The Impact of Time on the Session Problem

Injong Rhee Jennifer L. Welch
Department of Computer Science
CB 3175 Sitterson Hall
University of North Carolina at Chapel Hill
Chapel Hill, N.C. 27599-3175

Abstract

The *session* problem is an abstraction of synchronization problems in distributed systems. It has been used as a test-case to demonstrate the differences in the time needed to solve problems in various timing models, for both shared memory (SM) systems [2] and message-passing (MP) systems [4]. In this paper, the session problem continues to be used to compare timing models quantitatively. The session problem is studied in two new timing models, the *periodic* and the *sporadic*. Both SM and MP systems are considered. In the periodic model, each process takes steps at a constant unknown rate; different processes can have different rates. In the sporadic model, there exists a lower bound but no upper bound on step time, and message delay is bounded. We show upper and lower bounds on the time complexity of the session problem for these models. In addition, upper and lower bounds on running time are presented for the semi-synchronous SM model, closing an open problem from [4]. Our results suggest a hierarchy of various timing models in terms of time complexity for the session problem.

1 Introduction

Early work in distributed computing usually assumed one of two extreme timing models: either the completely synchronous model, in which processes operate in lockstep rounds of computation, or the completely asynchronous, in which there are no upper bounds on process step time or message delay. Since both of these timing assumptions are often unrealistic, researchers began to investigate the impact on distributed computing if those timing assumptions are relaxed or tightened to some extent in order to reflect the real time situation. This question has been studied for a variety of problems, including Byzantine agreement [7, 8, 1, 13], mutual exclusion [3], leader election [5], transaction commit [6], and the session problem [2, 4].

The (s,n)-*session* problem, first presented in [2], is an abstraction of the synchronization needed to solve many distributed computing problems. Therefore, it is an important tool for understanding the behavior of distributed systems under different timing constraints. Informally, a *session* is a minimal-length computation fragment that involves at least one "synchronization" step by every process in a distinguished set of n processes. An algorithm that solves the (s,n)-session problem must guarantee that in every computation there are at least s disjoint sessions and eventually all the n processes become idle.

We study the problem in two different interprocess communication models: *shared memory* and *message passing*. In the shared memory model, processes communicate only by means of shared variables. Each variable is shared by no more than b processes, where b is a constant relative to the total number of processes. In the message passing model, communication is done by exchanging messages across a network. A process can broadcast a message at a step; the message is guaranteed to be delivered to every process after some finite time.

The relevant timing aspects of a model are the

0

lower bound on process step time, c_1, the upper bound on process step time, c_2, and additionally, for the message passing model, the lower bound on message delay, d_1, and the upper bound on message delay, d_2. The running time of an algorithm for the (s, n)-session problem is the maximum time, over all computations, until all the n processes become idle. If there is an upper bound in real time on c_2 and d_2, then it makes sense to measure the running time in terms of real time. If not, then the common way to measure the running time is with rounds. A *round* is a minimal computation fragment in which every process takes at least one step.

Arjomandi, Fischer and Lynch [2] studied the (s, n)-session problem in synchronous and asynchronous shared memory models. *Synchronous* means that $c_1 = c_2$, a finite number. *Asynchronous* means that c_1 and c_2 are infinite. Their results showed a significant time complexity gap between the synchronous and asynchronous models, namely that s rounds are sufficient for the synchronous case but $(s-1)\lfloor \log_b n \rfloor$ rounds are necessary for the asynchronous case, where n is the size of the distinguished set of processes. The implication is that no communication is needed at all in the synchronous case, but it is needed for every session in the asynchronous case. (The $\lfloor \log_b n \rfloor$ factor is essentially the cost of communication when no more than b processes can access any shared variable.)

Attiya and Mavronicolas [4] addressed the problem in semi-synchronous and asynchronous message passing systems. *Semi-synchronous* means that $c_1 > 0$, c_2 and d_2 are finite, and these constants are known to the processes. They modeled the asynchronous system differently than [2]: they let $c_1 = 0$ and $d_1 = 0$, while c_2 and d_2 are finite. Their results also indicated an important time separation between semi-synchronous and asynchronous networks, again based on whether or not communication is necessary.

We present almost matching upper and lower bounds for the session problem in the semi-synchronous shared memory model. Our bounds are similar to those in [4] for the message passing model when the cost for information propagation in the shared memory model is substituted for the message delay. They indicate that if the time for one communication is less than that for one step multiplied by the ratio of c_2 and c_1, the model behaves like the asynchronous; otherwise it behaves like the synchronous (inflated by the ratio). Mavronicolas [12] has also independently developed the same lower and upper bounds for the shared memory semi-synchronous model.

We introduce two new timing models for the (s, n)-session problem: the periodic and the sporadic. In the *periodic* model, for each process there exists an unknown constant such that the process makes one step at every period of the constant. In the message-passing variant, d_2 is finite and known. The upper bounds for both the shared memory and message passing models are the time for the slowest process to take s steps plus the time for one communication. The lower bounds for both are the maximum of the time for the slowest process to take s steps and approximately the time for one communication. Our results indicate that the periodic model, which requires one communication, falls in between the synchronous and asynchronous models, which require no and $s-1$ communications respectively.

In the *sporadic* model, there exists a nonzero lower bound c_1, but no upper bound, on the time between any two consecutive steps of any process. The sporadic shared memory model is essentially equal to the asynchronous shared memory model and is not considered. For the message passing model, the message delay is within $[d_1, d_2]$, where $d_1 \geq 0$, d_2 is finite, and both are known. The combination of the lower bound on step time and upper bound on message delay allows processes to make inferences about the computation, namely, that enough time has elapsed so that a message must have arrived. The lower bound on the per-session time is $\max\{\lfloor \frac{u}{4c_1} \rfloor \cdot K, c_1\}$, where $u = d_2 - d_1$ and $K = \frac{2d_2 c_1}{d_2 - u/2}$. The upper bound on the per-session time is $\min\{(\lfloor \frac{u}{c_1} \rfloor + 3) \cdot \gamma + u, d_2 + \gamma\}$, where γ is the largest step time by a process before termination. As the message delay approaches a constant (i.e., d_1 approaches d_2), the per-session time becomes $\max\{0, c_1\} = c_1$ for the lower bound and $\min\{3\gamma, d_2 + \gamma\} = O(\gamma)$ for the upper bound, which is like the synchronous model.

