Yale University
Department of Computer Science

Real-Time Sequence Transmission Problem

Da-Wei Wang Lenore Zuck

YALEU/DCS/TR-856
May 1991

This work was supported in part by the National Science Foundation under grants CCR-8910289 and IRI-9015570 and by an IBM graduate fellowship.

Real-Time Sequence Transmission Problem

Da-Wei Wang Lenore D. Zuck
Department of Computer Science
Yale University

Abstract

In the *sequence transmission problem* one process, the transmitter, wishes to reliably communicate a sequence of data items (messages) to another process, the receiver. We study a real-time version of the sequence transmission problem (*RSTP*) where the messages are taken from a binary domain and we assume three constants, c_1, c_2, and d, $c_1 \leq c_2 \ll d$, such that each process takes a step at least every c_1 and at most every c_2 units of time and each packet which is sent is received within d time units. We define the *effort* of a solution to be the average time it takes the receiver to learn a message.

We study the effort of solution to *RSTP* as a function of the size of c_1, c_2, d, and k—the size of the transmitter's packet alphabet. We show tight bounds on the effort of solutions to *RSTP* for both the *r-passive* case, where the receiver sends no packets, and the general case.

Keywords: reliable communication, real time, I/O automaton.

This work was supported in part by the National Science Foundation under grants CCR-8910289 and IRI-9015570 and by an IBM graduate fellowship.

1 Introduction

One of the simplest and most basic problems of distributed computing is for one process, the *transmitter*, to reliably communicate a finite sequence of data items (*messages*) to another process, the *receiver*. We call this the *sequence transmission problem (STP)*. Solving STP with a perfect channel—one that preserves packet order and delivers each packet within a known amount of time—is trivial: the transmitter simply sends each data item in turn. The receiver passively waits for each packet and processes it when it arrives.

Real channels are not perfect. They may delay packets for arbitrary amounts of time, they may deliver packets out of order, and they may lose, duplicate, or corrupt packets. The *data link layer* ([BSW69, Car, Zim80]) in a standard protocol model attempts to solve STP under a particular set of assumptions about the underlying physical link layer (channel). Other common communication protocols such as virtual circuits, file transfer, and electronic mail are often built on top of this layer since the protocol designer does not then have to be concerned about the faultiness of the physical channel.

Solutions to STP date back to the early work on communication protocols (cf. [BSW69, Ste76, AUY79, AUWY82]). Much of this early work was concerned with optimizing the number of states or the number of packets under various assumptions about the channel; for example, [AUY79, AUWY82] assume synchronous channels in which the loss of a packet can be detected by the recipient at the next time step.

More recently, there have been extensive studies of STP, over asynchronous channels, with finite packet alphabets. It was shown that STP can be solved for channels that reorder and delete packets ([AAF+91, TL90]), although the solution cannot be efficient ([WZ89, MS89, TL90]). It was also shown that STP cannot be solved for channels that duplicate and reorder packets ([WZ89]). The Alternating Bit protocol ([BSW69]) is a solution to the problem for channels that lose and duplicate packets.

Most studies of STP assume that channels are "malicious", i.e., that they can display arbitrary faulty behavior. Hence, the impossibility results indicate that there are no reasonable solutions to the problem if the channel's faulty behavior is unrestricted. This, however, might not be the case in many applications. We therefore ask whether STP is solvable for channels whose faulty behavior is restricted. The work in [MS89] is the first step in this direction. There, it is assumed that the channel's faulty behavior is probabilistic, and it is shown that there are no efficient solutions to STP if the channel can lose and reorder packets with some given probability.

Here we pursue another direction by making *real-time* assumptions about both asynchronicity of the processes and the behavior of the channel. In particular, we follow [AL89, ADLS90] and assume three constants, c_1, c_2, and d, $c_1 \leq c_2 \ll d$, such that each process takes a step at least every c_1 and at most every c_2 units of time, and each packet which is sent on the channel is received within d time units. We call this version of the problem *RSTP*.

We distinguish between two types of solutions to *RSTP*: *r-passive* solutions are those where the receiver sends no packets, and *active* solutions are those in which the receiver sends some packets. We define the *effort* of a solution to be the average time it takes the receiver to learn a message.

We first prove lower bound results on the effort of both r-active and passive solutions to *RSTP* as a function of c_1, c_2, d, and k—the size of the transmitter's packet alphabet. We next present a family of r-passive and active solutions to *RSTP*, all dependent on k—the size of the transmitter's

packet alphabet. The effort of these solutions is only a constant factor worse than the corresponding lower bound.

Our results are presented in the timed Input/Output automaton formalism introduced in [MMT90], which extends the I/O automata model introduced in [LT87] and summarized in [LT89]. The formalism is used when presenting the programs run by the transmitter and the receiver, as well as the channel, by I/O automata. A solution to *RSTP* is then the timed I/O automaton defined by the composition of the I/O automata for both processes and the channel, with the appropriate timing assumptions. We review the relevant parts of the timed I/O automata model in Section 2.

We would like to remark that, although our results are presented in the timed I/O automata formalism, they hold for other models. We chose this formalism because it has been thoroughly studied and used in many previous works on STP.

2 The I/O Automata Model

The results in this paper are presented in the I/O automata formalism. We note that the ideas in the paper are not dependent on this formalism. The I/O automata model was first defined in [LT87]. See [LT87, LT89] for a complete description of the model. Here, we provide a brief summary of those parts of the model used in this paper. The timed I/O automata model was introduced and studied in [MMT90]. Again, we provide only a brief summary of the parts relevant to our work.

2.1 I/O Automata

We assume a universal set of *actions* which describe the activities that occur during a computation. We refer to a particular occurrence of an action as an *event.*

I/O automata are state machines whose state to state transitions are caused by actions. Actions can be internal actions, which have no effect on the environment, or external actions, which describe interactions with the environment. In describing these interactions, it is helpful to classify the actions involving the automaton as "input" or "output" actions. Input actions originate in the environment and are imposed by the environment on the automaton, whereas output actions are generated by the automaton and imposed by it on the environment. In addition, I/O automata have *states* and *steps* (i.e., *transitions*). Formally, an *I/O automaton* A (which we often call simply an *automaton*) is described by:

1. Three mutually disjoint sets of actions: $in(A)$, $out(A)$, and $int(A)$. We denote $acts(A) = in(A) \cup out(A) \cup int(A)$, and $loc(A) = int(A) \cup out(A)$, i.e., $acts(A)$ is the set of A's actions, and $loc(A)$ is the set of A's local actions, namely, the actions that A controls.
2. A set $states(A)$ and a set $start(A) \subseteq states(A)$ of A's *start states.*
3. A transition relation, $steps(A) \subseteq states(A) \times acts(A) \times states(A)$, that is *input enabled*, i.e., for every input action π and state s, there exists some state s' such that $(s, \pi, s') \in steps(A)$. (In general, an action π is *enabled* from a state s if for some s', $(s, \pi, s') \in steps(A)$.)
4. A fairness condition, $fair(A)$, described as a partition on A's local actions with countably many equivalence classes.

