Chapter 1
On the Microscopic View of Time and Messages

Nicola Santoro

Abstract In distributed message-passing systems, *synchronous* computations rely on and exploit for their correctness and/or efficiency the existence of some reliable mechanism, which provides all system entities with a globally consistent view of *time*, e.g., a common global clock. Many of these computations, however, exploit time at a *macroscopic* level: they assume that transmission of an unbounded amount of information can be done in constant time. We are instead interested in the *microscopic* level of synchronous computations; that is, the study of computability and complexity when, in a constant amount of time, only a constant number of bits can be transmitted. Our general interest includes the extreme case, when a message contains only a single bit. We discuss the basics of computing at the microscopic level, describing simple but powerful computational tools, and analyzing their use.

1.1 Introduction

A *message-passing system* is a model of a distributed computing environment, which in turn models many artificial systems (e.g., distributed systems, communication networks, systolic architectures, etc.); it provides a language to describe its components, its behaviour, its properties; furthermore, it includes the tools for the analysis and the measurement of such an environment.

Time is an human artifact superimposed on nature in our attempt to quantify its qualitative processes. This quantification has the immediate effect of discretizing the perceived continuum and enabling its measurement.

The presence of time and the discretization it imposes are the predominant aspects in *synchronous* message-passing system. Indeed, messages and time are the

School of Computer Science, Carleton University, Ottawa, Canada.
e-mail: santoro@scs.carleton.ca

two crucial components of synchronous computations, and their interplay is the determining element for both feasibility and complexity of the computations.

Many synchronous computations however view and exploit time only at a *macroscopic* level: they assume that a unit of time is large enough for the transmission of an unbounded amount of information.

Instead, like several other researchers, we are interested in the *microscopic* analysis of synchronous computations; that is, we are interested in the study of computability and complexity when, in a unit of time, only a constant number of bits can be transmitted. Our interest covers also the most extreme case, when a message contains only a single bit. Not surprisingly, great part of the nature and beauty of synchronous computing, is revealed only under the microscope.

The aim of this chapter is to introduce the basics of computing at the microscopic level, describing simple but powerful computational tools, and analyzing their use. The terminology and notation are from [37].

1.1.1 Message-Passing Systems

In the language of distributed computing, a message-passing system is a collection of computational *entities* which communicate by sending and receiving bounded sequences of bits called *messages*. A binary relation, called out-neighbour, defines for each entity x the subset of the other entities, called out-neighbourhood, to which x can send a message; analogous is the definition of in-neighbourhood of an entity. If, for each entity, its in-neighbourhood coincides with its out-neighbourhood, we will use the terms neighbour and neighbourhood. The couple $G = (V,E)$ where V is the set of entities and E is the out-neighbour relation defines a graph G which describes the communication topology of the system. Hence, graph-theoretic concepts and terminology (e.g., nodes, edges, diameter, etc.) can be used to describe distributed algorithms and analyze their performance. In the following, the terms vertex, node, site, and entity will have the same meaning; analogously, the terms edge, arc, link and line will be used interchangeably. Messages received at an entity are processed there in the order they arrive; if more than one message arrives at the same entity at the same time, they will be processed in arbitrary order.

Each entity is provided with local processing and storage capabilities, and a local clock. The behaviour of the entities can be conveniently described as finite-state and event-driven; that is, each entity at any time is in a particular system state (from a finite set of states) and, when a predefined external event occurs (e.g., a message is received, the local clock is increased by one unit, etc.), it will serially perform some operations whose nature depends on the current state and on the occurred event. The operations that can be performed are local computations, transmission of messages, and changes of state. Thus, the behaviour of an entity is a set of rules of the form *State x Event* $\rightarrow$ *Action*, where *State* is a system state, *Event* is one of a predefined set of external events, and *Action* is an indivisible sequence of local operations. The set of rules, the same for all entities, is called a *distributed algorithm* or *protocol*.

The entities might have distinguished initial values, e.g., an identity; if this is not the case, the system is said to be *anonymous*.

The basic model is based on only two simple axioms:

- *Local Orientation*: Every entity can distinguish between its (in- and out-) neighbours, and can detect from which in-neighbour a received message was sent.
- *Finite Delays*: In absence of failure, a message sent to an out-neighbour is eventually received there in its integrity.

As a consequence of the Local Orientation axiom, it can be assumed that each entity x has a distinct label associated to each out-edge (i.e., edge connecting x to an out-neighbour) and in-edge (i.e., edge connecting an in-neighbour to x).

Note that the Finite Delays axiom does not imply the existence of any bound on transmission delays; it only states that, in absence of failure, a message will arrive after a finite delay in its integrity.

Any additional restriction of the general model defines a specific submodel. For example, the following additional axiom, called Message Ordering, defines a system where the transmission of messages obeys a FIFO discipline: messages sent to the same out-neighbour, if they arrive, will do so in the same order in which they were sent. By convention, all axioms defining a submodel are common knowledge to all entities. Common restrictions usually relate to reliability, time, or communication.

1.1.2 Synchronous Systems

With respect to *time*, the basic model does not make any assumption on the local clocks nor (except for the fact that it is finite) on transmission time (which include both processing and queueing delays). For these reason, the systems described by the basic model are referred to as *asynchronous*, and represent one end of the spectrum of message-passing systems with respect to time. On the other end are *synchronous systems*; that is, systems defined by two assumptions about *time*:

- *Synchronized Clocks* (SC): all local clocks 'tick' simultaneously (although they might not sign the same value).
- *Bounded Transmission Delays* (BTD): there exists a known upper bound on the number of clock ticks required for message transmission (including processing and queueing delays).

