Efficiency of Semi-Synchronous versus Asynchronous Networks[1]

Hagit Attiya[2]
Department of Computer Science
The Technion
Haifa 32000, Israel

Marios Mavronicolas[3]
Aiken Computation Laboratory
Harvard University
Cambridge, MA 02138

September 1992

[1]A preliminary version of this paper appeared in proceedings of *the 28th annual Allerton Conference on Communication, Control and Computing,* October 1990, pp. 578–587.

[2]Partially supported by B. and G. Greenberg Research Fund (Ottawa) and by Technion V.P.R. funds. Part of this work was performed when the author was at the Laboratory for Computer Science, MIT, supported by ONR contract N00014-85-K-0168, by NSF grant CCR-8915206 and by DARPA contract N00014-87-K-0825.

[3]Supported by ONR contract N00014-91-J-1981. Current address: Department of Computer Science, University of Crete, Heraklion, Crete 71110, Greece.

Abstract

The *s-session problem* is studied in *asynchronous* and *semi-synchronous* networks. Processes are located at the nodes of an undirected graph G and communicate by sending messages along links that correspond to the edges of G. A session is a part of an execution in which each process takes at least one step; an algorithm for the s-session problem guarantees the existence of at least s disjoint sessions. The existence of many sessions guarantees a degree of interleaving which is necessary for certain computations. It is assumed that the (real) time for message delivery is at most d. In the asynchronous model it is assumed that the time between any two consecutive steps of any process is in the interval $[0, 1]$; in the semi-synchronous model the time between any two consecutive steps of any process is in the interval $[c, 1]$ for some c such that $0 < c \leq 1$, the *synchronous* model being the special case where $c = 1$. All processes are initially synchronized and take a step at time 0.

For the asynchronous model, an upper bound of $diam(G)(d+1)(s-1)$ and a lower bound of $diam(G)d(s-1)$ are presented; $diam(G)$ is the *diameter* of G. For the semi-synchronous model, an upper bound of $1 + \min\{\lfloor \frac{1}{c} \rfloor + 1, diam(G)(d+1)\}(s-2)$ is presented. The main result of the paper is a lower bound of $1 + \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}(s-2)$ for the time complexity of any semi-synchronous algorithm for the s-session problem, under the assumption that $d \geq \frac{d}{\min\{\lfloor 1/2c \rfloor, diam(G)d\}} + 2$. These results imply a time separation between semi-synchronous (in particular, synchronous) and asynchronous networks. Similar results are proved for the case where delays are not uniform.

1 Introduction

Most distributed systems are based on a *communication network* — a collection of n processes arranged at the nodes of an undirected graph G and communicating by sending messages across links of this graph. Central to the programming of distributed systems are *synchronization* problems, where a process is required to guarantee that *all* processes have performed a particular set of steps. Naturally, the timing information available to processes has critical impact on the time complexity of synchronization.

Arjomandi, Fischer and Lynch ([1]) introduced the *session problem* to study the impact of timing information on the time complexity of synchronization. Roughly speaking, a *session* is a sequence of events that contains *at least* one step by each process. An algorithm for the *s-session problem* guarantees that each execution of the algorithm includes at least s disjoint sessions.

The session problem is an abstraction of the synchronization needed for the execution of some tasks that arise in a distributed system, where separate components are each responsible for performing a small part of a computation. Consider, for example, a system which solves a set of equations by successive relaxation, where every process holds part of the data (cf. [5]). Interleaving of steps by different processes is necessary in order to ensure that a correct value was computed, since it implies sufficient interaction among the intermediate values computed by the processes. Any algorithm which ensures that sufficient interleaving has occurred also solves the s-session problem. The session problem is also an abstraction of some problems in real-time computing which involve synchronization of several computer system components, in order that they cooperate in performing a task involving real-world components. For example, multiple robots might cooperate to build a car on an assembly line, with each robot responsible for assembling a small piece of the machinery. Interleaving of assembly actions by different robots is necessary to ensure that pieces are assembled in the right order; a robot should not put the next item on the assembly line before all robots have completed a particular set of assembly actions making it possible for the item to "fit in". Clearly, any algorithm which ensures that sufficient interleaving has occurred also solves the s-session problem. Thus, the difficulty of solving the s-session problem reflects those of implementing the successive relaxation method and building the car on the assembly line.

Arjomandi, Fischer and Lynch ([1]) assume that processes communicate via *shared variables* and the time complexity of the session problem was studied in *synchronous* and *asynchronous* models. Informally, in a *synchronous* system, processes operate in lock-step, taking steps

simultaneously, while in an *asynchronous* system, processes work at completely independent rates and have no way to estimate time. The results of Arjomandi, Fischer and Lynch ([1]) show that there is a significant gap between the time complexities of solving the session problem in the synchronous and the asynchronous models.

In reality, however, there is an important middle ground between the synchronous and the asynchronous models of computation: in most distributed systems, processes operate neither at lock-step nor at a completely independent rate. For example, processes may have access to inaccurate clocks that operate at approximately, but not exactly, the same rate. We model these *semi-synchronous* systems by assuming that there exist a lower and an upper bound on processes' step time that enable processes to estimate time.

Following Arjomandi, Fischer and Lynch ([1]), we address the cost of synchronization in semi-synchronous and asynchronous communication networks by presenting upper and lower bounds for the time complexity of solving the s-session problem.

Informally, the *time complexity* of an algorithm is the maximal time, over all executions, until every process stops executing the algorithm. The following timing assumptions are made on the system. Messages sent over any communication link incur a delay in the range $[0, d]$, where $d \geq 0$ is a known constant. In the *asynchronous* model, processes' step time is in the range $[0, 1]$; in the *semi-synchronous* model, processes' step time is in the range $[c, 1]$, for some parameter c such that $0 < c \leq 1$.[1] Processes are initially synchronized and take a step at time 0.

We start with upper bounds. The first algorithm relies on explicit communication to ensure that the needed steps have occurred and does not use any timing information. In the asynchronous model, this algorithm has time complexity $diam(G)(d+1)(s-1)$, where the *diameter*, $diam(G)$, of an undirected graph G is the *maximum* distance between any two nodes. In the semi-synchronous model, this algorithm can be improved to take advantage of the initial synchronization and achieve a time complexity of $1 + diam(G)(d+1)(s-2)$. The second algorithm does not use any communication and relies only on timing information; it works only in the semi-synchronous model. The time complexity of this algorithm is $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$. These algorithms can be combined to yield a semi-synchronous algorithm for the s-session problem whose time complexity is $1 + \min\{\lfloor \frac{1}{c} \rfloor + 1, diam(G)(d+1)\}(s-2)$.

We then present lower bounds. For the asynchronous model, we prove an almost matching lower bound of $diam(G)d(s-1)$ on the time complexity of any algorithm for the s-session

[1] The synchronous model is a special case of the semi-synchronous model where $c = 1$.

problem. For the semi-synchronous model, we prove two lower bounds. We first show a simple lower bound of $\lfloor \frac{1}{c}(s-2) \rfloor$ for the case where no communication is used. We then present a lower bound of $1 + \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}(s-2)$ for the time complexity of any semi-synchronous algorithm for the s-session problem; the proof relies on the assumption that $d \geq \frac{d}{\min\{\lfloor 1/2c \rfloor, diam(G)d\}} + 2$.

These bounds extend in a straightforward way to the case where delays on the communication links of $G = (V, E)$ are not uniform. That is, for every communication link $(i, j) \in E$, the delivery time for a message sent over (i, j) is in the interval $[0, d(i, j)]$ for some fixed $d(i, j)$, $0 \leq d(i, j) < \infty$.

For appropriate values of the various parameters, our results imply a time separation between semi-synchronous (in particular, synchronous) and asynchronous networks. The lower bound for the semi-synchronous model shows the inherent limitations on using timing information. In addition, it can also be used to derive a lower bound of $1 + diam(G)d(s-2)$ for a model in which processes' step time is in the range $(0, 1]$ (rather than in $[0, 1]$, as in the asynchronous model). This is equivalent to requiring that two steps by the same process may not occur at the same time.[2] Fix some $c' > 0$ such that $\lfloor \frac{1}{2c'} \rfloor \geq diam(G)d$, and use the proof of the lower bound for the model where processes' step time is in the range $[c', 1]$; since $[c', 1] \subset (0, 1]$, the claim follows. This implies the first time separation between this model and the synchronous model. (The proof in [1] relies heavily on the ability to schedule many steps by the same process at the same time.) Our first algorithm for the semi-synchronous model only requires that two steps by the same process do not occur at the same time. This implies that the almost matching upper bound of $1 + diam(G)(d+1)(s-2)$ holds also for the asynchronous model where processes' step time is in the range $(0, 1]$.

The lower bounds presented in this paper use the same general approach as in [1]. However, since we assume processes communicate by sending messages while [1] assumes processes communicate via shared memory, the precise details differ substantially. The lower bound proof in [1] uses fan-in arguments, while our lower bounds are based on information propagation arguments using long delays of messages, combined with appropriate selection of processes and careful timing arguments. Our asynchronous lower bound is based and improves on a result of Lynch ([6]) showing a lower bound of $rad(G)d(s-1)$.[3]

Awerbuch ([4]) introduced the concept of a *synchronizer* as a way to translate algorithms

[2] We remark that this is the more common way of measuring time in an asynchronous system (e.g., [11]).

[3] The *radius*, $rad(G)$ of G is the *minimum*, over all nodes in V of the maximum distance from that node to any other node in V. For any undirected graph G, $rad(G) \leq diam(G) \leq 2rad(G)$.

designed for synchronous networks to asynchronous networks. Although the results of [4] may suggest that *any* synchronous network algorithm can be translated into an asynchronous algorithm with constant time overhead, our results imply that this is *not* the case: for some values of the parameters, any translation of a semi-synchronous (in particular, synchronous) algorithm for the s-session problem to an asynchronous algorithm *must* incur a non-constant time overhead.

