], d_R ← 0
  send(R, u, ∗, j mod 2)
  inc(d_R)
  REPEAT
    pkt ← receive
    CASE
      ∘ pkt = (R, w, msg, sc)
        IF sc = j mod 2 THEN
          inc(rReq[msg])
        ENDIF
        IF rReq[msg] ≤ u_R THEN
          send(R, w, ∗, j mod 2)
        ENDIF
      ∘ pkt = φ
        send(R, u, ∗, j mod 2)
        inc(d_R)
    END CASE {ignore normal packets}
  UNTIL rReq[msg] > u_R
  Y[j] ← msg

NormalMode(j) RETURNS [failure]
  ∀x ∈ Σ, nReq[x], failure ← 0
  send(N, u, ∗, j mod 2)
  REPEAT
    pkt ← receive
    CASE
      ∘ pkt = (N, w, msg, sc)
        IF sc = j mod 2 THEN
          inc(nReq[msg])
        ENDIF
        IF nReq[msg] < j THEN
          send(N, w, ∗, j mod 2)
        ENDIF
      ∘ pkt = φ
        failure ← 1
    END CASE {ignore recovery packets}
  UNTIL nReq[msg] ≥ j ∨ failure = 1
  IF failure = 0 THEN
    Y[j] ← msg
  ENDIF
```

Figure 8: The Receiver's protocol for $\mathcal{P}_{\text{lin}}$

clearly distinguishes between normal and recover mode however the Sender's protocol does not. This is because $\mathcal{P}_{\text{lin}}$ is receiver-driven, so it is the Receiver that makes the decisions as to what mode to be in and when to advance to the next message; the Sender just replies to any packets it receives. The reply the Sender sends depends on the packet it receives and the message it has most recently read.

Assuming our definition of stop-and-wait ideal runs, we now state our Upper Bound theorem.

Theorem 5.1 (Upper Bound) *$\mathcal{P}_{\text{lin}}$ is a $(2n, 1, 0)$-recoverable sequence transmission protocol.*

Because $\mathcal{P}_{\text{lin}}$ is $(2n, 1, 0)$-recoverable then it is also $2n$-weakly-bounded.

Theorem 5.2 *$\mathcal{P}_{\text{lin}}$ is a $2n$-weakly-bounded sequence transmission protocol.*

The remainder of this section is devoted to the proof of the Upper Bound theorem. We will use our runs model notation in the proof. In the sequel we will use $\mathcal{R}_{\text{lin}}$ to denote the set of runs consistent with $\mathcal{P}_{\text{lin}}$, and $\mathcal{F}_{\text{lin}}$ to denote the fair runs of $\mathcal{P}_{\text{lin}}$. Also, we will use $\mathbf{U}(C, v, r, t)$ to represent the number of undelivered class C, type v packets at time t in run r and $\texttt{Sent}(P, C, v, r, t)$ (respectively $\texttt{Rec}(P, C, v, r, t)$) to denote the number of packets of class C and type v sent (received) by process P in r at t. The process P is either $\mathbf{S}$ for the Sender or $\mathbf{R}$ for the Receiver. Recall that an action occurs at (r, t) means that protocol proceeded from state $r(t-1)$ to $r(t)$ because of the action. The value of a variable v in a global state $r(t)$ is denoted by $v^{r(t)}$.

Our first lemma describes basic invariants of the protocol, that is, the number of undelivered packets and the number of undelivered w packets at appropriate times are bounded by the values of variables maintained by the Sender and Receiver.

Lemma 5.1 (Invariant) *Let r be any run in $\mathcal{R}_{\text{lin}}$ and consider $t \geq 0$.*

1. *If* **NormalMode** *is called at (r, t) then $\mathbf{U}(N, u, r, t) + \mathbf{U}(N, w, r, t) \leq j^{r(t)} - 1$.*
2. *If* **RecoverMode** *is called at (r, t) then $\mathbf{U}(R, u, r, t) + \mathbf{U}(R, w, r, t) \leq u_R^{r(t)}$.*
3. *If* **SendMsg** *is called at (r, t) then*
   (a) $\mathbf{U}(N, w, r, t) \leq j^{r(t)}$ and
   (b) $\mathbf{U}(R, w, r, t) \leq u_S^{r(t)}$.

Proof:

1. Let **NormalMode** be called at (r, t). The number of undelivered normal packets is just the difference between the number sent and received in both directions.

$$\begin{aligned}
\mathbf{U}(N, u, r, t) + \mathbf{U}(N, w, r, t) \quad = \quad & \texttt{Sent}(\mathbf{S}, N, w, r, t) - \texttt{Rec}(\mathbf{R}, N, w, r, t) + \\
& \texttt{Sent}(\mathbf{R}, N, w, r, t) - \texttt{Rec}(\mathbf{S}, N, w, r, t) + \\
& \texttt{Sent}(\mathbf{S}, N, u, r, t) - \texttt{Rec}(\mathbf{R}, N, u, r, t) + \\
& \texttt{Sent}(\mathbf{R}, N, u, r, t) - \texttt{Rec}(\mathbf{S}, N, u, r, t).