Since the bound is known a priori to all entities and all local clocks tick simultaneously, the unit of time can be redefined so that the BTD axiom can be replaced (as is almost always done) by the axiom

- *Unitary Transmission Delays* (UTD): message transmission is performed in a single unit of time.

In other words, if an entity sends a message at local clock tick t to a neighbour, the message is received and processed there at time $t+1$ (sender's time). To avoid

paradoxical situations, it is assumed that at any clock tick only one message can be send to the same neighbour.

In the following, like in almost all the literature, *synchronous* means simultaneous presence of SC and UTD.

1.1.3 Macro vs Micro

In a synchronous message-passing system $\mathscr{S}$, the complexity of a distributed algorithm is evaluated with respect to two basic parameters: the number of message transmissions performed during the execution, and the number of clock ticks elapsed from the time the first entity starts the execution to the time the last entity terminates its participation in the computation.

The interval of time between successive clock ticks (sometimes called a *round*[1]) is bounded by some system parameter but, by definition, is long enough so that any message sent at a clock tick t arrives and is processed at its destination at clock tick $t+1$. As a consequence,

two very different assumptions on the message size are possible, and have been made in the literature, each offering a different view of synchronous computations:

1. message size is unbounded (e.g., all the data to be transmitted always fits in a single message); this is the *macroscopic view*.
2. the message size is bounded by some system constant B; this is the *microscopic view*.

In the microscopic view, if a computation requires an entity to transmit $M > B$ bits, it is actually requiring the transmission of at least $\lceil M/B \rceil$ messages (and not one, like in the macro level). Furthermore, since at a clock tick the entity can send at most one message to the same neighbour, the transmission of $M > m$ bits will require at least $\lceil M/B \rceil$ clock ticks (and not one, like in the macro level).

These microscopic facts are clearly invisible to the commonly used macroscopic view (e.g., the `LOCAL` model of computation [35]). Indeed, as already mentioned, a macroscopic view hides great part of the nature and beauty of synchronous computing, revealed only under the microscope. From this moment on, we will consider only the microscopic viewpoint.

1.1.4 Under the Microscope: the Difference Time Makes

Under the microscopic view, the unique characteristics of synchronous computations appear very clear. Among the many examples and results, the best known is the one expressed by the following "folk" theorem:

[1] These intervals are usually assumed to have all the same length, but such a condition is not necessary.

Property 1. Any finite sequence of bits can be communicated in $\mathscr{S}$ transmitting two messages, regardless of the message size.

Proof. Let α be the sequence of bits, and let u and v be the transmitting entity and its receiving neighbour, respectively. Consider the following protocol for u: 1. send a message; 2. wait $g(\alpha)$ clock ticks; and 3. send another message, where $g(\alpha)$ is the integer whose binary representation is 1α. The protocol for v is the following: 1. upon receiving the first message, set count to zero; 2. at each clock tick, if no message is received, increase count by one, otherwise stop. Obviously, when y stops, count=$g(\alpha)$. □

The above property is a striking example of the difference that computing with time makes: since the message size is irrelevant and since the string α is finite but arbitrary, and the content of the transmitted messages is irrelevant, the property states that any amount of information (e.g., several Facebook datasets) can be communicated by transmitting just two bits.

The property, as stated, is incomplete from a complexity point of view. In fact, in a synchronous system, time and transmission complexities are intrinsically related to a degree non existent in asynchronous systems. In the example above, the *constant* bit complexity is achieved at the cost of a time complexity which is *exponential* in the length of the sequence of bits to be communicated, as stated by the following reformulation of the above property:

Lemma 1. *Any finite sequence of bits α can be communicated in $\mathscr{S}$ transmitting two bits in time $2^{\alpha+1}$.*

1.1.5 Organization

In the following, we present in some details some interesting aspects of computing with time, always in the inherent interplay between time and transmissions.

In Section 1.2 we discuss the most basic distributed computation, *two-party communication*: the communication of information between two neighbouring entities; the described results are from [32]. In Section 1.3 we present a simple yet powerful technique, *waiting*, that exploits the availability of time as a computational element; the results described in Section 1.3.1 are from [38], those in Section 1.3.2 from [13]. A general technique, *guessing*, which can be used to avoid the transmission of unbounded values, is discussed in Section 1.4; the described results are from [24]. Finally, we look at another basic activity, *wakeup*, and again investigate the time vs bits tradeoffs that it offers in the case of complete network; the discussed results are from [19].

1.2 Two-Party Communication

In a system of communicating entities, the most basic and fundamental problem is obviously the process of efficiently and accurately communicating information between two neighbouring entities.

This problem is sometimes called *TWO-PARTY COMMUNICATION* problem, and any solution algorithm is called a TPC protocol or *communicator*. Due to the basic nature of the process, the choice of a communicator can greatly affect the overall performance of the higher-level protocols employed in the system. Associated with any communicator are clearly two related cost measures: the total number of bit transmissions and the total number of clock ticks elapsed during the communication; as we will see, the study of the two-party communication problem in synchronous networks is really the study of the *trade-off* between time and transmissions.

1.2.1 Basic Communicators

Consider two entities, called the *sender* and the *receiver*, connected by a direct link; at each time unit, the sender can either transmit a bit or remain silent; a bit transmitted by the sender at time t will be received and processed by the receiver at time $t+1$ (sender's time). A *quantum of silence* (or, simply, quantum) is the number of clock ticks between two successive bit transmissions; the quantum is zero if the bits are sent at two consecutive clock ticks.

Given a countable (and possibly infinite) universe U, the *two-party communication problem* for U, denoted by TPC(U), is the problem of the sender communicating without ambiguity to the receiver arbitrary elements of U using any combination of bit transmissions and silence. Since U is countable, we will assume without loss of generality that U is a set of consecutive integers starting from 0.