The rest of this paper is organized as follows. Section 2 presents the system model, defines the session problem and introduces some notation. Section 3 includes our upper bounds for both models, while Section 4 includes our lower bounds for both models. In Section 5, we consider the non-uniform case and state the corresponding upper and lower bounds. We conclude, in Section 6, with a discussion of the results and some open problems.

2 Definitions

In this section, we present the definitions for the underlying formal model,[4] define what it means for an algorithm to solve the *s-session problem* and introduce some notation.

2.1 The System Model

A *system* consists of n processes $p_1, \ldots, p_n$. Processes are located at the nodes of a graph $G = (V, E)$, where $V = [n]$. For simplicity, we identify processes with the nodes they are located at and we refer to nodes and processes interchangeably. Each process p_i is modeled as a (possibly infinite) state machine with state set Q_i. The state set Q_i contains a distinguished *initial state* $q_{0,i}$. The state set Q_i also includes a subset I_i of *idle* states; we assume $q_{0,i} \notin I_i$. We assume that any state of p_i includes a special component, $buffer_i$, which is p_i's message buffer. A *configuration* is a vector $C = (q_1, \ldots, q_n)$ where q_i is the local state of p_i; denote $state_i(C) = q_i$. The *initial configuration* is the vector $(q_{0,1}, \ldots, q_{0,n})$. Processes communicate by sending *messages*, taken from some alphabet $\mathcal{M}$, to each other. A *send action* $send(j, m)$ represents the sending of message m to a neighboring process p_j. Let $\mathcal{S}_i$ denote the set of all send actions $send(j, m)$ for all $m \in \mathcal{M}$ and all $j \in [n]$, such that $(i, j) \in E$; that is, $\mathcal{S}_i$ includes all the send actions possible for p_i.

[4] These definitions are similar in style to those in [2] and could be expressed in terms of the general *timed automaton model* described in [10, 3, 7].

We model computations of the system as sequences of *atomic events*, or simply *events*. Each event is either a *computation event*, representing a computation step of a single process, or a *delivery event*, representing the delivery of a message to a process. Each computation event is specified by $comp(i, S)$ for some $i \in [n]$. In the computation step associated with event $comp(i, S)$, the process p_i, based on its local state, changes its local state and performs some set S of send actions, where S is a finite subset of $\mathcal{S}_i$. Each delivery event has the form $del(i, m)$ for some $m \in \mathcal{M}$. In a delivery step associated with the event $del(i, m)$, the message m is added to $buffer_i$, p_i's message buffer.[5]

Each process p_i follows a deterministic local algorithm $\mathcal{A}_i$ that determines p_i's local computation, i.e., the messages to be sent and the state transition to be performed. More specifically, for each $q \in Q_i$, $\mathcal{A}_i(q) = (q', S)$ where q' is a state and S is a set of send actions. We assume that once a process enters an idle state, it will remain in an idle state, i.e., if q is an idle state, then q' is an idle state. An *algorithm* (or a *protocol*) is a sequence $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ of local algorithms.

An *execution* is an infinite sequence of alternating configurations and events

$$\alpha = C_0, \pi_1, C_1, \ldots, \pi_j, C_j, \ldots ,$$

satisfying the following conditions:

1. C_0 is the initial configuration;
2. If $\pi_j = del(i, m)$, then $state_i(C_j)$ is obtained by adding m to $buffer_i$.
3. If $\pi_j = comp(i, S)$, then $state_i(C_j)$ and S are obtained by applying $\mathcal{A}_i$ to $state_i(C_{j-1})$;
4. If π_j involves process i, then $state_k(C_{j-1}) = state_k(C_j)$ for every $k \neq i$;
5. For each $m \in \mathcal{M}$ and each process p_i, let $S(i, m)$ be the set of j such that π_j contains a $send(i, m)$ and let $D(i, m)$ be the set of j such that π_j is a delivery event $del(i, m)$. Then there exists a one-to-one onto mapping $\sigma_{i,m}$ from $S(i, m)$ to $D(i, m)$ such that $\sigma_{i,m}(j) > j$ for all $j \in S(i, m)$.

That is, in an execution the changes in processes' states are according to the transition function, only a process which takes a step or to which a message is delivered changes its state, and

[5] The system model can be extended to allow arbitrary state change upon message delivery without changing the results; for clarity of presentation, we chose not to do so.

each sending of a message is matched to a later message delivery and each message delivery to an earlier send. We adopt the convention that finite prefixes of an execution end with a configuration, and denote the last configuration in a finite execution prefix α by $last(\alpha)$. We say that $\pi_j = comp(i, S)$ is a *non-idle step* of the execution if $state_i(C_{j-1}) \notin I_i$, i.e., it is taken from a non-idle state.

A *timed event* is a pair (t, π), where t, the "time", is a nonnegative real number, and π is an event. A *timed sequence* is an infinite sequence of alternating configurations and timed events

$$\alpha = C_0, (t_1, \pi_1), C_1, \ldots, (t_j, \pi_j), C_j, \ldots ,$$

where the times are nondecreasing and unbounded.

Timed executions in this model are defined as follows. Fix real numbers c and d, where $0 \leq c \leq 1$ and $0 \leq d < \infty$. Letting α be a timed sequence as above, we say that α is a *timed execution* of $\mathcal{A}$ provided that the following all hold:

1. $C_0, \pi_1, C_1, \ldots, \pi_j, C_j, \ldots$ is an execution of $\mathcal{A}$;
2. (Synchronous start) There are computation events for all processes with time 0.
3. (Upper bound on step time) If the jth timed event is $(t_j, comp(i_j, S))$, then there exists a $k > j$ with $t_k \leq t_j + 1$ such that the kth timed event is $(t_k, comp(i_j, S'))$;
4. (Lower bound on step time) If the jth timed event is $(t_j, comp(i_j, S))$, then there does not exist a $k > j$ with $t_k < t_j + c$ such that the kth timed event is $(t_k, comp(i_j, S'))$;
5. (Upper bound on message delivery time) If message m is sent to p_i at the jth timed event, then there exists $k > j$ such that the kth timed event is the matching delivery $(t_k, del(i, m))$ (i.e., $\sigma_{i,m}(j) = k$) and $t_k \leq t_j + d$.

We say that α is an *execution fragment* of $\mathcal{A}$ if there is an execution α' of $\mathcal{A}$ of the form $\alpha' = \beta\alpha\beta'$. This definition is extended to apply to timed executions in the obvious way. For a finite execution fragment $\alpha = C_0, (t_1, \pi_1), C_1, \ldots, (t_k, \pi_k), C_k$, we define $t_{start}(\alpha) = t_1$ and $t_{end}(\alpha) = t_k$.

The *asynchronous* model is defined by taking $c = 0$, while the *semi-synchronous* model is defined by taking $0 < c \leq 1$; the *synchronous model* is a special case of the latter. Note that the asynchronous model, as defined above, allows two computation steps of the same process to occur at the same time (Condition 4 is vacuous when $c = 0$). We remark that our proofs, as

well as the proof in [1], use this property. If we want to define the more common asynchronous model, where a process can have at most one computation step at each time, we have to replace Condition 4 above with:

> (Lower bound on step time) If the jth timed event is $(t_j, comp(i_j, S))$, then there does not exist a $k > j$ with $t_k = t_j$ such that the kth timed event is $(t_k, comp(i_j, S'))$;

In both models, we say that a process p_i *enters an idle state by time* t' (in a timed execution α) if there exists a timed event (t_{j-1}, π_{j-1}) in α such that $t_{j-1} \leq t'$, $\pi_{j-1} = comp(i, S)$, and $state_i(C_j) \in I_i$. We say that a process p_i *receives the message* m *by time* t' (in a timed execution α) if, by time t', p_i has a computation event that is preceded in α by a delivery event $del(i, m)$. For the rest of the paper let D denote $d + 1$. Note that if m is sent to p_j at time t, then p_j receives m by time $t + D$.

2.2 The Session Problem

An execution fragment $C_1, \pi_1, C_2 \ldots, \pi_m, C_m$ is a *session* if for each i, $i \in [n]$, there exists at least one event $\pi_j = comp(i, S)$, for some $j \in [m]$, which is a non-idle step of the underlying execution. Intuitively, a session is an execution fragment in which each process takes at least one non-idle step. An execution α *contains* s *sessions* if it can be partitioned into at least s execution fragments with pairwise disjoint sets of events such that each of them is a session. These definitions are extended to apply to timed executions in the obvious way.

An algorithm *solves the* s*-session problem within time* t (on G) if each of its timed executions α satisfies the following: α contains s sessions and all processes enter an idle state no later than time t in α.

2.3 Notation

Consider an undirected graph $G = (V, E)$. For any $i, j \in V$, let $dist(i, j)$ be the *distance* of i and j in G, i.e., the number of edges in the shortest path in G ¿from i to j. The *diameter* of G, $diam(G)$, is the maximum distance between any two nodes in V, i.e., $diam(G) = \max_{i,j \in V} dist(i, j)$.

A node $i \in V$ is a *peripheral* node of G if $\max_{j \in V} dist(i, j) = diam(G)$; informally, a peripheral node "realizes" the diameter of G. A node $j \in V$ is *antipodal* to a node $i \in V$ if $dist(i, j) = \max_{k \in V} dist(i, k)$; informally, j is a "farthest neighbor" of i in G. Note that if j is antipodal to a peripheral node then j is peripheral.