The automaton A is *deterministic* if for every state $s \in states(A)$, (1) for every action $\pi \in acts(A)$ there is at most one state $s' \in states(A)$ such that $(s, \pi, s') \in steps(A)$, and (2) at most one local action π is enabled from s.

An *execution* α of A is a (possibly infinite) sequence of the form:

$$s_0 \xrightarrow{\pi_1} s_1 \xrightarrow{\pi_2} \cdots$$

where s_0 is an initial state of A, and for every $i \geq 0$, $(s_i, \pi_{i+1}, s_{i+1})$ is a transition of A. If α is finite then it terminates in a state. The execution α is *fair* if one of the following holds:

1. α is finite and no local action is enabled from the final state of α.
2. α is infinite, and for every set of local actions $L \in fair(A)$, either actions from L are taken infinitely many times in α (i.e., for infinitely many i's, $\pi_i \in L$), or actions from L are disabled infinitely many times in α (i.e., for infinitely many i's, no action of L is enabled from s_i).

Let B be some set and let B' be some subset of B. For every sequence α over B, we denote by $\alpha|B'$ the *restriction* of α on B', that is, the sequence obtained from α when all non-B' elements are deleted.

The *behavior* of an execution α of A (and more generally, of any sequence of actions and states of A), $beh(\alpha)$, is defined to be the sequence $\alpha|in(A) \cup out(A)$. A *fair behavior* of an automaton A is any sequence of the form $beh(\alpha)$, where α is a fair execution of A.

Let A and B be I/O automata. We say that A and B are *composable* if the only mutual actions are input of one and output of the other, or input of both. If A and B are composable, their *composition*, written as $A \circ B$, is an automaton C such that:

1. C's output and internal actions are the union of the output and internal actions respectively of A and B, and C's input actions are the union of A's and B's input actions that are not C's output actions (i.e., $in(C) = in(A) \cup in(B) - (out(A) \cup out(B))$).
2. C's state set is the Cartesian product of its component state sets, i.e., $states(C) = states(A) \times states(B)$, and C's initial state set is the Cartesian product of its components' initial state sets.
3. C's transitions are such that only the components to which the action belongs are affected, i.e., $((s_A, s_B), \pi, (s'_A, s'_B)) \in steps(C)$ iff:

$$\begin{cases} (s_A, \pi, s'_A) \in steps(A) \text{ and } s_B = s'_B & \text{if } \pi \in acts(A) - acts(B), \\ s_A = s'_A \text{ and } (s_B, \pi, s'_B) \in steps(B) & \text{if } \pi \in acts(B) - acts(A), \text{and} \\ (s_A, \pi, s'_A) \in steps(A) \text{ and } (s_B, \pi, s'_B) \in steps(B) & \text{if } \pi \in acts(A) \cap acts(B). \end{cases}$$

4. The fairness condition of C, $fair(C) = fair(A) \cup fair(B)$. (In other words, actions are in the same class in C's partition exactly if they are in the same class in either A's or B's partition.) Note that this is a partition of $loc(C)$ since A and B do not have any locally controlled actions in common.

Let α be an execution of C. Then α defines an execution of A obtained by deleting from α every occurrence of $\xrightarrow{\pi} s$ for actions $\pi \notin acts(A)$, and replacing every remaining state in α with its A component. We denote the resulting execution by $\alpha|A$. Similarly, α defines an execution of B denoted by $\alpha|B$. Note that α is a fair execution of C iff $\alpha|A$ and $\alpha|B$ are fair executions of A and B respectively.

2.2 Timed I/O Automata

In order to reason about time in I/O automata model, we assign "times" to events appearing in executions of automata. The "times" we assign are real nonnegative numbers. The assignment is such that if event π_1 proceeds event π_2, then the time assigned to π_1 is not greater than the time assigned to π_2, and if an execution is infinite, then the times assigned to its actions grow with no bounds. Formally, a *timing t for an execution* $\eta = s_0, a_1, s_1, a_2, \ldots$ is a mapping from η's events to $\mathbf{R}^*$ (non-negative reals), such that the first event in η is mapped to 0, t is monotonically increasing, and if η is infinite then t grows to infinity. Formally, we require the following:

1. If π is the first event in η, then $t(\pi) = 0$.
2. For every two events, π_1 and π_2 in η, if π_1 precedes π_2, then $t(\pi_1) \leq t(\pi_2)$.
3. For every interval $[t_1, t_2]$ where $t_1, t_2 \in \mathbf{R}^*$, $t(\pi) \in [t_1, t_2]$ for at most finitely many events π in η.

Given an automaton A, an execution η of A, and a timing t for η, we denote by η^t the pair (η, t) and term it *timed execution.* A *timing property* P for an I/O automaton A is a subset of the set of A's timed executions. A *timed automaton* (A, P) consists of an an I/O automaton A and a timing property P for A. Let σ be a sequence over $acts(A)$. A *timing t for σ* is defined in the obvious way, and the pair (σ, t) is denoted by σ^t and termed *timed sequence.* We define *timed behaviors* similarly.

Let A and B be I/O automata, and suppose that $C = A \circ B$. Every timed execution η^t of C induces a timed execution on both A and B, which we denote by $\eta^t|A$ and $\eta^t|B$ respectively. Similarly, if β^t is a timed behavior of C, β^t then defines timed behaviors $\beta^t|A$ and $\beta^t|B$ of A and B respectively.

3 Multisets and Related Definitions

Let U be a finite set. A *multiset over a universe U* is a function $Q: U \to \mathbf{N}$. For every element $u \in U$, we define $mult(u, Q) = Q(u)$, which denotes the number of occurrences of u in Q. For two multisets Q and Q' over the same universe U, we say that Q' is a *submultiset* of Q, written $Q' \sqsubseteq Q$, if $mult(u, Q') \leq mult(u, Q)$ for every $u \in U$. The *empty* multiset is denoted by $\emptyset$, is the multiset which has 0 occurences each element, that is, $mult(u, \emptyset) = 0$ for every $u \in U$.

For every multiset Q over U and every $u \in U$, $Q \sqcup \{u\}$ is the multiset Q' such that $mult(u, Q') = mult(u, Q) + 1$ and $mult(u', Q') = mult(u', Q)$ for all $u' \neq u$.

