The Impact of Timing Knowledge on the Session Problem*

Injong Rhee† Jennifer L. Welch‡

March 14, 1999

*A preliminary version of this paper appeared as [RW92]. Much of this work was done while the authors were with the Department of Computer Science, University of North Carolina at Chapel Hill. This work was supported by an IBM Faculty Development Award, NSF Presidential Young Investigator Award CCR-9158478, and TAMU Engineering Excellence funds.

†Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7534. Email: rhee@csc.ncsu.edu

‡Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. Email: welch@cs.tamu.edu

Abstract

The *session* problem is an abstraction of fundamental synchronization problems in distributed systems. It has previously been used as a test-case to demonstrate the differences in the time needed to solve problems in several timing models.

The goal of this paper is to compare the computational power of a family of partially synchronous models by studying the time needed to solve the session problem. Four timing parameters are considered: the maximum and minimum process step times and message delays. Timing models are obtained by considering independently whether each parameter is known (i.e., is hard-wired into processes' code) or unknown, giving rise to four shared memory models and 16 message passing models. The models are compared based on the time complexity, measured in real time, of the session problem.

This paper presents a modular proof technique for obtaining asymptotically tight bounds on the time complexity of the session problem for the four shared memory models and the 16 message passing models. Timing information known in each particular model can be exploited by algorithms to count sessions in different ways. This paper reports five different counting algorithms. The matching lower bound for each model suggests that[[]] they are the [[optimal]] ways to count sessions. Based on these bounds, a lattice among unknown parameter models is constructed, which confirms the common belief that as more timing information is known in a model, the model behaves more like a synchronous system.

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 and a message sent in a round is delivered in the next round, or the completely asynchronous, in which there are no bounds on process step time or message delay. However, in most distributed systems, processes operate neither in lock-step nor at completely independent rates. Furthermore, the asynchrony assumption makes it very difficult to design and verify distributed algorithms whereas the perfect synchrony assumption is very expensive, if not impossible, to implement in real distributed systems.

Based on these observations, researchers (e.g., [AAT94, ADLS94, AL89, AM94, CT90, CW90, DDS87, DLS88, Po91]) began to investigate the impact on distributed computing if those timing assumptions are relaxed or tightened to some extent in order to reflect more realistic situations. The new timing models that are obtained by relaxing or tightening the two extreme timing assumptions are called *partially synchronous models*.

The goal of this paper is to compare the computational power of a family of partially synchronous models by studying the time needed to solve a distributed computing problem, called *the session problem*.

1.1 The Session Problem

The (s, n)-session problem was first presented in [AFL83] and further studied in [AM94]. Informally, a *session* is a minimal-length computation fragment that involves at least one special "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.

The (s, n)-session problem is an abstraction of the synchronization used in many distributed computing settings. The (s, n)-session problem, like the mutual exclusion and dining philosophers problems, concerns possible ordering of process events (e.g., a process finishing its assigned task) rather than the computation of particular outputs.

Consider, for example, *barrier synchronization* [GVW89], a fundamental mechanism in concurrent systems which guarantees that all processes have finished a specified task in their execution before any proceeds. Barrier synchronization is a special case of the (s, n)-session problem when $s = 2$, and solutions for the $(2, n)$-session problem can be used to construct barrier synchronization: after each process finishes its specified task, it keeps taking synchronization steps until the $(2, n)$-session algorithm terminates.

As discussed in [AM94], another example of the (s, n)-session problem can be found in a distributed linear equation solver, where each process holds part of the input data (cf. [Ba78]) and iterates to solve equations by relaxation. Each process takes one synchronization step when it changes its data. Sufficient interleavings of synchronization steps by different processes ensure

a correct output since they imply sufficient interaction among the intermediate values computed by the processes.

The (s, n)-session problem is also an astraction of a simple message distribution system in which a sending process writes a sequence of s messages one at a time on a board (e.g., port or mailbox) visible to all and waits after each message until all $n-1$ other processes have read the message before writing the next one. Each reading step by a process is one synchronization step of the process. Any protocol which ensures that the sender has waited sufficiently long solves the (s, n)-session problem.

Since the time complexity of the session problem is very sensitive to the timing assumptions of the underlying model, it has been used as a test-case to demonstrate the theoretical differences in the time needed to solve problems in various timing models [AFL83, AM94, Ma93, RW92]. Using the session problem, we can quantify differences between various models in terms of the time complexity needed to solve distributed computing problems. Precise time complexities for various timing models allow us to show complexity gaps among the models. Time complexity gaps can provide valuable information to system designers in evaluating and comparing the various timing models and deciding what timing guarantees they have to provide or do not have to provide to build efficient, yet cost-effective distributed systems.

A solution for the (s, n)-session problem normally involves several methods to count sessions during execution. In particular, the first or last session is often counted in a different way than the other sessions. Hence, the time complexity of a solution is usually expressed as a function of s, n and some additional terms to account for the complexity of counting the first or last session. When we qualitatively evaluate relative time complexities of different timing models, the term associated with s has more weight than the other terms in deciding the time complexity hierarchy.

1.2 Timing Models

We consider two different interprocess communication models: *shared memory* (SM) and *message passing* (MP). In the shared memory model, processes communicate only by means of shared variables.

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.

Process step time is the amount of time between two consecutive steps of the same process and *message delay* is the amount of time between when a message is sent and when the message is received. The relevant timing parameters of a model are the minimum step time, c_1, the maximum step time, c_2, and additionally, for the message passing model, the minimum message delay, d_1, and the maximum message delay, d_2.

We consider families of timing models for both SM and MP systems. The timing models are obtained by considering independently whether each parameter is known (i.e., can be hard-wired

into processes' code) or unknown, giving rise to four SM models and 16 MP models. Some of these models have been studied previously in both practical and theoretical contexts.

Models with known maximum and minimum step times are commonly called *semi-synchronous models* and have been previously studied for various distributed computing problems, including the consensus problem, the mutual exclusion problem, and the session problem, in the literature (see [AM94, AAT94, ADLS94, AL89, AT92, CT90, LS92, Po91, RW92]). The semi-synchrony models systems where information about timing parameters, such as process step time, is only approximately known, e.g., processes may have access to inaccurate clocks that operate at approximately, but not exactly, the same rate.

Models with unknown step times have been studied for the consensus problem [DLS88] and for the mutual exclusion problem [AAT94]. As those papers argue, these models provide a useful abstraction of the timing constraints in real systems.

Models in which the minimum step time is known, but the maximum step time is unknown abstract event-driven processing such as responding to user inputs or non-periodic device interrupts [RW92]. In these models, processes 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 cannot take two consecutive steps faster than a certain amount of time.

A lower bound result or impossibility result shown for an asynchronous model does not automatically carry over to a model with unknown bounds. For instance, the work on the consensus problem in [DLS88] showed that fault-tolerant consensus can be solved in a model with unknown bounds, although it cannot be solved in an asynchronous system [FLP85]. There is more leeway in constructing "bad" executions in an asynchronous system than there is in one with unknown bounds. Thus, it is worth investigating how knowledge of step time and message delay affects the session problem.

1.3 Previous Work on the Session Problem

The upper and lower bounds on the time required to solve the session problem in an asynchronous shared memory system shown by Arjomandi, Fischer and Lynch [AFL83] demonstrated the first such case where asynchronous systems are less efficient than synchronous systems. In the synchronous model, all processes run in lockstep, while in the asynchronous model, no bounds on process running rates exist. Their result showed an inherent time complexity gap between the synchronous and asynchronous models: s steps are sufficient for s sessions in the synchronous model, i.e., no interprocess communication is needed, but $(s-1)\lfloor \log_a n \rfloor$ steps are necessary for the asynchronous model. The $\lfloor \log_a n \rfloor$ factor is essentially the cost of communication, where a is the maximum number of distinct processes that are ever allowed to access any given shared variable. Thus, one interprocess communication per session is needed in the asynchronous shared memory model.

Attiya and Mavronicolas [AM94] show a similar result for an asynchronous message passing system in which there is a maximum message delay d_2, but the minimum message delay d_1 is

zero. Their results show that the asynchronous message passing model requires $(s-1) \cdot d_2$ time to solve the (s,n)-session problem (at least one message delay per session).

The session problem has been studied in a semi-synchronous model as well, in which there are known minimum and maximum step times and there is a (not necessarily known) maximum message delay. The upper bound in the message passing model shown by Attiya and Mavronicolas [AM94] is $(s-1) \cdot \min\{\frac{c_2}{2c_1}c_2, d_2\}$. A nearly matching lower bound (within a factor of 2 of the upper bound) also appears in [AM94]. These results imply that the efficiency of the semi-synchronous shared memory model lies between those of the synchronous and asynchronous models.

In a *periodic* model where processes run at a fixed unknown periodic rate, nearly matching lower and upper bounds shown by [RW92] indicate that at least one communication is required to solve the session problem. These bounds also indicate that the inherent cost of synchronizing periodically running processes and the existence of time complexity gaps among the synchronous, periodic, and asynchronous timing models.

1.4 Our Results

Our complexity results are organized around "ways to count" s sessions in a computation. The intuition is that processes must have some way to count the passage of other processes' steps in order to "know" when a session has occurred.

Note that $s \cdot c_2$ is an obvious lower bound for all models because each process has to take at least s steps to solve the (s,n)-session problem and each step takes up to c_2 time [AFL83]. We omit from the discussion the obvious lower bound $s \cdot c_2$.

1.4.1 Shared Memory Results

In order for our results to be comparable with prior work, we study shared memory systems with a constant parameter a, which is the maximum number of distinct processes that are ever allowed to access any given shared variable. When a is smaller than the total number of processes in the system, it is not possible for all processes to exchange information in a single step. Instead, information must be propagated from process to process. Thus, as a gets smaller, the amount of propagation required increases. The motivation for this restriction on communication comes ¿from the fact that in a distributed shared memory system, some part of memory is local to a process and can be accessed quickly, while the rest is remote and requires more time for accesses.

Table 1 summarizes our results on the time complexity of solving the (s,n)-session problem in shared memory models.

Our results indicate that if either the minimum or maximum step time (or both) is unknown, then the running time for the (s,n)-session problem is $(s-1) \cdot c_2 \cdot \Theta(\log n)$, i.e., roughly one communication cost $(c_2 \cdot \Theta(\log n))$ is required for each session. On the other hand, if both step times are known, then the running time is $(s-1) \cdot c_2 \cdot \min\{\frac{c_2}{2c_1}, \Theta(\log n)\}$. In this model,

c_1	c_2	Lower bound	Upper bound
unknown	unknown	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
unknown	known	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
known	unknown	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
known	known	$(s-1)\cdot c_2\cdot \min\{\frac{c_2}{2c_1}, \log_a n\}$	$(s-1)\cdot c_2\cdot \min\{\frac{c_2}{2c_1}, \Theta(\log n)\}$

Table 1: Time bounds for the (s,n)-session problem in shared memory models; a is the maximum number of processes that can access a shared variable, c_1 and c_2 are the minimum and maximum step times.

processes can use timing information about relative step times to count locally in order to determine when enough sessions have elapsed. We call this counting technique the *step time method* (ST). It was first proposed in [AM94] for the semi-synchronous message passing model. However, if the gap between the minimum and maximum step times is sufficiently large, then it is more cost-effective to use explicit communication. We call this counting technique the *explicit communication method* (EC). It was first proposed in [AFL83] for the asynchronous shared memory model.

These results are analogous to those of [AFL83]: intuitively, if either bound is unknown, then the system can be considered somewhat "asynchronous", otherwise the system behaves "more synchronously". As we discussed in the introduction, the asynchronous lower bound of [AFL83] does not automatically imply any of the lower bounds in the unknown bound models; however, it is the case that the proof in [AFL83] also works in the case where both bounds are unknown. Mavronicolas [Ma93] independently [[and concurrently]] also developed the same bounds for the model where both minimum and maximum step time are known.

1.4.2 Message Passing Results

In the message-passing case, we discovered a pattern of upper bounds consisting of eight different groups of models. As in the shared memory case, the pattern is based on different counting methods. However, there are three additional counting methods available in message passing, so the relationships are more involved.

In the following, we specify each model by a tuple (c_1, c_2, d_1, d_2). Each entry in a tuple is a real value if that parameter is known, and '?' if it is unknown. For example, we denote the model in which only the maximum step time is known by $(?, c_2, ?, ?)$.

In addition to the two counting methods available in the shared memory model (EC and ST), three other counting methods are used in the message passing model. (1) The *message delay method* (MD) uses the known difference between the minimum and maximum message delays; (2) *combination method 1* (CB1) uses the known minimum step time in combination with the difference between the minimum and maximum message delays; and (3) *combination method 2*

counting methods	required knowledge	approximate per-session cost
explicit communication (EC)	none	d_2
step times (ST)	c_1, c_2	$\frac{c_2}{c_1}c_2$
message delays (MD)	d_1, d_2	$\frac{d_2}{d_1}u$
combination 1 (CB1)	c_1, d_1, d_2	$\frac{c_2}{c_1}u$
combination 2 (CB2)	c_2, d_1	$\frac{d_2}{d_1}c_2$

Table 2: The approximate per-session cost of each counting method used to solve the session problem when the required timing knowledge is available; c_1 and c_2 are the minimum and maximum step times, d_1 and d_2 are the minimum and maximum message delays, and $u = d_2 - d_1$ is the uncertainty in message delay.

(CB2) uses the known maximum message delay in combination with the known minimum step time.

Table 2 shows the approximate per-session cost for each counting method that is applicable when specific timing information about the system is available. The upper bound on the time complexity for a particular timing model is the minimum, over all applicable counting methods, of the time complexity of the counting methods.

These counting methods divide the models into eight groups, as shown in Table 3. Figure 1 shows a lattice of timing models based on the counting methods that a model can use. The results for group G4, when step time bounds are known, were previously shown by Attiya and Mavronicolas [AM94].