As the message delay fluctuates within a bigger interval (i.e., d_1 approaches 0), the per-session time becomes $\max\{d_2, c_1\} = d_2$ for the lower bound and $\min\{(\lfloor \frac{d_2}{c_1} \rfloor + 3) \cdot \gamma + d_2, d_2 + \gamma\} = O(d_2 + \gamma)$ for the upper bound, which is like the asynchronous model.

These two timing constraints are inspired by constraints with the same names commonly used in many real-time problems, especially in scheduling of real time tasks for a uniprocessor [9, 10, 11] where the period of task occurrences conforms to the constraints. In practice, as quoted in [10], periodic timing constraints are used in applications such as avionics and process control when accurate control requires continual sampling and processing of data. The sporadic timing constraint is associated with event-driven processing such as responding to user inputs or non-periodic device interrupts; these events occur repeatedly, but the time interval between consecutive occurrences varies and can be arbitrarily large. Therefore, the sporadic timing constraint models processes that can be blocked for an arbitrarily long (but finite) time waiting for a certain condition to be true or a certain event to occur, but that cannot make two consecutive steps faster than a certain lower bound.

Table 1 summarizes the bounds. L means lower bound, U means upper. In the periodic model, c_{min} and c_{max} are the smallest and the largest step times respectively of all processes. The bounds for the asynchronous shared memory case are in rounds. The bounds from [4] have been converted in three aspects for purposes of comparison: (1) That paper considers point-to-point networks; thus the results include a factor of the network diameter. In our model, d_2 subsumes the diameter factor; we have replaced all occurrences of the diameter factor with 1. (2) In [4], the constant 1 is used as the value of c_2; we have replaced all appropriate occurrences of 1 with c_2. (3) [4] assumes that all processes take their synchronized first steps at time 0, resulting in one session at time 0; although we assume that all processes start at time 0, we don't assume that all take a synchronized step at time 0. We rather assume that all steps (including the first step) obey the timing constraints of a specific model starting time 0.

Our results indicate that the periodic model is more efficient than the semi-synchronous system when $c_{max} = c_2$, $2c_1 < c_2$ and n is constant relative to s. The lower bound for the sporadic system and the upper bound for the periodic system suggest that the periodic system is more efficient than the sporadic system if c_{max} is smaller than $\lfloor \frac{u}{4c_1} \rfloor \cdot K \leq \frac{d_2 u}{2(d_2 - u/2)}$ and n is constant relative to s. In shared memory, the sporadic system is clearly less efficient than the semi-synchronous one, but the relationship between the sporadic and the semi-synchronous systems for message passing is rather unclear and understanding it requires further study.

The rest of the paper is organized as follows. Section 2 contains the definition of system models and Section 3 describes how to accomplish communication in the shared memory model. Section 4 concerns the periodic model, Section 5 the shared memory semi-synchronous, and Section 6 the message-passing sporadic. Please note that our lower bound proof technique combines those in [2, 4]. Some proofs are omitted or only sketched due to space constraints.

2 Definitions

2.1 Systems

The generalized system model definition for shared memory and message passing models is similar to that defined in [2].

There are finite sets P of processes and X of shared variables. A *process* consists of a set of internal states, including an initial state. Each *shared variable* has a set of values that it can take on, including an initial value. A *global state* is a tuple of internal states, one for each process, and values, one for each shared variable. The *initial* global state contains the initial state for each process and the initial value for each shared variable. A *step* π consists of simultaneous changes to the state of some process and the values of some number of variables, depending on the current state of that process and current values of the variables. More formally, we represent the step π with a tuple $((s,p,r),(u_1,x_1,v_1),\ldots(u_k,x_k,v_k))$, where s and r are old and new states of a process $p \in P$; u_i and v_i are old and new values of a shared variable $x_i \in X$ for all i. We say that step π is *applicable* to a global state if p is in state s and x_i has value u_i for all i in the global state.

A *system* is specified by describing P, X, and set Σ of possible steps. For all processes $p \in P$ and all global states g, there must exist some step involving process p that is applicable to global state g. A *computation* of a system is a sequence of steps $\pi_1, \pi_2, \ldots$ such that: (1) π_1 is applicable to the initial global state, (2) each subsequent step is applicable to the global state resulting from the previous steps, and (3) if the sequence is infinite, then every process takes an infinite number of steps. That is, there is no process failure. A *timed computation* of

Model		Shared Memory		Message Passing	
Sync.	L	$s \cdot c_2$	[2]	$s \cdot c_2$	
	U	$s \cdot c_2$	[2]	$s \cdot c_2$	
Periodic	L	$\max\{s \cdot c_{max}, \lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}\}$		$\max\{s \cdot c_{max}, d_2\}$	
	U	$s \cdot c_{max} + O(\log_b n) \cdot c_{max}$		$s \cdot c_{max} + d_2$	
Semi-sync.	L	$\min\{\lfloor \frac{c_2}{2c_1} \rfloor \cdot c_2, \lfloor \log_b n \rfloor \cdot c_2\} \cdot (s-1)$		$\min\{\lfloor \frac{c_2}{2c_1} \rfloor \cdot c_2, d_2 + c_2\} \cdot (s-1)$	[4]
	U	$\min\{(\lfloor \frac{c_2}{c_1} \rfloor + 1) \cdot c_2, O(\log_b n) \cdot c_2\} \cdot (s-1) + c_2$		$\min\{(\lfloor \frac{c_2}{c_1} \rfloor + 1) \cdot c_2, d_2 + c_2\} \cdot (s-1) + c_2$	[4]
Sporadic	L	See Async. SM		$\max\{\lfloor \frac{u}{4c_1} \rfloor \cdot K, c_1\} \cdot (s-1)$	
	U	See Async. SM		$\min\{(\lfloor \frac{u}{c_1} \rfloor + 3) \cdot \gamma + u, d_2 + \gamma\} \cdot (s-1) + \gamma$	
Async.	L	$(s-1) \cdot \lfloor \log_b n \rfloor$	[2]	$(s-1) \cdot d_2$	[4]
	U	$(s-1) \cdot O(\log_b n)$	[2]	$(s-1) \cdot (d_2 + c_2) + c_2$	[4]

Table 1: Bounds for the Session Problem

a system is a computation $\pi_1, \pi_2, \ldots$ together with a mapping T from positive integers to real numbers that associates a real time with each step in the computation. T must be nondecreasing and, if the computation is infinite, increase without bound. We will abuse notation and let $T(\pi_i)$ indicate the time at which step π_i occurs.