3 Upper Bounds

3.1 The Asynchronous Model

We start with a simple asynchronous algorithm in which processes communicate in order to learn about completion of a session before advancing to the next session. Each process maintains as part of its state a variable that gives its current session number; upon hearing that every other process has reached its current session, it increments its session number by one and notifies all other processes. Notification is done by sending messages along a shortest-path tree rooted at it. The process enters an idle state when its session number is set to s. We prove:

Theorem 3.1 *Let G be any graph. There exists an asynchronous algorithm, $\mathcal{A}^{as}$, that solves the s-session problem on G within time $diam(G)D(s-1)$.*

Proof: We describe an asynchronous algorithm, $\mathcal{A}^{as}$, that solves the s-session problem on G within time $diam(G)D(s-1)$; that is, in any execution of $\mathcal{A}^{as}$ there are at least s sessions and all processes enter an idle state no later than $diam(G)D(s-1)$. The algorithm is described here informally; this description can be easily translated into a state transition function.

For each $i \in [n]$, the state of p_i consists of the following components: *buffer* — a buffer, an unordered set of elements of $\mathcal{M}$, initially $\emptyset$; *session* — a nonnegative integer, initially 1. The message alphabet, $\mathcal{M}$, consists of the pairs (i,k), where $i \in [n]$ and $1 \leq k \leq s-1$. The initial state of p_i is non-idle.

The algorithm is as follows. Upon taking its first computation step, p_i broadcasts $(i,1)$. If for all $j \in [n]$, $(j, session_i) \in buffer_i$, p_i increments $session_i$ by 1. If $session_i = s$, p_i enters an idle state and remains in this state forever. Otherwise, p_i broadcasts $(i, session_i)$.

We assume that messages from a process are flooded on a *shortest path tree* rooted at this process. That is, $\mathcal{A}^{as}$ uses a routing algorithm by which, for any nodes $u, v \in G$, a message from u to v is routed through *exactly* $dist(u,v)$ communication links in G. The details of how this is done are not discussed here; the reader is referred to, e.g., [13].

If $session_i = k$, we say that p_i is in its kth session. The message (i,k) can be interpreted as "process i executed a step in the kth session".

We start by showing that in any execution of $\mathcal{A}^{as}$ there are at least s sessions. Fix an arbitrary timed execution α of $\mathcal{A}^{as}$. Clearly, each process p_i receives $(j,1)$ for all $j \in [n]$ and

sets $session_i$ to 2. By induction, it is simple to show that for any k, $1 \leq k \leq s$, p_i sets $session_i$ to k in α. For any k, $1 \leq k \leq s$, define α_k to be the longest prefix of α that does not include a configuration in which for some $i \in [n]$, $session_i \geq k$, i.e., no process has passed its kth session. Note that $\alpha_1 = \lambda$, the *empty sequence*, and that for each k, $1 \leq k \leq s-1$, α_k is a prefix of α_{k+1}. For each k, $1 \leq k \leq s-1$, let β_k be such that $\alpha_{k+1} = \alpha_k \beta_k$; let β_s be such that $\alpha = \alpha_s \beta_s$.

Lemma 3.2 *For each k, $1 \leq k \leq s-1$, there is a session in β_k.*

Proof: Let p_i be a process which sets $session_i$ to $k+1$. By definition, this event is not in β_k. By the algorithm, this implies that for each j, $j \in [n]$ and $j \neq i$, p_i has received a (j,k) message. However, by definition, no process p_j has $session_j \geq k$ in α_k. Thus, by the algorithm, no process p_j sends a (j,k) message in α_k. Hence, there is a step by every process, and, therefore, a session in β_k. ■

In addition, there is a session in β_s, since, for every $i \in [n]$, a computation step is included in β_s at which p_i sets $session_i$ to s. (Note that, by the definition of α_s, such a step cannot be included in α_s.) This implies that there are at least s sessions in α. Since α was chosen arbitrarily, this implies the correctness of $\mathcal{A}^{as}$. We now analyze the time complexity of $\mathcal{A}^{as}$.

Informally, the next definition captures the latest time at which the kth session can be completed. For each k, $1 \leq k \leq s$, define

$$T_k = \max_{i \in V} \{t : p_i \text{ sets } session_i \text{ to } k \text{ at time } t \text{ in } \alpha\} .$$

By the algorithm, $T_1 = 0$. We have:

Lemma 3.3 *For each k, $1 \leq k < s$, $T_{k+1} \leq T_k + diam(G)D$.*

Proof: Fix some process p_i, and let t be the time at which p_i broadcasts (i,k); note that by definition $t \leq T_k$. Clearly, for every process p_j, the delivery event $del(j,(i,k))$, delivering the message (i,k) to p_j, will occur at a time $\leq t + (diam(G)-1)D + d$. Thus, by time $t + diam(G)D$ every process has a computation step in which (i,k) is in the buffer. Thus, by time $T_k + diam(G)D$ every process has a computation step in which (i,k) is in the buffer, for any $i \in [n]$. By the algorithm, at this step the process sets its *session* variable to $k+1$. The claim follows. ■

Since $T_1 = 0$, it follows that $T_s \leq diam(G)D(s-1)$. Hence, every process enters an idle state after setting *session* to s, no later than time $diam(G)D(s-1)$. Thus, $\mathcal{A}^{as}$ solves the s-session problem on G within time $diam(G)D(s-1)$. ■

3.2 The Semi-Synchronous Model

In the semi-synchronous model we can slightly improve $\mathcal{A}^{as}$ by taking advantage of the available initial synchronization; specifically, each process operates exactly as in $\mathcal{A}^{as}$, except that it does not wait to hear that every other process has completed its first session, but passes directly to the second one upon taking its second step. We prove:

Theorem 3.4 *Let G be any graph. There exists a semi-synchronous algorithm, $\mathcal{A}_1^{ss}$, that solves the s-session problem on G within time $1 + diam(G)D(s-2)$.*

Proof: We describe a semi-synchronous algorithm, $\mathcal{A}_1^{ss}$ which is very similar to $\mathcal{A}^{as}$ and solves the s-session problem on G within time $1+diam(G)D(s-2)$. For each $i \in [n]$, the state of p_i consists of the following components: *buffer* — a buffer, an unordered set of elements of $\mathcal{M}$, initially $\emptyset$; *session* — a nonnegative integer, initially 1. The message alphabet, $\mathcal{M}$, consists of the pairs (i,k) where $i \in [n]$ and $2 \leq k \leq s-1$. The initial state of p_i is non-idle.

Upon taking its second computation step, p_i increments $session_i$ to 2 and broadcasts $(i,2)$. If for all $j \in [n]$, $(j, session_i) \in buffer_i$, p_i increments $session_i$ by 1. If $session_i = s$, p_i enters an idle state and remains in this state forever. Otherwise, p_i broadcasts $(i, session_i)$. As in $\mathcal{A}^{as}$, we assume that messages from a process are flooded on a shortest path tree rooted at this process. We say that p_i is in its kth session if $session_i = k$ and we interpret the message (i,k) as "process i executed a step in the kth session".

We start by showing that in any execution of $\mathcal{A}_1^{ss}$ there are at least s sessions. Fix an arbitrary execution α of $\mathcal{A}_1^{ss}$. For each k, $1 \leq k \leq s$, define α_k to be the longest prefix of α that does not include a configuration in which, for some $i \in [n]$, $session_i \geq k$, i.e., no process has passed its kth session. Note that $\alpha_1 = \lambda$, and that for each k, $1 \leq k \leq s-1$, α_k is a prefix of α_{k+1}. For each k, $1 \leq k \leq s-1$, let β_k be such that $\alpha_{k+1} = \alpha_k \beta_k$; let β_s be such that $\alpha = \alpha_s \beta_s$.

Lemma 3.5 *There is a session in β_1.*

Proof: Note that $\beta_1 = \alpha_2$, since $\alpha_1 = \lambda$. For every process p_i, the steps of p_i that are included in α_2 are exactly those that occur at time 0. Since every process has a step at time 0, there is a session in $\alpha_2 = \beta_1$. ∎

As in Lemma 3.2, we can prove:

Lemma 3.6 *For each k, $1 \leq k \leq s-1$, there is a session in β_k.*

In addition, there is a session in β_s. This implies that there are at least s sessions in α. Since α was chosen arbitrarily, this implies the correctness of $\mathcal{A}_1^{ss}$.

We now analyze the time complexity of $\mathcal{A}_1^{ss}$. For each k, $2 \leq k \leq s$, we define:

$$T_k = \max_{i \in V} \{t : p_i \text{ sets } session_i \text{ to } k \text{ at time } t \text{ in } \alpha\} .$$

Note that $T_2 \leq 1$. In addition, as in Lemma 3.3, we have:

Lemma 3.7 *For each k, $2 \leq k < s$, $T_{k+1} \leq T_k + diam(G)D$.*

Since $T_2 \leq 1$, it follows that $T_s \leq 1 + diam(G)D(s-2)$. Every process enters an idle state after setting *session* to s, no later than time $1 + diam(G)D(s-2)$. Thus, $\mathcal{A}_1^{ss}$ solves the s-session problem on G within time $1 + diam(G)D(s-2)$. ■

We next show that the timing information available in the semi-synchronous model can be exploited to obtain a bound which is sometimes better than the previous bound. This algorithm uses no communication; intuitively, this means that no process state transition can result in a send action. Formally, an algorithm $\mathcal{A}$ *uses no communication* if for every i, $i \in [n]$, for every $q \in Q_i$, $\mathcal{A}_i(q) = (q', \emptyset)$ for some $q' \in Q_i$. We prove:

Theorem 3.8 *Let G be any graph. There exists a semi-synchronous algorithm, $\mathcal{A}_2^{ss}$, which solves the s-session problem on G within time $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$. Furthermore, $\mathcal{A}_2^{ss}$ uses no communication.*

Proof: We describe a semi-synchronous algorithm, $\mathcal{A}_2^{ss}$, which solves the s-session problem on G within time $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$. For each $i \in [n]$, the state of p_i consists of a *counter*, an integer, initially -1. The initial state of p_i is non-idle. At each computation event, p_i increments $counter_i$; p_i enters an idle state when $counter_i$ is equal to $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$.