All the remaining results are new; as mentioned before, the asynchronous results in [AM94] do not automatically imply the same results in the unknown bound cases, although the proof techniques are similar.

We show that the upper bounds on the time complexity for the timing models are asymptotically optimal. Some of our lower bounds require certain relationships to hold between some of the parameters. For example, consider the model in which only c_2 and d_1 are known. The only applicable counting methods are EC and CB2. Thus, the per-session cost is (approximately) $\min\{d_2, \frac{d_2}{d_1} \cdot c_2\}$. The lower bound we prove for this model gives a per-session cost of approximately $\frac{2d_2}{3d_1} \cdot c_2$, assuming $2c_2 \leq d_1$. Since $2c_2 \leq d_1$, algebraic manipulation shows that this lower bound is less than d_2. Thus our upper and lower bounds are asymptotically tight, if $2c_2 \leq d_1$.

As in the case of the shared memory models, the general trend of these bounds is that if a smaller number of the parameters in a model are known, the model behaves more like "'asynchronous", and otherwise, more like "synchronous".

As the time complexity gaps (i.e., the difference between the upper bound in a model and the lower bound in another model) among the models sometimes overlap, it is rather difficult to

group	models	usable counting method(s)
$G1$	$(?,?,?,?), (?,?,?,d_2), (?,?,d_1,?), (?,c_2,?,?),$ $(?,c_2,?,d_2), (c_1,?,?,?), (c_1,?,?,d_2), (c_1,?,d_1,?)$	EC
$G2$	$(?,?,d_1,d_2)$	EC, MD
$G3$	$(?,c_2,d_1,?)$	EC, CB2
$G4$	$(c_1,c_2,?,?), (c_1,c_2,?,d_2)$	EC, ST
$G5$	$(c_1,?,d_1,d_2)$	EC, MD, CB1
$G6$	$(?,c_2,d_1,d_2)$	EC, CB2, MD
$G7$	$(c_1,c_2,d_1,?)$	EC, CB2, ST
$G8$	(c_1,c_2,d_1,d_2)	EC, CB2, CB1, ST, MD

Table 3: Groups of models that can use the same counting methods.

analyze the relative strength qualitatively without making assumptions on parameters. However, when specific values for each known parameters are given, the actual bounds can be used to analyze the relative strength of the models quantitatively.

As process step times become more synchronous (i.e., $c_1 \simeq c_2$) and message delays become erratic (i.e., $c_2 \ll d_1 \ll d_2$), the general trend of the bounds is that $\{G1, G2, G5\} > \{G3, G4\} > \{G6, G7, G8\} > S$ where S is the synchronous model and '$>$' denotes that it takes more time to solve the session problem. As message delays become more synchronous and smaller (i.e., $d_1 \simeq d_2$ and $d_2 \geq c_2^2/2c_1$) and process step times become more erratic (i.e., $c_1 \ll c_2$), the trend is that $\{G1, G6\} > \{G2, G5, G3, G4, G7, G8\} > S$. These trends suggest that when process step times are fairly "synchronous", ST can be more cost-effective than MD, CB2 and CB1, while when process step times are more "asynchronous" than message delays, the opposite is true.

1.4.3 Proof Techniques

We unify the lower bound proof techniques of [AFL83] in the shared memory model and [AM94] in the message passing model into one "modular" lower bound proof. Our technique is unique in that, instead of obtaining a lower bound for each model independently, we develop one sufficient condition for any given lower bound to hold in any given timing model. This sufficient condition consists of a set of algebraic relations involving (1) the timing parameters of the given model; (2) the given lower bound; and (3) some input parameters that need to be provided to prove the lower bound. Testing whether a lower bound holds in a timing model is a simple algebraic exercise of finding those input parameters that satisfy the relations.

The upper bounds are also obtained in a modular way. We first find algorithms (i.e., ways to count sessions) that work correctly when a certain set of timing parameters is known. Since several algorithms can be applicable to a model, we provide a scheme to combine these algorithms without increasing the time complexity of any of its applicable algorithms. The resulting upper

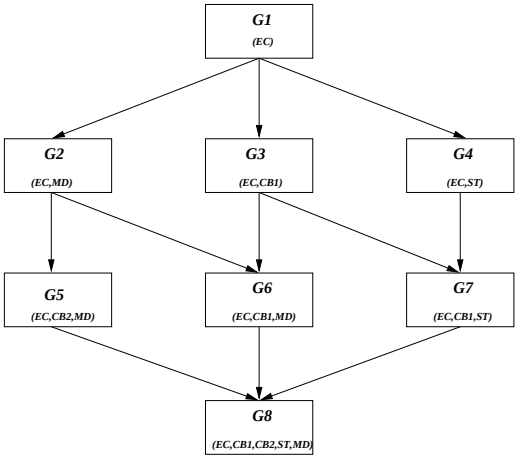

Figure 1: A lattice of model groups can be formed based on the counting methods (shown in parenthesis) and tight time complexity bounds of the timing models.

bound of a model is simply the minimum of the time complexity of all the algorithms applicable to the model.

1.5 Organization

The rest of this paper is organized as follows. Section 2 gives the definition of the system model. Section 3 contains our modular lower bound result for the time complexity of the session problem. Our algorithmic counting methods are presented in Section 4. Section 5 draws together the results for shared memory and Section 6 does the same for message passing. We conclude in Section 7.

2 Definitions

2.1 Systems

The system model definition is similar to that defined in [AFL83].

There are finite sets P of processes and V of shared variables. A *process* has a set of internal states, including an initial state. Each *shared variable* has a set of values that it can contain, including an initial value. A *global state* is a tuple of internal states of each process, and values of each shared variable. The *initial* global state contains the initial state for each process and the initial value for each shared variable.

A process can both read and write a shared variable in a single atomic step (i.e., the variable supports read-modify-write operations); we do not assume any upper bound on the size of the variables. A *step* π consists of simultaneous changes to the state of some process p and the

value of some set of variables $x_1, \ldots x_k$ (for some integer k), where p is allowed to accessed x_i, $1 \leq i \leq k$, depending on the current state of that process and current values of the variables. More formally, we represent the step π with a tuple $((q, p, r), (u_1, x_1, v_1), \ldots (u_k, x_k, v_k))$, where q and r are old and new states of a process $p \in P$, and u_i and v_i are old and new values of a shared variable $x_i \in V$. We define $proc(\pi) = p$ and $var(\pi) = \{x_1, \ldots x_k\}$. We say that step π is *applicable* to a global state if p is in state q and x_i has value u_i for all i in the global state.

An *algorithm* consists of P, V, and set Σ of possible steps. For all processes $p \in P$ and all global states g, there must exist some step in Σ involving process p that is applicable to global state g. This condition ensures that p never blocks. 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 step, 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* (α, T) of a system is a computation $\alpha = \pi_1, \pi_2, \ldots$ together with a mapping T from positive integers to nonnegative 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. This way of modeling processes assumes that the time taken for local computation at a step is negligible.

2.1.1 Shared Memory (SM) Model

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 involves only one shared variable. Associated with each variable is a set of at most a processes that are allowed to access that variable.

2.1.2 Message Passing (MP) Model

We specialize the general system into the message passing system, in which processes communicate with each other by exchanging messages. P consists of the *regular* processes, denoted by the set R, plus a distinguished process N, called the *network*. The network schedules the delivery of messages sent among the regular processes. V, the set of shared variables, equals $\{net\} \cup \{buf_p : p \in R\}$, where the values taken on by each variable are sets of messages. The variable net models the state of the network, i.e., the set of messages in transit. The variable 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 messages and its current state, changing its local state and sending out some message m to all the regular processes. More formally, the result of the step is to set buf_p to empty (i.e., receive messages), to add (m, q) to net for all q in R (i.e., send a message), and to change state. 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. More formally, the result of the step is to

remove (m, q) from net and add m to buf_q. We call this step the *delivery step* of m. Accordingly, the step also involves two shared variables, net and buf_q. We define $msg(\sigma)$ to be the message that is involved in a step σ of a process in P.

This definition of the MP model is an abstract model of a reliable strongly connected network with any topology (i.e., for every pair of processes, there exists a communication path between the two processes).

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. Note that after a message is delivered to a destination process p, p has to take at least one step to receive the message. That is, the time elapsed between the delivery step of a message m and the step of the destination process which finally removes m from the buffer is not counted toward the message delay.

2.2 Timing Models

First we consider shared memory models. Let v and w be two positive real numbers with $v \leq w$. $M(v, w)$, called a *submodel*, is the set of all timed computations of a system in which all step times (the time between two consecutive steps by the same process) are within $[v, w]$.

A shared memory model is specified by indicating whether the minimum and maximum step times are known; and if so, what their values are. Formally, a *shared memory model* is denoted $\mathcal{M}[c_1, c_2]$ where $c_i \in \{?\} \cup \mathcal{R}^+$ ($\mathcal{R}^+$ is the set of positive reals). If the minimum step time is known, then c_1 is some positive real; otherwise, $c_1 =?$. If the maximum step time is known, then c_2 is some positive real; otherwise $c_2 =?$.

We define the four shared memory models of interest as follows.

$$\begin{aligned}
\mathcal{M}[c_1, c_2] &= \{M(c_1, c_2)\} \text{ if } c_1 \in \mathcal{R}^+ \text{ and } c_2 \in \mathcal{R}^+. \\
\mathcal{M}[c_1, ?] &= \{M(c_1, w) : w \geq c_1\} \text{ if } c_1 \in \mathcal{R}^+. \\
\mathcal{M}[?, c_2] &= \{M(v, c_2) : v \leq c_2\} \text{ if } c_2 \in \mathcal{R}^+. \\
\mathcal{M}[?, ?] &= \{M(v, w) : 0 < v \leq w\}.
\end{aligned}$$

We now consider message passing models. Let v, w, x and y be four positive real numbers with $v \leq w$ and $x \leq y$. $M(v, w, x, y)$, called a *submodel*, is the set of all timed computations in which all step times are within $[v, w]$ and all message delays are within $[x, y]$.

The 16 message passing models are defined analogously to the four shared memory models. For example,

$$\mathcal{M}[?, c_2, d_1, ?] = \{M(v, c_2, d_1, y) : 0 < v \leq c_2 \text{ and } y \geq d_1\} \text{ if } c_2 \text{ and } d_1 \in \mathcal{R}^+.$$

We number the models 0 through 15 using the binary representation, assuming a parameter that equals ? is replaced with 0 and otherwise with 1. For instance, $\mathcal{M}[?, c_2, d_1, ?]$ is numbered $0110_2 = 6$.

We say that a timed computation α is *admissible* for a submodel M if α is in M, and is *admissible* for a model $\mathcal{M}$ if α is admissible for some M in $\mathcal{M}$.

2.3 The (s, n)-Session Problem

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

There is a distinguished set Y of n shared variables called *ports*; Y is a subset of V in SM; and Y is the set of *buf* variables in MP. There is a unique process in P (in R in MP) corresponding to each port, which is called a *port process*, and no two port processes can be assigned to the same port. A *port step* is any step involving a port and its corresponding port process. A port can be accessed by processes in addition to its corresponding port process, but such a step is not a port step. There may be some processes which are not port processes, i.e., it is possible for $|P|$ to be larger than n.[1]

Each port process in P must have a subset of special states, called *idle* states. The set Σ of steps of the system must guarantee that once a process is in an idle state, it always remains in an idle state, and after a process enters an idle state, it does not access a port.

A *session* is a minimal sequence of steps containing at least one port step for each port in Y. A computation *performs* s sessions if it can be partitioned into s segments, each of which is one session. Every infinite admissible timed computation must perform at least s sessions and eventually all port processes must be in idle states.

2.4 Time Complexity

We give the definitions for the shared memory models. The time complexity definitions for message passing models are analogous to those for shared memory models.

An algorithm A in a submodel $M(x, y)$ has *running time* t if t is the maximal time, over all admissible computations of A for $M(x, y)$, until all port processes become idle.

Let f be a function from $\mathcal{R}^+ \times \mathcal{R}^+$ to $\mathcal{R}^+$. We abuse notation and say that an algorithm A in model $\mathcal{M}[c_1, c_2]$ has *upper bound* $f(c_1, c_2)$ if A has running time at most $f(x, y)$ in every submodel $M(x, y)$ in $\mathcal{M}[c_1, c_2]$. (This is an abuse of notation because c_1 or c_2 might equal ? instead of being a positive real constant.)

$\mathcal{M}[c_1, c_2]$ has *lower bound* $f(c_1, c_2)$ if for every algorithm A, there is a submodel $M(x, y)$ such that A has running time at least $f(x, y)$ in that submodel.

[1] This possibility is implicitly contained in [AFL83], which refers to making the port processes the leaves of a tree network.

3 Modular Lower Bound

In this section, we give a modular lower bound proof that holds for all the timing models, both shared memory and message passing. Our lower bound proof is motivated by the proofs in [AFL83] and [AM94]. Our technique is unique in that, instead of obtaining a lower bound for each model independently, we develop a sufficient condition for a lower bound to hold in any given timing model. This sufficient condition consists of a set of algebraic relations on (1) the timing parameters of the given model; (2) the given lower bound; and (3) some other input parameters (shown below). Thus, testing whether a lower bound holds in a timing model is a simple algebraic exercise of finding those input parameters that satisfy the relations. The theorem below proves the sufficient condition; in its statement, c, c_1', c_2', d_1', d_2', and B are the input parameters, and SC1 to SC3 and MC1 to MC5 are the algebraic relations. In the proof of the theorem, we present some intuitive ideas behind the theorem and then formalize the ideas.