2.1.1 Shared Memory Model (SMM)

We specialize the general system into the shared memory system in which processes communicate with each other by means of shared variables. Each step π has the property that $k = 1$. That is, it involves only one shared variable. A process can read and write a shared variable in a single atomic step, and we don't assume any upper bound on the size of the variables. We let b be the maximum number of processes that access any single variable, in all the steps of the system. We assume b is constant relative to the number of processes.

2.1.2 Message Passing Model (MPM)

We specialize the general system into the message passing system, in which processes communicate with each other by exchanging messages. $P = R \cup \{N\}$, where R is the set of *regular* processes and N is the network. The network schedules the delivery of messages sent among the regular processes. $X = \{net\} \cup \{buf_p : p \in R\}$, where the values taken on by each variable are sets of messages. net models the state of the network, i.e., the set of messages in transit. buf_p holds the set of messages that have been delivered to p by the network but not yet received by p.

A step of a process p in R consists of p receiving the set M of messages in its buffer buf_p, and based solely on those message and its current state, changing its local state and sending out some message m to all the regular processes. The result is to set buf_p to empty and to add (m, q) to net, for all q in R. So, the step involves two shared variables, buf_p and net. A step of N is to deliver some message of the form (m, q) in net to q. The result is to remove (m, q) from net and add m to buf_q. Accordingly, the step also involves two shared variables, net and buf_q.

This definition of the MPM is an abstract model of a reliable strongly connected network with any topology.

In a timed computation, each message has a *delay*, defined to be the difference between the time of the step that adds it to net and the time of the step that removes it from net. If the message is never removed, then it has infinite delay. The delay only counts the time in transit in the network and does not include the time that the recipient takes to receive the message. That is, the time elapsed between the delivery step and the step which finally removes the message from the buffer is not counted toward the message delay, even if the message remains in the buffer for a long time before the recipient picks it up from its buffer.

2.2 The Real Time Constraints

For each timing model considered, we define the set of *admissible* timed computations to consist of timed computations which obey the stated condition on the step times of all processes in the SMM (all regular processes in the MPM) and, additionally for the MPM, the stated condition on the message delay.

Synchronous There exist constants c_2 and d_2 such that in every timed computation, for every p in P (p in R for MPM), the time between every pair of consecutive steps of p is c_2, and the delay of every message is d_2. Thus c_2 and d_2 are "known" to the processes and can be used in algorithms.

Asynchronous In every timed computation, every process takes an infinite number of steps and every message is eventually delivered.

Periodic There exists a constant d_2 such that in every timed computation, for every p_i in P (p_i in R for MPM), there exists constant c_i such that the time between every pair of consecutive steps of p_i is c_i, and the delay of every message is in $[0, d_2]$. Thus the c_i's are unknown but d_2 is known.

Semi-Synchronous There exist constants $c_1 > 0$, c_2 and d_2 such that in every timed computation, for every p in P (p in R for MPM), the time between every pair of consecutive steps of p is in $[c_1, c_2]$ and the delay of every message is in $[0, d_2]$. Thus c_1, c_2 and d_2 are known.

Sporadic There exist constants c_1, d_1, and d_2 such that in every timed computation, for every p in P (p in R for MPM), the time between every pair of consecutive steps of p is at least c_1, and the delay of every message is in $[d_1, d_2]$. Thus c_1, d_1, and d_2 are known.

2.3 The Session Problem

We now state the conditions that must be satisfied for a system to *solve the* (s, n)*-session problem* (also called an (s, n)-session algorithm).

(1) Each process in P (in R for the MPM) has a subset of *idle* states. The set Σ of steps of the system guarantees that once a process is in an idle state, it always remains in an idle state.

(2) There is a distinguished set Y of n variables called *ports*; Y is a subset of X in the SMM and the set of buf's in the MPM. There is a unique process in P (in R for the MPM) corresponding to each port, which is called a *port process*.

(3) Let p be a port process which corresponds to a port y. A *port step* is any step involving p and y. A *session* is any minimal sequence of steps containing at least one port step for each port in Y. In every admissible timed computation, there are at least s disjoint sessions and eventually all port processes are in idle states.

In the timing models with finite upper bounds on step time and message delay, we measure the running time of an algorithm in real time as follows. An algorithm *runs in time* t if, for every admissible timed computation, every process is in an idle state by time t. In the case of the asynchronous and sporadic models, step time and/or message delay is unbounded (but finite). For these cases, we measure the running time in rounds [14, 2, 4]. A *round* is a minimal-length computation fragment in which every process appears at least once. An algorithm *runs in* r *rounds* if, in every admissible timed computation C, the prefix of C before all processes are idle consists of at most r disjoint rounds. It is also informative in these models to express the time complexity of an algorithm in terms of a new parameter γ, the largest step time during the computation of the algorithm before all the processes are idle. The values of γ is dependent on a particular computation of the algorithm. This type of per-computation based time complexity measure is also used in [1].

3 Communication in SMM

In describing our algorithms, we use a subroutine called *broadcast* as a generic operator for communication in both of the communication models.

In the MPM, the broadcasting of message m by process p is taken care of by the network. It takes at most $d_2 + c_2$ time for a message to be received by all processes in the MPM.

However, the communication in the SMM is constrained by the number of processes which can access a shared variable. Therefore, broadcasting in the SMM involves relaying messages from process to process by means of shared variables.