For every $k, n \geq 1$, let $\mathsf{multi}_k(n)$ denote the set of multisets of size n over $\{0, \ldots, k-1\}$. Note that

$$|\mathsf{multi}_k(n)| = \binom{n+k-1}{k-1}.$$

We denote $|\mathsf{multi}_k(n)|$ by $\mu_k(n)$. Define

$$\zeta_k(n) = \sum_{j=1}^{n} \mu_k(j),$$

that is, $\zeta_k(n)$ is the number of multisets over a universe of size k that contain n elements or less. Note that, since $\mu_k(j) < \mu_k(j+1)$ for every $j \geq 1$, $\zeta_k(n) \leq n\mu_k(n)$.

For every $k \geq 2$ and $n \geq 1$, let $\mathsf{toseq}_k(n)\colon \mathsf{multi}_k(n) \to \{0, \ldots, k-1\}^n$ be such that for every multiset $P \in \mathsf{multi}_k(n)$, $\mathsf{toseq}_k(n)(P)$ is a "linearization" of P, i.e., a sequnece that contains, for every $0 \leq j < k$, $mult(j, P)$ occurences of j. Let $\mathsf{tomulti}_k(n)\colon \{0,1\}^{\lfloor \log(\mu_k(n)) \rfloor} \to \mathsf{multi}_k(n)$ be some one-to-one function that maps every binary sequence x of length $\lfloor \log(\mu_k(n)) \rfloor$ to a multiset $P \in \mathsf{multi}_k(n)$.

4 Real-Time Sequence Transmission Problem

In the *sequence transmission problem* (STP) there are two processors, the *transmitter* and the *receiver*, which communicate over a bidirectional communication link (the *channel*). The transmitter t has some finite sequence of messages X, taken from some finite domain M, which it tries to transmit to the receiver r. r must write these messages onto an output tape Y. We require that for all X, at any time, Y is a prefix of X, and if the channel satisfies appropriate *fairness* conditions, then every message in the sequence X is eventually written by r.

We consider here a *real-time* version of STP, where the channel can delay messages up to d time units, and both t and r take steps within at least c_1 and at most c_2 time units. We restrict discussion to cases where $|M| = 2$ and $0 < c_1 < c_2 \ll d$. Without loss of generality, we assume that $M = \{0, 1\}$, and keep M, c_1, c_2, and d fixed for the duration of the paper. We denote the real-time version of the sequence transmission problem *RSTP*.

Both the transmitter and the receiver are represented by I/O automata, A_t and A_r. A_t and A_r communicate by sending *packets* to one another through the channel. In particular, t sends r packets from some alphabet P^{tr} of packets, and r sends t packets from some alphabet P^{rt} of packets. We assume that for every $p \in P^{\mathsf{tr}}$, $\mathsf{send}(p) \in out(A_\mathsf{t})$ and $\mathsf{recv}(p) \in in(A_\mathsf{r})$. Similarly, for every $p \in P^{\mathsf{rt}}$, $\mathsf{send}(p) \in out(A_\mathsf{r})$ and $\mathsf{recv}(p) \in in(A_\mathsf{t})$. The receiver writes the messages it learns, and we therefore assume that for every $m \in M$, $\mathsf{write}(m) \in out(A_\mathsf{r})$. In addition, each process can have internal actions, and we assume that the actions set of the processes are mutually disjoint.

Given a packet alphabet P (in this paper, P is always $P^{\mathsf{tr}} \cup P^{\mathsf{rt}}$), a *channel* over P is an I/O automaton $C(P)$, where $in(C(P)) = \{\mathsf{send}(p) : p \in P\}$, $out(C(P)) = \{\mathsf{recv}(p) : p \in P\}$, and $C(P)$'s fair executions consist of all the sequences over $acts(C(P))$ for which there exists a bijection between send events and recv events, such that no packet is received before it is sent[1]. Note that

[1] See [AAF+91] for a proof that $C(P)$ exists for every P.

in every fair execution of $C(P)$, every $\mathsf{send}(p)$ event has a unique corresponding $\mathsf{recv}(p)$ event and vice versa. When P is clear from the context, we denote $C(P)$ by C.

The definition of *RSTP* includes two assumptions about timing, the first is concerned with relative timing of A_r's and A_t's steps, and the second is concerned with delivery time of packets. The first clearly has to do with the composition $A_r \circ A_t$, the second has to do with the C's timed behaviors.

Let A_r and A_t be automata as described above. Assume that $A = A_t \circ A_r$ and that $P = P^{tr} \cup P^{rt}$. We define $\Sigma(A_t, A_r)$ to be the set of A's timed executions in which the difference of timing between every two consecutive local (output or internal) events of each of component automaton is at least c_1 and at most c_2. Similarly, we define $\Delta(C(P))$ to be the set of $C(P)$'s timed executions where the difference of timing between every **send** event and its corresponding **recv** event is at most d.

Let A_r, A_t, A, and P be as above. For every execution η of A, let $X(\eta)$ denote the input sequence of η. Similarly, let $Y(\eta)$ denote the sequence of messages written in η, i.e., if $\eta|\{\mathsf{write}(m) : m \in M\} = \mathsf{write}(m_1), \mathsf{write}(m_2), \ldots$, then $Y(\eta) = m_1, m_2, \ldots$. Let $good(A)$ denote the set of A's timed executions that satisfy both timing assumptions, that it, $good(A)$ consists of the timed executions η^t of $(A, \Sigma(A_t, A_r))$ such that $\eta^t|C$ is a timed behavior of $(C(P), \Delta(C(P)))$. We say that the pair (A_t, A_r) *solve* *RSTP* if for every $\eta \in good(A)$, $Y(\eta) = X(\eta)$.

We refer to general solutions to *RSTP* as *active* solutions. Solutions where A_r sends no packets ($P^{rt} = \emptyset$) are termed r-*passive*.

For every timed execution η^t in $good(A)$, let $\mathsf{last\text{-}send}(\eta^t)$ denote the last **send** event in η^t. The *effort* of A, denoted *eff*(A), is defined by:

$$\operatorname*{suplim}_{n\to\infty} \frac{\max\{t(\mathsf{last\text{-}send}(\eta^t)) : \eta^t \in good(A(n))\}}{n}$$

where $good(A(n))$ is the set of timed sequences η^t in $good(A)$ such that $|X(\eta)| = n$. The numerator is the time it takes the transmitter to transmit the input sequence in "good" executions, and dividing it by n results in the average time it takes the transmitter to transmit a single message in "good" executions. We then choose the maximal such time among all "good" executions whose input is of length n, and *eff*$(\mathcal{S})$ is defined as the suplim of these as n approaches infinity.