As observed in Section 1.1.4, any positive integer x can be communicated transmitting only two bits. This is achieved by the well-kown 2-bits Communicator $\mathscr{C}_2$, to which Lemma 1 refers in the Introduction:

Communicator $\mathscr{C}_2$:

- To communicate a positive integer x, the sender transmits a first bit, waits a quantum of silence $q_1 = x$, and then sends the second and final bit b_1
- To reconstruct x , the receiver simply reconstructs the quantum of silence q_1 between the two received bits.

Using this communicator, the number of bits transmitted is 2 and the time is x.

Interestingly, if we increase the number of transmissions, time becomes sublinear. Consider the following protocol C_3 that uses 3 bits:

Communicator $\mathscr{C}_3$:

- To communicate a positive integer x, the sender transmits three bits in order: b_0, b_1, and b_2; the quantum of silence q_1 between the first two transmissions, and q_2 between the second and the last are: $q_1 = \lfloor\sqrt{x}\rfloor$ and $q_2 = x - \lfloor\sqrt{x}\rfloor^2$.
- To obtain x the receiver simply computes $(q_1)^2 + q_2$.

Notice that $q_1 = x - \lfloor\sqrt{x}\rfloor^2 \leq 2\sqrt{x}$; thus, protocol $\mathscr{C}_3$ has time *sublinear* time complexity $\leq 3\lfloor\sqrt{x}\rfloor + 3$. The method used by protocol C_3 can be easily extended to arbitrary $k = 2^r + 1$, obtaining a communicator $\mathscr{C}_k$ that communicates any integer x transmitting k bits using at most $k\, x^{\frac{1}{k-1}} + k$ time units.

Notice that here, as in the rest of this section, the transmitted bits are used only as delimiters; this renders the protocols resistant to message corruptions. In corruption-free systems, the bounds can obviously be improved by using the bits to convey information [32].

1.2.2 Optimal Communicators

At this point the natural question is what are the optimal communicators. We first discuss lower-bounds on the time-bits trade-off for the two-party communication problem both in the worst and in the average case. The bounds apply to any solution protocol, regardless of the schemes employed for encoding, transmitting and decoding. We then describe a solution protocol whose cost matches the lowerbounds.

1.2.2.1 Lower Bounds

Consider $C_b(U)$; i.e., the two-party communication problem for U using exactly b bit transmissions. Observe that b time units will be required to transmit the b bits; hence, the concern is on the amount of *additional* time required by the protocol. Obviously, the time before the first transmission and after the last transmission cannot be used to convey information.

Let $c(U,b)$ denote the number of time units needed in the worst case to solve $C_b(U)$. To derive a bound on $c(U,b)$, we will consider the dual problem of determining the size $\omega(t,b)$ of the largest set $\ddot{U}$ for which $c(\ddot{U},b) \leq t$; that is, $\ddot{U}$ is the largest set for which the two-party communication problem can always be solved using b transmissions and at most t additional time units. Notice that, with b bit transmissions, it is only possible to distinguish $k = b - 1$ quanta; hence, the dual problem can be rephrased as follows:

> Determine the largest positive integer $n = \omega(t,b)$ such that every $x \in Z_n = \{0,1,\ldots,n\}$ can be communicated using $k = b-1$ distinguished quanta whose total sum is at most t.

This problem has an exact solution which will enable us to establish the desired bounds. Let $\mathbf{Bin}(x,y)$ denote the binomial coefficient $\binom{x}{y}$.

Theorem 1. $\omega(t,b) = \mathbf{Bin}(t+k,k)$

Proof. Let $n = \omega(t,b)$; by definition, it must be possible to communicate any element in $Z_n = \{0,1,\ldots,n\}$ using $k = b-1$ distinguished quanta requiring at most time t. In other words, $\omega(t,k+1)$ is equal to the number of distinct k-tuples $\langle t_1,t_2,\ldots,t_k\rangle$ of positive integers such that $\sum_{1\leq i\leq k} t_i \leq t$. Given a positive integer x, let $T_k[x]$ denote the number of compositions of x of size k; i.e.,

$$T_k[x] = |\{\langle x_1,x_2,\ldots,x_k\rangle : \sum x_j = x, x_j \in Z^+\}|$$

Since $T_k[x] = \mathbf{Bin}(x+k-1,k-1)$, it follows that

$$\omega(t,k+1) = \sum_i T_k[i] = \sum_i \mathbf{Bin}(i+k-1,k-1) = \mathbf{Bin}(t+k,k)$$

which proves the theorem. □

We can now establish a *worst case* lower bound. Given two positive integers x and k, let $f(x,k)$ be the smallest integer t such that $x \leq \omega(t,k+1)$.

Theorem 2. *Any solution protocol for $C_{k+1}(U)$ requires $f(|U|,k)$ time units in the worst case.*

Proof. From Theorem 1, it follows that $c(U,b) = f(|U|,k)$. □

Theorem 3. *Let $f(|U|,k) = t$. For any solution protocol P for $C_{k+1}(U)$, there exists a partition of U into $t+1$ disjoint subsets $U_0,U_1,\ldots,U_t$ such that*

1. $|U_i| = \mathbf{Bin}(i+k-1,k-1), 0 \leq i < t; |U_t| \leq \mathbf{Bin}(t+k-1,k-1)$
2. *the time $P(x)$ required by P to communicate $x \in U_i$ is $P(x) \geq i$.*

Proof. Since $f(|U|,k) = t$, by Theorem 1, U is the largest set for which the two-party communication problem can always be solved using $b = k+1$ transmissions and at most t additional time units. Given a protocol P for $C_{k+1}(U)$, order the elements $x \in U$ according to the time $P(x)$ required by P to communicate them; let $\ddot{U}$ be the corresponding ordered set. Define $\ddot{U}_i$ to be the subset composed of the elements of $\ddot{U}$ whose ranking, with respect to the ordering defined above, is in the range $\sum_{0\leq j<i}\mathbf{Bin}(j+k-1,k-1), \sum_{0\leq j\leq i}\mathbf{Bin}(j+k-1,k-1)$. Since $f(|U|,k) = t$, it follows that $|\ddot{U}_i| = \mathbf{Bin}(i+k-1,k-1)$ for $0 \leq i < t$ and $|\ddot{U}_t| \leq \mathbf{Bin}(t+k-1,k-1)$ which proves part 1 of the Theorem.