We start by showing that in any execution of $\mathcal{A}_2^{ss}$ there are at least s sessions. Consider an arbitrary execution α of $\mathcal{A}_2^{ss}$. We partition α into execution fragments, $\alpha = \alpha_0 \alpha_1 \ldots \alpha_{s-1}$, such that: *(i)* α_0 consists only of the computation steps at time 0, and *(ii)* for each k, $1 \leq k \leq s-2$, $\alpha_0 \ldots \alpha_k$ is the shortest prefix of α that includes a configuration in which, for some $i \in [n]$, $counter_i = k(\lfloor \frac{1}{c} \rfloor + 1)$. We have:

Lemma 3.9 *For each k, $1 \leq k \leq s-2$, there is a session in α_k.*

Proof: Let p_i be the first process to set $counter_i$ to $k(\lfloor \frac{1}{c} \rfloor + 1)$ in α. By the definition of α_k, the steps at which $counter_i$ is equal to $(k-1)(\lfloor \frac{1}{c} \rfloor + 1) + j$, for $1 \leq j \leq \lfloor \frac{1}{c} \rfloor + 1$, are included in α_k. Thus, there are at least $\lfloor \frac{1}{c} \rfloor + 1$ steps by p_i in α_k. These steps take time at least $c(\lfloor \frac{1}{c} \rfloor + 1) > c\frac{1}{c} = 1$; thus, there exists a computation step by every process and, therefore, a session in α_k. ■

In addition, there is a session in α_0 since every process takes a step at time 0.

There is also a session in α_{s-1}, since, by the definition of α_{s-1}, each process sets its counter to $1 + (s-2)(\lfloor \frac{1}{c} \rfloor + 1)$ at its last non-idle step. Together with Lemma 3.9, this implies that there are at least s sessions in any timed execution of $\mathcal{A}_2^{ss}$.

Each process will enter an idle state no later than time $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$, since for any process the time between successive computation steps is at most 1. Thus, $\mathcal{A}_2^{ss}$ solves the s-session problem within time $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$. ■

When d and $diam(G)$ are known, it is possible to calculate in advance which of the algorithms of Theorems 3.4 and 3.8 is faster, and run it. Furthermore, even if d and $diam(G)$ are not known, it is possible to run the algorithms of Theorems 3.4 and 3.8 "side by side," halting when the first of them does. In both cases we get:

Theorem 3.10 *Let G be any graph. There exists a semi-synchronous algorithm, $\mathcal{A}^{ss}$, which solves the s-session problem on G within time $1 + \min\{\lfloor \frac{1}{c} \rfloor + 1, diam(G)D\}(s-2)$.*

4 Lower Bounds

In all our lower bounds we use an infinite timed execution in which processes take steps in round-robin order, starting with p_1, with step time close to 1, and all messages are delivered after exactly d delay. It is called a *slow, synchronous* timed execution.

4.1 The Asynchronous Model

We start by showing that for the asynchronous model, the algorithm presented in Theorem 3.1 is optimal. The proof of the following theorem is based on delaying information propagation

and then perturbing an execution to obtain an execution of the algorithm which does not include s sessions.

Theorem 4.1 *Let G be any graph. There does not exist an asynchronous algorithm which solves the s-session problem on G within time strictly less than $diam(G)d(s-1)$.*

Proof: Assume, by way of contradiction, that there exists an asynchronous algorithm, $\mathcal{A}$, which solves the s-session problem on G within time strictly less than $diam(G)d(s-1)$. We construct a timed execution of $\mathcal{A}$ which does not include s sessions.

The following is an informal outline of the proof. We start with a slow, synchronous timed execution of $\mathcal{A}$ and partition it into $s-1$ execution fragments each of which is completed within time $< diam(G)d$. Since communication is slow, there is no communication between any pair of antipodal nodes during a fragment. By "retiming", we will perturb each fragment to get a new execution fragment in which there is a "fast" peripheral node which takes all of its steps before a "slow" antipodal node takes any of its steps. Our construction will have the "slow" node of each execution fragment be identical to the "fast" node of the next execution fragment. In each execution fragment, a session can be completed as soon as the "slow" peripheral node takes its first computation step; since the "fast" peripheral node does not take any more computation steps, no more sessions can be completed in this execution fragment. This will guarantee that at most one session is contained in each execution fragment; thus, the total number of sessions in the "retimed" execution is at most $s-1$, contradicting the correctness of $\mathcal{A}$.

We now present the details of the formal proof.

Pick some ϵ such that $0 < \epsilon \leq (diam(G)d(s-1)+1)^{-1}$. Consider a slow, synchronous timed execution $\gamma = \alpha\alpha'$ of $\mathcal{A}$, with step time $1-\epsilon$, where α is the shortest prefix of γ such that all processes are in an idle state in $last(\alpha)$ and α' is the remaining part of γ. We perturb α and α' to obtain timed sequences β and β', respectively, such that $\beta\beta'$ is a timed execution that does not include s sessions."

We first show how to modify α to obtain β. By assumption, $t_{end}(\alpha) < diam(G)d(s-1)$. Write $\alpha = \alpha_0\alpha_1\ldots\alpha_{s-1}$, where $\alpha_0 = \lambda$ and for each k, $1 \leq k \leq s-1$, $t_{end}(\alpha_k) - t_{end}(\alpha_{k-1}) \leq diam(G)d(1-\epsilon) < diam(G)d$. (We adopt the convention that $t_{end}(\alpha_0) = 0$.) For some sequence $i_0, \ldots, i_{s-1}$ of peripheral nodes, we construct from each execution fragment α_k an execution fragment $\beta_k = \rho_k\sigma_k$, such that:

(1) ρ_k contains no computation step of $p_{i_{k-1}}$, and

(2) σ_k contains no computation step of p_{i_k}.

In this construction, i_k is the "fast" node which takes all of its steps in the execution fragment ρ_k, before the "slow" node i_{k-1} takes any of its steps. (All the steps of i_k are in σ_k.) Our construction uses peripheral nodes since they maximize the time to transfer information to other nodes, which is, roughly, $diam(G)d$. In particular, i_{k-1} will be antipodal to i_k.

We now show, for each k, $1 \leq k \leq s-1$, how to construct β_k, by induction on k. For the base case, let i_0 be an arbitrary peripheral node of G, and take β_0 to be the empty execution.

Assume we have picked $i_0, \ldots, i_{k-1}$ and constructed $\beta_0, \ldots, \beta_{k-1}$. Let i_k be some node that is antipodal to i_{k-1}, i.e., $dist(i_{k-1}, i_k) = diam(G)$; note that i_k is also peripheral. We now show how to construct β_k.

For any node u, ρ_k includes all events at u that occur at time $< t_{end}(\alpha_{k-1}) + dist(u, i_{k-1})d$ in α_k; σ_k includes all events at u that occur at time $\geq t_{end}(\alpha_{k-1}) + dist(u, i_{k-1})d$ in α_k. Events at each process occur in the same order as in α_k and all occur at time 0, in both ρ_k and σ_k. In addition, ordering of events across different processes that occur at the same time in α_k is preserved within each of ρ_k and σ_k.

Since

$$t_{end}(\alpha_{k-1}) + dist(i_k, i_{k-1})d = t_{end}(\alpha_{k-1}) + diam(G)d > t_{end}(\alpha_k) \ ,$$

and all events at i_k occur at time $\leq t_{end}(\alpha_k)$ in α_k, this implies that all events at i_k will appear in ρ_k. On the other hand, since

$$t_{end}(\alpha_{k-1}) + dist(i_{k-1}, i_{k-1})d = t_{end}(\alpha_{k-1}) \ ,$$

and all events at i_{k-1} in α_k occur at time $\geq t_{end}(\alpha_{k-1})$, all events at i_{k-1} in α_k will appear in σ_k. Thus, $\beta_k = \rho_k \sigma_k$ has properties (1) and (2) above.

Let $\beta = \beta_0 \beta_1 \ldots \beta_{s-1}$.

By construction, events at each process p_i, $i \in [n]$, occur in the same order in β as in α. Hence, p_i undergoes the same state changes in β as in α and therefore, $state_i(last(\beta)) = state_i(last(\alpha))$.

We now modify α' to obtain β'. The first computation step of any process in β' will occur at time 1 and all later computation steps of it are 1 time unit apart. Any message delivery event at a process will occur at time d after the corresponding message sending event.

We next establish that $\beta\beta'$ is a timed execution of $\mathcal{A}$. We start by showing:

Lemma 4.2 *Each receive event is after the corresponding send event in $\beta\beta'$.*

Proof: Consider the message send event π_1 at node u_1 which occurs at time t_1 in $\alpha\alpha'$ and let π_2 be the corresponding message receive event at node u_2 which occurs at time t_2 in $\alpha\alpha'$. Note that u_1 and u_2 are neighboring processes, i.e., $dist(u_1, u_2) = 1$. Hence, $dist(u_1, i_{k-1}) + 1 \geq dist(u_2, i_{k-1})$. The only non-trivial case is when π_1 and π_2 occur in the same α_k, for some k, $1 \leq k \leq s-1$. We show that the ordering of π_1 and π_2 is the same in β_k as in α_k.