We end this section with an example of a solution, (A_t^α, A_r^α), to *RSTP*. A_t^α and A_r^α are presented in Figure 1. Obviously, (A_t^α, A_r^α) is an r-passive solution to *RSTP*. We assume that $P = P^{tr} = M$, and that the input sequence is given by $x_1, \ldots, x_n$.

The execution of A_t^α proceeds in rounds, in each A_t^α transmits a single input element (the input elements are transmitted in sequence). To transmit an input m, A_t^α performs a single $\mathsf{send}(m)$ action. It then waits idly for d/c_1 steps. A_t^α has two local counters, i that points to the next message to be sent, and j that counts the idle steps. Initially both counters are 0. A_t^α has one internal action, wait_t, used to enforce A_t to wait d/c_1 step in between transmission of consecutive messages.

A_r^α, following every $\mathsf{recv}(p)$ action, performs a $\mathsf{write}(p)$ action. Since I/O automata do not allow actions (e.g., $\mathsf{write}(m)$) to be the effects of other actions (e.g., $\mathsf{recv}(m)$), A_r^α has to store the messages it learns and write them when possible. To this end, A_r^α has an array $y_1, \ldots$ in which it stores the messages it receives. It also has two local counters, i (initially 0) that counts the number of messages received, and k (initially 1) that counts the number of messages written. A_r has one local action, idle_r: since A_r is required to take local actions at least every c_2 time units, it

must have some local action enabled when no packet arrives and there are no messages to write. Consequently, the precondition of idle_r is true only when A_r has nothing else to do, and idle_r has no effect.

Remark: The assumption that $c_2 \ll d$, guarantees that A_r^α has to store only two messages. We chose the unbounded array for sake of simplicity.

The fairness partition of (A_t^α, A_r^α) has all the local actions in one class.

Transmitter	**Receiver**
send(p): **precondition:** $j = 0$ and $p = x_i$ **effect:** $j := 1$ wait_t: **precondition:** $0 < j < d/c_1$ **effect:** $j := j + 1$ if $j = d/c_1$ then $i := i + 1$ $j := 0$	recv(p): **effect:** $i := i + 1$ $y_i := p$ write(m): **precondition:** $k \leq i$ and $m = y_k$ **effect:** $k := k + 1$ idle_r: **precondition:** $k > i$

Figure 1: A Simple r-passive solution to *RSTP*

To see why the protocol presented in Figure 1 indeed solves *RSTP*, observe that once A_t^α sends a packet, it sends no other packet until it performs $d/c_1 - 1$ idle steps, or, alternatively, packets are sent no less than $c_1 d/c_1 = d$ time units apart. Since packets can be delayed at most d units of time, if follows that packets are delivered in the order they are sent. Hence, A_r^α writes messages in the order they are sent. Let $A^\alpha = A_t^\alpha \circ A_r^\alpha$. It is easy to see that

$$\mathit{eff}(A^\alpha) = c_2 \frac{d}{c_1}.$$

The term d/c_1 denote the maximal number of steps a process can take in d units of time and occurs frequently in our discussions. We therefore find it convenient to denote it by δ_1. Similarly, the term d/c_2 denotes the minimal number of steps a process can take in d units of time, and we denote it by δ_2.

5 Lower Bounds on Effort of *RSTP* Solutions

For sake of clarity, we consider only solutions (A_t, A_r) to *RSTP* where both A_t and A_r are deterministic. All the results reported in this paper apply to nondeterministic processes as well.

5.1 The r-passive case

Consider an r-passive solution (A_t, A_r) to *RSTP*, and let $A = A_t \circ A_r$. Assume that $P^{tr} = \{0, \ldots, k-1\}$, $k \geq 2$. Fix A_t, A_r, and A for the duration of this section.

Since (A_t, A_r) is r-passive, $in(A_t) = \emptyset$, so that A_t's actions depend only on the input sequence. Since A_t is deterministic, it follows that the input sequence determines A_t's actions. Hence, there exists some function f_t that maps input sequences to sequence of actions, such that for every input sequence $X \in M^*$, for every execution η of A_t such that $X(\eta) = X$, $\eta | acts(A) = f_t(X)$.

Let X be some input sequence, and assume $f_t(X) = a_1, a_2, \ldots$. Consider now "fast" timed executions of A, namely, ones in which A_t is scheduled every c_1 time units. In such executions, the packets that are sent in any consecutive δ_1 actions may be received in any order before the next action (following the δ_1 actions) is taken. We next divide $f_t(X)$ to intervals of δ_1 actions, and consider the multisets of packets sent in each interval. Consequently, we define a function $P^{tr}(X)$ from N to multisets over P^{tr} such that for every $i \geq 1$, $P^{tr}(X)[i]$ is the multiset of packets sent in $a_{\delta_1(i-1)+1}, \ldots, a_{\delta_1 i}$.

Obviously, if there exists some number, say ℓ, such that in any timed execution $\eta^t \in good(A)$ whose input is X, **last-send**(η^t) occurs at or before the $(\ell\delta_1)^{th}$ local action of A_t in η^t, then the only "meaningful" information of $P^{tr}(X)$ is contained in $P^{tr}(X)[1], \ldots, P^{tr}(X)[\ell]$. Such an $\ell(X)$ is defined, for every input sequence X, by:

$$\left\lceil \frac{\max\{|\hat{\eta}^t| : \eta^t \in good(A) \text{ and } X(\eta^t) = X\}}{\delta_1} \right\rceil,$$

$\hat{\eta}^t = \bar{\eta}^t \,|loc(A_t)$ where $\bar{\eta}^t$ denotes η^t, truncated after the last **send** event (**last-send**(η^t)).

Define an equivalence relation $\approx$ on M^* by

$$\ell(X_1) = \ell(X_2) \text{ and } P^{tr}(X_1)[i] = P^{tr}(X_2)[i] \text{ for every } i \leq \ell(X_1)$$

The next lemma establishes that $X_1 \approx X_2$ iff $X_1 = X_2$.

Lemma 5.1 *For every $X_1, X_2 \in M^*$, if $X_1 \approx X_2$ then $X_1 = X_2$.*

Proof: We show, by contradiction, that if $X_1 \approx X_2$ and $X_1 \neq X_2$, then there exist two good executions η_1^t and η_2^t, whose input sequences are X_1 and X_2 respectively, such that the receiver cannot tell these two executions apart. Consequently, it writes the same messages in both executions, contradicting the assumption that $X_1 \neq X_2$.