In a *b-bounded* shared memory system, we can build a tree networks of processes and shared variables by making port and port processes the leaves of the tree. This network can accomplish the necessary communication to propagate a peice of information originaing from a process to all other processes in $O(\log_b n)$ steps.

In this paper, when we say *broadcast* in the SMM, it implies all the interaction of processes in the tree network to accomplish the broadcasting. Throughout this paper, we only describe the role of port processes in an algorithm and assume that broadcast encapsulates the interactions among port processes and other processes which participate in the tree-network communication. In addition, we use the term "step" interchangeably with "port step"; when necessary, we make the proper distinction.

4 The Periodic Model

The periodic model and the synchronous model are similar in that a process takes steps at regular time intervals, yet they differ from each other in that there is no bound on the relative speed of processes in the periodic model. We first present an algorithm $A(p)$ for the (s,n)-session problem in this model and then show that for all periodic algorithms which solve the (s,n)-session problem, there exists a computation of A which takes at least $\max\{s \cdot c_{max}, d_2\}$ time for the MPM and $\max\{s, \lfloor \log_b n \rfloor\} \cdot c_{max}$ for the SMM.

Algorithm $A(p)$: (This algorithm runs in the MBM and the SMM.) Each port process accesses its own port $s-1$ times and at its $s-1$th step, broadcasts the fact. It enters an idle state after it hears that all other processes have taken $s-1$ steps and it has taken at least one more port step.

Theorem 4.1 *$A(p)$ solves the (s,n)-session problem in time $s \cdot c_{max} + d$ in the MPM and time $s \cdot c_{max} + O(\log_b n) \cdot c_{max}$ in the SMM, where $c_{max} = \max\{c_i : p_i \in P\}$.*

Theorem 4.2 *No MP periodic algorithm for the (s,n)-session problem runs in time less than $\max\{s \cdot c_{max}, d_2\}$.*

Theorem 4.3 *No SM periodic algorithm for the (s,n)-session problem runs in time less than $max\{s \cdot c_{max}, \lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}\}$.*

Proof: Suppose that $s \cdot c_{max} \geq \lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}$. Since all processes must take at least s steps to have s sessions, $s \cdot c_{max}$ is obviously the lower bound.

Suppose that $s \cdot c_{max} < \lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}$. By way of contradiction we assume that there exists an algorithm A which solves the (s,n)-session problem in the periodic SMM in time Z strictly less than $\lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}$. We prove that there exists an infinite admissible computation of A that contains less than s sessions.

Let (α, T) be the admissible timed computation in which processes take steps in round robin order and each process's ith step occurs at time $i \cdot c_{min}$ Each consecutive group of steps for p_1 through $p_{|P|}$ is a round. (Round i occurs at time $i \cdot c_{min}$ and consists of the i th step of each process.) Since all processes should enter idle states by time Z in α and all the step time periods are equal to c_{min} in (α, T), there are at most $r = \lfloor Z/c_{min} \rfloor$ rounds required until termination in α.

We will perturb (α, T) in order to get a new admissible timed computation (α', T'). We will prove that there exists at least one port process in (α', T') which enters an idle state before another port process takes any step, resulting in an admissible computation that contains less than s sessions.

Fix any port process p' and change p''s step time period to be $\lfloor \log_{2b-1}(2n-1) \rfloor \cdot c_{min}$. Run A with this modified set of processes to get a new timed admissible computation (α', T').

We define a *subround* to be a minimal computation fragment of α' that involves all processes except p'. A variable v is *contaminated* in subround k of α' if there exists $j \leq k$ and process $p \neq p'$ such that v's value in the global state of α' following p's step on subround j is not equal to v's value in the global state of α following p's step in round j. We define no variable to be contaminated in subround 0. A process p is *contaminated* in subround k of α' if $p \neq p'$ and there exists $j \leq k$ such that in subround j of α', p accesses a variable that is contaminated in subround j. We define no processes to be contaminated in subround 0.

Let $P(t)$ be the set of processes that are contaminated in subround t, and let $V(t)$ be the set of variables that are not contaminated in subround $t-1$ but are contaminated in subround t. Let P_t and V_t satisfy the recurrence equations: $P_0 = V_0 = 0$, $V_t = 2 \cdot P_{t-1} + 1$, and $P_t = (b-1) \cdot V_t + P_{t-1}$.

Lemma 4.4 *$|P(t)| \leq P_t$ and $|V(t)| \leq V_t$ for $0 \leq t \leq r$, where $r = \lfloor Z/c_{min} \rfloor$.*

Proof: By induction on t. The key points are that p' contributes at most one variable to $V(t)$, while each contaminated process contributes at most two. Also, in the worst case a process becomes contaminated as soon as possible, processes only become contaminated due to the variables that just become contaminated, and each variable contaminates at most $b-1$ other processes. ∎

Soving the recurrence equation, we get

$$P_t = \frac{(2b-1)^t - 1}{2}.$$

Thus the total number of processes that are contaminated in subround r is at most $n-1$.

Since less than n processes are contaminated in subround r, at least one port process $p \neq p'$ is in the same state at the end of subround r in α' as it is at the end of round r in α — an idle state. But p' has not taken a step yet. Thus (α', T') is an admissible timed computation that contains less than s sessions. Contradiction. (Note that $\log_{2b-1}(2n-1)$ approaches $\log_b n$ as b and n increase.) ∎

5 Semi-Synchronous Model

In this section, we address the upper and lower bounds in the semi-synchronous shared memory model. The semi-synchronous algorithm in [4] can be adapted to work in the shared memory semi-synchronous model simply by replacing the communication primitives (send and receive) with the explicit propagation of information through the tree network of shared variables using the *broadcast* subroutine described in Section 2.

The proof of the lower bound for the semi-synchronous SMM is rather complicated, because the propagation of information relies on reading and writing shared variables, and also because computations constructed in the proof must satisfy the real time constraints.

Theorem 5.1 *There is no semi-synchronous algorithm which solves the (s,n)-session problem in the SMM within time strictly less than* $\min\{\lfloor \frac{c_2}{2c_1} \rfloor, \lfloor \log_b n \rfloor\} \cdot c_2 \cdot (s-1)$.