The only case of interest is when π_1 occurs in σ_k, while π_2 occurs in ρ_k. In this case, $t_1 \geq t_{end}(\alpha_{k-1}) + dist(u_1, i_{k-1})d$, while $t_2 < t_{end}(\alpha_{k-1}) + dist(u_2, i_{k-1})d$. Then,

$$\begin{aligned} t_2 &= t_1 + d \\ &\geq t_{end}(\alpha_{k-1}) + dist(u_1, i_{k-1})d + d \qquad \text{(since } \pi_1 \text{ occurs in } \sigma_k\text{)} \\ &= t_{end}(\alpha_{k-1}) + (dist(u_1, i_{k-1}) + 1)d \\ &\geq t_{end}(\alpha_{k-1}) + dist(u_2, i_{k-1})d \ , \end{aligned}$$

a contradiction. ■

All events in β occur at time 0 and computation steps in β' occur with step time 1. Since there are no lower bounds on either process step time or message delivery time in the asynchronous model, we have:

Lemma 4.3 *Lower and upper bounds on step time are preserved in $\beta\beta'$.*

Lemma 4.4 *Lower and upper bounds on message delay time are preserved in $\beta\beta'$.*

To derive a contradiction, we prove:

Lemma 4.5 *There are at most $s-1$ sessions in β.*

Proof: We show, by induction on k, that $\beta_0 \ldots \beta_{k-1}\rho_k$ does not contain k sessions, for $1 \leq k \leq s-1$.

For the base case, note that, by construction, $\beta_0 = \lambda$ and ρ_1 does not include a computation step of p_{i_0}. Thus, $\beta_0\rho_1$ cannot contain one session.

For the induction step, assume that the claim holds for $k-1$, i.e., $\beta_0 \ldots \beta_{k-2}\rho_{k-1}$ does not contain $k-1$ sessions, for $1 \leq k \leq s$. Hence, the kth session does not start within

$\beta_0 \ldots \beta_{k-2}\rho_{k-1}$. Since neither σ_{k-1} nor ρ_k contains a computation step of $p_{i_{k-1}}$, $\sigma_{k-1}\rho_k$ does not contain a session. Thus, $\beta_0 \ldots \beta_{k-1}\rho_k$ does not contain k sessions.

To complete the proof, note that σ_{s-1} does not contain a session since, by construction, it does not contain a computation step of $p_{i_{s-1}}$. ■

Thus, there are strictly less than s sessions in β; however, in β' no process takes a non-idle step, so there cannot be an additional session in β'. A contradiction. ■

We remark that the general outline of this lower bound proof follows [1, 6]. However, while the proofs in [1, 6] use causality arguments to reorder the events in the execution, our proof presents an explicit reordering and retiming of the events. We do so because this provides a basis for the retiming arguments used in the proof of the lower bound for the semi-synchronous model. Our improvement over [6] is achieved by carefully choosing only peripheral nodes in the construction of β.

4.2 The Semi-Synchronous Model

In Section 3.2, we have seen two algorithms that solve the s-session problem in the semi-synchronous model. The first of them, $\mathcal{A}_1^{ss}$, solves the s-session problem on G within time $1 + diam(G)D(s-2)$. Designed for the asynchronous model, $\mathcal{A}_1^{ss}$ has the interesting property that processes do not use any timing information. Loosely speaking, the lower bound proved in Theorem 4.1 says that if processes have no timing information, then $diam(G)d(s-1)$ is a lower bound for any asynchronous algorithm which solves the s-session problem on G.

Recall, also, that $\mathcal{A}_2^{ss}$ uses no communication, but relies only on timing information to achieve an upper bound of $1 + (\lfloor \frac{1}{c} \rfloor + 1)(s-2)$. We first show that this upper bound is close to optimal in the absence of communication:

Theorem 4.6 *Let G be any graph. There does not exist a semi-synchronous algorithm which solves the s-session problem on G within time strictly less than $\lfloor \frac{1}{c}(s-2) \rfloor$ and uses no communication.*

Proof: Assume, by way of contradiction, that there exists a semi-synchronous algorithm, $\mathcal{A}$, which solves the s-session problem on G within time strictly less than $\lfloor \frac{1}{c}(s-2) \rfloor$, and uses no communication. We construct a timed execution of $\mathcal{A}$ which does not include s sessions.

Let α be a slow, synchronous timed execution of $\mathcal{A}$ with step time 1. Assume, without loss of generality, that p_n is the last process to enter an idle state in α. Let $\alpha = \alpha_0\alpha'$, where α_0 includes events at time 0, while α' is the remaining part of α. Let m be the number of non-idle steps taken by any process in α'. It must be that $m < \lfloor \frac{1}{c}(s-2) \rfloor$, since $\mathcal{A}$ solves the s-session problem within time $< \lfloor \frac{1}{c}(s-2) \rfloor$, and α is slow.

Now modify α' to get a new timed execution fragment β in which all processes except p_n, operate with fastest step time, i.e., c. This can be done since there are no receive events in α.

In β, each process but p_n enters an idle state at some time $\leq cm < c\lfloor \frac{1}{c}(s-2) \rfloor \leq c\frac{1}{c}(s-2) = s-2$. Thus, in β p_n performs strictly less than $s-2$ steps when all other processes are not in an idle state. Therefore, at most $s-2$ sessions can be completed in β; hence, at most $s-1$ sessions can be completed in $\alpha_0\beta$. A contradiction. ■

We show next that communication and timing information *cannot* be combined to get an upper bound that is significantly better than the upper bound achieved in Theorem 3.10. We prove:

Theorem 4.7 *Let G be any graph and assume that $d \geq \frac{d}{\min\{\lfloor 1/2c \rfloor, diam(G)d\}} + 2$. There does not exist a semi-synchronous algorithm which solves the s-session problem on G within time strictly less than $1 + \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}(s-2)$.*

Proof: Assume, by way of contradiction, that there exists a semi-synchronous algorithm, $\mathcal{A}$, which solves the s-session problem on G within time strictly less than $1+\min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}(s-2)$. We construct a timed execution of $\mathcal{A}$ which does not include s sessions.

The general structure of our lower bound proof closely follows that of Theorem 4.1, though there are several complications: First, the early events of the execution, happening at time < 1 and including processes' steps occurring at time 0, are handled separately (unlike the proof of Theorem 4.1). Second, the additional timing requirements placed in the semi-synchronous model require more careful arguing to show the correctness of the construction.

We start with a slow, synchronous timed execution of $\mathcal{A}$ and partition it into an execution fragment containing the events at time 0 and $s-2$ execution fragments each of which is completed within time $< \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}$. Since communication is slow, there is no communication between any pair of antipodal nodes during a fragment. Furthermore, since the execution is slow, a process takes, roughly, at most $\frac{1}{2c}$ steps, so it is possible to have these all steps occur at the same time another process takes only one step. By "retiming", we will

perturb each fragment to get a new execution fragment in which there is a "fast" peripheral node which takes all of its steps before a "slow" antipodal node takes any of its steps. The part of the proof that shows that the "retimed" execution preserves the timing constraints of the semi-synchronous model requires substantially more careful arguments than the corresponding part in the proof of Theorem 4.1. In particular, we need to choose the execution fragments to take time $< \lfloor \frac{1}{2c} \rfloor$, so that it will be possible for a process not to have a computation step during a large part of the execution fragment. Our construction will have the "slow" node of each execution fragment be identical to the "fast" node of the next execution fragment. Arguing as in Theorem 4.1, this will guarantee that at most one session is contained in each execution fragment. Thus, the total number of sessions in the "retimed" execution is at most $s-2$, contradicting the correctness of $\mathcal{A}$.

We now present the details of the formal proof.

Denote $b = \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}$.

If $b \leq 1$, then the lower bound we are trying to prove is $\leq 1 + 1(s-2) = s-1$. Since s steps of each process are necessary if s sessions are to occur and they can occur 1 time unit apart, it follows that $s-1$ is a lower bound. Thus, we assume, without loss of generality, that $b > 1$. It follows that $c < \frac{1}{2}$. Note that, by assumption, $d \geq \frac{d}{b} + 2$, i.e., $d \geq \frac{2b}{b-1}$. Since $b > 1$, it follows that $d > 1$.

Let γ be a slow, synchronous timed execution of $\mathcal{A}$ with step time 1. Assume $\gamma = \beta_0 \alpha \alpha'$, where β_0 contains only events that occur at time < 1, and $\beta_0 \alpha$ is the shortest prefix of γ such that all processes are in an idle state in $last(\beta_0 \alpha)$, and α' is the remaining part of γ. Denote $T = t_{end}(\beta_0 \alpha)$. Since γ is slow and s steps of each process are necessary to guarantee s sessions, $T \geq s-1$. Since $\mathcal{A}$ solves the s-session problem within time strictly less than $1 + b(s-2)$, it follows that $T < 1 + b(s-2)$. Note that, by construction, $t_{start}(\alpha) = 1$. Thus, $t_{end}(\alpha) - t_{start}(\alpha) = T - 1 < b(s-2)$, and hence $\lceil \frac{T-1}{b} \rceil \leq (s-2)$. Denote $s' = \lceil \frac{T-1}{b} \rceil$; it follows that $s' \leq (s-2)$.

We write $\alpha = \alpha_1 \alpha_2 \ldots \alpha_{s'}$, where:

- For each k, $1 \leq k < s'$, α_k contains all events that occur at time t, where $1 + (k-1)b \leq t < 1 + kb$, and
- $\alpha_{s'}$ contains all events occurring at time t, where $1 + (s'-1)b \leq t \leq T$.

That is, we partition α into execution fragments, each taking time $< b$.

Figure 1 should appear here.

Figure 1: The timed execution $\beta_0\alpha\alpha'$

Figure 1 depicts the timed execution $\beta_0\alpha\alpha'$. Each horizontal line represents events happening at one process. We use the symbol $\bullet$ to mark non-idle process steps; similarly, we use the symbol $\times$ to mark idle process steps. Arrows show typical message delay times between pairs of processes; dashed vertical lines mark time points that are used in the proof.

We reorder and retime events in α to obtain a timed sequence β and reorder and retime events in α' to obtain a timed sequence β', such that $\beta_0\beta\beta'$ is a timed execution of $\mathcal{A}$ that does not include s sessions.