We will now show that, for every $x \in \ddot{U}_i, P(x) \geq i$. By contradiction, let this not be the case. Let $j \leq t$ be the smallest index for which there exists an $x \in \ddot{U}_i$ such that $P(x) < j$. This implies that there exists a $j' < t$ such that $|\{x \in U : P(x) = j'\}| > \mathbf{Bin}(j'+k-1,k-1)$. In other words, in protocol P, the number of elements which are uniquely identified using k quanta for a total of j' time is greater than the number $T_k[j'] = \mathbf{Bin}(j'+k-1,k-1)$ of compositions of j' of size k; a clear contradiction. Hence, for every $x \in \ddot{U}_i, P(x) \geq i$, proving part 2 of the theorem. □

This gives us an *average case* lower bound:

Theorem 4. *Any solution protocol for $C_{k+1}(U)$ requires*

$$\frac{tm + \sum_{0 \le i < t} i\,\mathbf{Bin}(i+k-1, k-1)}{|U|}$$

time on the average where $t = f(|U|, k)$ and $m = t(|U| - \sum_{0 \le i < t} i\,\mathbf{Bin}(i+k-1, k-1))$.

Proof. From Theorem 3. □

1.2.2.2 An Optimal Solution

We now introduce a protocol whose cost matches both the worst and the average case lower bounds; we can actually show that this communicator is optimal at any point of the time-bits tradeoff.

Given two k-tuples $q = \langle q_1, q_2, \ldots, q_k \rangle$ and $q' = \langle q'_1, q'_2, \ldots, q'_k \rangle$ of positive integers, we say that $q < q'$ if $q_j = q'_j$ for $1 \le j < l$, and $q_l < q'_l$ for some index l, $1 \le l \le k+1$. For a given k, let V_t be the ordered set of k-tuples $q = \langle q_1, q_2, \ldots, q_k \rangle$ where $q_i \in Z^+$ and $\sum_i q_i \le t$; that is $V_t[i] < V_t[i+1]$. Obviously, the size of V_t is $\mathbf{Bin}(t+k, k)$. Any two integers t and i, $1 \le i \le \mathbf{Bin}(t+k, k)$, uniquely identifies a k-tuple $V_t[i] = \langle q_1, q_2, \ldots, q_k \rangle$ where $\sum_i q_i \le t$; conversely, any k-tuple $\langle q_1, q_2, \ldots, q_k \rangle$ uniquely identifies the integers $t = \sum_i q_i$ and $i, 1 \le i \le \mathbf{Bin}(t+k, k)$, such that $V_t[i] = \langle q_1, q_2, \ldots, q_k \rangle$.

The solution algorithm, $P1$, is described below; it comprises of an encoding scheme, a decoding scheme, and a communication protocol.

<u>Encoding Scheme:</u> Given X and k,

1. Let t be the smallest integer such that $X \le \mathbf{Bin}(t+k, k)$; i.e., $t = f(X, k)$.
2. Determine $V_t[X] = \langle q_1, q_2, \ldots, q_k \rangle$
3. Set *encoding*$(X) = \langle p_0, p_1, \ldots, p_{2k} \rangle$, where $p_{2i} = b \in \{0, 1\}$ and $p_{2i+1} = q_i, (0 \le i < k)$.

The value X to be communicated will be encoded as a $(2k+1)$-tuple $\langle p_0, p_1, \ldots, p_{2k} \rangle$, where the even elements $p_0, p_2, \ldots, p_{2k}$ are arbitrary bits and the odd elements $p_1, p_3, \ldots, p_{2k-1}$ form the k-tuple corresponding to the X-th element of the set $V_{f(X,k)}$; i.e., $\langle p_1, p_3, \ldots, p_{2k-1} \rangle = V_{f(X,k)}[X]$.

Once the $(2k+1)$-tuple $\langle p_0, p_1, \ldots, p_{2k} \rangle$ corresponding to the encoding of X has been determined, the actual communication can start. The encoded information is communicated as follows: the element $p_{2i} = b \in \{0, 1\}$ is transmitted and the element $p_{2i+1} = q_i$ is communicated by waiting a quantum of silence of length q_i.

Communication Protocol

SEND(X):
    Compute *encoding*$(X) = \langle p_0, p_1, \ldots, p_{2k} \rangle$;
    for $0 \leq i \leq 2k$
        if even(i) then transmit p_i else wait p_i time units;
    endfor
RECEIVE(Z):
    $i := 0$;
    receive(b);
    $p_0 := b$;
    Repeat until $i = k$
        wait q until receive(b);
        $p_{2i+1} := q$; $i := i+1$; $p_{2i} := b$;
    $Z := \langle p_0, p_1, \ldots, p_{2k} \rangle$;
    Compute *decoding*(Z);

Once the last bit p_{2k} has been received, the receiving entity has received the $(2k+1)$-tuple $\langle p_0, p_1, \ldots, p_{2k} \rangle$ and will apply to it the decoding scheme. To decode $\langle p_0, p_1, \ldots, p_{2k} \rangle$, the receiver will extract the $(k+1)$-tuple $\langle q_1, q_2, \ldots, q_k \rangle$ formed by the odd elements $q_i = p_{2i+1}, (0 \leq i < k)$ and compute $t = \sum_i q_i$; at this point X, the communicated value, is the unique integer such that $1 \leq X \leq \mathbf{Bin}(t+k,k)$ and $V_t[X] = \langle q_1, q_2, \ldots, q_k \rangle$.