Theorem 3.1 *Let M be a timing model that satisfies the following.*

If $M = \mathcal{M}[c_1, c_2]$ is a shared memory model, then there exist positive real numbers c, c_1', c_2' and a function f with $B = f(c_1', c_2')$ such that:

$$
\begin{array}{lllll}
\text{SC1.} & (B \leq c_2' \cdot \log_a n) & \wedge & (c_1' \leq c_2') & \\
\text{SC2.} & (c_1' \leq \frac{1}{2}c) & \wedge & ((c_1' = c_1) & \textit{if } c_1 \neq ?) \\
\text{SC3.} & (c_2' \geq B\frac{c}{c_2'}) & \wedge & ((c_2' = c_2) & \textit{if } c_2 \neq ?)
\end{array}
$$

If $M = \mathcal{M}[c_1, c_2, d_1, d_2]$ is a message passing model, then there exist positive real numbers c, c_1', c_2', d_1', d_2' and a function f with $B = f(c_1', c_2', d_1', d_2')$ such that:

$$
\begin{array}{llllll}
\text{MC1.} & (B \leq d_2') & \wedge & (c_1' \leq c_2') \wedge (d_1' \leq d_2') & & \\
\text{MC2.} & (c_1' \leq \frac{1}{2}c) & \wedge & ((c_1' = c_1) & \textit{if } c_1 \neq ?) \\
\text{MC3.} & (c_2' \geq B\frac{c}{c_2'}) & \wedge & ((c_2' = c_2) & \textit{if } c_2 \neq ?) \\
\text{MC4.} & (d_1' \leq (d_2' - B)\frac{c}{c_2'} + c) & (\wedge & (d_1' = d_1) & \textit{if } d_1 \neq ?) \\
\text{MC5.} & (d_2' \geq (d_2' + B)\frac{c}{c_2'} - c) & (\wedge & (d_2' = d_2) & \textit{if } d_2 \neq ?)
\end{array}
$$

Then a lower bound on the time complexity of the (s, n)-session problem for M is $(s-1) \cdot f(c_1, c_2)$ if M is a shared memory model, and $(s-1) \cdot f(c_1, c_2, d_1, d_2)$ if M is a message passing model.

Informal description By way of contradiction we assume that there exists such an algorithm A that solves the (s,n)-session problem in model M within time less than the stated lower bound. We prove that there exists an infinite timed computation of A that is admissible for M yet contains fewer than s sessions, contradicting the assumed correctness of A.

More specifically, we first fix a submodel M' of M and pick an infinite timed computation (α, T) of A that is admissible for M'. Then we retime and reorder some steps in α to obtain a new infinite timed computation (α', T') that has only $s-1$ sessions, yet is admissible for some submodel M'' of M. (M' and M'' are not necessarily the same.)

Real numbers c_1', c_2', d_1', and d_2' are the minimum and maximum step times and message delays of submodel M' respectively for which (α, T) is admissible. Real numbers $\frac{1}{2}c$, $B\frac{c}{c_2'}$, $(d_2' - B)\frac{c}{c_2'} + c$, and $(d_2' + B)\frac{c}{c_2'} - c$ are the minimum and maximum step times and message delays of M'' for which (α', T') is admissible.

The conditions in the theorem statement are used to prove that M' and M'' really are submodels of M. Below we provide some intuition for these conditions.

1. B is roughly the time for a process to "recognize" one session in a computation. The first clause in Conditions SC1 and MC1 states that B does not take more than the lower bound on the maximum communication delay in the SM and MP models.

2. Conditions SC1 and MC1 ensure that minimum and maximum step times and message delays (c_1', c_2', d_1' and d_2') in submodel M' satisfy the property that the minimum is not larger than the maximum.

3. Conditions SC2 and MC2 ensure that, if the minimum step time (c_1) is known in M, then in M' it is equal to that of M and in M'' it is at least as large as that of M.

4. Conditions SC3 and MC3 ensure that, if the maximum step time (c_2) is known in M, then in M' it is equal to that of M and in M'' it is not larger than that of M.

5. Condition MC4 ensures that, if the minimum message delay (d_1) is known in M, then in M' it is equal to that of M and in M'' it is at least as large as that of M.

6. Condition MC5 ensures that, if the maximum step time (d_2) is known in M, then in M' it is equal to that of M and in M'' it is not larger than that of M.

Since B is roughly the time upper-bound to have one session, the time complexity lower bound to solve the (s,n)-session problem cannot be less than $(s-1)$ multiple of the maximum B which is determined by the timing parameters of M.

We now need to prove that satisfying the above conditions is sufficient to prove the time complexity lower bound for solving the (s,n)-session problem. This involves several procedures.

1. We prove that (α, T) is admissible for M', and M' is a submodel of M. In (α, T), processes enter an idle state before $B \cdot (s-1)$.

2. We then reorder steps in α to obtain α' without violating the causal dependency among process steps. The causal dependency among two process steps happens, for example, because one step receives a message sent by the other step. Thus, for example, α' should not order the receive step before the send step. This procedure involves several other steps.

   (a) We first break α into two segments. The first segment, β, is up to the last step taken by any process before all processes enter the idle state. The second segment, γ, is the rest of α. It is clear that β should contain s sessions because α is a computation of A that solves the (s, n)-session problem.

   (b) We then break β into $s-1$ equal non-overlapping segments of time period B. We reorder the process steps only within each segment without violating their causal depedency so that each segment by itself does not contain one session. Since each segement contains less than one session, the reordered sequence β' does not contain more than $s-1$ sessions. Since each segment does not violate the causal depedency, all the port processes will be in the same state as they are in the corresponding segment in β. Thus, in the end of β', all the port processes are in the same state as in β, and $\alpha' = \beta'\gamma$ is an admissible computation for M because $\alpha = \beta\gamma$ is.

3. Reordering steps perturbs the timings of process steps. We show a timing mapping T' for in β' whose minimum and maximum step times are $c/2$ and $B\frac{c}{c_2'}$, and whose minimum and maximum message delays are $(d_1' \leq (d_2' - B)\frac{c}{c_2'} + c)$ and $(d_2' \geq (d_2' + B)\frac{c}{c_2'} - c)$. The conditions in the theorem is used to show that these retimed parameters are within the constraints of a submodel M'' of M.

4. Since there is a timed computation (α', T') that is admissible for a submodel of M which contains less than $s-1$ sessions, this is a contradiction.

Proof: We now formalize these ideas.

Let M' be the submodel of M that has minimum and maximum step times and message delays equal to c_1', c_2', d_1' and d_2' respectively where the followings hold. (a) $c_1' \leq c_2'$; (b) $c_1' = c_1$ if the minimum step time of M (c_1) is known; and (c) $c_2' = c_2$ if the maximum step time of M (c_2) is known. Furthermore, if M is a message passing model, (d) $d_1' \leq d_2'$; (e) $d_1' = d_1$ if the minimum message delay of M (d_1) is known; and (f) $d_2' = d_2$ if the maximum message delay of M (d_2) is known.

Since M' is a submodel of M, by the assumption that algorithm A is correct for M, A has running time in M' less than $(s-1) \cdot B$, where $B = f(c_1', c_2')$ if M is a shared memory model, and $B = f(c_1', c_2', d_1', d_2')$ if M is a message passing model.

Let (α, T) be the infinite timed computation of A in which all the regular processes take steps at the same speed in round robin order and each process' ith step occurs at time $i \cdot c_2'$. Furthermore, if M is a message passing model, all the message delays in (α, T) are exactly d_2'. Note that (α, T) is admissible for M' (and thus for M).

Let $\alpha = \beta\gamma$, where β contains all the steps that occur in (α, T) during time interval $[0, (s-1)\cdot B)$ and γ contains the rest of α. Note that all the states in γ are idle states because A solves the (s,n)-session problem in time less than $(s-1)\cdot B$.

Case 1: If $B \leq c_2'$, then β contains only $s-1$ sessions because each process takes only $s-1$ steps in β. This is a contradiction since all regular processes are in an idle state in γ.

Case 2: For the rest of the proof, assume that $B > c_2'$. For convenience of presentation, we assume that B is divisible by c_2'.[2]

We will reorder and retime (i.e., assign new times to) steps in β to obtain (β', T'). To ensure that the retimed computation leads to the same global state, this retiming should not violate the dependencies among process steps. Informally, a dependency arises between two steps if they are steps of the same process; if one step reads a variable previously accessed by the other; or if one step is the receipt of a message sent by the other. A more precise definition of dependency is given below.

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 that τ *depends* on σ, if:

- $\sigma = \tau$, or
- $T(\sigma) < T(\tau)$ and $proc(\sigma) = proc(\tau) \neq N$, or
- $T(\sigma) < T(\tau)$ and $msg(\sigma) = msg(\tau)$ if M is a message passing model, or
- $T(\sigma) < T(\tau)$ and $var(\sigma) = var(\tau)$ if M is a shared memory model.

Close $\leq_\beta$ under transitivity.

Let $\beta = \beta_1 \ldots \beta_{s-1}$ such that each β_k, which we call *segment* k, $1 \leq k \leq s-1$, consists of all the steps in (α, T) during time interval $[(k-1)B, kB)$. Note that in a message passing model, no message sent in β_k is received in β_k since $B \leq d_2'$ by MC1 and d_2' is the message delay in (α, T).

Informally speaking, we will reorder steps in each segment β_k without violating the dependencies as follows. We first pick one port variable for each segment in such a way that the same port variable is not picked for two consecutive segments. Let y_i be the port variable picked for segment β_i. Then, we reorder the steps of each segment, resulting in two "subsegments" such that the first subsegment does not contain any port event accessing y_i and the second subsegment does not contain any port event accessing y_{i+1}. The reordered sequence will contain only $s-1$ sessions. Figure 2 illustrates an example when $s = 4$.

However, for the reordered computation to end in the same state as β, this reordering should not violate relation $\leq_\beta$ within each segment (relation $\leq_\beta$ is not violated across segments by the

[2] If B is not divisible by c_2', the lower bound we obtain is $(s-1)\cdot\lfloor\frac{f(c_1,c_2)}{c_2}\rfloor\cdot c_2$ instead of $(s-1)\cdot f(c_1,c_2)$ in the shared memory case, and similarly for the message passing case.

reordering because steps are not reordered out of their own segments). Thus, we must choose the port y_k for each segment β_k such that the first step in β_k to access y_{k-1} does not depend on the last step in β_k to access y_k. The following claim shows that this can be done.

Claim 3.2 *Let y_0 be an arbitrary port in Y. For all k, $1 \leq k \leq s-1$, there exists a port variable y_k such that it is false that $\tau_k \leq_\beta \sigma_k$, where τ_k is the first step in β_k that accesses y_{k-1} and σ_k is the last step in β_k that accesses y_k.*

Proof: If M is a message passing model, no message sent in β_k is delivered in β_k because the size of each segment is less than d_2', and d_2' is the message delay. Because there is no $\leq_\beta$ relation between any two steps of different processes in β_k, any port variable except y_{k-1} can be chosen as y_k.

If M is a shared memory model, part of the proof of Theorem 2 of [AFL83] [3] proves that there exists such y_k if each process takes fewer than $\log_a n$ steps in a segment. Because in β_k, each process takes fewer than $\log_a n$ steps (cf. SC1), there exists such y_k in β_k. ■

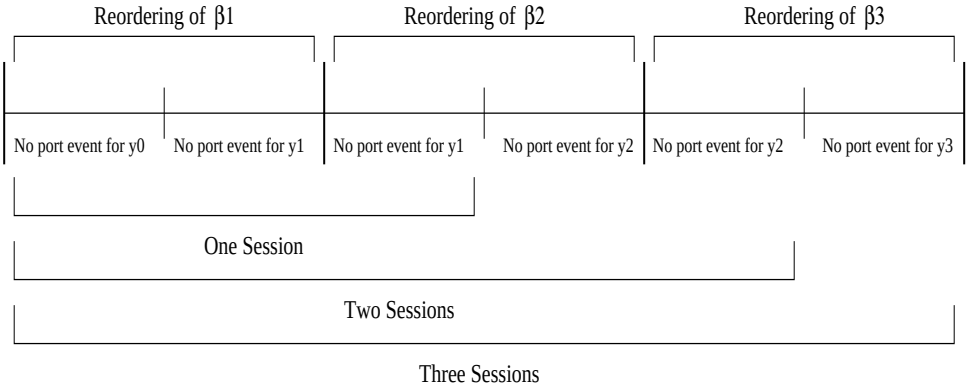

Figure 2: An example of reordering when $s = 4$.

Claim 3.2 allows us to use the y_k's to reorder β in order to obtain a new sequence with less than s sessions (shown in 3.4). However the steps in the sequence are not mapped to time. Thus, we need a scheme to assign new times to the reordered steps so that the new timed computation does not violate the timing constraints of M. This is the major difference between our lower bound proof for unknown parameter models and that in [AFL83] for the asynchronous model; in the latter no timing scheme is necessary because the asynchronous model imposes no timing constraints.

[3] The statement of Theorem 2 in [AFL83] does not explicitly include this result. For conciseness, we do not include a copy of the relevant part of the proof (the middle of pp. 453–455).

In the following, we define a retiming scheme that also encompasses the reordering scheme presented above.

Let us first assign a new mapping T'' to every step π in $\beta\gamma$, including all the steps of the network N if M is a message passing model, such that $T''(\pi) = T(\pi) \cdot \frac{c}{c_2'}$. That is, every process (except N) takes a step at every time that is a multiple of c. Since in a message passing model, the delivery steps of N are retimed along with other steps, the message delay is now changed from d_2' to $d_2' \cdot \frac{c}{c_2'}$. Note that the assignment of T'' does not change the relative ordering of the steps in β because it changes every step time by the same proportion (namely, $\frac{c}{c_2'}$).

We now reorder and assign new times (the mapping T') to every step in (β_k, T''). Intuitively, in order to obtain a timed computation as in Figure 2, σ_k and every step that σ_k depends on have to move earlier in time into the first half of (β_k, T''), and τ_k and every step that depends on τ_k have to move later in time into the second half of (β_k, T''). As a result, σ_k will occur before τ_k.