We first show how to modify α to obtain an execution fragment $\beta = \beta_1\beta_2\ldots\beta_{s'}$ that includes at most $s' \leq s-2$ sessions. For some sequence $i_0, \ldots, i_{s'}$ of peripheral nodes, we construct from each execution fragment α_k an execution fragment $\beta_k = \rho_k\sigma_k$, such that:

(1) ρ_k contains no computation step of $p_{i_{k-1}}$, and

(2) σ_k contains no computation step of p_{i_k}.

Like in the proof of Theorem 4.1, in this construction, i_{k-1} is the "fast" node which takes all its steps in the execution fragment ρ_k, before the "slow" node i_k takes any of its steps. (All the steps of i_k are in σ_k.) Our construction uses peripheral nodes since they maximize the time to transfer information to other nodes, which is, roughly, $diam(G)d$. In particular, i_{k-1} will be antipodal to i_k.

For each k, $1 \leq k \leq s'$, we show how to construct β_k inductively. For the base case, let i_0 be an arbitrary peripheral node of G.

Assume we have picked $i_0, \ldots, i_{k-1}$ and constructed $\beta_1, \ldots, \beta_{k-1}$. Let i_k be some node that is antipodal to i_{k-1}, i.e., $dist(i_{k-1}, i_k) = diam(G)$; note that i_k is also peripheral. We now show how to construct β_k. For any node u, ρ_k includes all events at u that occur at time $< 1 + (k-1)b + dist(u, i_{k-1})d$ in α_k; σ_k includes all events at u that occur at time $\geq 1 + (k-1)b + dist(u, i_{k-1})d$ in α_k. Events at each process occur in the same order as in α_k and all occur at step time of c, in both ρ_k and σ_k. In addition, ordering of events that occur at the same time, in different processes, in α_k is preserved within each of ρ_k and σ_k. Since

$$1 + (k-1)b + dist(i_k, i_{k-1})d = 1 + (k-1)b + diam(G)d \geq 1 + (k-1)b + b = 1 + kb > t_{end}(\alpha_k)\ ,$$

Figure 2 should appear here.

Figure 2: The timed execution $\beta_0\beta\beta'$

and all events at i_k occur at time $\leq t_{end}(\alpha_k)$ in α_k, this implies that all events at i_k will appear in ρ_k. On the other hand, since

$$1 + (k-1)b + dist(i_{k-1}, i_{k-1})d = 1 + (k-1)b \leq t_{start}(\alpha_k) \ .$$

and all events at i_{k-1} in α_k occur at time $\geq t_{start}(\alpha_k)$, all events at i_{k-1} in α_k will appear in σ_k. Thus, $\beta_k = \rho_k\sigma_k$ has properties (1) and (2) above.

To complete our construction, we assign times to events in β_k. Let $t_{start}(\rho_1) = c$. The first and last computation steps of i_k in ρ_k occur at times $t_{start}(\rho_k) = t_{end}(\sigma_{k-1}) + c$ and $t_{end}(\rho_k)$, respectively. Similarly, the first and last computation steps of i_{k-1} in σ_k occur at times $t_{start}(\sigma_k) = t_{end}(\rho_k)$ and $t_{end}(\sigma_k)$, respectively. Steps are taken c time units apart. For each process p_j, we schedule each computation step π_j of p_j in ρ_k to occur simultaneously with a computation step, π_{i_k}, of i_k which is such that π_j and π_{i_k} occurred at the same time in α_k. Similarly, for each process p_j, we schedule each computation step π_j of p_j in σ_k to occur simultaneously with a computation step, $\pi_{i_{k-1}}$, of i_{k-1} which is such that π_j and $\pi_{i_{k-1}}$ occurred at the same time in α_k. Any message delivery event at a process will occur right after and at exactly the same time as the computation step of the process which immediately precedes the delivery event in α_k. We shall shortly show that assigning times in this manner is consistent with the requirements from a timed execution.

We now modify α' to obtain β'. The first computation step of any process in β' will occur at time c after its last computation step in β and all later computation steps of it will occur at c time units apart in β'. Any message delivery event at a process will occur at time d after the corresponding message send event.

Figure 2 depicts the timed execution $\beta_0\beta\beta'$ using the same conventions as in Figure 1.

We remark that what allowed us to "separate" the steps at i_{k-1} ¿from those at i_k in each of the execution fragments was the assumption that the length of each execution fragment is less than $diam(G)d$ which is the time needed for a communication between an antipodal pair of nodes to be established.

We first show that $\beta_0\beta\beta'$ is a timed execution of $\mathcal{A}$. By Lemma 4.5, since $s' \leq s-2$ and β_0 contains exactly one session, we derive a contradiction.

By the same arguments as in Lemma 4.2, we prove:

Lemma 4.8 *Each receive event is after the corresponding send event in $\beta_0\beta\beta'$.*

Before showing that the timing constraints are preserved in $\beta_0\beta\beta'$, we prove the following simple fact:

Claim 4.9 *(1) For any k, $1 \leq k \leq s'-1$, $t_{end}(\rho_{k+1}) - t_{end}(\rho_k) \leq 1-c$.*
(2) For any k, $1 \leq k \leq s'$, $t_{end}(\beta_k) - t_{end}(\beta_{k-1}) \leq 1-c$.

Proof: We first show that for any k, $1 \leq k \leq s'-1$, $t_{end}(\rho_{k+1}) - t_{end}(\beta_k) \leq \frac{1}{2}$, and for any k, $1 \leq k \leq s'$, $t_{end}(\beta_k) - t_{end}(\rho_k) \leq \frac{1}{2} - c$.

Fix some k, $1 \leq k \leq s'$. By construction,

$$t_{start}(\alpha_k) \geq 1 + (k-1)b,$$

while

$$t_{end}(\alpha_k) < 1 + kb.$$

Thus

$$t_{end}(\alpha_k) - t_{start}(\alpha_k) < 1 + kb - 1 - (k-1)b = b \leq \lfloor \frac{1}{2c} \rfloor.$$

Let m be the maximum number of steps over all processes that some process takes within α_k.

If both $t_{start}(\alpha_k)$ and $t_{end}(\alpha_k)$ are integral, $t_{end}(\alpha_k) - t_{start}(\alpha_k) \leq \lfloor \frac{1}{2c} \rfloor - 1$; then, since α is a slow execution,

$$m \leq t_{end}(\alpha_k) - t_{start}(\alpha_k) + 1 \leq \lfloor \frac{1}{2c} \rfloor \leq \frac{1}{2c}.$$

If at least one of $t_{start}(\alpha_k)$ and $t_{end}(\alpha_k)$ is not integral, then, since α is a slow execution,

$$m \leq \lceil t_{end}(\alpha_k) - t_{start}(\alpha_k) \rceil \leq \lceil \lfloor \frac{1}{2c} \rfloor \rceil = \lfloor \frac{1}{2c} \rfloor \leq \frac{1}{2c}.$$

Thus, in any case, $m \leq \frac{1}{2c}$. Let n_k be the number of computation steps of process $p_{i_{k-1}}$ in α_k and n_{k+1} be the number of computation steps of process $p_{i_{k+1}}$ in α_{k+1}. (Recall that, by construction, in β_k, $p_{i_{k-1}}$ will have all its steps in σ_k, while in β_{k+1}, $p_{i_{k+1}}$ will have all its steps in ρ_{k+1}.) Thus

$$t_{end}(\rho_{k+1}) - t_{end}(\beta_k) = n_{k+1}c \leq mc \leq \frac{1}{2c}c = \frac{1}{2}.$$

Also, since $p_{i_{k-1}}$ takes n_k steps in σ_k with the first occurring at time $t_{start}(\sigma_k) = t_{end}(\rho_k)$ and the last occurring at time $t_{end}(\sigma_k) = t_{end}(\beta_k)$, we have

$$t_{end}(\beta_k) - t_{end}(\rho_k) = (n_k - 1)c \leq (m-1)c \leq (\frac{1}{2c} - 1)c = \frac{1}{2} - c \ .$$

Now, we have

$$t_{end}(\rho_{k+1}) - t_{end}(\rho_k) = t_{end}(\rho_{k+1}) - t_{end}(\beta_k) + t_{end}(\beta_k) - t_{end}(\rho_k) \leq \frac{1}{2} + \frac{1}{2} - c = 1 - c,$$

which proves (1). Also,

$$t_{end}(\beta_k) - t_{end}(\beta_{k-1}) = t_{end}(\beta_k) - t_{end}(\rho_k) + t_{end}(\rho_k) - t_{end}(\beta_{k-1}) \leq \frac{1}{2} - c + \frac{1}{2} = 1 - c,$$

which proves (2). ■

We next show:

Lemma 4.10 *Lower and upper bounds on step time are preserved in $\beta_0\beta\beta'$.*

Proof: By construction, no two computation steps are closer than c in $\beta_0\beta\beta'$; so, the lower bound on step time is preserved. Note also that the difference between consecutive computation steps of a process is maximized when the process is a peripheral node, i_k, for some k such that $1 \leq k \leq s' - 1$, that has no computation steps in either σ_k or ρ_{k+1}. By Claim 4.9(1), this is less than or equal to 1. ■

To complete the proof that $\beta_0\beta\beta'$ is a timed execution we show:

Lemma 4.11 *The time between a send event and the corresponding receive event in $\beta_0\beta\beta'$ is at most d.*

Proof: Let π_1 be a computation event at node u_1 which occurs at time t_1 in $\beta_0\alpha\alpha'$, in which a message is sent; let π_2 be the corresponding delivery event at node u_2, occurring at time t_2 in α. Assume π_1 is scheduled to occur at time t_1' and π_2 occur at time t_2' in $\beta_0\beta\beta'$.