\end{aligned}$$

First note that the Sender does not send any type u packets, that is $\texttt{Sent}(\mathbf{S}, N, u, r, t) = \texttt{Rec}(\mathbf{R}, N, u, r, t) = 0$. Also, the protocol is such that the Sender sends a type w normal packet every time it receives a normal packet, that is, $\texttt{Sent}(\mathbf{S}, N, w, r, t) \leq \texttt{Rec}(\mathbf{S}, N, u, r, t) + \texttt{Rec}(\mathbf{S}, N, w, r, t)$. Thus,

$$\mathbf{U}(N,u,r,t)+\mathbf{U}(N,w,r,t) \leq \texttt{Sent}(\mathbf{R},N,w,r,t)-\texttt{Rec}(\mathbf{R},N,w,r,t)+ \texttt{Sent}(\mathbf{R},N,u,r,t).$$

The Receiver sends a type w normal packet only after it has received a type w normal packet, that is, $\texttt{Sent}(\mathbf{R},N,w,r,t) \leq \texttt{Rec}(\mathbf{R},N,w,r,t)$. This means that, when **NormalMode** is called at (r,t),

$$\mathbf{U}(N,u,r,t)+\mathbf{U}(N,w,r,t) \leq \texttt{Sent}(\mathbf{R},N,u,r,t).$$

Exactly one u normal packet is sent by the Receiver each time **NormalMode** is called. The value of j is incremented just before a call to **NormalMode** and the type u normal packet sent just after the call. Thus, if **NormalMode** is called at (r,t) then $\texttt{Sent}(\mathbf{R},N,u,r,t) = j^{r(t)}-1$. It follows that, $\mathbf{U}(N,u,r,t)+\mathbf{U}(N,w,r,t) \leq j^{r(t)}-1$.

2. Let **RecoverMode** be called at (r,t). This argument is similar, by substituting recovery packets for normal packets, to the argument above up to the point of showing that $\mathbf{U}(R,u,r,t)+\mathbf{U}(R,w,r,t) \leq \texttt{Sent}(\mathbf{R},R,u,r,t)$. Each time that the Receiver sends a type u recovery packet it increments the counter d_R. Just before a call to **NormalMode** the value of the counter d_R is added into u_R. No more type u recovery packets are sent until **RecoverMode** is called again. Thus, when **RecoverMode** is called the total number of type u recovery packets sent by the Receiver so far equals the value of u_R. That is, if **RecoverMode** is called at (r,t) then $\texttt{Sent}(\mathbf{R},R,u,r,t) = u_R^{r(t)}$. It follows that, $\mathbf{U}(R,u,r,t)+\mathbf{U}(R,w,r,t) \leq u_R^{r(t)}$.

3. Let **SendMsg** be called at (r,t).

   (a) The number of undelivered type w normal packets is just the difference between the number sent and received in both directions. Stated precisely,

   $$\mathbf{U}(N,w,r,t) = \texttt{Sent}(\mathbf{S},N,w,r,t)-\texttt{Rec}(\mathbf{R},N,w,r,t)+ \texttt{Sent}(\mathbf{R},N,w,r,t)-\texttt{Rec}(\mathbf{S},N,w,r,t).$$

   The protocol is such that the Sender, within a call to **SendMsg**, sends a normal packet if and only if it receives a normal packet. Thus, when **SendMsg** is called at (r,t), $\texttt{Sent}(\mathbf{S},N,w,r,t) = \texttt{Rec}(\mathbf{S},N,u,r,t)+\texttt{Rec}(\mathbf{S},N,w,r,t)$. Hence,

   $$\mathbf{U}(N,w,r,t) = \texttt{Rec}(\mathbf{S},N,u,r,t)-\texttt{Rec}(\mathbf{R},N,w,r,t)+ \texttt{Sent}(\mathbf{R},N,w,r,t).$$

   The Receiver sends a type w packet only after it has received a type w packet, that is, $\texttt{Sent}(\mathbf{R},N,w,r,t) \leq \texttt{Rec}(\mathbf{R},N,w,r,t)$. Thus, when **SendMsg** is called at (r,t),

   $$\mathbf{U}(N,w,r,t) \leq \texttt{Rec}(\mathbf{S},N,u,r,t).$$

   The only times that the Receiver sends type u normal packets is just after a call to **NormalMode** and just before **NormalMode** is called the value of j is incremented. Thus, $\texttt{Rec}(\mathbf{S},N,u,r,t) \leq \texttt{Sent}(\mathbf{R},N,u,r,t) \leq j^{r(t)}$, and consequently $\mathbf{U}(N,w,r,t) \leq j^{r(t)}$.

   (b) This argument is similar, by substituting recovery packets for normal packets, to the argument above up to the point of showing that $\mathbf{U}(R,w,r,t) \leq \texttt{Rec}(\mathbf{S},R,u,r,t)$. Each time that the Sender receives a type u recovery packet it increments the

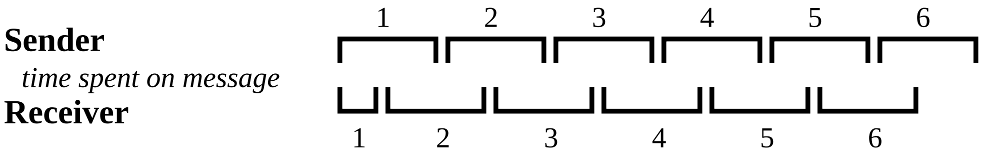

Figure 9: Description of the general "zipper property."