Let $t_k = k \cdot \frac{Bc}{c_2'}$, where $0 \leq k \leq s-1$. Thus, β_k occurs during $[t_{k-1}, t_k)$ under T''. Note that $t_k - t_{k-1} = \frac{Bc}{c_2'}$.

1. (Earlier retiming) Let π be any step in β_k by a process in P that σ_k depends on (in a message passing model, this means $proc(\pi) = proc(\sigma_k)$). Retime π such that $T'(\pi) = \frac{t_{k-1}+T''(\pi)}{2}$. The step is moved backward halfway to the beginning of β_k.

2. (Later retiming) Let σ be any step in β_k by a process in P that depends on τ_k (in a message passing model, this means $proc(\sigma) = proc(\tau_k)$). Retime σ such that $T'(\sigma) = \frac{T''(\sigma)+t_k-c}{2}$. The step is moved forward approximately halfway to the end of β_k.

3. (Stationary retiming) All other steps in β_k and all steps in γ are assigned the same times as in T''.

For all k, $1 \leq k \leq s-1$, let β_k'' be the result of reordering and retiming steps in β_k according to T', and let $\beta'' = \beta_1''\beta_2'' \ldots \beta_{s-1}''$.

If M is a message passing model, let β' be the result of changing the states of the network in β'' so that in each step of the network, the state of the network is consistent with all the send steps of regular processes and all the deliver steps of the network in β'' that have happened so far ([["consistent" means that]] a delivery step of a message happens after its send step). If M is a shared memory model, let $\beta' = \beta''$.

In summary, (β, T) is now transformed to (β', T') using (β'', T'') as an intermediate computation. To show the theorem, it suffices to show that (1) β' is a computation leading to the same global state as β (Claim 3.3); (2) β' contains less than s sessions (Claim 3.4); and (3) $(\beta'\gamma, T')$ is admissible for M (Claim 3.5).

Claim 3.3 *β' is a computation that leaves the system in the same global state as β does.*

Proof: We first prove that $\leq_\beta$ holds in β'.

For any k, $1 \leq k \leq s-1$, pick any two steps, π and π' in β_k such that $\pi \leq_\beta \pi'$. Thus $T(\pi) \leq T(\pi')$ (recall that T is the original timing). We only need to prove that $T'(\pi) \leq T'(\pi')$ because π occurs earlier than π' in T.

Each of π and π' was retimed by one of the earlier, later or stationary retiming methods. Clearly the desired ordering is preserved:

(1) when π stays the same and π' either stays the same or moves later in time, or

(2) when π' stays the same and π either stays the same or moves earlier in time, or

(3) when π and π' both move in the same direction.

The other cases cannot occur, since if π moves later in time, then so does π', and if π' moves earlier in time then so does π.

Since β' does not violate $\leq_\beta$ in both SM and MP, and all the states of *net* in β' are consistent with the steps of β'' by the definition of β' in MP, it follows that β' is a computation resulting in the same global state as β. ■

Claim 3.4 *β' contains at most $s-1$ sessions.*

Proof: For all k, $1 \leq k \leq s-1$, let $h_k = T'(\tau_k)$. We show, by induction on k, that time period $[0, h_k)$ in (β', T') contains at most $k-1$ sessions.

For the basis, $[0, h_1)$ contains no session because it does not contain any port event for y_0.

Assume, by way of induction, that $[0, h_{k-1})$ contains at most $k-2$ sessions. We prove that $[0, h_k)$ contains at most $k-1$ sessions. It is clear from the construction that $(h_{k-1}, t_{k-1}]$ contains no session because it does not contain any port event of y_{k-1} and similarly, $[t_{k-1}, h_k)$ does not contain any session because it contains no port event of y_{k-1}. Since the port process of y_{k-1} takes only one step at h_{k-1}, it is clear that $[h_{k-1}, h_k)$ contains at most one session. Therefore, $[0, h_k)$ contains at most $k-1$ sessions.

A similar argument shows that $[h_{s-1}, t_{s-1}]$ contains at most one session. Since $[0, h_{s-1})$ contains at most $s-2$ sessions, it follows that there are at most $s-1$ sessions in $[0, t_{s-1}]$. ■

Claim 3.5 *$(\beta'\gamma, T')$ is a timed computation that is admissible for M.*

Proof: By Claim 3.3, β' is a computation that leads to the same global state as β. Therefore, $\beta'\gamma$ is also a computation because $\beta\gamma$ is a computation. Since γ is infinite, so is $\beta'\gamma$. It remains to show that the timing T' conforms to the timing constraints of model M.

Constraints on step time:

Let π_i and π_{i+1} be consecutive steps of a single process in β_k for some k. Then, for some $\Delta \geq 0$, it is true that $T''(\pi_i) = t_{k-1} + \Delta$ and $T''(\pi_{i+1}) = t_{k-1} + \Delta + c$.

The distance between the two steps is minimized when both steps are retimed by the same retiming method because it is not possible that π_i is retimed by the later retiming while π_{i+1} is retimed by the earlier retiming. If both π_i and π_{i+1} are retimed by the same retiming method (either later or earlier) retiming, $T'(\pi_{i+1}) - T'(\pi_i)$ is equal to $c/2$.

None of the retiming methods retimes a step outside its original segment: If a step is in segment β_k, after the retiming, it is still in segment β'_k. Thus, neither π_i nor π_{i+1} is retimed outside the segment. Since the maximum time elapsed between π_i and π_{i+1} in β_k is B, and in T'', all times are shrunk in proportion to $\frac{c}{c'_2}$]], the distance between the two steps can never be larger than $t_k - t_{k-1} \leq B\frac{c}{c'_2}$.

For the timing of steps in different segments, let π_j and π_{j+1} be the consecutive steps of a process, each of which are in the different segments, say β_k and β_{k+1} respectively. Then, $T''(\pi_j) = t_k$ and $T''(\pi_{j+1}) = t_k + c$ (this is because B is divisible by c'_2). Since the two steps are in different segments, the distance between the two steps is minimized when π_j is retimed by the later retiming while π_{j+1} is retimed by the earlier retiming. However, neither of these retimings results in a change, i.e., $T'(\pi_j) = T''(\pi_j)$ and $T'(\pi_{j+1}) = T''(\pi_{j+1})$. Therefore, $T'(\pi_{i+1}) - T'(\pi_i) = c$. Since the two steps are in different segments, the distance between the two steps is maximized when π_j is retimed by the earlier retiming while π_{j+1} is retimed by the later retiming. In this case, π_j moves to $(t_{k-1}+t_k-c)/2$ and π_{j+1} moves to $(t_k+t_{k+1}-c)/2$. Therefore, the distance between the two steps cannot be larger than $B\frac{c}{c'_2}$ because $T'(\pi_{i+1}) - T'(\pi_i) = \frac{t_{k+1}-t_{k-1}}{2} \leq B\frac{c}{c'_2}$.

We first check the minimum step time.

- If c_1 is unknown, then there exists a positive constant (namely $c/2$) which is the minimum step time in (β', T').
- If c_1 is known, by condition SC2 (or MC2), the minimum step $(c/2)$ in (β', T') is bigger than or equal to the minimum step time of M.
- If c_2 is known, the minimum step $(c/2)$ is less than or equal to c_2 because (1) by SC3 (MC3), $c'_2 = c_2$ and $B\frac{c}{c_2} \leq c_2$, and (2) by the assumption for Case 2 of the main proof, $B > c_2$.

Now we check the maximum step time.

- If c_2 is unknown, then there exists a positive constant (namely $B\frac{c}{c'_2}$) which is the maximum step time in (β', T').

- If c_2 is known, the maximum step time ($B\frac{c}{c_2'}$) in (β', T') is less than or equal to c_2 because of SC3 and MC3.
- If c_1 is known, the maximum step time ($B\frac{c}{c_2'}$) in (β', T') is bigger than or equal to c_1 because (1) by the assumption for Case 2, $B > c_2'$ and thus $B\frac{c}{c_2'} > c$, and (2) by SC2 and MC2, $c \geq c_1$.

Constraints on message delay:

The steps that move the farthest due to the earlier retiming ¿from T'' to T' are those at the end of each segment because they do not move outside their segments. Let π_{last} be the step at the end of β_k. $T''(\pi_{last}) = t_k - c$. By the earlier retiming, $T'(\pi_{last}) = (t_{k-1} + t_k - c)/2 = t_k - \frac{1}{2}B\frac{c}{c_2'} - c/2$.

The steps that move the farthest due to the later retiming from T'' to T' are those at the start of each segment because they do not move outside their segments. Let π_{start} be the step at the start of β_k. $T''(\pi_{start}) = t_{k-1}$. By the later retiming, $T'(\pi_{start}) = (t_{k-1} + t_k - c)/2 = t_k - \frac{1}{2}B\frac{c}{c_2'} - c/2$. Therefore, the maximum distance that a step can move ¿from T'' to T' is bounded by $\frac{1}{2}B\frac{c}{c_2'} - c/2$.

Let π_s and π_r be the send and receive steps of any message in β. Recall that under T'', all the message delays are $d_2'\frac{c}{c_2'}$. Thus, $T''(\pi_r) - T''(\pi_s) = d_2'\frac{c}{c_2'}$. Let $d' = d_2'\frac{c}{c_2'}$. Because both π_r and π_s can move $\frac{1}{2}(B\frac{c}{c_2'} - c)$ time from T' to T'', each in opposite directions, $d' - B\frac{c}{c_2} + c \leq T'(\pi_r) - T'(\pi_s) \leq d' + B\frac{c}{c_2} - c$.

We first check the minimum message delay.

- If d_1 is unknown, then there exists a positive constant that is the minimum message delay in (β', T') because (1) $T'(\pi_r) - T'(\pi_s) \geq d' - B\frac{c}{c_2} + c$, (2) by MC1, $d' - B\frac{c}{c_2'} = (d_2' - B)\frac{c}{c_2'} \geq 0$, and (3) c is a positive real.
- If d_1 is known, the minimum message delay ($d' - B\frac{c}{c_2'} + c$) in (β', T') is bigger than or equal to d_1 because of MC4.

We now check the maximum message delay.

- If d_2 is unknown, then there exists a positive constant (namely $d' + B\frac{c}{c_2'} - c$) that is the maximum message delay in (β', T') because $d' + B\frac{c}{c_2'} - c \geq T'(\pi_r) - T'(\pi_s) > 0$.
- If d_2 is known, the maximum message delay ($d' + B\frac{c}{c_2'} - c$) is less than or equal to d_2 because of MC5.

■

Technique	Timing Information	Running Time
Explicit Communication (EC)	None	$(s-1)\cdot d_2 + c_2$ or $(s-1)\cdot c_2\Theta(\log n) + c_2$
Step Time (ST)	c_1, c_2	$(s-1)\cdot \frac{c_2}{c_1}\cdot c_2 + c_2$
Combination 2 (CB1)	c_1, d_1, d_2	$(s-2)\cdot(\frac{c_2}{c_1}\cdot u + u + 2c_2) + d_2 + 3c_2$
Message Delay (MD)	d_1, d_2	$(s-2)(\frac{d_2}{d_1}\cdot u + u + 2c_2) + d_2 + 2c_2$
Combination 1 (CB2)	c_2, d_1	$(s-1)\cdot c_2\cdot\frac{d_2+c_2}{d_1} + c_2$

Table 4: Counting methods

To finish the proof of the main theorem, $(\beta'\gamma, T')$ is an infinite timed computation admissible for M but with fewer than s sessions, by Claims 3.4 and 3.5. This contradicts the assumed correctness of A. ■

4 Algorithmic Counting Methods

We develop five methods to count the number of sessions during a computation (cf. Table 4). These method differ in the ways they use the known timing information of a model to count sessions. An (s, n)-session algorithm can be obtained for a model simply by combining all the applicable methods to the model without increasing the asymptotic time complexity of any of those methods. This can be done by running each method "side by side", halting when the first of them finishes [AM94]. Since there are only a constant number of methods running at the same time, the combination does not affect the asymptotic time complexity of the algorithm. The resulting upper bound on the time needed to solve the session problem in a model is the minimum of the time complexity of all the methods applicable to the model.

We now describe each of the counting methods. A message is denoted $m(i, j, k)$, where i is the identifier of the sending process p_i, j is an integer in $[0, s-1]$ and k is an integer. We let $*$ be a *don't care* value. The port for a port process i is denoted y_i. In a message passing model, y_i denotes process i's buffer of incoming messages.

In describing the methods, we use a subroutine called *broadcast* as a generic operator for communication. In MP models, *broadcast* is accomplished by having each process send a message to all the processes, including itself.

We now explain how to achieve a broadcast in an SM model. Recall that at most a processes can access any specific shared variable. We conceptually organize the processes and shared variables into a tree with $\Theta(\log n)$ levels. In order for a port process to broadcast information to all other port processes, the information travels up the tree to the root and then down from the root to all the leaves. See Appendix A for more details.

4.1 Explicit Communication (EC)

The explicit communication method (see Figure 3), originally presented and analyzed in [AFL83] for the asynchronous SM model, and in [AM94] for the asynchronous MP model, does not require any timing information to solve the (s, n)-session problem. It can be used in any timing model because the correctness of the method does not depend on specific step time or message delay.

```
session := 0; msgs := ∅;
while ( session < s − 1 )
    msgs := msgs ∪ y_i; /* port event; recall y_i = buf_i in MP */
    if for all j ∈ {1, ..., n}, m(j, session, *) is in msgs
    then
        session := session + 1;
    end if;
    broadcast m(i, session, *); /* port event */
end while;
Enter an idle state.
```

Figure 3: Technique EC for process i

Theorem 4.1 *EC solves the (s, n)-session problem in time $(s-1) \cdot c_2 \cdot \Theta(\log n) + c_2$ in a shared memory model and in time $(s-1) \cdot d_2 + c_2$ in a message passing model.*

The basic intuition for the method is that since it does not use any timing information, a process relies only on communication with other processes at every session to recognize there is one session. Each process executes one port event, broadcasts the fact to every process at each step and repeats this step until it hears that every process has executed another port event. Then it increments *session*. It performs these steps $s-1$ times. Then, after it executes one additional port event, it enters an idle state.