If π_2 occurs in α' then, by construction, $t_2' - t_1' = d$, in $\beta_0\beta\beta'$. So assume π_1 and π_2 occur in $\beta_0\alpha$. We first consider the case where both π_1 and π_2 occur in α. Assume π_1 appears in α_{k_1} and π_2 appears in α_{k_2}, where $1 \leq k_1 \leq k_2 \leq s'$. Clearly, in β, π_1 appears in β_{k_1} and π_2

appears in β_{k_2}. Note that, by construction, $d = t_2 - t_1 > (k_2 - 1)b - k_1 b = (k_2 - k_1 - 1)b$, i.e., $k_2 - k_1 - 1 < \frac{d}{b}$. It follows that

$$\begin{aligned} t'_2 - t'_1 &\leq t_{end}(\beta_{k_2}) - t_{start}(\beta_{k_1}) \leq t_{end}(\beta_{k_2}) - t_{end}(\beta_{k_1-1}) \\ &= \sum_{j=k_1}^{k_2} (t_{end}(\beta_j) - t_{end}(\beta_{j-1})) \\ &\leq (k_2 - k_1 + 1) \qquad \text{(by Claim 4.9(2))} \\ &\leq \frac{d}{b} + 2 \leq d \qquad \text{(by assumption)} , \end{aligned}$$

as needed.

Finally, we consider the case where π_1 occurs in β_0, i.e., $t_1 = 0$. Assume that π_2 occurs in α_{k_2}. By construction, $d = t_2 - t_1 = t_2 > (k_2 - 1)b$, i.e., $k_2 - 1 < \frac{d}{b}$. Reasoning as in the previous case, we get:

$$t'_2 - t'_1 \leq t_{end}(\beta_{k_2}) = \sum_{j=1}^{k_2} (t_{end}(\beta_j) - t_{end}(\beta_{j-1})) \leq k_2 < \frac{d}{b} + 1 < d ,$$

as needed. ■

Lemma 4.5 implies that β contains at most $s' \leq s - 2$ sessions; also, β_0 contains exactly one session. Therefore, there are at most $s - 1$ sessions in $\beta_0\beta$. Since in β' no process takes a non-idle step, there is no additional session in β'. Thus, there are at most $s - 1$ sessions in $\beta_0\beta\beta'$. A contradiction. ■

5 The Non-Uniform Case

In this section we consider the case problem where delays on communication links are not uniform. Specifically, we assume that for each $(i, j) \in E$, the delay of any message along (i, j) is in the interval $[0, d(i, j)]$ for some $d(i, j)$ such that $0 \leq d(i, j) < \infty$.

We first develop some notation that is necessary for stating our results. Let p be a path from node v_0 to node v_k in G, i.e., a sequence of nodes $v_0, v_1, \ldots, v_k$ such that for each i, $1 \leq i < k$, $(v_i, v_{i+1}) \in E$. Denote by $l(p)$ the *length*, k, of p. We define the *delay on* p, $d(p)$, to be the sum of the delay on its edges, i.e.,

$$d(p) = \sum_{i=0}^{k-1} d(v_i, v_{i+1}) .$$

We define the *delay from node i to node j*, $del(i,j)$, to be the minimum of $d(p)$ over all paths p between i and j. Naturally, the *delay on G*, $\hat{d}(G)$, is the maximum of the delay from one node of G to another, over all pairs of nodes in G, i.e.,

$$\hat{d}(G) = \max_{i,j \in V} del(i,j) \ .$$

Intuitively, $\hat{d}(G)$ is the worst-case delay that a message between a pair of nodes may incur along a "shortest-delay" path from i to j in G. However, because of local processing time a message that is sent along a path p can effectively incur a delay of up to $d(p)+l(p)$, since each process in the path can incur a local processing delay of at most 1 and postpone forwarding the message until its next computation step. Thus, we define the *effective delay on G* to be

$$\hat{D}(G) = \max_{i,j \in V} \min_{p \text{ a path from } i \text{ to } j} (d(p) + l(p)) \ .$$

Clearly, in the uniform case, when all delays are equal to d, $\hat{d}(G)$ and $\hat{D}(G)$ are equal to $diam(G)d$ and $diam(G)D$, respectively. Also,

$$\hat{d}(G) \leq \hat{D}(G) \leq \hat{d}(G) + diam(G) \ .$$

Denote $d_{\min} = \min_{(i,j) \in E} d(i,j)$.

To obtain bounds for the non-uniform case, we observe that $\hat{D}(G)$ naturally replaces $diam(G)D$ in the upper bounds for the uniform analogs, while $\hat{d}(G)$ naturally replaces $diam(G)d$ in the corresponding lower bounds.

Also, for the lower bounds for the semi-synchronous model, let $b' = \min\{\lfloor \frac{1}{2c} \rfloor, \hat{d}(G)\}$ and assume, as in the proof of Theorem 4.7, that $b' > 1$. Note that if the condition $d \geq \frac{d}{b'} + 2$ holds with $d_{\min}$ for d, then it also holds with $d(i,j)$, for any $(i,j) \in E$, for d. This implies that the non-uniform analog of the condition $d \geq \frac{d}{b} + 2$ is $d_{\min} \geq \frac{d_{\min}}{b'} + 2$. We next state our upper and lower bound results for the non-uniform case. Their proofs exactly follow those of their uniform analogs and are omitted.

Theorem 5.1 *Let G be any graph. There exists an asynchronous algorithm $\mathcal{A}_w^{as}$ which solves the s-session problem on G within time $\hat{D}(G)(s-1)$.*

Theorem 5.2 *Let G be any graph. There exists a semi-synchronous algorithm $\mathcal{A}_w^{ss}$ which solves the s-session problem on G within time $1 + \min\{\lfloor \frac{1}{c} \rfloor + 1, \hat{D}(G)\}(s-2)$.*

Figure 3 should appear here.

Figure 3: Summary of the results

Theorem 5.3 *Let G be any graph. There does not exist an asynchronous algorithm which solves the s-session problem on G within time strictly less than $\hat{d}(G)(s-1)$.*

Theorem 5.4 *Let G be any graph and assume that $d_{\min} \geq \frac{d_{\min}}{\min\{\lfloor 1/2c \rfloor, diam(G)\hat{d}(G)\}} + 2$. There does not exist a semi-synchronous algorithm which solves the s-session problem on G within time strictly less than $1 + \min\{\lfloor \frac{1}{2c} \rfloor, \hat{d}(G)\}(s-2)$.*

6 Discussion and Directions for Future Research

Assuming that $1 \ll d$, i.e., that $D \approx d$, we have almost matching upper and lower bounds of $diam(G)D(s-1)$ for the asynchronous model. For the semi-synchronous model, we have proved an upper bound of $1 + \min\{\lfloor \frac{1}{c} \rfloor + 1, diam(G)D\}(s-2)$ and a lower bound of $1 + \min\{\lfloor \frac{1}{2c} \rfloor, diam(G)d\}(s-2)$. The proof of the last lower bound relies on the assumption $d \geq \frac{d}{\min\{\lfloor 1/2c \rfloor, diam(G)d\}} + 2$. We remark that this condition holds for large enough values of $\frac{1}{c}$ and d. Neglecting roundoffs, the upper bound is within a factor of 2 of the lower bound. Similar results were proved for the cases where message delays are not uniform. We summarize our main results in Figure 3. The case where processes do not start simultaneously is studied elsewhere ([9]), where our techniques are extended to yield similar results for this case.

The work presented in this paper continues the study of time bounds in the presence of timing uncertainty within the framework of the semi-synchronous model ([2, 3]). Our results give a time separation between semi-synchronous (in particular, synchronous) and asynchronous networks. Unlike previous separation results ([1, 6]), our results do not rely on the ability to schedule several steps by the same process at the same real time.

The results presented in this paper have been extended to the model where processes communicate via shared memory ([8, 12]). Rhee and Welch also studied the session problem in two intermediate timing models—the *sporadic* model and the *periodic* model ([12]).

Our work leaves open several interesting problems. An obvious open problem is to close the gap between the lower and the upper bounds for the semi-synchronous case. It will be interesting to relax the assumption $d \geq \frac{d}{\min\{\lfloor 1/2c \rfloor, diam(G)d\}} + 2$ used to prove the lower bound

for the semi-synchronous model. The definition of a session does not require processes to be "aware" of a session's end; how do the bounds change if this requirement is imposed?

Our results show that there are some synchronous algorithms that *cannot* be simulated by asynchronous algorithms without significant time overhead (e.g., algorithms for the s-session problem). In contrast, the results of Awerbuch ([4]) indicate that there are some synchronous algorithms which *can* be simulated by asynchronous algorithms with only constant time overhead. Perhaps the most interesting extension of our research is to characterize the synchronous algorithms which can (respectively, cannot) be efficiently simulated by asynchronous algorithms.

Acknowledgements:

The authors thank Nancy Lynch for making [6] available to us and for helpful discussions, and Pino Persiano and the anonymous referees for comments on earlier versions of the paper. The second author would like to thank Harry Lewis for his constant advice and encouragement.

References

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

[2] H. Attiya, C. Dwork, N. Lynch and L. Stockmeyer, "Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty," in *Proceedings of the 23rd ACM Symp. on Theory of Computing,* pp. 359–369, May 1991. Also: Technical Memo MIT/LCS/TM-435, Laboratory for Computer Science, MIT.

[3] H. Attiya and N. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," in *Proceedings of the 10th Real-Time Systems Symp.,* pp. 268–284, December 1989. To appear in *Information and Computation.* Expanded version: Technical Memo MIT/LCS/TM-403, Laboratory for Computer Science, MIT.

[4] B. Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM,* Vol. 32, No. 4 (1985), pp. 804–823.

[5] G. M. Baudet, "Asynchronous Iterative Methods for Multi-Processors," *Journal of the ACM,* Vol. 25, No. 2 (April 1978), pp. 226-244.

[6] N. Lynch, *Asynchronous versus Synchronous Computation* (guest lecture), lecture notes for MIT 18.438: Network Protocols and Distributed Graph Algorithms, Spring 1989.