Decoding Scheme: Given $Z = \langle p_0, p_1, \ldots, p_{2k} \rangle$ and k,

1. Let $Y = \langle q_1, q_2, \ldots, q_k \rangle$ where $q_i = p_{2i+1}, (0 \leq i < k)$; let $t = \sum_i q_i$.
2. Find X such that $V_t[X] = Y$.
3. Set decoding(Z) = X.

For a fixed k, let $P(X)$ denote the amount of time required by algorithm P to communicate integer X using k bit transmissions. Recall (from Section 3) that $f(X,k)$ is the smallest integer t such that $x \leq \omega(t, k+1)$.

Lemma 2. *For a fixed k, $P(X) = f(X,k)$ for every integer X.*

Proof. By construction. □

Theorem 5. *P is worst-case optimal for every $Z_n = \{0, 1, \ldots, n\}$.*

Proof. By Lemma 2 and Theorem 2. □

Protocol P actually satisfies a much stronger notion of optimality. A solution protocol A is *everywhere optimal* for U if, for every solution protocol B and $\forall b \geq$

2, there exists a permutation π of the elements of U such that $\forall x \in U : A(x,b) \leq B(\pi(x),b)$. In other words, for every choice of the number of transmitted bits, A requires no more time to communicate any element of U (within a relabelling) than any other solution algorithm. Obviously, everywhere optimality implies both worst-case and average-case time-bits optimality.

Theorem 6. *For a fixed k, P is everywhere optimal for every* $Z_n = \{0,1,\ldots,n\}$.

Proof. Given Z_n, let $t = f(n,k)$ be the smallest integer such that $n \leq \omega(t,k+1)$. Assume for simplicity that $n = \mathbf{Bin}(t+k,k)$. Let $S_i = \{x \in Z_n : P1(x) = i\}$. By Lemma 2, for every $x \in Z_n, P1(x) = f(x,k) \leq t$; hence, $|S_i| = \mathbf{Bin}(i+k-1,k-1), 0 \leq i \leq t$. Recall that, by Theorem 2, for any solution algorithm A, there exists a partition of Z_n into $t+1$ disjoint subsets $A_0, A_1, \ldots, A_t$ such that $|A_i| = \mathbf{Bin}(i+k-1,k-1)$ and $A(x) \geq i$ for every $x \in A_i$. Therefore, there exists a permutation π of Z_n such that $P1(x) \leq A(\pi(x))$ for all $x \in Z_n$, proving the theorem. □

1.3 Waiting

In synchronous systems, time can be used to avoid the transmission of messages of unbounded length, i.e., unbounded values. The communicators described in the previous section are an instance of a simple and direct way of exploiting time to communicate unbounded values transmitting only a constant number of bits.

In this section, we describe another technique that makes an explicit use of time and that can be efficiently used as an alternative to transmitting possibly unbounded values. The technique assumes that every entity x, in addition to its own integer value $v(x)$ (not necessarily unique), has locally available a bound w on the number n of entities and a monotonically increasing integer function f, the same for all entities With respect to this technique, an entity can be either *active*, *processing* or *passive*. Initially, all entities are *active*.

The technique applies to both undirected and (strongly connected) directed graphs (i.e., bidirectional and unidirectional networks). In the following, the term 'neighbours' and the phrase 'all other neighbours' are assumed to mean for digraphs 'out-neighbours' and 'all out-neighbours', respectively. The technique is as follows:

Waiting Technique

1. An *active* entity x waits $f(v(x),w)$ time units.
2. If, during this time, it receives any message, it will forward it to all its other neighbours and become *passive*; otherwise it becomes *processing* and sends a message to all its neighbours.

We will now show how to "mutate" the basic technique so to work in different environments and different problems.

1.3.1 Minimum Finding and Election

Consider the situation where each entity x has a a positive integer value $v(x)$; values might not be distinct. *MINIMUM FINDING* is the problem of moving from an initial configuration where all entities are in the same state *available*, to a configuration where every entity whose associated value is the minimum of all the values is in a predefined state *minimum* and all others are in a different predefined state *large*.

To deal with different initiation times, a pre-processing phase is added to the basic technique so to bound the delay between distinct starting times. Following is the algorithm where $w \geq \Delta(G)$ is known to all nodes, and $f(a,b) = 2ab$.

Algorithm WaitMinElect

- **Rule 0**. If an *available* entity wants to start the algorithm or receives an ACTIVATION message, it sends an ACTIVATION message to all other neighbours and becomes *active*.
- **Rule 1**. An *active* entity x waits $f(v(x),w)$ time units, ignoring any ACTIVATION message. If, during this time it receives an END message, it forwards it to all other neighbours and becomes *large*; otherwise, it sends an END message to all its neighbours and becomes *minimum*.
- **Rule 2**. A *large* entity ignores all END messages.