Proof: Let $B = \min\{\lfloor \frac{c_2}{2c_1} \rfloor, \lfloor \log_b n \rfloor\}$.

If $c_2 \leq 2c_1$, then $B \leq 1$ and it is obvious that the bound holds since every process must take at least s steps to have s sessions.

Suppose $c_2 > 2c_1$. Assume, by way of contradiction, that there exists a semi-synchronous algorithm, A, which solves the problem in SMM within time Z strictly less than $B \cdot c_2 \cdot (s-1)$. Then $\lceil Z/(B \cdot c_2) \rceil \leq (s-1)$.

Let (α, T) be the admissible timed computation in which processes take steps in round robin order and each process' ith step occurs at time $i \cdot c_2$. Each consecutive group of steps for p_1 through $p_{|P|}$ is a round.

There are $t = \lceil Z/c_2 \rceil$ rounds required until termination in α. Let $\alpha = \beta\gamma$, where β contains the first t rounds of α.

Following the proof of Theorem 1 in [2], we will show that there is a reordering β' of β that results in the same global state as β but that contains at most $s-1$ sessions. Thus $\alpha' = \beta'\gamma$ is a computation with at most $s-1$ sessions. We then will show how to time the events in α' to produce an admissible timed computation (α', T') with at most $s-1$ sessions, a contradiction.

We construct a partial order $\leq_\beta$ on the steps in β, representing dependency. Let $\sigma \leq_\beta \tau$ for every pair of steps σ and τ in β, and say τ is *dependent* on σ, if $\sigma = \tau$ or if σ precedes τ in α and σ and τ either involve the same process or involve the same variable. Close $\leq_\beta$ under transitivity. The following claim is not difficult to prove.

Claim 5.2 *$\leq_\beta$ is a partial order, and every total order of steps of β consistent with $\leq_\beta$ is a computation which leaves the system in the same global state as β does.*

Let $\beta = \beta_0, \beta_1, \ldots, \beta_m$ where $m = \lceil Z/(B \cdot c_2) \rceil$. Each β_k (except possibly the last one) consists of B rounds. Let y_0 be an arbitrary port in Y. For all k, $1 \leq k \leq s-1$, we show that there exists a port y_k and two sequences of steps ϕ_k and ψ_k, such that the following properties hold. (p_{y_i} is the corresponding port process to y_i, $1 \leq i \leq s-1$.)

(i) $\phi_k \psi_k$ is a total ordering of the steps in β_k, consistent with $\leq_\beta$.

(ii)ϕ_k does not contain any step by process $p_{y_{k-1}}$ which accesses y_{k-1}.

(iii)ψ_k does not contain any step by process p_{y_k} which accesses y_k.

Then define β' to be $\phi_1\psi_1\phi_2\psi_2 \ldots \phi_m\psi_m$.

For all k, $1 \le k \le m$, y_k, ϕ_k, and ψ_k are defined inductively. If there is some port variable that is not accessed by any step in β_k, then let y_k be that port, ϕ_k the empty sequence, and $\psi_k = \beta_k$. Otherwise (every port variable is accessed in β_k), let τ_k be the first step in β_k that accesses y_{k-1}. As a consequence of a very general result proved in [1], there exists a port variable y_k such that:

(iv) it is not the case that $\tau_k \le_\beta \sigma_k$, where σ_k is the last step in β_k that accesses y_k.

We now assign times (the mapping T') to every step in β_k and then let β'_k be any total ordering of the steps in β_k consistent with the times. We then define ϕ_k and ψ_k.

- For each process $p \in P$, if there are some steps of p in β_k which σ_k is dependent on, we let π be the step that occurred last among them. We retime π and all the steps of p that happened earlier than π such that the first step of p in β_k occurs at $2c_1B(k-1)+c_1$, the next step occurs c_1 time later, and so on.
- For each process p, if there are some steps of p in β_k which are dependent on τ_k, we let π be the step that occurred first among them. We retime π and all the steps of p that occurred later than π such that the last step of p in β_k occurs at $2c_1Bk$, the step before that occurs c_1 time earlier, and so on.
- All other steps in β_k are assigned the same time as they are under T (the original timing).

Let ϕ_k be all the steps that happened up to $time(\sigma_k)$ including σ_k, and let ψ_k be the remainder.

Lemma 5.3 *β' is consistent with $\le_\beta$.*

Proof: For any k, $1 \le k \le s-1$, pick any two steps, π and π' in β_k such that $\pi \le_\beta \pi'$. Thus $T(\pi) \le T(\pi')$. (Recall that T is the original timing.) We only need to prove that $T'(\pi) \le T'(\pi')$, where T' is the new timing.

Each of π and π' belongs to either ϕ_k or ψ_k and is either retimed or not. If π is retimed in ϕ_k and π' is not retimed in ϕ_k, then $T'(\pi) \le T'(\pi')$ since $T'(\pi) \le T(\pi)$ and $T'(\pi') \ge T(\pi')$. All other cases can be proved similarly. ∎

Lemma 5.4 *(β', T') is admissible.*

Proof: We need to prove that all steps in (β', T') satisfy the real time constraint imposed on the semi-synchronous model.

By the construction, no two consecutive steps by a process in the system are closer than c_1 in (β', T'); therefore, the lower bound on step time is preserved.

We now show that the maximum time between any two consecutive steps of a process is c_2. Let π and π' be two consecutive steps of some process p. First assume that π and π' both occur in β_k for some k, π is the i-th step of p in β_k and π' is the $i+1$-st. If π and π' are both in ϕ_k or are both in ψ_k, then either there is no change in their times or they are retimed to be c_1 apart.

Now suppose π is in ϕ_k and π' is in ψ_k. By construction,

$$
\begin{aligned}
& T'(\pi') - T'(\pi) \\
&\quad = \quad B \cdot c_1 + c_1 \\
&\quad = \quad \min\{\lfloor \frac{c_2}{2c_1} \rfloor, \log_b n\} \cdot c_1 + c_1 \\
&\quad \le \quad \frac{c_2}{2} + c_1 .
\end{aligned}
$$

Since $c_1 < c_2/2$, this difference is less than c_2.