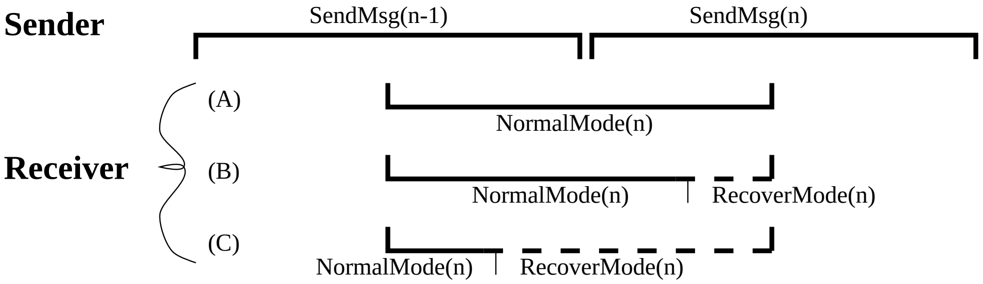

Figure 10: Description of Zipper Property of the protocol $\mathcal{P}_{\text{lin}}$.

counter d_S. Just before a call to **SendMsg** the value of the counter d_S is added into u_S. Thus, when **SendMsg** is called the total number of type u recovery packets received by the Sender so far equals the value of u_S. That is, if **SendMsg** is called at (r,t) then $\mathtt{Rec}(\mathbf{S}, R, u, r, t) = u_S^{r(t)}$. It follows that, $\mathbf{U}(R, w, r, t) \leq u_S^{r(t)}$.

■

The next step is to show that $\mathcal{P}_{\text{lin}}$ has what we call the "zipper" property exhibited by all stop-and-wait, receiver-driven communications protocols. This property, illustrated in figure 9, represents the fact that, at any time, either the Sender is sending the current message to the Receiver, or the Receiver is ready for the next message and is trying to convince the Sender to advance to the next message. In the case of $\mathcal{P}_{\text{lin}}$, there are three possibilities, depending on the behavior of the channel. Figure 10 shows the different cases. In case (A) **RecoverMode** is not called. In case (B) **RecoverMode** is called after the Sender has moved on to the next message. In case (C) **RecoverMode** is called before the Sender has moved on the the next message.

This behavior is formalized in the following lemma:

Lemma 5.2 (Zipper Property) *Let r be any run in $\mathcal{R}_{\text{lin}}$, $t \geq 0$, and $n \geq 2$.*

1. *If* **NormalMode** *is called at (r,t) and $j^{r(t)} = n$, then the Sender is in the call to* **SendMsg** *with $i^{r(t)} = n-1$ at (r,t).*

2. *If* **RecoverMode** *is called at (r,t) and $j^{r(t)} = n$, then at (r,t) the Sender is in one of or between the calls to* **SendMsg** *with $i^{r(t)} = n-1$ and* **SendMsg** *with $i^{r(t)} = n$.*

3. *If* **SendMsg** *is called at (r,t) and $i^{r(t)} = n$, then at (r,t) the Receiver is in one of or between the calls to* **NormalMode** *with $j^{r(t)} = n$ and* **RecoverMode** *with $j^{r(t)} = n$.*

Proof: The proof is by induction on n.
Base case: $n = 2$.

1. When **NormalMode** is called with $j = 2$ the Receiver must have received a packet with sync count 1 to complete the transmission of the first message, which means the

Sender must have called **SendMsg** with $i = 1$. In order for the Sender to terminate the call to **SendMsg** with $i = 1$ the Sender must receive a w packet with sync count 0. Since the Receiver has sent no type w packets with sync count 0, the Sender's call to **SendMsg** with $i = 1$ has not terminated.

2. When **RecoverMode** is called with $j = 2$ at (r, t), by the same argument as above, the Sender must have already called **SendMsg** with $i = 1$. If the Sender has not called **SendMsg** with $i = 2$ by time t then we are done. So, assume that **SendMsg** was called with $i = 2$ at time $t' < t$. If this call to **SendMsg** was completed before time t, then either the Sender received 2 packets of the form $(N, w, *, 1)$ or $u_S^{r(t')} + 1$ packets of the form $(R, w, *, 1)$ before time t.

   Let t'' be the time at which **NormalMode** is called with $j = 2$. By part 1 of the base case $t'' < t'$. The Receiver sends no packets (normal or recovery) with sync count 1 between times t'' and t. By part 1 of the Invariant lemma, there is at most 1 undelivered normal packet at time t'' and by part 3b of the Invariant lemma there are at most $u_S^{r(t')}$ undelivered type w recovery packets at time t'. Thus, the call to **SendMsg** with $i = 2$ could not have completed before time t. Hence, the Sender is still in its call to **SendMsg** with $i = 2$ at (r, t).

3. When **SendMsg** is called with $i = 2$, by part 1 of the base case, the Receiver must have previously called **NormalMode** with $j = 2$ and may have already called **RecoverMode** with $j = 2$. The only way the call to **NormalMode** or any subsequent call to **RecoverMode** with $j = 2$ can complete is for the Receiver to receive packets with sync count 0. However this is not possible since the Sender has not sent any such packets before **SendMsg** is called with $i = 2$.

Inductive step: Assume the claim is true for all k with $2 \leq k \leq n$ and consider the case $n + 1$.

1. Let **NormalMode** be called at (r, t) with $j^{r(t)} = n + 1$. Let t' be the time at which **SendMsg** is called with $i = n$. By part 3 of the induction hypothesis, $t' < t$. For this call to **SendMsg** to complete, the Sender must receive either n type w normal packets with sync count $n + 1 \bmod 2$ or $u_S^{r(t')} + 1$ type w recovery packets with sync count $n + 1 \bmod 2$.