Theorem 7. *The minimum value v_{min} in any synchronous graph G with n nodes and e edges can be found with at most $4e$ bits in at most $2wv_{min} + 2\Delta(G)$ time units, provided $w \geq \Delta(G)$ is known.*

Proof. Let $t(x)$ denote the time delay, from the start of the execution of the algorithm, to the time entity x becomes *active*. Let x and y be two nodes such that $v(x) < v(y)$. Entity x will become active at time $t(x)$ and will wait $f(x,w) = 2v(x)w$ time units; a message broadcasted by x would reach y after $d(x,y)$ time units, where $d(x,y)$ denotes the length of the shortest path from x to y in G. Since (from Rule 0) $t(x) \leq t(y) + d(y,x)$, it follows that

$$t(x) + 2v(x)w + d(x,y) \leq t(y) + d(y,x) + 2v(x)w + d(x,y) < t(y) + 2v(x)w + 2\Delta(G)$$

This implies that the entities with smallest value become *active* within at most $\Delta(G)$ time units from the time the algorithm is first started; they will finish waiting before everybody else, and thus send an END message; furthermore, this message will reach every other entity while they are still waiting. Thus, any entity z with the smallest identity (i.e., $v(z) = v_{min}$) will become *minimum* while all others will become *large*. This process will require at most $2v_{min}w + 2\Delta(G)$ time units. Each edge will be traversed by at most two ACTIVATION messages and two END messages; since a single bit is sufficient to distingush between the two types of messages, a total of at most $4e$ bits will be transmitted. □

Notice that, if all initial values $v(x)$ are distinct, only one entity will become *minimum*, while all others become *large*. This means, that protocol *WaitMinElect* actually solves the *LEADER ELECTION* problem; this problem requires moving the system from an initial configuration where all entities are in the same state ("candidate"), each with a distinct value, to a final configuration where all entities are in the same predefined state ("defeated"), except one which is a distinguished state ("leader"). Hence, the unique *minimum* is the elected *leader*.

Theorem 8. *A* leader *can be elected in any synchronous graph G with n nodes and e edges with at most* $4e$ *bits in at most* $2wv_{min} + 2\Delta(G)$ *time units, where* v_{min} *is the smallest value, provided* $w \geq \Delta(G)$ *is known.*

In specific classes of graphs more specific bounds apply:

Corollary 1. *Knowing n, an election can be performed in a unidirectional* ring *exchanging* $2n$ *bits in time* $(n+1)v_{min} + 2n - 1$*, provided that the entities are aware of being in a ring.*

Proof. To prove the time, choose $f(a,b) = a(b+1)$ and observe that, in a unidirectional ring, $d(x,y) + d(y,x) = n$ for all x and y. For the bit complexity, observe that $e = n$ and that each edge will be traversed by exactly one ACTIVATION and one entities message.

Corollary 2. *Knowing* n_1 *and* n_2*, an election can be performed in a* $n_1 \times n_2$ mesh *exchanging* $O(n)$ *bits in time* $O((n_1+n_2)v_{min})$*, provided that the entities are aware of being in a mesh.*

Proof. In a mesh of $n = n_1 \times n_2$ nodes, $\Delta(G) = n_1 + n_2$; by choosing $w = d(G)$ and $f(a,b) = a(b+1)$ the time bounds is achieved. Since $e = O(n)$, the message result follows.

Corollary 3. *Knowing n, an election can be performed in an unlabelled* hypercube *exchanging* $O(n \log n)$ *bits in time* $O(\log n v_{min})$*, provided that the entities are aware of being in a hypercube.*

Proof. In a hypercube of $n =$ nodes, $\Delta(G) = \log n$; by choosing $w = \Delta(G)$ and $f(a,b) = a(b+1)$ the time bounds is achieved. Since $e = O(n \log n)$, the message result follows.

Corollary 4. *With simultaneous initiation, an election can be performed in a* complete graph *exchanging* $n-1$ *bits in time* $2v_{min} + 1$*, provided the entities are aware of being in a complete graph.*

Proof. Remove Rule 0 (unnecessary because of simultaneous initiation) and Rule 2; modify Rule 1 so that received END message is not forwarded, and choose $f(a,b) = 2a$. This choice of f ensures that the entity with smallest identity v_{min} will finish waiting at least two time units before everybody else; hence, all other entities will become *passive* after $2v_{min} + 1$ time units. Following these modifications, the only communication occurring in this election process will be the bit sent from the entity with smallest identity to all other entities. □

1.3.2 Symmetry Breaking in Rings

If the assumption on the uniqueness of the values $v(x)$ does not hold, the election problem cannot obviously be solved by an extrema-finding process. If the nodes have no identities (i.e., the system is anonymous) then no deterministic solution exists for the election problem, duly renamed *SYMMETRY BREAKING*, regardless of whether the network is synchronous or not [3]. Thus, if any solution exists, it must be a randomized algorithm.

.

We now shown that, using the *Waiting Technique*, symmetry can be broken in a ring with $O(n)$ bits and time units on the average without any assumption on simultaneous initiation.

The algorithm is composed of a sequence of rounds; in each round, all nodes become *awake*. In round i, upon becoming *awake*, a node x chooses a random value $v(x,i) \in \{0,1\}$ with a biased coin: it selects 0 with probability $\frac{1}{n}$ and 1 with probability $\frac{n-1}{n}$. All nodes participate in determining whether exactly one node has chosen 0 (Situation 1), or not (Situation 2). If Situation 1 has occured, the only node that has chosen 0 becomes *leader*, all other nodes become *defeated*, and the algorithm terminates; if Situation 2 has occured, all nodes start a new round.

Initially, all nodes are in a *sleeping* state. Any *sleeping* node can spontaneously become awake at any time and start the first round. To deal with different initiation times, a pre-processing phase is added in each round so to bound the delay between distinct starting times in that round.

A detailed description of the algorithm is as follows.