Now suppose π occurs in β_{k-1} and π' occurs in β_k. In the worst case, π is retimed to occur at $x - c_2/2$ and π' is retimed to occur at $x + c_2/2$, where x is the time at the end of β_{k-1}. (This is true by the definition of B.) So the time between π and π' is at most c_2. ∎

Lemma 5.5 *β' contains less than s sessions.*

Lemma 5.5 is true because of the way ψ_{k-1} and ϕ_k are defined. The theorem now follows. ∎

6 The Sporadic Model

In the MPM, a lower bound c_1 on step time and lower and upper bounds $[d_1, d_2]$ on message delay are imposed. The correctness of our sporadic algorithm $A(sp)$ depends on the following observation: If a process p_i receives a message m from a process p_j at time t, then the message must have been sent no later than $t - d_1$, because it takes at least d_1 time for a message to be delivered. All the messages received by p_i after $t + d_2 - d_1$ must have been sent after m was, because it takes at most d_2 time for a message to be delivered.

Using the above fact, each process broadcasts a message at every step carrying its knowledge on the number of sesssions happened by the time that the step occurs. After receiving a message m which says there are at least $k-1$ sessions in the system, a process waits for $d_2 - d_1$ time. After that, the process waits to receive at least a message from every process. it is clear that there are at least k sessions in the system by the time because every message received after $t + d_2 - d_1$, where t is the time that m is sent, must have been sent after the time there are at least $k-1$ sessions.

We first proceed by presenting the algorithm $A(sp)$. A message is denoted $m(i, V)$, where i is the identifier of a sending process p_i and V is an integer in $[0, s-1]$. We let $*$ be a *don't care* value for either of the fields and $u = d_2 - d_1$.

```
A(sp) for process p_i:
B := ⌊u/c_1⌋ + 1;
count := session := 0;
msg_buf := temp_buf := ∅;
while( session < s − 1 )
    read buf_i and let the set of messages
        obtained be M;
    msg_buf := msg_buf ∪ M;
    if for all j ∈ [n], m(j, session) is in msg_buf
    then                                /* condition 1 */
        count := 0;
        session := session + 1;
    elsif (count > B)
    then
        temp_buf := temp_buf ∪ M;
        if for all j ∈ [n], at least one m(j, *)
                is in temp_buf
        then                        /* condition 2 */
                count := 0;
                session := session + 1;
                temp_buf := ∅;
        end if;
    end if;
    broadcast m(i, session);
    count := count + 1;
end while;
Enter an idle state.
```

Theorem 6.1 *$A(sp)$ solves the (s,n)-session problem within time*
$\min\{\lfloor \frac{u}{c_1} + 1 \rfloor \gamma + (u + 2\gamma), d_2 + \gamma\}(s-2) + d_2 + 2\gamma$.

Proof: Consider an arbitrary admissible timed computation C of $A(sp)$. The following lemma (proof omitted) can be used to prove the theorem.

Lemma 6.2 *There exists at least one step in C in which a process sets its session to k, $0 \le k \le s-1$.*

Let p_{i_k} be the first process which sets $session_{i_k}$ to $k \ge 0$. To increment $session$, a process must receive a message(s) which notifies the process that there is at least one session after the last update to $session$. Let M_k be the set of messages received by p_{i_k} that causes p_{i_k} to set $session_{i_k}$ to k. (We define M_0 to be the empty set.) In more detail: If condition 1 was true, M_k is the set of message $m(j, session_{i_{k-1}})$ for all integers $j \in [n]$ in msg_buf. If condition 2 was true, M_k is the set of messages in $temp\_buf$ at the time. Assuming that m_k is the message which is sent last among M_k, we prove the following lemma.

Lemma 6.3 *Let π be the step which sends m_k. There are at least k sessions by the time π occurs in C.*

Proof: We proceed by induction on k.

For the basis, when $k = 0$, it is always true that there are at least 0 sessions in C.

Inductively when $k > 0$, assuming the lemma is true for $k-1$, we show that when π occurs, there are at least k sessions.

Let τ be the step that sent m_{k-1} and σ be the step in which $p_{i_{k-1}}$ sets $session_{i_{k-1}}$ to $k-1$. For p_{i_k} to update its $session$, one of conditions 1 and 2 in the algorithm must hold.

First, assume that condition 1 is true. According to the algorithm, a process broadcasts a message with a new session value $k-1$ after it sets its $session$ to the new value $k-1$, before which time there were $k-1$ sessions in C because the induction hypothesis dictates that there were $k-1$ sessions in C when m_{k-1} was sent. Because σ is the first step to set $session_{i_{k-1}}$ to $k-1$, all messages in M_k, must have been sent when or after σ occurs. Because all processes take at least one step to send messages in M_k after there were at least $k-1$ sessions, there must be at least k sessions in C by the time that message m_k is sent.

Second, assume that condition 2 is true. Since p_{i_k} is the first process which sets $session_{i_k}$ to k,

$session_{i_k}$ must have taken on $k-1$ according to the proof of Lemma 6.2. Let t be the time when τ occurs at which time m_{k-1} was sent and t' be the time that p_{i_k} sets $session_{i_k}$ to $k-1$. The message m_{k-1} must arrive at $buf_{i_{k-1}}$ at time between $[t+d_1, t+d_2]$ because of the bounds on message delay. Thus, $t'-t \geq d_1$. Since *count* in the algorithm is reset whenever *session* is updated, when $count_{i_k}$ is equal to B most recently before when p_{i_k} sets $session_{i_k}$ to k, say, at time t'', at least time $B \cdot c_1 > u$ must have elapsed since t'. So, $t'' > t' + u \geq t + d$. Therefore, all messages received at t'' or later must be sent after time t, at which time there were $k-1$ sessions by the assumption. Since at least one message is sent by each process after time t, there must be at least one additional step by all processes between time t and the time π occurs. Therefore, there must be at least k sessions by the time π occurs. ∎

To analyze the time complexity of the algorithm $A(sp)$, we use the actual maximum step time γ since in our sporadic model the upper bound on the step time is not available.

We define for each k, $2 \leq k \leq s-1$, $T_k = \max\{t : p_i$ sets $session_i$ to k at time t in C for all $p_i \in R\}$.