The two timed executions, η_1^t and η_2^t, are constructed by letting both A_t and A_r take steps every c_1 units of time, starting at 0, and guaranteeing that A_r received the same packet, at the same time, in both executions. This is possible since $X_1 \approx X_2$ and the packets sent in δ_1 consecutive steps of A_t can be grouped and delivered in the same order and at the same time in both η_1^t and η_2^t.

It therefore follows that $\eta_1^t|A_r = \eta_2^t|A_r$. Consequently, $Y(\eta_1) = Y(\eta_2)$. Since both executions are in $good(A)$, $Y(\eta_1) = X_1$ and $Y(\eta_2) = X_2$. Hence, $X_1 = X_2$, contradicting the assumption that $X_1 \neq X_2$. ■

Let $n \geq 0$ and consider the set of input sequences whose length is n, denoted by X^n. Since $|M| = 2$, $|X^n| = 2^n$. Define $\ell(n)$ to be $\max\{\ell(X) : X \in X^n\}$. There are at most $\zeta_k(\delta_1)$ different multisets over P^{tr} that contain δ_1 or less packets. From Lemma 5.1 it follows that for every X_1 and X_2 in X^n, $P^{tr}(X_1)[i] \neq P^{tr}(X_2)[i]$ for at least one $i \leq \ell(n)$. It therefore follows that

$$(\zeta_k(\delta_1))^{\ell(n)} \geq 2^n.$$

Consequently,

$$\ell(n) \geq \frac{n}{\log(\zeta_k(\delta_1))}.$$

The following lemma establishes a lower bound on A's effort:

Lemma 5.2

$$eff(A) \geq \frac{c_2\delta_1}{\log(\zeta_k(\delta_1)}.$$

Proof: Fix some $n \geq 0$. Recall that for every timed execution in $good(A)$ whose input is in X^n, A_t goes through no more than $\ell(n)$ intervals of δ_1 steps before it performs the last send event. Each such interval takes at most $c_2\delta_1$ units of time, hence,

$$\max\{T(\eta^t) : \eta^t \in good(A(n))\} \geq (\ell(n) - 1)\delta_1 c_2,$$

where $good(A(n))$ denotes the set of timed executions in $good(A)$ whose input sequence is of length n. It therefore follows that

$$\begin{aligned} eff(A) &= \lim_{n\to\infty} \frac{\max\{T(\eta^t) : \eta^t \in good(A(n))}{n} \\ &\geq \lim_{n\to\infty} \frac{(\ell(n) - 1)\delta_1 c_2}{n} \\ &\geq \lim_{n\to\infty} \frac{n\delta_1 c_2}{\log(\zeta_k(\delta_1))n} \\ &= \frac{\delta_1 c_2}{\log(\zeta_k(\delta_1))} \end{aligned}$$ ■

From the discussion above we derive:

Theorem 5.3 *Let (A_t, A_r) be an r-passive solution to RSTP, then*

$$eff(A_t \circ A_r) \geq \frac{c_2\delta_1}{\log(\zeta_k(\delta_1))} = \Omega\Big(\frac{c_2\delta_1}{\log(\mu_k(\delta_1))}\Big).$$

5.2 The active case

Consider an active solution (A_t, A_r) to *RSTP*, and let $A = A_t \circ A_r$. Fix A_t, A_r, and A for the duration of this section. Unlike the r-passive case where every execution η of A, $\eta|A_t$ uniquely depends on $X(\eta)$, in the active case $\eta|A_t$ also depends on packets sent by A_r. However, for every $X \in M^*$, we can define some unique timed execution in $good(A)$ whose input is X:

Let $\epsilon > 0$ be arbitrarily small. Define two sequences of intervals over the real nonnegative line, $t_0, t_1, \ldots$ such that for every $i \geq 0$, $t_i = [i(d-\epsilon), (i+1)(d-\epsilon))$, and $\hat{t}_1, \ldots$ such that for every $i \geq 1$, $\hat{t}_i = [i(d-\epsilon), i(d-\epsilon)+\epsilon)$. The intervals are illustrated Figure 5.2.

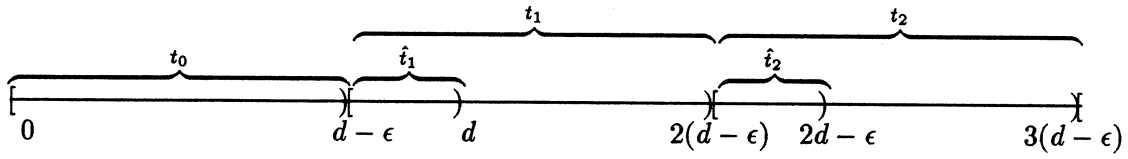

Figure 2: Dividing the real line into intervals

For every $X \in M^*$, let $\eta(X)$ be the (unique) timed sequence in $good(A)$ whose timed events satisfy all the following:

1. For every $i \geq 0$, every message sent during t_i is received during $\hat{t}_{i+1}$.
2. For every $i > 1$, the packets received during $\hat{t}_i$ are received uniformally time-wise, that is, if k packets are received during t_i, then they are received ϵ/k time units apart, the first received at $i(d-\epsilon)$. Also, for every $j = 1, \ldots, k-1$, no j-packets is received in $\hat{t}_i$ before all $(j-1)$-packets have been received.
3. Both A_t and A_r take steps every c_2 units of time, starting at time 0.

Similarly to the r-passive case, for every $X \in M^*$, define a function $P^{tr}(X)$ from $\mathbf{N}$ to multisets over P^{tr} such that for every $i \geq 0$, $P^{tr}(X)[i]$ is the multiset of packets sent during t_i in $\eta^t(X)$. Let $\ell(X)$ be such that **last-send**$(\eta^t(X))$ occurs during $t_{\ell(X)}$.