Algorithm SymmBreak

- **Rule 1.** A *sleeping* node:
  1. It can become spontaneously *awake* and execute the Wake-up routine.
  2. If it receives a WAKE-UP message, it becomes *awake* and executes the Wake-up routine.
- **Rule 2.** An *awake* node:
  1. It ignores any received WAKE-UP message.
  2. If it receives a CLAIM message, it becomes *half-awake* and sends the message on.
  3. If it receives a END message, it becomes *defeated* and passes the message on. (* Situation 1 *)
  4. When clock$= n$, if no CLAIM is received (see rule 2.2) and the number it selected is 0 it becomes *candidate* and sends a CLAIM message.
  5. When clock$= 2n$, if no CLAIM is received (see rule 2.2) it executes the Wake-up routine. (* Situation 2 *)
- **Rule 3.** A *candidate* node:
  1. If it receives a CLAIM with clock$< 2n$, it becomes *awake* and executes the Wake-up routine. (* Situation 2 *)

2. If it receives a CLAIM and its clock equals $2n$, it becomes *elected* and sends a END message. (* Situation 1 *)

- **Rule 4.** A *half-awake* node:
  1. If it receives a WAKE-UP message, it becomes *awake* and executes the Wake-up routine.
  2. If it receives a END message, it becomes *defeated* and passes the message on. (* Situation 2 *)

where the Wake-up routine is as follows

Wake-up Routine

1. choose 0 with probability $\frac{1}{n}$ and 1 with probability $\frac{n-1}{n}$;
2. set clock:=0 and send a WAKE-UP message.

An important property of the algorithm is expressed by the following

Lemma 3.
(i) Every node starts its execution of a round within $n-1$ time units from the start of that round.
(ii) If exactly one node becomes candidate *during this round, that node becomes* elected *and all others become* defeated*; otherwise, all nodes start another round.*

Proof. Call a round a *success* if Situation 1 occurs. Assume the algorithm has performed $s-1$ unsuccessful rounds and that (i) holds at the beginning the s-th round ($s \geq 1$). Let $t(x)$ denote the time at which node x becomes awake in this round; a node x becomes *candidate* if and only if it has choosen 0 and it has not received any CLAIM in the (global) time interval $(t(x), t(x)+n)$; furthermore, only *candidate* nodes originate CLAIM messages. Three cases are possible depending on whether exactly one, more than one, or no node becomes *candidate* in the round, respectively.

Case 1: exactly one node x becomes *candidate*. (Note: this case occurs if and only if only one node chooses 0 in this round.) In this case, x will send a CLAIM at time $t(x)+n$. This message will reach node y at time $t(x)+d(x,y)+n$; since $t(x) \leq t(y)+d(y,x)$, it follows that node y will receive the CLAIM at time $t(x)+d(x,y)+n \leq t(y)+d(y,x)+d(x,y)+n = t(y)+2n$. Thus, by rule 2.2, y becomes *half-awake* and sends the message on. In other words, the CLAIM message originated by x will travel along the ring transforming every node (except x) into *half-awake* and will arrive at x at time $t(x)+2n$; when this occurs, x becomes *elected* and originates an END message (rule 2.2) which will make all other nodes *defeated* (rule 4.2).

Case 2: more than one node becomes *candidate*. Let $x_1, x_2, ..., x_k$ become *candidate* in this round; w.l.g. assume $t(x_i) \leq t(x_{i+1})$, and let $r(x_i)$ denote the *candidate* nearest to x_i clockwise. First observe that, for all *candidate* nodes x_i and x_j,

$t(x_j) < t(x_i) + d(x_i, x_j)$ (otherwise $t(x_i) + d(x_i, x_j) + n \leq t(x_j) + n$, and x_j would receive a CLAIM with clock$\leq n$ becoming *half-awake* and not *candidate* by rule 2.2). This implies that $t(x_i) + n < t(r(x_i)) + d(r(x_i), x_i) + n$; that is,

$$t(x_i) + d(x_i, r(x_i)) + n < t(r(x_i)) + d(r(x_i), x_i) + d(x_i, r(x_i)) + n = t(r(x_i)) + 2n$$

In other words, a CLAIM from x_i will reach node $r(x_i)$ before $r(x_i)$ counts $2n$. By rule 3.1, $r(x_i)$ will then kill the CLAIM and start the next round by becoming *awake* and sending a WAKE-UP message; thus, within at most $n-1$ additional time units from the time the first x_i becomes *awake* again, all nodes are *awake*.

Case 3: nobody becomes *candidate*. (Note: this case occurs if and only if nobody chooses 0.) In this case, no CLAIM will be sent, and each node x will start the next round by becoming *awake* at time $t(x) + 2n$ (rule 2.5).

Summarizing, if part (i) of the lemma holds for the s-th round, then part (ii) will also hold; furthermore, if the round is not a success, part (i) will hold for the $(s+1)$-th round. Since (i) holds initially (i.e., for $s = 1$), the lemma is proved. □

The only thing left now is to see after how many rounds a leader will be elected. Perhaps surprisingly, the process terminates after less than 3 expected rounds.