4.2 Step Time (ST)

The Step Time method (see Figure 4), originally presented and analyzed in [AM94] for the semi-synchronous model, requires information about the maximum and minimum step times (c_2 and c_1). For convenience of presentation, we assume that c_2 is divisible by c_1. This method can also be used for both a shared memory model and a message passing model.

In this method, processes use timing information about relative step times to determine when a session occurs. Each process executes $\frac{c_2}{c_1}$ port events. During this interval, at least $\frac{c_2}{c_1} \cdot c_1 = c_2$ time elapses, since c_1 is the minimum step time. Since every process performs at least one port event within time c_2 (since c_2 is the maximum step time), at least one session has occurred by

$B := \frac{c_2}{c_1}$;
$count := session := 0$;
while ($session < s - 1$)
  if ($count \geq B$)
  then
    $count := 0$;
    $session := session + 1$;
  end if;
  $count := count + 1$;
  access y_i; /* port event; recall $y_i = buf_i$ in MP */
end while;
Enter an idle state.

Figure 4: Technique ST for process i

the time that the process finishes $\frac{c_2}{c_1}$ port events. Each process repeats the above procedure $s-1$ times. After it executes one additional port event, it enters an idle state.

Theorem 4.2 *ST solves the (s, n)-session problem in time $(s-1) \cdot \frac{c_2}{c_1} c_2 + c_2$ in both a shared memory model and a message passing model.*

4.3 Combination 1 (CB1)

CB1 (see Figure 5) requires information about the minimum step time (c_1), the minimum and maximum message delays (d_1 and d_2). For convenience of presentation, we assume that $u = d_2 - d_1$ is divisible by c_1.

The correctness of this method relies 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. Based on this idea, each process broadcasts a message at every step. (1) When a process initially receives one message from every process, it recognizes there is one session because each process must have taken one step to send the message. (2) Then, when it receives another set of messages from every process after time $u = d_2 - d_1$, there must have been another session after the initial session because the second set of messages must have been sent after the first session occurred. Time u can be measured by counting steps, using the known minimum step time (c_1) (i.e., when a process takes $\frac{u}{c_1}$ local steps, at least u time is guaranteed to be elapsed.)

```
B := u/c_1; /* u = d_2 - d_1 */
count := session := 0;
msgs := ∅;
while (session < s - 1)
    msgs := msgs ∪ buf_i; /* port event */
    if (session = 0) or (count ≥ B)
    then
        if for all j ∈ {1, ..., n}, m(j, *, *) is in msgs
        then /* condition 1 */
            count := 0;
            session := session + 1;
            msgs := ∅;
        end if;
    end if;
    broadcast m(i, *, *); /* port event */
    count := count + 1;
end while;
Enter an idle state.
```

Figure 5: Technique CB1 for process i

Each process performs the second procedure $s-2$ times. Then it enters an idle state after taking an additional step. The running time of CB1 is $(\frac{c_2}{c_1}u + u + 2c_2)(s-2) + d_2 + 3c_2$. (1) It takes at most $\frac{u}{c_1} \cdot c_2$ time to count $\frac{u}{c_1}$ steps; (2) at most $u + 2c_2$ time for a process to receive another set of message after it recognizes there was a session; (3) $d_2 + 2c_2$ time to receive the initial set of messages; and finally (4) one more step to accomplish the last session. The detailed proof of the following theorem can be found in Appendix A.

Theorem 4.3 *CB1 solves the (s,n)-session problem within time $(s-2)\cdot(\frac{c_2}{c_1}u+u+2c_2)+d_2+3c_2$ if c_1, d_1 and d_2 are known.*

4.4 Message Delay

Message Delay (MD) (see Figure 6) requires information about the lower bound d_1 and upper bound d_2 on message delay. MD differs from CB1 only in one way: a process recognizes that time u has elapsed by counting the number of times that a certain message is being passed between two processes, using the known minimum message delay. For example, when a process p_i broadcasts a message at time t, the message is received by p_j no earlier than time $t + d_1$. So if the message is passed between them (or any process because of the minimum message delay)

```
B := u/d_1;
count := session := 0;
msgs := ∅;
while( session < s − 1 )
    msgs := msgs ∪ buf_i;
    if (session = 0) or (count ≥ B)
    then
        if for all j ∈ [n], m(j, ∗, ∗) is in msgs
        then /∗ condition 1 ∗/
            count := 0;
            session := session + 1;
            msgs := ∅;
        end if;
    end if;
    if there is any m(∗, session, ∗) in msgs
    then count := max{count, k : m(∗, session, k) ∈ msgs };
    broadcast m(i, session, count + 1);
end while;
Enter an idle state.
```

Figure 6: Technique MD for process i

more than u/d_1 times, then we know at least u time has elapsed. The running time of MD is equal to that of CB1 with $\frac{c_2}{c_1}$ factor replaced by $\frac{d_2}{d_1}$.

The detailed proof of the following theorem can be found in Appendix A.

Theorem 4.4 *MD solves the (s, n)-session problem within time $(s-2)\cdot(\frac{d_2}{d_1}u+u+2c_2)+d_2+3c_2$ if d_1 and d_2 are known.*

4.5 Combination 2

Combination 2 (CB2) (see Figure 7) can be used if the maximum step time c_2 and the minimum message delay d_1 are known. The known minimum message delay (d_1) can be used to measure the elapsed time between the send time of a message and the receive time of the same message; We know at least d_1 time has passed between the send and receive. In addition, because of the known maximum step time, it is possible to estimate how many steps a process takes within time d_1 (at least $\frac{d_1}{c_2}$ steps). Therefore, a process can deduce that if it receives a message sent after the last session, there have been at least $\frac{d_1}{c_2}$ sessions after that last session.

We can inductively apply the above argument starting from session 0. Initially, a process starts by sending a message to all, and as soon as it receives a message from all other processes,

$count := 0$;
$msgs := \emptyset$;
while($count < s - 1$)
  $msgs := msgs \cup buf_i$;
  $count := \max\{k, count : m(*, *, k) \in msgs\ \}$;
  broadcast $m(i, *, count + \frac{c_2}{d_1})$;
end while;
Enter an idle state.

Figure 7: Technique CB2 for process i

it knows that there are at least $\frac{d_1}{c_2}$ sessions in the computation. It increments its counter by $\frac{d_1}{c_2}$, and sends another message piggybacking the value of that counter. If it receives a message with a counter x, it knows that there are at least $x + \frac{d_1}{c_2}$ sessions. Then it updates its counter to $x + \frac{d_1}{c_2}$, and sends another message with the value of that counter. It continues the above until its counter is larger than $s - 1$. Then, it makes one more step and enters an idle state.

The detailed proof of the following theorem can be found in Appendix A.

Theorem 4.5 *Technique CB2 solves the (s, n)-session problem in time $(s - 1) \cdot \frac{d_2 + c_2}{d_1} c_2 + c_2$ if c_2 and d_1 are known.*

5 Shared Memory Results

In this section, we show that the upper bounds we presented in Section 4 for the shared memory models are asymptotically tight by obtaining the matching lower bounds. We use Theorem 3.1 to obtain the lower bounds. To prove a given lower bound for a shared memory model, we simply check whether there exist c, c_1' and c_2' that satisfy SC1, SC2 and SC3.

5.1 Counting with Explicit Communication

Suppose that only EC is used. The resulting upper bound is $(s - 1) \cdot c_2 \cdot \Theta(\log n)$.

We now show that this bound is asymptotically tight. In particular, we show that if either c_1 or c_2 (or both) is unknown, then the lower bound is $(s - 1) \cdot c_2 \cdot \log_a n$.

Corollary 5.1 *Let $c_2 \in \mathcal{R}^+$. For $\mathcal{M}[?, c_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s - 1) \cdot c_2 \cdot \log_a n$.*

Proof: Let $c = c_2/\log_a n$. Let c_1' be some constant less than or equal to c_2. Let $c_2' = c_2$. Let $f(x,y) = y \cdot \log_a n$.

As $c_1' \leq c_2$ and $B = f(c_1', c_2') = c_2' \cdot \log_a n$, SC1 is satisfied. As $B\frac{c}{c_2'} = c_2'$, SC3 is satisfied. As c_1 is unknown, we do not consider SC2. ■

Corollary 5.2 *Let $c_1 \in \mathcal{R}^+$. For $\mathcal{M}[c_1, ?]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot c_2 \cdot \log_a n$.*

Proof: Let $c = 2 \cdot c_1$. Let $c_1' = c_1$. Let c_2' be some constant greater than or equal to c_1. Let $f(x,y) = y \cdot \log_a n$.

As $B = f(c_1', c_2') = c_2' \cdot \log_a n$, SC1 is satisfied. As $\frac{1}{2}c = c_1$ and $c_1' = c_1$, SC2 is satisfied. As c_2 is unknown, we do not consider SC3. ■

Each of Corollaries 5.1 and 5.2 separately implies that a lower bound for $\mathcal{M}[?,?]$ is $(s-1) \cdot c_2 \cdot \log_a n$.

5.2 Counting with Explicit Communication and Step Time Bounds

If processes know c_1 and c_2, then they can use both methods ST and EC in order to count. This results in an algorithm with running time $(s-1) \cdot c_2 \cdot \min\{\frac{c_2}{2c_1}, \Theta(\log n)\} + c_2$.

We now show that this bound is asymptotically tight.

Corollary 5.3 *Let $c_1, c_2 \in \mathcal{R}^+$. For $\mathcal{M}[c_1, c_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot c_2 \cdot \min\{\frac{c_2}{2c_1}, \log_a n\}$.*

Proof: Let $c = 2c_1$. Let $c_1' = c_1$. Let $c_2' = c_2$. Let $f(x,y) = y \cdot \min\{\frac{y}{x}, \log_a n\}$.

SC1 holds because $B = f(c_1', c_2') = c_2' \cdot \min\{\frac{c_2'}{2c_1'}, \log_a n\} \leq c_2' \cdot \log_a n$. SC2 holds because $c_1' = c_1 \leq \frac{c}{2}$. SC3 holds because $B\frac{c}{c_2'} = f(c_1', c_2') \cdot \frac{c}{c_2'} = c_2' \cdot \min\{\frac{c_2'}{2c_1'}, \log_a n\}\frac{2c_1}{c_2'} \leq c_2'$. ■

6 Message Passing Results

In this section, we show that the upper bounds we presented in Section 4 for the message passing models are asymptotically tight by obtaining the matching lower bounds. We use Theorem 3.1 to obtain the lower bounds. To prove a given lower bound for a message passing model, we simply check whether there exist c, c_1', c_2', d_1', and d_2' that satisfy MC1 through MC5.

6.1 Counting with Explicit Communication

The use of EC alone in a message passing model gives an upper bound of $(s-1) \cdot d_2 + c_2$.

We show that if no other method can be used, then this bound is asymptotically tight. The models that allow the use of EC only are models 0, 1, 2, 4, 5, 8, 9, and 10. Using corollaries to Theorem 3.1, we show that the lower bounds for models 5, 9 and 10 are $(s-1) \cdot d_2$. The result for model 5 implies the same lower bound for models 1 and 4 (since 1 and 4 have less timing information than does 5). The result for model 10 implies the same lower bound for models 2 and 8. The result for model 1 implies the same lower bound for model 0.

First, we give the corollary for model 5.

Corollary 6.1 *Let c_2 and d_2 be positive reals. For $\mathcal{M}[?, c_2, ?, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot d_2$.*

Proof: Let $c = \min\{\frac{c_2^2}{d_2}, \frac{c_2}{2}\}$. Let c_1' be some constant less than or equal to c_2. Let $c_2' = c_2$. Let d_1' be some constant less than or equal to d_2. Let $d_2' = d_2$. Let $f(w,x,y,z) = z$.

MC1 is satisfied by the choice of c_1', c_2', d_1' and d_2', and because $B = f(c_1', c_2', d_1', d_2') = d_2' = d_2$.

MC3 is satisfied because $B \cdot \frac{c}{c_2'} \leq d_2 \cdot \frac{c_2^2/d_2}{c_2'} \leq c_2$.

MC5 is satisfied because $(d_2' + B)\frac{c}{c_2'} - c \leq 2d_2 \cdot \frac{c_2/2}{c_2} - c < d_2$.

MC2 and MC4 are not considered because c_1 and d_1 are unknown. ■

Next we give the corollary for model 9.

Corollary 6.2 *Let c_1 and d_2 be positive reals. For $\mathcal{M}[c_1, ?, ?, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)d_2$.*

Proof: Let $c = 2c_1$. Let $c_1' = c_1$. Let $c_2' = 4c_1$. Let d_1' be some constant less than or equal to d_2. Let $d_2' = d_2$. Let $f(w,x,y,z) = z$.

MC1 is satisfied because $B = f(c_1', c_2', d_1', d_2') = d_2' = d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1$.

MC5 is satisfied because $(d_2' + B)\frac{c}{c_2'} - c \leq 2d_2 \frac{2c_1}{4c_1} = d_2$.

MC3 and MC4 are not considered because c_2 and d_1 are unknown. ■

Finally, we give the corollary for model 10.