Define an equivalence relation $\approx$ on M^* by:

$$X_1 \approx X_2 \text{ iff } \ell(X_1) = \ell(X_2) \text{ and } P^{tr}(X_1)[i] = P^{tr}(X_2)[i] \text{ for every } i \leq \ell(X_1)$$

Similarly to Lemma 5.1, we have:

Lemma 5.4 *For every $X_1, X_2 \in M^*$, if $X_1 \approx X_2$ then $X_1 = X_2$.*

Proof: The proof is similar to this of Lemma 5.1. ■

Since in each $\eta^t(X)$ the transmitter takes a step every c_2 units of time, the maximum number of packets that can be sent during any t_i is $\hat{\delta}_2 = (d-\epsilon)/c_2$. A simple combinatorial argument establishes that there are $\zeta_k(\hat{\delta}_2)$ different multisets over a universe of size k that contain no more than $\hat{\delta}_2$ elements. Using a counting argument, similar to the one used in the proof of Lemma 5.2, we get:

Lemma 5.5

$$\mathit{eff}(A) \geq \frac{d}{\log(\zeta_k(\hat{\delta}_2))}.$$

Since ϵ is arbitrarily small,

$$\mathit{eff}(A) = \Omega(d/\log(\zeta_k(\hat{\delta}_2))) = \Omega(d/\log(\zeta_k(\delta_2))).$$

The discussion above is summarized in:

Theorem 5.6 *Let (A_t, A_r) be an r-active solution to RSTP, then*

$$\mathit{eff}(A_t \circ A_r) = \Omega\Big(\frac{d}{\log(\zeta_k(\delta_2))}\Big) = \Omega\Big(\frac{d}{\log(\mu_k(\delta_2))}\Big).$$

6 Solutions to *RSTP*

As shown in Section 5, the effort of a solution to *RSTP* depends on the size of t's packets alphabet: the larger P^{tr} is, the least effort the solution requires. Assume that we are given some $k \geq 2$. Our goal is to find a solution to *RSTP* with $P^{\mathrm{tr}} = \{0, \ldots, k-1\}$ whose effort is optimal.

6.1 An r-passive Solution to *RSTP*

Consider $A^\alpha = (A_t^\alpha, A_r^\alpha)$ of Section 4. The effort of A^α can be reduced if the transmitter sends the messages in a somewhat more compact form. In particular, the transmitter can *encode* several messages together and transmit them in δ_1 consecutive packets before waiting δ_1 steps for their safe reception. For every $k \geq 2$, we present an r-passive solution $(A_t^{\beta(k)}, A_r^{\beta(k)})$ to *RSTP* with $|P^{\mathrm{tr}}| = k$, where t uses $(\mathsf{toseq}_k(\delta_1) \circ \mathsf{tomulti}_k(\delta_1))$ to encode sequences of consecutive $\lfloor \log(\mu_k(\delta_1)) \rfloor$ messages into k-ary sequences of length δ_1, which it transmits in δ_1 steps.

For simplicity, assume that $|X| \equiv 0 \pmod{\lfloor \log(\mu_k(\delta_1 k)) \rfloor}$.

The execution of $A_t^{\beta(k)}$ then proceeds in rounds, each consists of $2\delta_1$ steps in which $A_t^{\beta(k)}$ sends δ_1 packets containing δ_1 consecutive elements of $\hat{X}$, followed by δ_1 idle steps, to ensure safe delivery of the previously sent δ_1 packets. The receiver waits for the arrival of δ_1 packets, decodes them using $\mathsf{toseq}_k(\delta_1)$, and writes them.

Protocol $(A_t^{\beta(k)}, A_r^{\beta(k)})$, which is based on the above ideas, is presented in Figure 3. Since the encoding/decoding parts are straightforward but tedious, they are omitted. Instead, we assume that $A_t^{\beta(k)}$ is given $\hat{X}$, and that $A_r^{\beta(k)}$ has only to write the elements of $\hat{X}$.

The code for $A_t^{\beta(k)}$ uses three local variables, i, initially 1, is the index of the next $\hat{X}$ value to be sent, c, initially 0, counts steps in each round: when A_t sends packets, c goes from 0 to δ_1, when $A_t^{\beta(k)}$ waits, c goes from δ_1 to $2\delta_1$. $A_t^{\beta(k)}$ has one internal action, wait_t whose role is just like that of wait_t in A_t^α.

$A_r^{\beta(k)}$ uses i, initially 0, to count the $\hat{X}$ values received, $\hat{y}_1, \ldots$ to store these values, and k, initially 1, to count the values written. It also uses a multiset A, initially empty, to store the packets it receives in each round. $A_r^{\beta(k)}$ has one internal action, idle_r, which is just like that of idle_r in A_r^α.

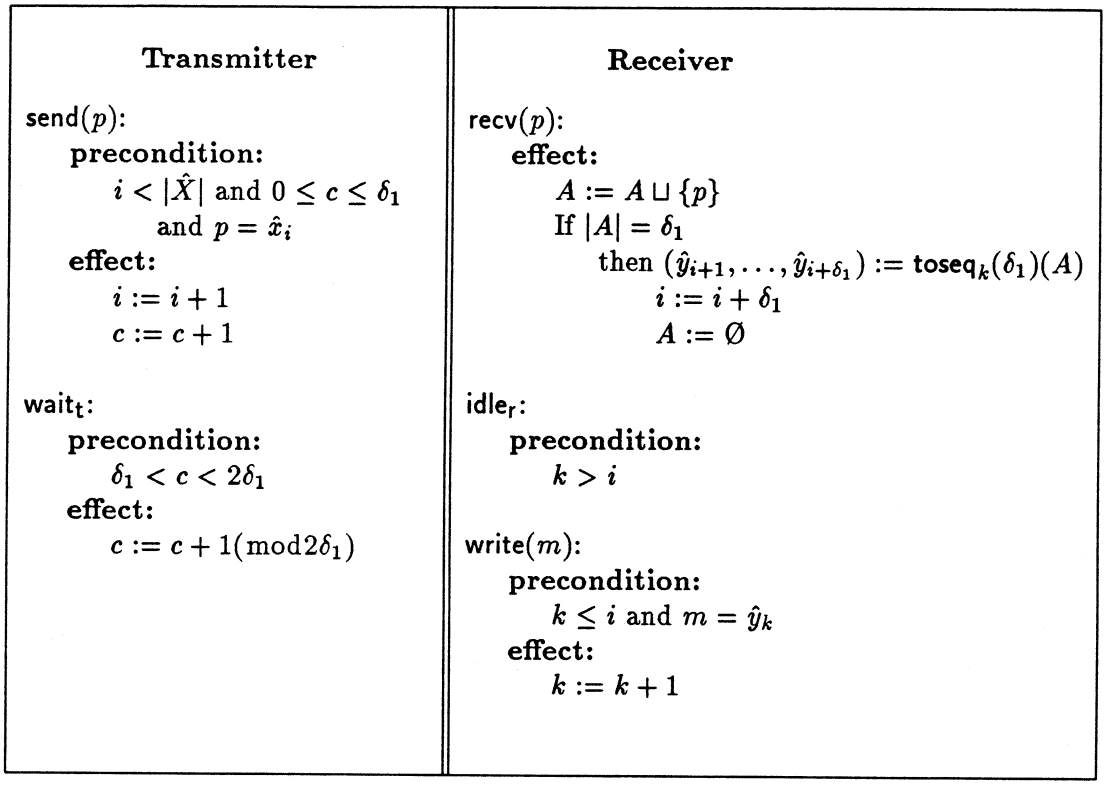

Figure 3: An r-passive solution to *RSTP*

Lemma 6.1 *For every $k \ge 2$, $(A_t^{\beta(k)}, A_r^{\beta(k)})$ is an r-passive solution to RSTP with $|P^{tr}| = k$.*