   Let t'' be the time at which **NormalMode** is called with $j = n$. By part 1 of the induction hypothesis, $t'' < t'$. The Receiver sends no packets (normal or recovery) with sync count $n+1 \bmod 2$ between times t'' and t. By part 1 of the Invariant lemma, there are at most $n - 1$ undelivered normal packets at time t'' and by part 3b of the Invariant lemma there are at most $u_S^{r(t')}$ undelivered type w recovery packets at time t'. Thus, the call to **SendMsg** with $i = n$ could not have completed before time t. Hence, the Sender is in its call to **SendMsg** with $i^{r(t)} = n$ at (r, t).

2. This case is similar to the same part in the base case. Suppose **RecoverMode** is called with $j = n + 1$ at (r, t). The Receiver must have called **NormalMode** with $j = n + 1$ previously. Thus, by part 1 of the inductive step, the Sender must have already called **SendMsg** with $i = n$ by time t. If the Sender has not called **SendMsg** with $i = n + 1$ by time t then we are done. So, assume that **SendMsg** was called with $i = n + 1$ at time $t' < t$. If this call to **SendMsg** was completed before time t, then either the Sender received $n + 1$ type w normal packets with sync count $n + 2 \bmod 2$ or $u_S^{r(t')} + 1$ type w recovery packets before time t with sync count $n + 2 \bmod 2$.

Let t'' be the time at which **NormalMode** is called with $j = n+1$. By part 1 just shown for $n+1$, $t'' < t'$. The Receiver sends no packets (normal or recovery) with sync count $n+2 \bmod 2$ between times t'' and t. By part 1 of the Invariant lemma, there are at most n undelivered normal packets at time t'' and by part 3b of the Invariant lemma there are at most $u_S^{r(t')}$ undelivered type w recovery packets at time t'. Thus, the call to **SendMsg** with $i = n+1$ could not have completed before time t. Hence, the Sender is still in its call to **SendMsg** with $i = n+1$ at (r, t).

3. Let **SendMsg** with $i^{r(t)} = n+1$ be called at (r, t). By part 1 just shown for $n+1$, the Receiver must have previously called **NormalMode** with $j = n+1$. In addition, **RecoverMode** with $j = n+1$ may also have been called prior to time t. Let $t' < t$ be such that **NormalMode** with $j = n+1$ is called at (r, t').

   Let t'' be the time at which **SendMsg** is called with $j = n$. By part 3 of the induction hypothesis, $t'' < t'$ and $j^{r(t'')} = n$. By part 3a of the Invariant lemma, there are at most n undelivered type w normal packets at time t''. Since the Sender does not send any w packet with sync count $n+1 \bmod 2$ during its call to **SendMsg** with $i = n$, the call to **NormalMode** with $i = n+1$ cannot terminate before time t by the Sender having received $n+1$ type w normal packets.

   Suppose **RecoverMode** with $j = n+1$ is called at time t'''. If $t''' > t$ then we are done. Otherwise, we must have $t' < t''' < t$. By part 2 just shown for $n+1$, between the time t''' and t the Sender is either in **SendMsg** with $i = n$ or between the termination of **SendMsg** with $i = n$ and the call to **SendMsg** with $i = n+1$. During this period the Sender sends no packets with sync count $n+1 \bmod 2$. By part 2 of the Invariant lemma, the number of undelivered recovery packets is bounded by $u_R^{r(t''')}$. Thus, the number of recovery packets with sync count $n+1 \bmod 2$ which the Receiver could receive before time t is also bounded by $u_R^{r(t''')}$. Hence, the Receiver must still be in its call to **RecoverMode** with $j = n+1$ at time t.

■

5.1.1 Safety

We can now show that $\mathcal{P}_{\text{lin}}$ satisfies safety, that is:

Lemma 5.3 (Safety) *For any $r \in \mathcal{R}_{\text{lin}}$, $t \geq 0$, and $n \geq 1$, if $t_n^r \leq t$ then $y_n^{r(t)} = x_n^{r(t)}$.*

Proof: Let r be a run of the protocol $\mathcal{P}_{\text{lin}}$. Let y_n^r be written at point (r, t_n^r). It suffices to show $y_n^r = x_n^r$.

Let us first consider the case $n = 1$. In order for the Receiver to write its first message it must receive one type w packet (either normal or recovery) with sync count 1. By part 3 of the Zipper Property, the Sender could not have called **SendMsg** with $i = 2$ by time t_1^r. Thus, the only packets that the Receiver could have received before time t_1^r have the form $(*, *, x_1^r, 1)$. Thus, the Receiver must have written x_1^r at time t_1^r.

Let us now consider the case $n \geq 2$. The time t_n^r must occur just after **NormalMode** or **RecoverMode** with $j = n$ has completed and before **NormalMode** with $j = n+1$ is called. There are now two cases to consider.

Case 1: The Receiver calls **NormalMode** with $j = n$ at time $t < t_n^r$ and does not call **RecoverMode** with $j = n$ (that is, y_n^r is written on completion of the call to **NormalMode**).

By part 1 of the Invariant lemma there are not more than $n-1$ undelivered normal packets at time t. However, the Receiver receives n copies of packets of the form $(N, w, y_n^r, n \bmod 2)$

in the period t to t_n^r, thus at least one of these packets must have been sent by the Sender after time t. By the Zipper property, during the period from t to t_n^r the Sender is in one of or between calls to **SendMsg** with $i = n - 1$ or **SendMsg** with $i = n$. During this period the only packets the Sender sends with sync count $n \bmod 2$ are of the form $(N, w, x_n^r, n \bmod 2)$. Thus, we conclude $y_n^r = x_n^r$.