[7] N. Lynch and H. Attiya, "Using Mappings to Prove Timing Properties," in *Proceedings of the 9th Annual ACM Symp. on Principles of Distributed Computing,* pp. 265-280, August 1990. To appear in *Distributed Computing*, Vol. 6, No. 2. Expanded version: Technical Memo MIT/LCS/TM-412.d, Laboratory for Computer Science, MIT, October 1991.

[8] M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Systems: Atomic Shared Memory," to appear in *Computers and Mathematics with Applications.* Also: Technical Report TR-03-92, Aiken Computation Laboratory, Harvard University, January 1992.

[9] M. Mavronicolas, *Timing-Based Distributed Computation: Algorithms and Impossibility Results,* Ph. D. Thesis, Harvard University, 1992. Also: Technical Report TR-13-92, Aiken Computation Laboratory, Harvard University, July 1992.

[10] M. Merritt, F. Modugno and M. Tuttle, "Time Constrained Automata," in *Proceedings of the 2nd International Conference on Concurrency,* Amsterdam, Lecture Notes in Computer Science (Vol. 527), pp. 408–423, Springer-Verlag, 1991.

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

[12] I. Rhee and J. L. Welch, "The Impact of Time on the Session Problem," in *Proceedings of the 11th Annual ACM Symp. on Principles of Distributed Computing,* pp. 191–202, August 1992.

[13] A. Segall, "Distributed Network Protocols," *IEEE Transactions on Information Theory,* Vol. IT-29, No. 1 (January 1983), pp. 23–35.

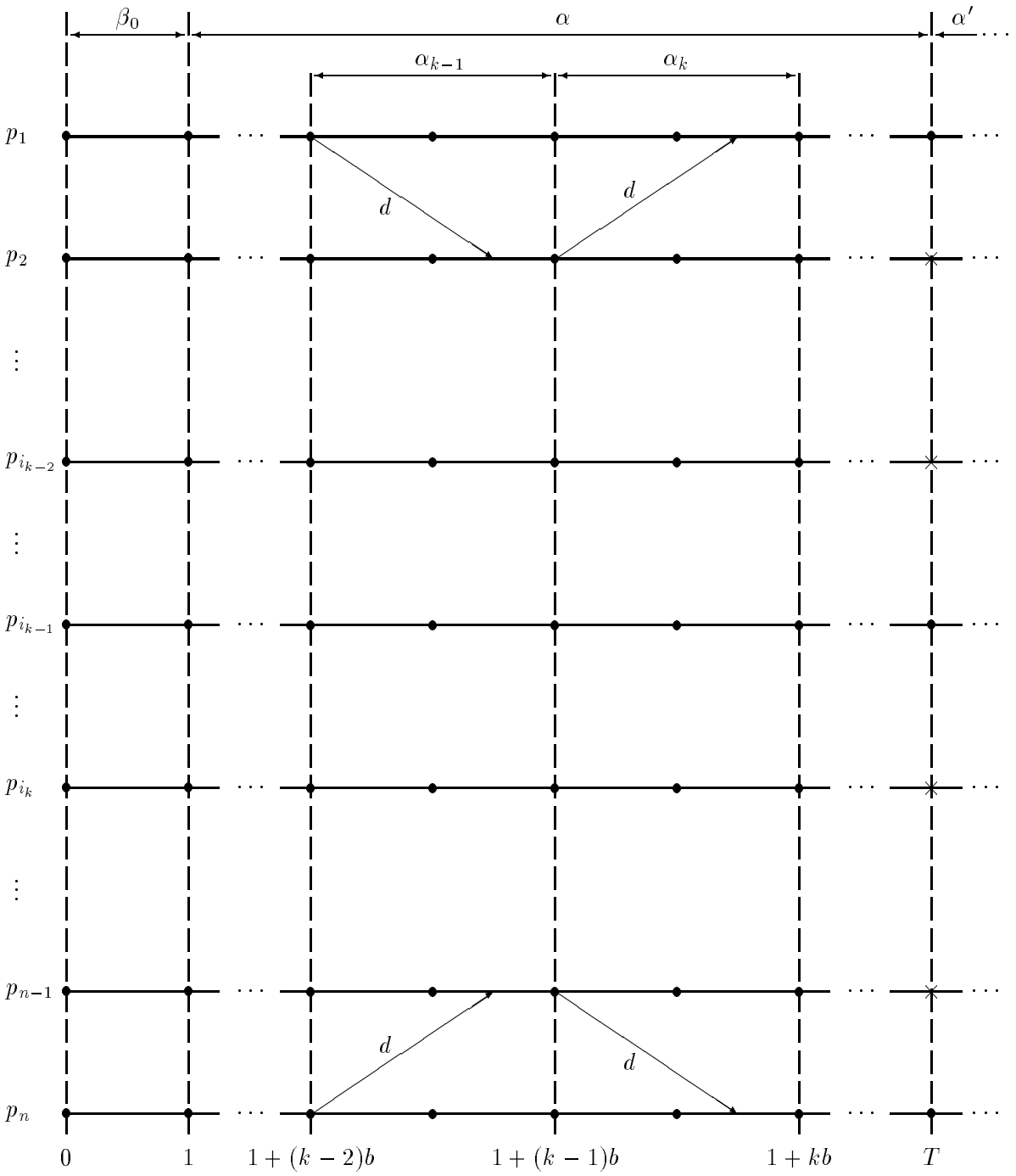
β_0
α
α'
α_{k-1}
α_k
p_1
p_2
$p_{i_{k-2}}$
$p_{i_{k-1}}$
p_{i_k}
p_{n-1}
p_n
d
d
d
d
0
1
$1+(k-2)b$
$1+(k-1)b$
$1+kb$
T

Figure 1

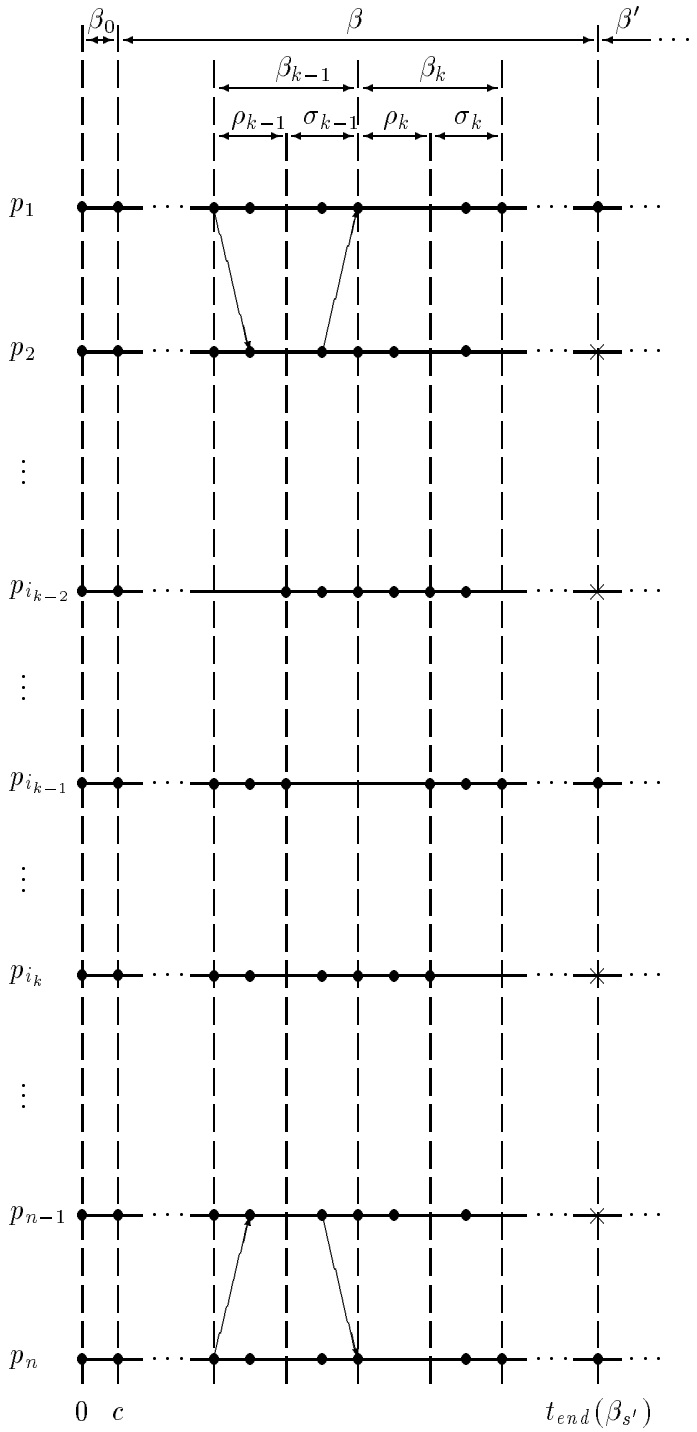
β_0
β
β'
β_{k-1}
β_k
ρ_{k-1}
σ_{k-1}
ρ_k
σ_k
p_1
p_2
$p_{i_{k-2}}$
$p_{i_{k-1}}$
p_{i_k}
p_{n-1}
p_n
0
c
$t_{end}(\beta_{s'})$

Figure 2

Step time range	Upper bound	Lower bound
$[0,1]$	$diam(G)D(s-1)$	$diam(G)d(s-1)$
$(0,1]$	$1+diam(G)D(s-2)$	$1+diam(G)d(s-2)$
$[c,1]$ $(0<c\leq 1)$	$1+\min\{\lfloor\frac{1}{c}\rfloor+1, diam(G)D\}(s-2)$	$\lfloor\frac{1}{c}(s-2)\rfloor$, if no communication is used
		$1+\min\{\lfloor\frac{1}{2c}\rfloor, diam(G)d\}(s-2)$, if $d\geq\frac{d}{\min\{\frac{1}{2c}, diam(G)d\}}+2$

Figure 3.