Proof: We show, by induction on l, that in every timed execution η^t of $(A_t^{\beta(k)}, A_r^{\beta(k)})$, for every $l \le |\hat{X}(\eta)|$ such that $l \equiv 1 \pmod{\delta_1}$, (1) eventually the precondition of wait$_t$ is true and then $A_t^{\beta(k)}$'s i equals $l > \delta_1$, and (2) if the precondition of wait$_t$ is true with the transmitter's i equals to l, then eventually $A_r^{\beta(k)}$ sets $\hat{y}_{l-\delta_1}, \ldots, \hat{y}_{l-1}$ to $\hat{x}_{l-\delta_1}, \ldots, \hat{x}_{l-1}$.

(1) follows immediately from the code. We show only the base case of (2) and leave the inductive step to the reader. The precondition of wait$_t$ becomes true only after $A_t^{\beta(k)}$ sends δ_1 packets. All of these packets are received by time $t + d$, where t denotes the time $A_t^{\beta(k)}$ sends the last among those δ_1 packets. After $A_t^{\beta(k)}$ sends the δ_1 packets, it sends nothing until it takes additional δ_1 steps where it sends no packets, or, equivalently, $A_t^{\beta(k)}$ sends no packet during $(t, t + d]$. Consequently, by $t + d$, $A_r^{\beta(k)}$ receives exactly the δ_1 packets $A_t^{\beta(k)}$ sent before the precondition of wait$_t$ became true. The claim trivially follows. ■

Let $A^{\beta(k)}$ be $A_t^{\beta(k)} \circ A_r^{\beta(k)}$. Let η^t be a timed execution in $good(A^{\beta(k)})$ and assume that $|X(\eta)| = n$. η^t can be divided to $n/\lfloor \log(\delta_1 + 1) \rfloor$ blocks, each containing $2\delta_1$ steps of $A_t^{\beta(k)}$. Since each step

takes at most c_2 time units, it follows that

$$t(\textsf{last-send}(\eta^t)) \leq \frac{n}{\lfloor \log(\mu_k(\delta_1)) \rfloor} 2\delta_1 c_2.$$

Consequently,

$$\mathit{eff}(A^{\beta(k)}) \leq \frac{2c_2\delta_1}{\lfloor \log(\mu_k(\delta_1)) \rfloor}.$$

6.2 An Active Solution to *RSTP*

The active protocol is similar to the r-passive protocol[2]. The differences are that in the active protocol δ_2 replaces δ_1 of the r-passive protocol, and that in the active protocol the receiver acknowledges each packet received. Consequently, The transmitter, after sending δ_2 values of $\hat{X}$, waits until it receives δ_2 acknowledgements, denotes by *ack* packets, before sending the next δ_2 values of $\hat{X}$.

Protocol $(A_t^{\gamma(k)}, A_r^{\gamma(k)})$, which is based on the above ideas, is presented in Figure 4. As in Figure 3, we assume that $A_t^{\gamma(k)}$ is given $\hat{X}$, and that $A_r^{\gamma(k)}$ has only to write the elements of $\hat{X}$.

In addition to the local variables that $A_t^{\beta(k)}$ uses, $A_t^{\gamma(k)}$ uses a, initially 0, to count the number of *ack* packets received in each round. In addition to the local variables that $A_r^{\beta(k)}$ uses, $A_r^{\gamma(k)}$ uses j, initially 0, to count the number of of unacknowledged packets, so that at any time, $0 \leq j \leq |A| \leq \delta_2$. Note that $A_r^{\gamma(k)}$ has, in addition to all the **write** output actions, also a **send**(*ack*) output action, and $A_t^{\gamma(k)}$ has a **recv**(*ack*) input action.

Similarly to Lemma 6.1, we can prove:

Lemma 6.2 $(A_t^{\gamma(k)}, A_r^{\gamma(k)})$ *is a solution to RSTP.*

Let $A^{\gamma(k)} = A_t^{\gamma(k)} \circ A_r^{\gamma(k)}$. Consider any execution η^t in $\mathit{good}(A^{\gamma(k)})$. Let t' be such that in η^t, the transmitter sends the first packet in some block of δ_2 values of $\hat{X}$ at time t'. Obviously, the transmitter's first packet arrives to the receiver by time $t'+d$, the receiver sends an *ack* to the packet by time $t' + d + c_2$, and the *ack* packet arrives to the transmitter by time $t' + 2d + c_2$. It therefore follows that the transmitter's δ_2^{th} packet is sent by time $t' + \delta_2 c_2 = t' + d$, and is acknowledged by time $t' + 3d + c_2$. By this time, the transmitter can start sending the next $\hat{X}$ block. Hence, each block of δ_2 values of $\hat{X}$, which corresponds to a block of $\lfloor \log(\mu_k(\delta_2)) \rfloor$ values of X, is transmitted in at most $3d + c_2$ units of time. It therefore follows that:

$$\mathit{eff}(A^{\gamma(k)}) \leq \frac{3d + c_2}{\lfloor \log(\mu_k(\delta_2 k) \rfloor}.$$

Note that $\mathit{eff}(A^{\beta(k)})$ and $\mathit{eff}(A^{\gamma(k)})$ are asympotically optimal according to the results of Section 5.