Case 2: The Receiver calls **RecoverMode** with $j = n$ at time $t < t_n^r$.

By part 2 of the Invariant lemma there are not more than $u_R^{r(t)}$ undelivered recovery packets at time t. However, the Receiver receives $u_R^{r(t)} + 1$ copies of packets of the form $(R, w, y_n^r, n \bmod 2)$ in the period t to t_n^r. By the Zipper Property, the only packets the Sender sends with sync count $n \bmod 2$ during this period are of the form $(R, w, x_n^r, n \bmod 2)$. Thus, we conclude $y_n^r = x_n^r$. ■

5.1.2 Liveness

We now show that $\mathcal{P}_{\text{lin}}$ satisfies liveness, that is:

Lemma 5.4 (Liveness) *For any fair run, $r \in \mathcal{F}_{\text{lin}}$, and $n \geq 1$, $t_n^r < \infty$.*

Proof: Assume that claim is false. That is, there is a run $r \in \mathcal{F}_{\text{lin}}$ and $n \geq 1$ such that $t_n^r = \infty$. We can assume without loss of generality that n is the smallest with this property. Let **NormalMode** with $j = n$ be called at (r, t). The Receiver never calls **NormalMode** with $j = n + 1$. Since the run r is fair both the Sender and Receiver are each scheduled to run infinitely often.

If $n > 1$, then by the Zipper Property the Sender is in a call to **SendMsg** with $i = n - 1$. We first argue that the call to **SendMsg** with $i = n - 1$ must terminate.

Suppose the call to **NormalMode** with $j = n$ executes for infinitely many steps. By the structure of the procedure **NormalMode**, the Receiver executes infinitely many receive actions. In each of these receive actions an actual packet sent by the Sender must be received, otherwise the Receiver terminates **NormalMode** and switches to **RecoverMode**. The Receiver only sends normal packets after time t. Thus, the Sender will only send finitely many recovery packets after time t, since it can send a recovery packet only when it receives one. Hence, by copy fairness, the Receiver must receive infinitely many normal packets during its execution of **NormalMode** with $j = n$. With each such receipt the Receiver sends a packet of the form $(N, w, *, n \bmod 2)$. Since the channel is copy fair, at least $n - 1$ of these are received by the Sender, forcing it to terminate **SendMsg** with $i = n - 1$ and call **SendMsg** with $i = n$.

Suppose then the call to **NormalMode** with $j = n$ executes for only finitely many steps. Then, eventually **RecoverMode** with $j = n$ is called at a time $t' > t$. The Receiver only sends recovery packets after time t'. Thus, the Sender will only send finitely many normal packets after time t', since it can send a normal packet only when it receives one. Thus, the Receiver will send type w or u recovery packets with sync count $n \bmod 2$ infinitely often after time t'. By copy fairness, infinitely many of these packets are received by the Sender, and the Sender responds with a packet of the form $(R, w, x_{n-1}^r, n - 1 \bmod 2)$, unless the Sender has terminated its call to **SendMsg** with $i = n - 1$. If the call to **SendMsg** with $i = n - 1$ does not terminate then, by copy fairness, infinitely many of these are received by the Receiver, and the Receiver responds with a packet of the form $(R, w, *, n \bmod 2)$. Thus, the Receiver must send infinitely many type w recovery packets with sync count $n \bmod 2$ after time t'. By copy fairness, at least $u_S^{r(t'')} + 1$ of these packets will be received by the Sender forcing **SendMsg** with $i = n - 1$ to terminate.

By the argument above, if $n > 1$ then **SendMsg** with $i = n$ is eventually called and clearly **SendMsg** with $i = 1$ is called. Thus, for all $n \geq 1$, **SendMsg** with $i = n$ is called. By the Zipper property this call can never terminate because the Receiver never calls **NormalMode** with $j = n + 1$. We now argue that **NormalMode** with $j = n + 1$ must eventually be called contrary to our assumption.

Suppose the call to **NormalMode** with $j = n$ executes for infinitely many steps. The Receiver will send infinitely many packets of the form $(N, w, *, n \bmod 2)$ after time t. The Sender must receive infinitely many of these. Once, **SendMsg** with $i = n$ is called the Sender responds to each of these packets with a packet of the form $(N, w, x_n^r, n \bmod 2)$. The Receiver must receive at least n of these causing it to terminate **NormalMode** with $j = n$ and call **NormalMode** with $j = n + 1$.

Suppose then the call to **NormalMode** with $j = n$ executes for only finitely many steps. Then, eventually **RecoverMode** with $j = n$ is called at a time $t' > t$. The Receiver will send infinitely many type w or u recovery packets with sync count $n \bmod 2$ after time t'. Infinitely many of these packets are received by the Sender. After **SendMsg** with $i = n$ is called, the Sender responds to each of these packets with a packet of the form $(R, w, x_n^r, n \bmod 2)$. At least $u_R^{r(t')} + 1$ of these packets will be received by the Receiver, forcing it to terminate **RecoverMode** with $j = n$ and call **NormalMode** with $j = n + 1$. ■

5.1.3 Stop-and-Wait Ideal Runs

Intuitively, $\mathcal{P}_{\text{lin}}$ is running ideally if the Sender and Receiver are perfectly synchronized like in a traditional stop-and-wait protocol. When the Sender sends a packet the Receiver receives it and sends a packet in response and *vice versa*. There is a synchronous back-and-forth communication between Sender and Receiver. We begin by defining the synchronous behavior we want in an ideal run. By a *communication action* we mean a send action or receive action by either the Sender or Receiver.