Corollary 6.3 *Let c_1 and d_1 be positive reals. For $\mathcal{M}[c_1, ?, d_1, ?]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot d_2$.*

Proof: Let $c = \max\{2c_1, d_1\}$. Let $c'_1 = c_1$. Let c'_2 be some constant bigger than or equal to c_1. Let $d'_1 = d_1$. Let d'_2 be some constant bigger than or equal to d_1. Let $f(w, x, y, z) = z$.

MC1 is satisfied because $B = f(c'_1, c'_2, d'_1, d'_2) = d'_2$.

MC2 is satisfied because $\frac{c}{2} \geq \frac{2c_1}{2} = c_1$.

MC4 is satisfied because $(d'_2 - B)\frac{c}{c'_2} + c = c \geq d_1 = d'_1$.

MC3 and MC5 are not considered because c_2 and d_2 are unknown. ■

6.2 Counting with Explicit Communication and Step Time Bounds

If both methods EC and ST can be used, the resulting algorithm gives an upper bound of $(s-1) \cdot \min\{d_2, \frac{c_2}{c_1} \cdot c_2\} + c_2$.

We show that this bound is asymptotically tight if no other methods can be used and the following is true:

- $c_2 \leq d_2$.

The models that allow the use of EC and ST alone are models 12 and 13. We prove a corollary to Theorem 3.1 for model 13, which implies the same lower bound for model 12.

Corollary 6.4 *Let c_1, c_2, and d_2 be positive reals such that $c_2 \leq d_2$. For $\mathcal{M}[c_1, c_2, ?, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1) \cdot \min\{\frac{c_2}{2c_1}c_2, d_2\}$.*

Proof: Let $c = 2c_1$. Let $c'_1 = c_1$. Let $c'_2 = c_2$. Let d'_1 be some constant less than or equal to d_2. Let $d'_2 = d_2$. Let $f(w, x, y, z) = \min\{\frac{x}{2w}x, z\}$.

MC1 is satisfied because $B = f(c'_1, c'_2, d'_1, d'_2) = \min\{\frac{c_2^2}{2c_1}, d_2\} \leq d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1 = c'_1$.

MC3 is satisfied because $B \cdot \frac{c}{c'_2} = \min\{\frac{c_2^2}{2c_1}, d_2\} \cdot \frac{2c_1}{c_2} \leq c_2$.

For MC5, we need to show that $(d'_2 + B)\frac{c}{c'_2} - c \leq d_2$. Then, it suffices to prove that $(d_2 - ((d'_2 + B)\frac{c}{c'_2} - c)) \geq 0$.

$d_2 - ((d'_2 + B)\frac{c}{c'_2} - c) \geq d_2 - ((d_2 + \frac{c_2^2}{2c_1})\frac{2c_1}{c_2} - 2c_1) = d_2(1 - \frac{2c_1}{c_2}) - (c_2 - 2c_1) \geq 0$ since $c_2 \leq d_2$.

MC4 is not considered because d_1 is unknown. ■

6.3 Counting with Explicit Communication and Message Delay

Suppose methods EC and MD are used. The only model that can use these two methods alone is model 3 ($\mathcal{M}[?, ?, d_1, d_2]$). The resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{d_2}{d_1}u + 2c_2\}$, where $u = d_2 - d_1$.

We now argue that this bound is asymptotically tight if no other method can be used. First note that if the $2c_2$ term in the MD cost dominates the $\frac{d_2}{d_1}u$ term, then the upper bound is $\Theta(c_2)$ per session, which is obviously tight. Thus we ignore the $2c_2$ term.

Corollary 6.8 in Section 6.7 below shows that the lower bound for model 11 ($\mathcal{M}[c_1, ?, d_1, d_2]$) is $(s-1) \cdot \frac{d_2}{d_2+d_1}u$. Since model 3 is weaker than model 11, the same lower bound holds for model 3.

(Case 1) Suppose $d_2 \leq \frac{d_2}{d_1}u$. Then $\frac{d_2}{d_2+d_1}u \geq \frac{d_2}{3}$. Thus the asymptotic per session upper bound for model 3 is d_2 and the lower bound is $\frac{d_2}{3}$.

(Case 2) Suppose $\frac{d_2}{d_1}u < d_2$. Then $\frac{d_2}{d_2+d_1}u \geq \frac{d_2}{3d_1}u$. Thus the asymptotic per session upper bound for model 3 is $\frac{d_2}{d_1}u$ and the lower bound is $\frac{d_2}{3d_1}u$.

6.4 Counting with Explicit Communication and Combination 2

If methods EC and CB2 are used, the resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model that can use these two methods alone is model 6 ($\mathcal{M}[?, c_2, d_1, ?]$).

We now show this bound is asymptotically tight if $2c_2 \leq 3d_1$.

Under this assumption, the CB2 term, $\frac{d_2+c_2}{d_1} \cdot c_2$, is at most $\frac{5d_2}{2d_1} \cdot c_2$, which is at most $\frac{15}{4}d_2$.

Corollary 6.5 *Let c_2 and d_1 be positive reals such that $c_2 \leq \frac{3}{2}d_1$. For $\mathcal{M}[?, c_2, d_1, ?]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1) \cdot \frac{2d_2}{3d_1} \cdot c_2$.*

Proof: Let c_1' be some constant less than or equal to c_2. Let c_2' equal c_2. Let d_1' equal d_1. Let d_2' be some constant bigger than d_1. Let $c = \frac{3d_1}{2d_2'} \cdot c_2$. Let $f(w, x, y, z) = \frac{2z}{3y} \cdot x$.

MC1 holds because $B = f(c_1', c_2', d_1', d_2') = \frac{2d_2'}{3d_1'} \cdot c_2'$. By the assumption that $c_2 \leq \frac{3}{2}d_1$, B is less than or equal to d_2'.

MC3 holds because $B \cdot \frac{c}{c_2'} = c_2'$.

MC4 holds because $c_2 \leq \frac{3}{2}d_1$, and $(d_2' - B) \cdot \frac{c}{c_2'} + c = 3d_1 - c_2 + c \geq 3d_1 - 3/2d_1 + c > d_1 = d_1'$.

MC2 and MC5 are not considered because c_1 and d_2 are unknown. ■

6.5 Counting with Explicit Communication, Step Time Bounds, and Combination 2

If methods EC, ST, and CB2 are used, the resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{c_2}{c_1} \cdot c_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model in which exactly these methods can be used is model 14 ($\mathcal{M}[c_1, c_2, d_1, ?]$).

We now argue that this bound is asymptotically tight if no other method can be used, assuming $2c_2 \leq d_1$.

Corollary 6.6 below shows that the lower bound for model 14 is $(s-1) \cdot \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$, assuming $2c_2 \leq d_1$. We show that this bound is asymptotically tight.

Because $2c_2 \leq d_1$, $\frac{2d_2}{3d_1}c_2 \leq d_2$, and $\frac{d_2+c_2}{d_1} \leq \frac{3d_2}{2d_1}$. Thus the per session lower bound is $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$, and the per session upper bound is $\min\{d_2, \frac{c_2}{c_1}c_2, \frac{3d_2}{2d_1}c_2\}$.

Corollary 6.6 *Let c_2 and d_1 be positive reals such that $2c_2 \leq d_1$. For $\mathcal{M}[c_1, c_2, d_1, ?]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1) \cdot \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$.*

Proof: By the hypothesis of the corollary,

$$d_1 \geq 2c_2. \tag{1}$$

Let $c_1' = c_1$. Let $c_2' = c_2$. Let $d_1' = d_1$. Let d_2' be some real number bigger than $\frac{c_2}{2c_1}c_2$. Let $c = \max\{2c_1, \frac{3d_1}{2d_2'}c_2\}$.

Let $f(w, x, y, z) = \min\{\frac{x}{2w}x, \frac{2z}{3y}x\}$.

We show they satisfy the hypothesis of Theorem 3.1. MC5 is not considered since d_2 is unknown.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2'}{3d_1}c_2\}$.

Case 1 Suppose $\frac{c_2}{2c_1}c_2 < \frac{2d_2}{3d_1}c_2$. Then

$$\frac{2c_1}{c_2} > \frac{3d_1}{2d_2'}. \tag{2}$$

and thus $c = 2c_1$. For MC1, clearly $B = \frac{c_2}{2c_1}c_2 < d_2'$ by the definition of d_2'

For MC2, $\frac{c}{2} = c_1$.

For MC3, $B\frac{c}{c_2'} = c_2$.

For MC4,

$$\begin{aligned}
(d_2' - B)\tfrac{c}{c_2'} + c &= d_2'\tfrac{2c_1}{c_2} - c_2 + 2c_1 \\
&\geq d_2'\tfrac{3d_1}{2d_2'} - c_2 && \text{because of Eq. 2} \\
&\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} && \text{because of Eq. 1} \\
&= d_1.
\end{aligned}$$

Case 2 Suppose $\frac{c_2}{2c_1}c_2 \geq \frac{2d_2'}{3d_1}c_2$. Then

$$\frac{2c_1}{c_2} \leq \frac{3d_1}{2d_2'}. \tag{3}$$

and thus $c = \frac{3d_1}{2d_2'} \cdot c_2$. For MC1, $B = \frac{2d_2'}{3d_1}\frac{d_1}{2} < d_2'$ because of Eq. 1.

For MC2, $\frac{c}{2} = \frac{1}{2}\frac{3d_1}{2d_2'}c_2 \geq c_1$ because of Eq. 3.

For MC3, $B\frac{c}{c_2'} \leq \frac{2d_2'}{3d_1}c_2\frac{\frac{3d_1}{2d_2'}c_2}{c_2} = c_2$.

For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &= (d_2' - \tfrac{2d_2'}{3d_1}c_2)\tfrac{c}{c_2'} + c \\ &= \tfrac{3d_1}{2} - c_2 + c \\ &> \tfrac{3d_1}{2} - c_2 \\ &\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} \text{ because of Eq. 1} \\ &= d_1.\end{aligned}$$

■

6.6 Counting with Explicit Communication, Message Delay, and Combination 2

If methods EC, MD, and CB2 are used, the resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{d_2}{d_1} \cdot u + 2c_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model in which exactly these methods can be used is model 7 ($\mathcal{M}[?, c_2, d_1, d_2]$).

We now show this upper bound is asymptotically tight if

- $2c_2 \leq d_1$ and
- $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$.

As in Section 6.3, we ignore the $2c_2$ term in the MD expression.

Corollary 6.7 below proves that the per session lower bound is $(s-1)\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. We show that this bound is asymptotically tight.

(Case 1) If $\frac{d_2}{d_1}u \leq \min\{d_2, \frac{d_2+c_2}{d_1}c_2\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_1}{3d_2}u$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$ (cf. Section 6.3); and since $c_2 \leq \frac{1}{2}d_1$, $\frac{d_2}{d_1}u \leq \frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$ and thus, $\frac{d_2}{3d_1}u < \frac{2d_2}{3d_1}c_2$.

(Case 2) If $\frac{d_2+c_2}{d_1}c_2 \leq \min\{d_2, \frac{d_2}{d_1}u\}$, then $\frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$ because $c_2 \leq \frac{1}{2}d_1$. Thus the per-session upper bound is $\Theta(\frac{d_2}{d_1}c_2)$. The lower bound is $\frac{d_1}{3d_2}c_2$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and because $\frac{d_2}{d_1}u \geq \frac{d_2+c_2}{d_1}c_2 > \frac{d_2}{d_1}c_2$, $\frac{d_2}{3d_1}u > \frac{d_2}{3d_1}c_2$.

(Case 3) If $d_2 \leq \min\{\frac{d_2}{d_1}u, \frac{d_2+c_2}{d_1}c_2\}$, then the upper bound is $\Theta(d_2)$. The lower bound is $\frac{1}{3}d_2$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and since $c_2 \leq \frac{1}{2}d_1$, $d_2 \leq \frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$, and thus, $\frac{1}{3}d_2 < \frac{2d_2}{3d_1}c_2$.

Corollary 6.7 *Let c_2, d_1, and d_2 be positive reals such that $2c_2 \leq d_1$ and either $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$. For $\mathcal{M}[?, c_2, d_1, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1)\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$,*

Proof: Case 1: $d_2 \leq \frac{3}{2}d_1$.

Then $\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\} \leq \frac{2d_2}{3d_1}c_2 = c_2$. The lower bound holds because a process has to take at least s steps to solve the (s, n) session problem.

Case 2: $d_2 \geq \frac{3}{2} \cdot d_1 + c_2$ and $c_2 \leq \frac{3}{2}d_1$.

Let $c = \frac{3d_1}{2d_2}c_2$. Let c_1' be some constant less than or equal to c_2. Let c_2' equal c_2. Let d_1' equal d_1. Let d_2' equal d_2. Let $f(w, x, y, z) = \min\{\frac{2z}{3y}x, \frac{z}{z+y}(z-y)\}$.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

MC1 holds because $B \leq \frac{2d_2}{3d_1}c_2 < d_2$, since $2c_2 \leq d_1$.

MC3 holds because $B \cdot \frac{c}{c_2} \leq \frac{2d_2}{3d_1} \cdot c_2 \cdot \frac{c}{c_2} \leq c_2$.

MC4 holds because $(d_2' - B)\frac{c}{c_2} + c > (d_2' - \frac{2d_2}{3d_1}c_2)\frac{3d_1}{2d_2}\frac{c_2}{c_2} = \frac{3}{2}d_1 - c_2$. This quantity is grater than or equal to $\frac{3}{2}d_1 - \frac{1}{2}d_1$ since $d_1 \geq 2c_2$. Thus the expression is greater than or equal to d_1. MC5 holds because $(d_2' + B)\frac{c}{c_2} - c < (d_2 + \frac{2d_2}{3d_1}c_2)\frac{3d_1}{2d_2}\frac{c_2}{c_2} = \frac{3}{2}d_1 + c_2 \leq d_2$ since $\frac{3}{2}d_1 + c_2 \leq d_2$.