Lemma 6.4 *For each k, $2 \leq k \leq s-1$,*
$T_{k+1} \leq T_k + \min\{\lfloor \frac{u}{c_1} + 1\rfloor)\gamma + (u+2\gamma), d_2+\gamma\}$.

Proof: According to the algorithm, a process broadcasts a message at every step. Thus, if process p_i receives a message from process p_j at time t, it will receive at least one more message from p_j by time $\leq t+u+2\gamma$. Let p_{i_k} be the last process to set $session_{i_k}$ to k and $p_{i_{k+1}}$ be the last process to set $session_{i_k}$ to $k+1$. We now look at each of the possible cases which may cause $session_{i_{k+1}}$ to be updated to $k+1$:

If condition 2 is true when $session_{i_{k+1}}$ is updated to $k+1$, $p_{i_{k+1}}$ has made at least $B = \lfloor \frac{u}{c} + 1 \rfloor$ steps since the last update to $session_{i_{k+1}}$. Because a process must wait, since then, at most $u+2\gamma$ time to receive another set of messages sent from all processes, at most $(\lfloor \frac{u}{c} + 1\rfloor)\gamma + (u+2\gamma)$ time has elapsed.

If condition 1 is true when $session_{i_{k+1}}$ is updated to $k+1$, let t be the time at which p_{i_k} broadcasts $m(i_k, k)$; note that by definition, $t = T_k$. Message $m(i,k)$ must be received by $p_{i_{k+1}}$ by time $t + d_2 + \gamma$.

Since both conditions take at most $\min\{\lfloor \frac{u}{c} + 1\rfloor\gamma + (u+2\gamma), d_2+\gamma\}$ time to be true since the last update to *session*, the lemma follows. ∎

From Lemmas 6.2 and 6.3, it follows that there are at least $s-1$ sessions at the time that m_{s-1} is sent. All processes will eventually set their *session*'s to $s-1$ (since *session* can't be bigger than $s-1$). Each process sets *session* to $s-1$ because it receives a certain message. Therefore, there is at least one additional step by all processes after there have been $s-1$ sessions in C. Thus, there are at least s sessions in C.

By the algorithm, initially it takes at most $d_2 + 2\gamma$ to receive at least one message from all processes in order to accomplish the first session. Using Lemma 6.4, it is clear now that it takes at most $\min\{\lfloor \frac{u}{c_1} + 1\rfloor\gamma + (u+2\gamma), d_2+\gamma\}(s-2) + d_2 + 2\gamma$. (This equals $\min\{(\lfloor \frac{u}{c_1}\rfloor + 3)\cdot\gamma + u, d_2+\gamma\}(s-1)+\gamma$ if $d_1 < \lfloor \frac{u}{c_1} + 1\rfloor \cdot \gamma$). ∎

We now prove the lower bound.

Theorem 6.5 *No sporadic algorithm solves the (s,n)-session problem in the MPM within time $< \max\{\lfloor \frac{u}{4c_1}\rfloor \cdot K, c_1\}(s-1)$ where $K = \frac{2d_2c_1}{(d_2 - \frac{u}{2})}$.*

Proof: The general structure of this proof follows that of Theorem 5.1.

Let $B = \lfloor \frac{u}{4c_1} \rfloor$.

When $B \cdot K \leq c_1$, the lower bound holds because a process must execute at least s steps to achieve s sessions.

Suppose that $B \cdot K > c_1$. Assume, by way of contradiction, that there exists a sporadic algorithm, A, which solves the (s,n)-session problem in the MPM within time Z strictly less than $B \cdot K \cdot (s-1)$. Then $\lceil Z/(B \cdot K)\rceil \leq (s-1)$. We show that there exists an admissible timed computation of A which does not include s sessions.

Let (α, T) be the admissible timed computation in which regular processes take steps in round robin order and each process' ith step occurs at time $i \cdot K$, and all message delays are exactly d_2. Each consecutive group of steps for p_1 through p_n is a round.

Therefore, there are $r = \lceil Z/K \rceil$ rounds required until termination in α. Let $\alpha = \beta\gamma$, where β contains the first r rounds of α.

We will show that there is a reordering β' of β that contains at most $s-1$ sessions. Thus $\alpha' = \beta'\gamma$ is a sequence with at most $s-1$ sessions. In order to get an admisssible computation β', we will assign new times (T') to every step in β and let β' be any total ordering of the steps in β consistent with the times, and then we will prove that (β', T') is an admissible timed computation which results in the same global state as β. A contradiction.

Then we will show how to reorder β to produce admissible timed computation (α', T') that results in the same global state as α.

Let $\beta = \beta_1 \dots \beta_m$ where $m = \lceil Z/(B \cdot K) \rceil$. Each β_k, $1 \leq k \leq m$ (except possible the last one) consists of B rounds, and for some sequence $i_0, i_1 \dots i_m$ of integers in $[1, n]$, each computation fragment β_k consists of $\phi_k \psi_k$ such that:

(i) ϕ_k does not contain any step by process $p_{i_{k-1}}$.

(ii) ψ_k does not contain any step by process p_{i_k}.

Then define β' to be $\phi_1\psi_1\phi_2\psi_2 \dots \phi_m\psi_m$. As in Lemma 5.5, we prove that β' has at most $s-1$ session.

Lemma 6.6 *β' has at most $s-1$ sessions.*

Since all processes in γ are in idle states, α' has at most $s-1$ sessions.

We need to show how to reorder every step in β to get an admissible timed computation (α', T') which preserves properties (i) and (ii), and results in the same global state as α.

Let us first assign times (a new mapping T'') to every step, π, in β including all the steps of the network N such that $T''(\pi) = T(\pi) \cdot \frac{2c_1}{K}$. That is, every process except N takes a step at every $2c_1$ and each round occurs at every $2c_1$. Since the delivery steps of N are also remapped, the message delay is reduced to $d_2 \cdot \frac{2c_1}{K} = d_2 - \frac{u}{2}$.

$C' = (\beta, T'')$ is an admissible timed computation because β is a computation, and the step times and message delays obey the sporadic time constraint.