We define a run $r \in \mathcal{R}_{\text{lin}}$ to have *synchronous communication from time t*, if $t \leq t_1 < t_2 < t_3 < ...$ are all the times of communication actions in the run r on or after time t, then for all i,

1. if the Sender (Receiver) receives at point (r, t_i) then the Sender (Receiver) sends at (r, t_{i+1}) and
2. if the Sender (Receiver) sends at point (r, t_i) then the Receiver (Sender) receives at point (r, t_{i+1}) the packet sent at time t_i.

Thus, for example, if the Sender sends packet p at time t_i then $(p, t_i) \in \mathcal{CR}^{r(t_{i+1}-1)} - \mathcal{CR}^{r(t_{i+1})}$. That is, p is delivered to the Receiver at time t_{i+1}. From the definition, if r has synchronous communication from time t and $t' \geq t$, then r has synchronous communication from time t'.

To properly define a run $r \in \mathcal{R}_{\text{lin}}$ to be stop-and-wait ideal from a time t we are presented with the difficulty that the run r may be out of synchrony at time t, that is, both the Sender and Receiver may be required to send as their next communication actions after time t. The run r must achieve synchronous communication as soon as possible after time t. We now define:

> A run $r \in \mathcal{R}_{\text{lin}}$ is *stop-and-wait ideal from time* $t \geq 0$ if the Sender's first communication action after t is at t_S and the Receiver's first communication action after t is at t_R then one of the following mutually exclusive statements holds:
>
> 1. the Sender sends at (r, t_S), the Receiver receives at (r, t_R), $t_S < t_R$, and r has synchronous communication from time t_S,

2. the Sender receives at (r, t_S),the Receiver sends at (r, t_R), $t_R < t_S$ and r has synchronous communication from time t_R,
3. the Sender sends at (r, t_S), the Receiver sends at (r, t_R), $t_S < t_R$, the packet sent by the Sender at time t_S is never delivered in r, and r has synchronous communication from time $t_S + 1$,
4. the Sender receives at (r, t_S), the Receiver receives at (r, t_R), and r has synchronous communication from time $\min(t_S, t_R)$

It should be clear from the definition that if r is stop-and-wait ideal from time t then r is a fair run of $\mathcal{P}_{\text{lin}}$.

We now argue that extension property (property 3.1), which was vital to the Lower Bound theorem, holds for the protocol $\mathcal{P}_{\text{lin}}$ using our definition of stop-and-wait ideal runs. Thus, $\mathcal{P}_{\text{lin}}$, with its notion of stop-and-wait ideal runs, is in the class of sequence transmission protocols to which our lower bound applies. Stated for the protocol $\mathcal{P}_{\text{lin}}$, the extension property becomes:

Lemma 5.5 *For every $r \in \mathcal{R}_{\text{lin}}$ and $t \geq 0$, there is a fair run r' extending (r, t) that is stop-and-wait ideal from time t and for any $t' \geq t$, $\mathcal{CR}^{r(t)} \subseteq \mathcal{CR}^{r'(t')}$ and $\mathcal{CS}^{r(t)} \subseteq \mathcal{CS}^{r'(t')}$.*

Proof: Let $r \in \mathcal{R}_{\text{lin}}$ and $t \geq 0$. There are four cases to consider depending on where the Sender and Receiver are in their protocols at point (r, t). Each of the Sender and Receiver is prescribed by its protocol to either send or receive as its next communication action. These four cases correspond to the four mutually exclusive cases in the definition of stop-and-wait ideal run.

1. The Sender would send and the Receiver receive. In this case we extend (r, t) with the run r' where in the run r' the Sender goes first, then synchronous communication can begin at that point. Thus, r' and t satisfy part 1 of the definition.

2. The Sender would receive and the Receiver send. In this case, in the run r' the Receiver would go first. Thus, r' and t satisfy part 2 of the definition.

3. Both Sender and Receiver would send. In this case, in the run r' we let the sender go first. This packet is never delivered. Then the Receiver sends and synchronous communication begins. Thus, r' and t satisfy part 3 of the definition.

4. Both Sender and Receiver would receive. In this case, in the run r' we let the Receiver receive a ϕ at which point synchronous communication begins. In r' the Sender does not receive anything until after the Receiver has received the ϕ and sent a type u recovery packet as the Receiver's protocol prescribes. Thus, r' and t satisfy part 4 of the definition.

In all cases r' extends (r, t), r' is stop-and-wait ideal from time t, and no packet sent on or before time t is delivered. ■

Finally, stop-and-wait ideal runs of the protocol $\mathcal{P}_{\text{lin}}$ have the desirable property that if r is stop-and-wait ideal from a time, then it is stop-and-wait ideal from that time on.

Property 5.1 *For all $r \in \mathcal{R}_{\text{lin}}$ and all $t \geq 0$, if r is stop-and-wait ideal from time t and $t' \geq t$, then r is stop-and-wait ideal from time t'.*

Proof: Suppose $r \in \mathcal{R}_{\text{lin}}$ is stop-and-wait ideal from time t and $t' \geq t$. Let t_S (t_R) be the time of the first communication action by the Sender (Receiver) after time t in the run r. By the definition, if $t' < \min(t_S, t_R)$, then r must be stop-and-wait ideal from time t'. Suppose then that $t' \geq \min(t_S, t_R)$. Unless both the Sender and Receiver would send at t_S and t_R, respectively, r has synchronous communication from time $\min(t_S, t_R)$. In the case where both the Sender and Receiver would send as their first communication action after time t, r has synchronous communication from time $t_S + 1$, the time just after the Sender's first communication action. Since $t' \geq \min(t_S, t_R) > t$ then it must be the case that r has synchronous communication from $t' + 1$.