MC2 is not considered because c_1 is unknown. ■

6.7 Counting with Explicit Communication, Message Delay, and Combination 1

Suppose methods EC, MD, and CB1 are used. The resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{d_2}{d_1} \cdot u + 2c_2, \frac{c_2}{c_1} \cdot u + 2c_2\}$. The only model in which exactly these methods can be used is model 11 ($\mathcal{M}[c_1, ?, d_1, d_2]$).

Corollary 6.8 below proves that the lower bound for model 11 is $(s-1) \cdot \frac{d_2}{d_2+d_1}u$. We now show this bound is asymptotically tight. As in Section 6.3, we ignore the $2c_2$ term in the MD and CB1 expressions.

(Case 1) If $\frac{c_2}{c_1}u \leq \frac{d_2}{d_1}u \leq d_2$, the upper bound is $\Theta(\frac{c_2}{c_1}u)$. Because $\frac{d_2}{d_1}u \leq d_2$, then $d_2 \leq 2d_1$. Also $\frac{d_2}{d_1+d_2}u \geq \frac{d_2}{3d_1}u \geq \frac{c_2}{3c_1}u$. Thus, the lower bound is $\frac{c_2}{3c_1}u$.

(Case 2) If $\frac{c_2}{c_1}u \leq d_2 < \frac{d_2}{d_1}u$, the upper bound is $\Theta(\frac{c_2}{c_1}u)$. Because $\frac{d_2}{d_1}u > d_2$, then $d_2 > 2d_1$. Also $\frac{d_2}{d_1+d_2}u \geq \frac{c_2}{c_1}u \cdot \frac{d_2-d_1}{d_2+d_1}$, which is greater than and equal to $\frac{d_2}{3c_1}u$ because $\frac{d_2-d_1}{d_2+d_1} \geq \frac{1}{3}$. Thus, the lower bound is $\frac{c_2}{3c_1}u$.

(Case 3) If $\frac{d_2}{d_1}u \leq \min\{d_2, \frac{c_2}{c_1}u\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_2}{3d_1}u$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$.

(Case 4) If $d_2 < \min\{\frac{d_2}{d_1}u, \frac{c_2}{c_1}u\}$, the upper bound is $\Theta(d_2)$. The lower bound is $\frac{d_2}{3}$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and $\frac{d_2}{3} \leq \frac{2d_2}{3d_1}c_2$.

Corollary 6.8 *Let c_1, d_1, and d_2 be positive reals. For $\mathcal{M}[c_1, ?, d_1, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1) \cdot \frac{d_2}{d_2+d_1}u$.*

Proof: Let $c = 2c_1$. Let $c'_1 = c_1$. Let $c'_2 = \frac{4d_2c_1}{d_2+d_1}$. Let $d'_1 = d_1$. Let $d'_2 = d_2$. Let $f(w, x, y, z) = \frac{z}{z+y} \cdot (z - y)$.

MC1 is satisfied because $B = f(c'_1, c'_2, d'_1, d'_2) = \frac{d'_2}{d'_2+d'_1} \cdot (d'_2 - d'_1) = \frac{d_2}{d_2+d_1} \cdot (d_2 - d_1) < d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1$.

MC4 is satisfied because $(d'_2 - B)\frac{c}{c'_2} + c = (d_2 - \frac{d_2u}{d_2+d_1})\frac{2c_1}{\frac{4d_2c_1}{d_2+d_1}} + c = d_1 + c > d_1$.

MC5 is satisfied because $(d'_2 + B)\frac{c}{c'_2} - c = (d_2 + \frac{d_2u}{d_2+d_1})\frac{2c_1}{\frac{4d_2c_1}{d_2+d_1}} - c = d_2 - c < d_2$.

MC3 is not considered because c_2 is not known. ■

6.8 Counting with All Methods

Suppose all five methods (EC, ST, MD, CB1 and CB2) are used. The resulting asymptotic upper bound on the per session cost is $\min\{d_2, \frac{c_2}{c_1} \cdot c_2, \frac{d_2}{d_1} \cdot u + 2c_2, \frac{d_2+c_2}{d_1} \cdot c_2, \frac{c_2}{c_1} \cdot u + 2c_2\}$. The only model in which all these methods can be used is model 15 ($\mathcal{M}[c_1, c_2, d_1, d_2]$).

We now show this upper bound is asymptotically tight if

- $2c_2 \leq d_1$ and
- $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$.

Corollary 6.9 below shows the per session lower bound for model 15 is $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. As in Section 6.3, we ignore the $2c_2$ term in the MD and CB1 expressions.

(Case 1) If $\frac{c_2}{c_1}c_2 \leq \min\{d_2, \frac{d_2}{d_1}u, \frac{d_2+c_2}{d_1}c_2, \frac{c_2}{c_1} \cdot u\}$, then the upper bound is $\Theta(\frac{c_2}{c_1}c_2)$. The lower bound is $\frac{c_2}{3c_1}c_2$ because (1) $\frac{c_2}{2c_1}c_2 \geq \frac{c_2}{3c_1}c_2$; (2) $\frac{3d_2}{2d_1}c_2 \geq \frac{d_2+c_2}{d_1}c_2 \geq \frac{c_2}{c_1}c_2$, and thus $\frac{2d_2}{3d_1}c_2 > \frac{c_2}{3c_1}c_2$; and (3) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \geq \frac{c_2}{3c_1}c_2$.

(Case 2) If $\frac{d_2}{d_1}u \leq \min\{d_2, \frac{c_2}{c_1}c_2, \frac{d_2+c_2}{d_1}c_2, \frac{c_2}{c_1}u\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_2}{3d_1}u$ because (1) $\frac{c_2}{c_1}c_2 \geq \frac{d_2}{d_1}u$, and thus, $\frac{c_2}{2c_1}c_2 \geq \frac{d_2}{3d_1}u$; (2) $\frac{3d_2}{2d_1}c_2 \geq \frac{d_2+c_2}{d_1}c_2 \geq \frac{d_2}{d_1}u$, and thus, $\frac{2d_2}{3d_1}c_2 > \frac{d_2}{3d_1}u$; and (3) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$ and $\frac{d_2}{3d_1}u$.

(Case 3) If $\frac{c_2}{c_1}u \leq \min\{d_2, \frac{c_2}{c_1}c_2, \frac{d_2+c_2}{d_1}c_2, \frac{d_2}{d_1}u\}$, then the upper bound is $\Theta(\frac{c_2}{c_1}u)$. The lower bound is $\frac{4c_2}{15c_1}u$ because (1) $\frac{c_2}{2c_1}c_2 \geq \frac{c_2}{2c_1}u \geq \frac{c_2}{3c_1}u$; (2) $\frac{3d_2}{2d_1}c_2 \geq \frac{d_2+c_2}{d_1}c_2 \geq \frac{c_2}{c_1}u$, and thus, $\frac{2d_2}{3d_1}c_2 > fracc_23c_1u$; and (3) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \geq \frac{c_2}{3c_1}u$.

(Case 4) $\frac{d_2+c_2}{d_1}c_2 \leq \min\{d_2, \frac{c_2}{c_1}c_2, \frac{c_2}{c_1}u, \frac{d_2}{d_1}u\}$, then $\frac{d_2+c_2}{d_1}c_2 \geq \frac{d_2}{d_1}c_2$, and the upper bound is $\Theta(\frac{d_2}{d_1}c_2)$. The lower bound is $\frac{d_2}{3d_1}c_2$ because (1) $\frac{c_2}{2c_1}c_2 \geq \frac{d_2}{2d_1}c_2 \geq \frac{d_2}{3d_1}c_2$; and (2) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \geq \frac{d_2+c_2}{3d_1}c_2 > \frac{d_2}{3d_1}c_2$; and (3) $\frac{d_2+c_2}{d_1}c_2 \geq \frac{d_2}{d_1}c_2 \geq \frac{d_2}{3d_1}c_2$.

(Case 5) If $d_2 \geq \min\{\frac{c_2}{c_1}c_2, \frac{c_2}{c_1}u, \frac{d_2+c_2}{d_1}c_2, \frac{d_2}{d_1}u\}$, then the upper bound is $\Theta(d_2)$. The lower bound is $\frac{d_2}{3}$ because (1) $\frac{c_2}{2c_1}c_2 \geq \frac{d_2}{3}$; (2) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$; and (3) $\frac{3d_2}{2d_1}c_2 \geq \frac{d_2+c_2}{d_1}c_2 \geq d_2$, and $\frac{3d_2}{3d_1}c_2 > \frac{d_2}{3}$.

Corollary 6.9 *Let c_1, c_2, d_1, and d_2 be positive reals such that $2c_2 \leq d_1$ and either $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$. For $\mathcal{M}[c_1, c_2, d_1, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}(s-1)$.*

Proof: Case 1: $c_2 \geq \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

The lower bound clearly holds because all processes have to take at least s steps to solve the (s, n)-session problem and c_2 is the upper bound on step time.

Case 2: $c_2 < \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

Note that in this case, by the hypothesis of the corollary, the following are true:

$$d_1 \geq 2c_2. \tag{4}$$

$$d_2 \geq \frac{3d_1}{2} + c_2. \tag{5}$$

Let $c = \max\{2c_1, \frac{3d_1}{2d_2}c_2\}$. Let $c_1' = c_1$. Let $c_2' = c_2$. Let $d_1' = d_1$. Let $d_2' = d_2$. Let $f(w, x, y, z) = \min\{\frac{x}{2w}x, \frac{2z}{3y}x, \frac{z}{z+y}(z-y)\}$.

We show they satisfy the hypothesis of Theorem 3.1.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

Case 2.A Suppose $\frac{c_2}{2c_1}c_2 < \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. Then $B = \frac{c_2}{2c_1}c_2$ and $c = 2c_1$. Furthermore,

$$\frac{c_2}{2c_1} \leq \frac{2d_2}{3d_1}. \tag{6}$$

For MC1, clearly $B = \frac{c_2}{2c_1}c_2 \leq \frac{d_2}{d_2+d_1}u < d_2$.
For MC2, $\frac{c}{2} = c_1$.

For MC3, $B\frac{c}{c_2'} = c_2$.
For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &= d_2\tfrac{2c_1}{c_2} - c_2 + 2c_1 \\ &\geq d_2\tfrac{3d_1}{2d_2} - c_2 && \text{because of Eq. 6} \\ &\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} && \text{because of Eq. 4} \\ &= d_1.\end{aligned}$$

For MC5, it suffices to prove that $(d_2 - ((d_2' + B)\frac{c}{c_2'} - c)) \geq 0$.

$$\begin{aligned}d_2 - ((d_2' + B)\tfrac{c}{c_2'} - c) &\geq d_2 - ((d_2 + \tfrac{c_2^2}{2c_1})\tfrac{2c_1}{c_2} - 2c_1) \\ &= d_2(1 - \tfrac{2c_1}{c_2}) - (c_2 - 2c_1) \\ &\geq 0 && \text{because of Eq. 4}\end{aligned}$$

Case 2.B Suppose $\frac{c_2}{2c_1}c_2 \geq \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. Then $B = \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$ and $c = \frac{3d_1}{2d_2}c_2$. Furthermore,

$$\frac{c_2}{2c_1} > \frac{2d_2}{3d_1}. \tag{7}$$

For MC1, $B \leq \frac{d_2}{d_2+d_1}u < d_2$.
For MC2, $\frac{c}{2} = \frac{1}{2}\frac{3d_1}{2d_2}c_2 \geq c_1$ because of Eq. 7.
For MC3, $B\frac{c}{c_2'} \leq \frac{2d_2}{3d_1}c_2\frac{\frac{3d_1}{2d_2}c_2}{c_2} = c_2$.
For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &\geq (d_2' - \tfrac{2d_2}{3d_1}c_2)\tfrac{c}{c_2'} + c \\ &= \tfrac{3d_1}{2} - c_2 + c \\ &> \tfrac{3d_1}{2} - c_2 \\ &\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} \text{ because of Eq. 4} \\ &= d_1.\end{aligned}$$

For MC5,

$$\begin{aligned}(d_2' + B)\tfrac{c}{c_2'} - c &\leq (d_2' + \tfrac{2d_2}{3d_1}c_2)\tfrac{c}{c_2'} - c \\ &= \tfrac{3d_1}{2} + c_2 - c \\ &< \tfrac{3d_1}{2} + c_2 \\ &\leq d_2 \text{ because of Eq. 5.}\end{aligned}$$

■

7 Conclusion and Discussion

This paper concerns timing models in distributed systems that lie between the synchronous and asynchronous models. Four timing parameters are considered: the maximum and minimum process step times and message delays. Timing models are obtained by considering independently

whether each parameter is known (i.e., is hard-wired into processes' code) or unknown, giving rise to four shared memory models and 16 message passing models.

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 systems and message passing systems. In this paper, the session problem is continued to be used to compare quantitatively a family of models in which various parameters are either known or unknown.

For each unknown parameter model, we obtain an asymptotically tight time complexity bound on the session problem. Two of the algorithms were previously known, while the other three are new. We categorize the algorithms in terms of "ways to count". The intuition is that processes must have some way to count the passage of other processes' steps in order to "know" when a session has occurred. Our matching lower and upper bounds indicate[[that the algorithms are the optimal]] ways to count, and allow us to construct a lattice of timing models in terms of the counting algorithms that are applicable to a model (cf. Figure 1). This hierarchy confirms the common belief that as a model has more timing knowledge, it behaves more like the synchronous model.

All but one of our lower bound results are new and all of them are obtained by one modular lower bound proof. The lower bound technique combines those of [AFL83] for the asynchronous model in the shared memory system and [AM94] for the asynchronous and semi-synchronous models in the message passing systems. Our technique identifies one sufficient condition for a lower bound in a timing model to hold in solving the (s, n)-session problem, and is applicable to every unknown parameter model as well as to those models previously studied for the session problem.