[2]The protocol described here is due to an idea of Richard Beigel

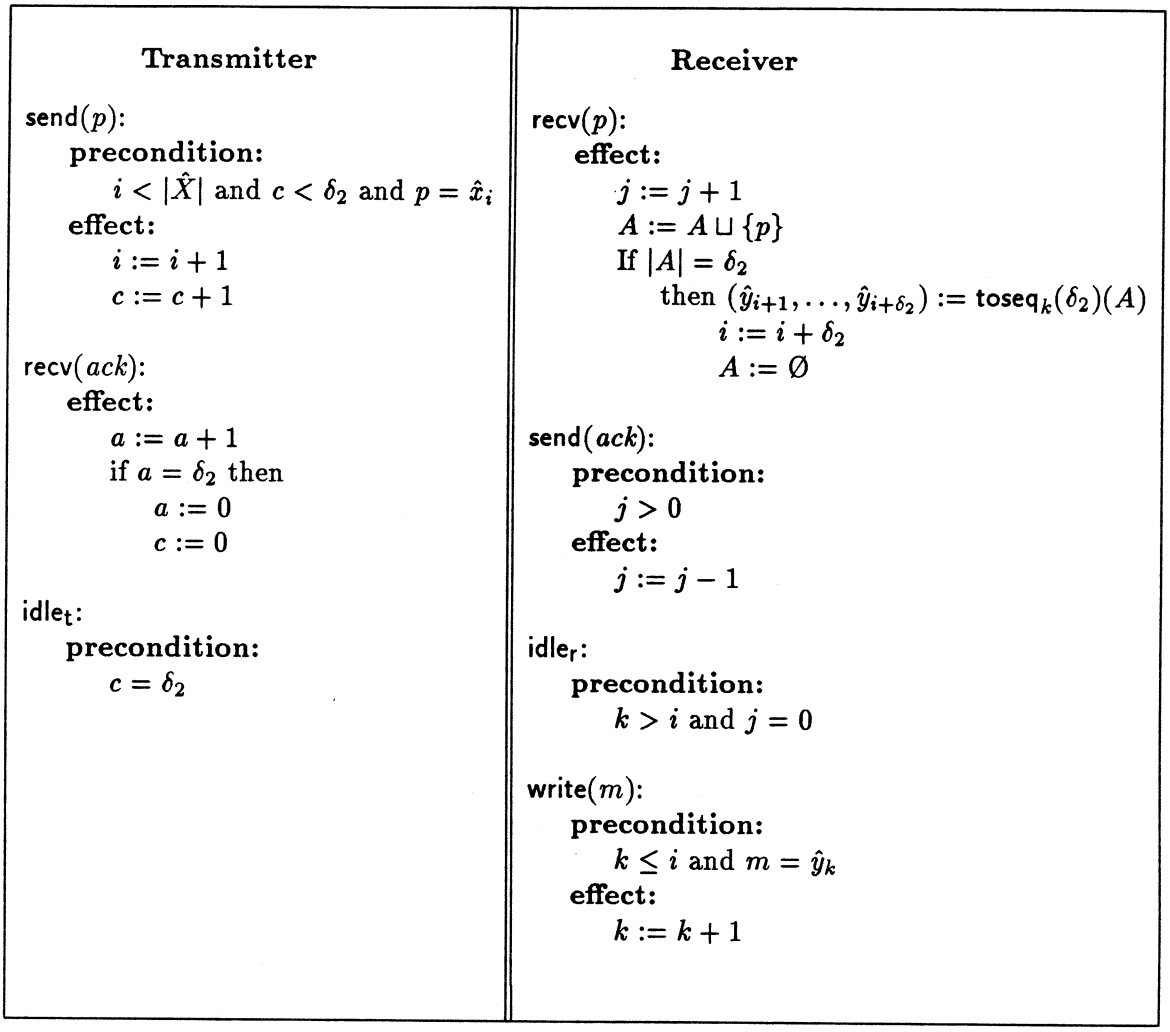

Figure 4: An active solution to *RSTP*

7 Conclusions and Future Work

In this paper we study *RSTP*, a real-time version of STP, where we assume three constants, $c_1 < c_2 \ll d$ such that each process takes step at least every c_1 and at most c_2 units of time, and each packet is delivered up to d units of time after it is sent. We present a notion of *effort*, which intuitively is the average length of time it takes the receiver to learn a single message. We then investigate the effortof solutions to *RSTP* as a function of c_1, c_2, d and k—the size of the transmitter's packet alphabet. We distinguish between r-passive solutions where the receiver sends no packets, and general (active) solutions.

In particular, we show that the effort of every r-passive solution to *RSTP* is $\Omega(\delta_1 c_2 / \log(\mu_k(\delta_1))$, where $\delta_1 = d/c_1$ and $\mu_k(\delta_1)$ is the number of multisets of δ_1 elements over a universe of k elements. We also show, for every k, an r-passive solution to *RSTP* whose effort is $\mathcal{O}(\delta_1 c_2 / \log(\mu_k(\delta_1)))$ where

the transmitter's alphabet is of size k.

Similarly, we show that the effort of every active solution to *RSTP* is $\Omega(d/\log(\mu_k(\delta_2))$, where $\delta_2 = d/c_2$ and $\mu_k(\delta_2)$ is the number of multisets of δ_2 elements over a universe of k elements. We also show, for every k, an active solution to *RSTP* whose effort is $\mathcal{O}(d/\log(\mu_k(\delta_2)))$ where the tranmistter's alphabet is of size k (and the receiver's alphabet consists of a single packet).

There are many other ways to impose real-time restrictions on the system. For example, we can replace d by two constants, $d_1 < d_2$, that determine the time range in which a packet is delivered, or we can assume that each process is associated with its own c_1 and c_2, within whose range it takes steps. It would be interesting to see whether our results can be generalized to other such real-time systems.

Acknowledgement:

We would like to thank David Greenberg and Mike Fischer for helpful discussions, Richard Beigel for suggesting an active protocol, Stan Eisenstat for pointing out to us that our "lim" should have been "suplim", and Drew McDermott for suggesting "effort" instead of "efficiency".

References

[AAF+91] Y. Afek, H. Attiya, A. Fekete, M. J. Fischer, N.A. Lynch Y. Mansour, D. Wang, and L. D. Zuck. Reliable communication over unreliable channels. Manuscript, 1991.

[ADLS90] H. Attiya, C. Dwork, N. A. Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. Manuscript, October 1990.

[AL89] H. Attiya and N. A. Lynch. Time bounds for real-time process control in the presence of timing uncertainty. In *Proc. 10th IEEE Real-Time Systems Symposium*, pages 268–284, 1989. To appear in Information and Computation.

[AUWY82] A. V. Aho, J. D. Ullman, A. D. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Comp. & Maths. with Appls.*, 8(3):205–214, 1982. This is a later version of [AUY79].

[AUY79] A. V. Aho, J. D. Ullman, and M. Yannakakis. Modeling communication protocols by automata. In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pages 267–273, 1979.

[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(5):260–261, May 1969.

[Car] D. E. Carlson. Bit-oriented data link control. In P. E. Green, editor, *Computer Network Architecture and Protocols.* Plenum New York.

[LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, August 1987.

[LT89] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[MMT90] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. Manuscript, August 1990.

[MS89] Y. Mansour and B. Schieber. The intractability of bounded protocols for non-FIFO channels. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 59–72, August 1989.

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

[TL90] Ewan Tempero and Richard E. Ladner. Tight bounds for weakly bounded protocols. In *Proc. 9th ACM Symp. 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 *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 73–83, August 1989.

[Zim80] H. Zimmermann. OSI reference model—the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communication*, COM–28:425–432, April 1980.