Let t'' be the time of the next communication action in r after time t'. There are four cases to consider depending on what and whose communication action happened at point (r, t''). If the Sender sent a packet at point (r, t''), then, because $t'' \geq t' + 1$, the first communication action by the Receiver after time t' must be a receive of the packet sent at time t'' and r has synchronous communication from time t''. Similarly, if the Receiver sent a packet at point (r, t''), then the first communication action by the Sender after time t' must be a receive of the packet sent at time t'' and r has synchronous communication from time t''. If the Sender received a packet at time t'', then, because $t'' \geq t' + 1$, the first communication action by the Receiver after time t' must be a receive of a packet sent by the Sender immediately after its receive at time t'' and r has synchronous communication from time t''. Similarly, if the Receiver received a packet at time t'', then, the first communication action by the Sender after time t' must be a receive of a packet sent by the Receiver immediately after its receive at time t'' and r has synchronous communication from time t''. Notice that case 3 never occurs. ■

It is worth noting that there are stop-and-wait ideal runs from a time t in which packets sent on or before time t are actually delivered. Such a time t would be when both Sender and Receiver would receive as their next communication action.

5.1.4 Proof that $\mathcal{P}_{\text{lin}}$ is $(2n, 1, 0)$-Recoverable

Upper Bound Theorem(Theorem 5.1) *$\mathcal{P}_{\text{lin}}$ is a $(2n, 1, 0)$-recoverable sequence transmission protocol.*

Proof: Consider any run $r \in \mathcal{R}_{\text{lin}}$ and time t, such that $t^r_{n-1} < t \leq t^r_n$, and any run $r' \in \mathcal{R}_{\text{lin}}$ which extends (r, t) and is stop-and-wait ideal from time t. We will first show that $\mathcal{P}_{\text{lin}}$ has ideal transmission cost $2n$ from $t^{r'}_n$. This will also demonstrate that $\mathcal{P}_{\text{lin}}$ has recovery cost 1.

Let $m \geq n$. Since r' is stop-and-wait ideal from time t and $t \leq t^{r'}_m$ then, by property 5.1, r is stop-and-wait ideal from time $t^{r'}_m$. Since the Receiver writes at $(r, t^{r'}_m)$, its next communication action is a send at the beginning of the call to **NormalMode** with $j = m+1$. Let $t' > t^{r'}_m$ be the time when the Receiver sends its first packet in **NormalMode** with $j = m + 1$. By the Zipper Property the Sender is in a call to **SendMsg** with $i = m$ at time $t^{r'}_m$. The Sender remains in its call to **SendMsg** with $i = m$ at least until time t'. By the definition of stop-and-wait ideal runs, r' has synchronous communication from time t'.

During the call to **NormalMode** with $j = m + 1$, the Receiver will only send normal packets with sync count $m+1 \bmod 2$. Each of these except the first is a type w packet. While the Sender is in **SendMsg** with $i = m$ each type w packet will cause the Sender to increase its counter $nReq$ by one. After receiving at most m such packets the Sender will terminate **SendMsg** with $i = m$ and call **SendMsg** with $i = m + 1$. Thus, while the Sender is in **SendMsg** with $i = m$ and the Receiver is in **NormalMode** with $j = m + 1$, the Receiver will receive at most $m + 1$ packets from the Sender, each one in response to a packet sent by the Receiver. Once the Sender calls **SendMsg** with $i = m + 1$ it will send packets of the form $(N, w, x^{r'}_{m+1}, m+1 \bmod 2)$. The receipt of each of these will cause the Receiver's counter $nReq[x^{r'}_{m+1}]$ to increase by one. When this counter reaches $m+1$, the call to **NormalMode** with $j = m+1$ terminates and $x^{r'}_{m+1}$ is written. Thus, the Receiver receives another $m+1$ packets. Hence, the total number of packets received by the Receiver from the time $t^{r'}_m$ until time $t^{r'}_{m+1}$ is bounded by $2m+2$.

That $\mathcal{P}_{\text{lin}}$ has lookahead 0 follows from the Zipper Property since that shows that read and write actions alternate in all runs in $\mathcal{R}_{\text{lin}}$. ■

6 Conclusions

We have proven lower and upper bounds for recoverable sequence transmission protocols. We have shown the Lower Bound theorem: if $\mathcal{P}$ is a $f(n)$-recoverable sequence transmission protocol then for some $c > 0$ and infinitely many n, $f(n) > cn$. We have also shown the upper bound that $\mathcal{P}_{\text{lin}}$ is $2n$-recoverable under the definition of stop-and-wait ideal runs. There seems to be no way to improve the lower bound, but there are a number of ways to improve our protocol $\mathcal{P}_{\text{lin}}$. We chose to present $\mathcal{P}_{\text{lin}}$ because it is our simplest recoverable protocol. We list below several directions we have taken to improve on the performance of $\mathcal{P}_{\text{lin}}$.