From C', we construct (β', T'), an admissible computation which results in the same global state as β'' (and β). We map T'' to T' in order to get i_k, i_{k-1}, ϕ_k and ψ_k for all k , $1 \leq k \leq m$.

For all k, choose i_k arbitrarily, as long as $i_k \neq i_{k-1}$. For all $0 \leq j \leq m$, let $t_j = B \cdot 2c_1 \cdot j$. t_j is equal to the ending time of β_j in C'.

1. Let π be all steps of p_{i_k} and all the steps of N that deliver messages to p_{i_k} in β_k. Retime π such that $T'(\pi) = t_{k-1} + (T''(\pi) - t_{k-1})/2$.
2. Let σ be all steps of $p_{i_{k-1}}$ and all the steps of N that deliver messages to $p_{i_{k-1}}$ in β_k. Retime σ such that $T'(\sigma) = t_k - (t_k - T''(\sigma))/2$.
3. All other steps in β_k are assigned the same time as they are under T''.

Fix up the states of the network in β so that the state of the network is consistent with all the send and receive steps of regular processes in β. Let ϕ_k be the prefix of β'_k up to the last step of p_{i_k}, and let ψ_k be the remainder. Let β'_k be any total ordering consistent with T'.

We now prove the following lemma:

Lemma 6.7 *(β', T'') is an admissible timed computation which results in the same global state as β.*

Proof: The time period of β_k in C' , $1 \leq k \leq m$, is equal to to $B2c_1$ since β_k in C' consists of B rounds (except the last one) and the step time is c_1. Since no step is retimed outside the time boundary of (β_k, T'), the time period of (β_k, T'') is also equal to (β'_k, T'').

In (β, T''), the message delay of all messages is bigger than $B2c_1$ by the definition of B, so that the messages sent in β_k are never received in β_k. In (β', T'), the messages sent in β'_k are never received in β'_k too because no step is retimed outside the the time boundary of (β_k, T'').

By the construction, in β', the delivery steps of all messages are retimed with the steps that receive the messages, so that every message is delivered always before received. Every step in β' receives the same set of messages as the corresponding step in β does. Since states of processes are updated based only on the current state and the set of message received, β' is a computation which leads to the same global state as β.

Now, to prove that (β', T') is admissible, we need to show that (β', T') obeys the sporadic time constraints.

First, it is clear that every computation step time in β' is bigger than the minimum step time c_1 by the construction.

Second, we need to prove the delay of any message sent in β' is within $[d_2 - u, d_2]$. For any message m sent in β', let π_i be the step of a process in R which sends m and π_j be the step of N which delivers m. We need to prove that $T'(\pi_j) - T'(\pi_i)$ is in $[d_2 - u, d_2]$. Without loss of generality, assuming π_i is in β'_k, $T'(\pi_j) - T'(\pi_i) = T''(\pi_j) - T''(\pi_i) + [T'(\pi_j) - T''(\pi_j)] - [T'(\pi_i) - T''(\pi_i)]$.

It can be proved that for any step π,

$$-u/4 \leq T'(\pi) - T''(\pi) \leq u/4.$$

$T''(\pi_j) - T''(\pi_i) = B \cdot 2c_1 \leq d_2 - u/2$ from the construction of C'. Therefore, it is clear that $T'(\pi_j) - T'(\pi_i)$ is always within $[d_2 - u, d_2]$ in β'. ■

Since there exists an admissible timed computation (β', T') of A which has at most $s-1$ sessions by Lemma 6.7 and Lemma 6.6, this contradicts the assumed existence of algorithm A. Therefore, Theorem 6.5 now follows. ■

Acknowledgments:

We thank Kevin Jeffay for helpful comments. This work was partially supported by an NSF PYI Award CCR-9158478, an IBM Faculty Development Award, and NSF grant CCR-9010730.

References

[1] H. Attiya, C. Dwork, N. Lynch and L. Stockmeyer, "Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty," *Proc. 23rd ACM Symp. on Theory of Computing,* 1991, pp. 359–369.

[2] E. Arjomandi, M. Fischer and N. A. Lynch, "Efficiency of Synchronous versus Asynchronous Distributed Systems," *Journal of the ACM*, Vol. 30, No. 3, 1983, pp. 449–456.

[3] H. Attiya and N. A. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," *Proc. 10th Real-Time Systems Symp.,* Dec. 1989, pp. 268–284.

[4] H. Attiya and M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Networks," *Proc. 28th Allerton Conf. on Communications, Control and Computing*, Oct. 1990, pp. 578–587.

[5] B. A. Coan and G. Thomas, "Agreeing on a Leader in Real-Time," *Proc. 11th Real-Time Systems Symp.,* Dec. 1990.

[6] B. A. Coan and J. L. Welch, "Transaction Commit in a Realistic Timing Model," *Distributed Computing,* Vol. 4, 1990, pp. 87–103.

[7] D. Dolev, C. Dwork and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *Journal of the ACM*, Vol. 34, No. 1, Jan. 1987, pp. 77–97.

[8] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *SIAM Journal of Computing,* Vol. 12, No. 13, Nov. 1983, pp. 656–666.

[9] K. Jeffay, "The Real-Time Producer/Consumer Paradigm: A Paradigm for the Efficient, Predictable Real-Time System," University of North Carolina at Chapel Hill, Department of Computer Science, submitted for publication. April 1991.

[10] K. Jeffay, D. F. Stanat and C. U. Martel, "On Optimal, Non-Preemptive Scheduling of Periodic and Sporadic Tasks," *Proc. 12th IEEE Real-Time Systems Symp.,* Dec. 1991, pp. 129–139.

[11] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46–61.

[12] M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Systems: Atomic Shared Memory," TR-03-92, Aiken Computation Lab., Harvard University, Jan. 1992.

[13] S. Ponzio, "Consensus in the Presence of Timing Uncertainty: Omission and Byzantine Failures," *Proc. ACM Symp. on Principles of Distributed Computing,* Oct. 1991, pp. 125–137.

[14] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System," *Proc. 9th ACM Symp. on Theory of Computing,* 1977, pp. 91–97.