For several unknown parameter models, we are not able to show the lower bound without making some assumptions about the timing parameters. It will be interesting to develop a new lower bound technique that can show either the same or tighter lower bounds without such assumptions.

It will be also interesting to see whether our modular lower bound proof technique can be applied to show lower bounds for other distributed computing problems, such as the mutual exclusion problem and the dining philosophers problem in many different timing models.

References

[AAT94] R. Alur, H. Attiya and G. Taubenfeld, "Time-Adaptive Algorithms for Synchronization," *SIAM Journal on Computing,* vol. 26, no. 2, 1997, pp. 539–556.

[ADLS94] H. Attiya, C. Dwork, N. A. Lynch and L. J. Stockmeyer, "Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty," *Journal of the ACM*, vol. 41, no. 1 (January 1994), pp. 122–152.

[AFL83] 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.

[AL89] H. Attiya and N. A. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," *Information and Computation,* vol. 110, no. 1 (April 1994), pp. 183–232.

[AM94] H. Attiya and M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Networks," *Mathematical Systems Theory*, vol. 27, 1994, pp. 547–571.

[AT92] Rajeev Alur and Gadi Taubenfeld, "Fast Timing-Based Algorithms," *Distributed Computing,* vol. 10, no. 1, 1996, pp. 1-10.

[Ba78] G. Baudet, "Asynchronous Interactive Methods for Multi-Processors," *Journal of the ACM*, vol. 32, no. 4, 1978, pp. 226–244.

[CT90] B. A. Coan and G. Thomas, "Agreeing on a Leader in Real-Time," *Proceedings of the IEEE 11th Real-Time Systems Symposium*, Lake Buena Vista, FL, Dec. 1990, pp. 1–7.

[CW90] B. A. Coan and J. L. Welch, "Transaction Commit in a Realistic Timing Model," *Distributed Computing,* vol. 4, 1990, pp. 87–103.

[DDS87] 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.

[DLS88] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM,* vol. 35, no. 2, 1988, pp. 288-323.

[FLP85] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, no. 2, 1985, pp. 374-382.

[GVW89] J. Goodman, M. Vernon, and P. Woest, "Efficient Synchronization Primitives for Large-Scale Cache Coherent Multiprocessors," *Proc. of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems* 1989, pp. 64–75.

[Ly96] N. Lynch, *Distributed Algorithms*, Morgan-Kaufman Publishers, Inc., San Francisco, CA, 1996.

[LS92] Nancy Lynch and Nir Shavit, "Timing-Based Mutual Exclusion," *Proceedings of the IEEE 13th Real-Time Systems Symposium*, San Antonio, TX, December, 1992, pp. 2–11.

[Ma93] M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Systems: Atomic Shared Memory," *Computers and Mathematics with Applications*, vol. 25, no. 2, Jan. 1993, pp. 81–91.

[Po91] S. Ponzio, "Consensus in the Presence of Timing Uncertainty: Omission and Byzantine Failures," *Proc. ACM Symposium on Principles of Distributed Computing,* Oct. 1991, pp. 125–137.

[RW92] I. Rhee and J. Welch, "The Impact of Time on the Session Problem," *Proc. of 11th ACM Symposimum on Principles of Distributed Computing,* Vancouver, Canada, August 1992, pp. 191–201.

A Communication in SM

Consider an $(a-1)$-ary tree with n leaves in which each level, except possibly the lowest, has the maximum number of nodes. The number of levels in the tree is $\lceil \log_{a-1} n \rceil + 1$. Associated with each node in the tree are a process and a shared variable. Each port process and its port variable are associated with a leaf node; the processes and variables associated with internal nodes are called *relay* processes and variables. The relay variable associated with a node is accessed by the process associated with the node and the processes associated with that node's children in the tree. Figure 8 illustrates a tree with $a = 4$ and $n = 7$.

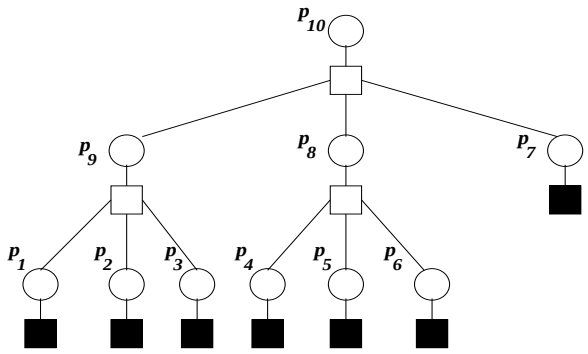

Figure 8: A tree network with $a = 4$ and $n = 7$ where circles represent processes, empty squares relay variables, dark squares port variables, solid lines memory access patterns of processes. Port processes are p_1 through p_7.

Each relay variable has two fields, *up* and *down*. Each process has two local variables, *lup* and *ldown*; initially they are empty except that *lup* at a port process holds the information to be propagated.

Each relay process p other than the root repeats the following two steps. First, p accesses its own shared variable, saving the contents of the *up* field in *lup* and appending the contents of *ldown* to the *down* field. Second, p accesses its parent's variable, appending *lup* to the *up* field and saving the *down* field in *ldown*.

The root continuously accesses its own variable; at each access it copies the *up* field to the *down* field.

In the example of Figure 8, a piece of information m is transferred from p_2 to p_6 as follows: p_2 appends m to $v_9.up$; p_9 obtains the contents of $v_9.up$ and appends them to $v_{10}.up$; p_{10} copies the contents of $v_{10}.up$ to $v_{10}.down$; p_8 obtains the contents of $v_{10}.down$ and appends them to $v_8.down$; p_6 obtains the contents of $v_8.down$ and gets m.

It takes at most $c_{max} \cdot \lceil \log_{a-1} n \rceil$ time for a piece of information (or a "message") to be relayed up to the root for the following reason. In each time interval of length c_{max}, every process takes at least one step, and thus every relay process other than the root passes the message up to its parent. Likewise, it takes additional $c_{max} \cdot (\lceil \log_{a-1} n \rceil + 1)$ time for the message at the root to be relayed down to a leaf node in the tree (the one additional c_{max} is for the root to move the message from its *up* to its *down*). Thus, the broadcast is accomplished in $c_{max} \cdot \Theta(\log_{a-1} n)$ time. A similar tree network is mentioned in [AFL83].

When we say *broadcast* in the SM model, it implies all the interaction of processes in the tree network to accomplish the broadcasting. 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.

B Correctness proofs of counting methods

B.1 Correctness proof of CB1

Theorem B.1 *CB1 solves the (s, n)-session problem within time $(s-2) \cdot (\frac{c_2}{c_1} u + u + 2c_2) + d_2 + 3c_2$ if c_1, d_1 and d_2 are known.*

Proof: Consider an arbitrary admissible timed computation C of Technique CB1.

For each k, $0 \leq k \leq s-1$, let p_{i_k} be the first process that sets its *session* variable to k in C. To increment *session*, a process must receive a set of messages that satisfy condition 1 in Figure 5. Let M_k be the set of messages received by p_{i_k} that cause p_{i_k} to set $session_{i_k}$ to k (M_0 is the empty set); let m_k be the message which is sent last among M_k (if there is a tie, choose an arbitrary message among them).

Lemma B.2 *Let π be the step which sends m_k. There are at least k sessions by the time that 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 sends m_{k-1} and σ be the step in which $p_{i_{k-1}}$ sets $session_{i_{k-1}}$ to $k-1$. Such a step exists because $session_{i_{k-1}}$ is always incremented by one. For $p_{i_{k-1}}$ to update $session_{i_{k-1}}$, condition 1 in Figure 5 must hold.

Let t be the time when τ occurs and t' be the time when σ occurs.

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 delays. Thus, $t'-t \geq d_1$ because σ occurs after $p_{i_{k-1}}$ receives m_{k-1}. Note that *count* in the algorithm is reset whenever *session* is updated. Let t'' be the time that p_{i_k} sets $session_{i_k}$ to k. From the code, because condition 1 should be true before *session* is updated, $count_{i_k}$ must equal to B at time t''. Thus, when $count_{i_k}$ is equal to B, at least $B \cdot c_1$ time has elapsed since time t' because t' is the time that *session* is updated to $k-1$ for the first time in computation C and $count_{i_k}$ must be reinitialized after time t. Thus, $t'' \geq t' + Bc_1 = t' + d_2 - d_1 \geq t + d_2$ because $t' - t \geq d_1$.

Therefore, the difference between times t and t'' is bigger than d_2. Thus, all messages received at time t'' or later must be sent after time t, at which time there were $k-1$ sessions by the inductive 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 are at least k sessions by the time π occurs. ■

From Lemma B.2, 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 all processes take additional steps after there are at least $s-1$ sessions (to receive a message), there are at least s sessions in C. Thus the algorithm is correct.

We now calculate the running time of the algorithm. We define for each k, $2 \leq k \leq s-1$, $T_k = \max\{t : p_i \text{ sets } session_i \text{ to } k \text{ at time } t \text{ in } C \text{ for all } p_i \in R\}$. T_k is the latest time that a process sets *session* to k.

Lemma B.3 *For each k, $2 \leq k \leq s-1$, $T_{k+1} \leq T_k + \frac{c_2}{c_1}u + u + 2c_2$.*

Proof: After a process p_i sets $session_i$ to k, it takes at most $\frac{u}{c_1}c_2$ time for *count* to be bigger than B. Then, it takes at most Δ time additionally for condition 1 to be true (i.e., for at least one message ¿from every process to be received after *count* becomes bigger than B). We prove that $\Delta \leq u + 2c_2$.

Let m_1 be the message that is received from process p_j by p_i just before condition 1 becomes true in p_i (i.e., p_i has waited $\frac{u}{c_1}c_2$ time). Message m_1 exists because condition 1 becomes true only if there are enough messages in *msgs* which is emptied only after condition 1 becomes true. Let t be the time that m_1 is sent. p_j must broadcast another message m_2 within $t+c_2$ to process i because according to the code, all processes broadcast a message at every step. m_2 will be delivered to buf_i by time $t + c_2 + d_2$ (because it takes at most d_2 delay for a message to arrive at a buffer) and be received by time $t + 2c_2 + d_2$ (because it takes at most c_2 time for a process to take a step).

Process p_i will receive m_1 at time bigger than or equal to $t + d_1$ because it is sent at time t and it takes at least d_1 time delay for a message to arrive at a buffer. Since process i receives m_2 by time $t + 2c_2 + d_2$, the maximum time difference between the time that process i receives m_1 and the time that it receives m_2 is $(t + 2c_2 + d_2) - (t + d_1) = d_2 - d_1 + 2c_1 = u + 2c_2$. Therefore, $\Delta = u + 2c_2$. ■

By the algorithm, initially it takes at most $d_2 + 2c_2$ time to receive at least one message from all processes in order to accomplish the first session. Therefore $T_1 = d_2 + 2c_1$. Using Lemma B.3, T_{s-1}, which is the latest time that a process sets *session* to $s - 1$, is at most $(s - 2) \cdot (\frac{c_2}{c_1}u + u + 2c_2) + T_1$. After T_{s-1}, a process takes one step to complete s sessions. Therefore, a process enters the idle state by time $(s - 2) \cdot (\frac{c_2}{c_1}u + u + 2c_2) + d_2 + 2c_2 + c_2$. ■

B.2 Correctness proof of MD

Theorem B.4 *MD solves the (s, n)-session problem within time $(s-2)\cdot(\frac{d_2}{d_1}u+u+2c_2)+d_2+2c_2$ if d_1 and d_2 are known.*

Proof: MD differs from CB1 only in the way that *count* is incremented and in that B is set to u/d_1. The rest of the code is same. The correctness proof is similar to that of Technique CB1.

Since *count* and B affect condition (*count* $\geq B$) in the code, we only need to prove that when $B <$ *count*, at least time u has passed since the last time *session* was incremented. *count* is incremented to k only when a process receives $m(j, session, k - 1)$ for any j and for some value of *session*. When we apply this argument inductively, we prove that there must be a chain of processes $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ where p_{i_r} receives $m(i_{r-1}, session, r - 1)$ from $p_{i_{r-1}}$. Thus, when *count* is B, at least time $Bd_1 = d_2 - d_1 = u$ has passed after p_{i_1} sent $m(i_1, session, 1)$ because it takes at least d_1 message delay for a message to be received after it is sent.

For time complexity, it takes at most time $d_2 + c_2$ for p_{i_k} to receive $m(i_{k-1}, session, k - 1)$ after $p_{i_{k-1}}$ receives $m(i_{k-2}, session, k - 2)$. Therefore, for *count* to be bigger than or equal to B, it takes at most $B(d_2 + c_2)$. After that, for condition 1 in the code to be true it takes at most $(u + 2c_2)$ as it is proved in the proof of Lemma B.3. Therefore, the running time of Technique MD is $(s - 2) \cdot (\frac{d_2+c_2}{d_1}u + u + 2c_2) + d_2 + 2c_2$. ■

B.3 Correctness proof of CB2

Theorem B.5 *Technique CB2 solves the (s, n)-session problem in time $(s - 1) \cdot \frac{d_2+c_2}{d_1}c_2 + c_2$ if c_2 and d_1 are known.*

Proof: *count* is incremented to k only when a process receives $m(j, *, k-1)$ for any j. When we apply this argument inductively, we prove that there must be a chain of processes $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ where p_{i_r} receives $m(i_{r-1}, *, r - 1)$ from $p_{i_{r-1}}$. Thus, when *count* is B, at least time $Bd_1 =$

$\frac{c_2}{d_1}(s-1)d_1$ has passed after p_{i_1} sent $m(i_1, *, 1)$ because it takes at least d_1 message delay for a message to be received after it is sent. Thus, when $count > B$, at least time $c_2(s-1)$ has passed since the start of the computation. The theorem follows. ■