First, it can be shown that for every constant $c > 0$ there is a sequence transmission protocol which is $(cn+3)$-recoverable. This protocol is very similar to $\mathcal{P}_{\text{lin}}$ except that it employs k normal modes where k is a positive integer greater than $2/c$. When the Receiver is requesting the n-th message it uses normal packets of the form $(n \bmod k, v, *, n \bmod 2)$, where $v = w$ or $v = u$. The Sender uses similar packets in reply. The effect is that **NormalMode** with $j = n$ can terminate with correct knowledge of x_n, after receiving only $\lceil n/k \rceil$ packets of the form $(n \bmod k, w, x_n, n \bmod 2)$. In a similar fashion a call to **SendMsg** with $i = n$ can terminate with the Sender having received $\lceil n/k \rceil$ packets of the form $(n+1 \bmod k, w, *, n+1 \bmod 2)$. It can be argued that in an ideal run the Receiver need only receive $2\lceil n/k \rceil + 1 \leq cn+3$ packets in order to write the n-th message.

Second, the protocol $\mathcal{P}_{\text{lin}}$ solves the sequence transmission problem for channels which are copy fair. Another $(2n, 1, 0)$-recoverable protocol, $\mathcal{P}_{\text{wb}}$, solves the sequence transmission problem for channels that are packet fair [Tem90]. The protocol $\mathcal{P}_{\text{wb}}$ employs a recovery method different from that used in $\mathcal{P}_{\text{lin}}$. The recovery method of $\mathcal{P}_{\text{wb}}$ is closely related to the original sequence transmission protocol of Attiya *et al* [AFWZ89]. We chose to present $\mathcal{P}_{\text{lin}}$ because it is simpler and the performance of its recovery mode is better than $\mathcal{P}_{\text{wb}}$.

Third, in $\mathcal{P}_{\text{lin}}$, even if the protocol is running well, the n-th message requires $2n$ packets. This requirement can be lifted by the Sender and Receiver each maintaining a count which bounds the number of undelivered normal packets. These counts can be used instead of the i and j to terminate **SendMsg** and **NormalMode** respectively. In addition, $\mathcal{P}_{\text{lin}}$ runs poorly in recovery mode because it does not take advantage of receipt of very old packets to decrement its counters so that future messages will take fewer packets per message. An improved protocol $\mathcal{P}_{\text{ave}}$ has been defined which employs counters for undelivered normal packets and takes advantage of the receipt of very old packets to reduce the size of its counters [Tem90].

Finally, although recoverable protocols are well motivated because, in favorable conditions, the time to deliver messages is predictable, $O(n)$ packets to deliver the n-th message is still intolerable for practical applications. In another paper [LLT92] we have explored other approaches to the sequence transmission problem which have more potential for practical application. Mansour and Schieber have shown out that any sequence transmission protocol, which is correct for a channel which both loses and reorders packets, requires, with high probability, α^n packets to deliver the n-th message in a FIFO-channel with positive loss rate, where α depends on the loss rate and header size [MS92]. This result seems to rule out any possibility of a practical sequence transmission protocol for channels which can capture packets for an arbitrary amount of time. However, using multi-bit headers, we have developed sequence transmission protocols which behave well for channels with a reasonable loss rate and for sequences of reasonable length. In these protocols the base of the exponent α can be made very close to 1, thereby allowing α^n to be reasonably small, even for relatively large n.

Acknowledgements

We would like to thank Anthony LaMarca for helping review the paper and the anonymous referees for many valuable suggestions for improvements.

References

[AAF+91] Y. Afek, H. Attiya, A. Fekete, M.J. Fischer, N. Lynch, Y. Mansour, D-W. Wang, and L.D. Zuck. Reliable communication using unreliable channels. Yale Technical Report YALEU/DCS/TR-853. To appear in *Journal of the ACM.*, 1991.

[AFWZ89] Hagit Attiya, Michael J. Fischer, Da-Wei Wang, and Lenore D. Zuck. Reliable communication using unreliable channels. Manuscript, May 1989.

[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.

[HZ93] Joseph Y. Halpern and Lenore D. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, 39(3):449–478, July 1993.

[LLT92] Richard E. Ladner, Anthony LaMarca, and Ewan Tempero. Counting protocols for reliable end-to-end transmission. Technical Report TR 92-10-06, University of Washington, October 1992.

[LMF88] Nancy Lynch, Yishay Mansour, and Alan Fekete. The data link layer: Two impossibility results. In *Seventh Annual ACM Symposium on Principles of Distributed Computing*, August 1988. Also available as Technical Report MIT/LCS/TM-355, MIT. May 1988.

[MS92] Yishay Mansour and Baruch Schieber. The intractability of bounded protocols for on-line sequence transmission over non-FIFO channels. *Journal of the ACM*, 39(4):783–799, October 1992.

[Sta88] William Stallings. *Data and Computer Communications.* Macmillan Publishing Company, second edition, 1988.

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

[Tan88] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall, Inc., Englewood Cliffs, N.J., second edition, 1988.

[Tem90] Ewan D. Tempero. *Network protocols for non-FIFO channels.* PhD thesis, University of Washington, Seattle, Washington. United States of America, August 1990. Available as a University of Washington, Department of Computer Science and Engineering Technical Report, No. 90-09-03.

[TL90] Ewan D. Tempero and Richard E. Ladner. Tight bounds for weakly bounded protocols. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 205–218, August 1990.

[WZ89] Da-Wei Wang and Lenore D. Zuck. Tight bounds for the Sequence Transmission Problem. In *Eighth Annual ACM Symposium on Principles of Distributed Computing*, August 1989. Also available as Technical Report YALEU/DCS/TR-705, Yale University.