Theorem 9. *Symmetry can be broken in a unidirectional ring using* $2n$ *bits and* $2\mathbf{e}n$ *time units on the average regardless of the initiation time, where c is a constant and* $\mathbf{e} = 2.7\ldots$ *is the basis of the natural logarithm.*

Proof. In any one round, each node will send exactly one WAKE-UP message. If at least one node becomes *candidate*, then each node will send or forward exactly one CLAIM message. Since there are a constant number of message types, each message will use a constant c number of bits. Thus, each round will use at most $2cn$ bits. Within $2n$ time steps when the first node on a round executed the Wake-up routine, either a unique node is *elected* or a new round is started. For any round of random selections, the probability that exactly one node selects 0 is

$$\mathbf{Bin}(n,1)\frac{1}{n}(\frac{n-1}{n})^{n-1} = (\frac{n-1}{n})^{n-1}$$

For n large enough, this quantity is easily bounded:

$$\lim_{n\to\infty}(\frac{n-1}{n})^{n-1} = \frac{1}{\mathbf{e}}$$

Thus, the expected number of rounds until this situation occurs is less than $\mathbf{e}$; by Lemma 3, if this event occurs, the algorithm terminates. □

In the above theorem, the factor 2 can be removed from both the time and bit complexity by allowing the nodes to immediately become *candidate* if they select 1 in the Wake-up routine, and modifying the algorithm appropriately.

1.4 Guessing

Another powerful technique that allows to compute functions on unbounded values without ever transmitting them is guessing. Let us consider again the *MINIMUM FINDING* problem, that is the problem of computing $v_{min} = \min\{v(x)\}$. Let us assume that all entities know n and start at the same time.

Consider the following distributed algorithm, where p is a parameter available to all entities:

Decide(p)
$clock := 0$; (* start counting *)
if $v(x) \leq p$ then
  send YES to all neighbors;
  $state := decided$;
else $state := undecided$;
if (YES is received and $clock < n$ and $state = undecided$) then
  send the message to all neighbors which have not sent any message to you;
  $state := decided$;
else ignore the message.

Note that forwarding a YES message can be done at most once by any entity since after sending it the entity becomes *decided*. Also note that, due to the synchrony in the network, this message could have been received from more than one neighbor in the current time slot, and that it is forwarded only to the other neighbors.

Lemma 4. *Let all entities know n and p, and simultaneously start the execution of* **Decide***(p) at time* 0*. Then, at time n:*

1. *if all local values are greater than p, then all entities are undecided;*
2. *if there is at least one local value $v(x) \leq p$, then all entities are decided.*

Furthermore, the number of bits transmitted is zero in case (1), and at most $2e$ *in case (2).*

Proof. At time zero entity x becomes *undecided* and sends no message iff its value $v(x)$ is greater than p. Thus, if all values are greater than p, no messages will be transmitted during the execution of **Decide**; furthermore, all entities will remain *undecided*, at time n. If entity x has local value $v(x) \leq p$, it will become *decided* at time zero and send YES messages to all its neighbours. An entity in state *decided* ignores all YES messages; an entity in state *undecided* receiving a YES message becomes *decided* and forwards the message only to the neighbours from which such a message has not yet been received; thus, at most two messages will be transmitted on each edge, for a total of at most $2e$ bits. Since the underlying communication graph is connected, it is easily shown that by time n each entity

that was not decided at time zero has received at least one YES message. Since an *undecided* entity becomes *decided* as soon as it receives a message, all entities become *decided* within $n-1$ time units. □

Using this property, we can effectively employ **Decide** as a building block for our computations.

1.4.1 Minimum Finding as a Guessing Games

A technique for minimum-finding can be developed by performing a sequence of executions of **Decide** as follows. Initially, all entities choose the same initial value g_1 and simultaneously perform **Decide**(g_1). After n time units, all entities will be aware of whether the minimum value is greater than g_1 (case (1) in Lemma 4) or not (case (2) in Lemma 4); note that the latter case means that we overestimated or guessed the minimum value; this case will be called *overestimate*, even if the correct value has been guessed. Based on the outcome, a new value g_2 will be chosen by all entities, which will then simultaneously perform **Decide**(g_2). In general, based on the outcome of the execution of **Decide**(g_i), all entities will choose a value g_{i+1} and simultaneously perform **Decide**(g_{i+1}); this process is repeated until the minimum value is unambiguously determined. Depending on which strategy is employed for choosing g_{i+1} given the outcome of **Decide**(g_i), different minimum-finding algorithms will result from this technique. This technique allows to reformulate the minimum-finding problem in terms of a *number-guessing game*, as follows.

Guessing Game

1. the network is a player;
2. the minimum value in the network is a number, previously chosen and unknown to the player;
3. the player has to guess the number, by only asking questions of the type "is the number greater than g?", where each question corresponds to a simultaneous execution of **decide**(g);
4. cases (1) and (2) of Lemma 4 correspond to a "yes" and a "no" answer to the question, respectively; the latter case will be termed an *overestimate*.

First observe that, by definition, to each solution strategy for this number-guessing game corresponds a solution algorithm for the minimum-finding problem. As for the complexity of these solution algorithms recall that, by Lemma 4, each execution of **Decide** (i.e., each question) requires n time units, while the number of bits transmitted is either zero or at most $2e$, depending on whether the answer is "yes"or "no", respectively. The following theorem has thus been proved.

Theorem 10. *Let S be a solution strategy for the number-guessing game which requires $b(X)$ overestimates and a total of $t(X)$ questions in the worst case, where X is the unknown number. Let v_{min} be the smallest value in the network, and assume n is known to all entities. Then*

1. *minimum-finding can be performed in an anonymous network using at most $n \cdot t(v_{min})$ time and $2 \cdot e \cdot b(v_{min})$ bits;*
2. *election in a network with distinct values can be performed using at most $n \cdot t(v_{min})$ time and $2 \cdot e \cdot b(v_{min})$ bits;*
3. *a spanning-tree in a network with distinct values can be constructed using at most $n \cdot t(v_{min})$ time and $2 \cdot e \cdot (b(v_{min}) + 2)$ bits.*

1.4.2 Optimal Solutions

We are interested in determining the guessing strategy that offers the best use of time for reducing the amount of bit transmissions. This means to find the strategy that solves the guessing games with the minimum number of questions (each question costs n time units) of which up to a given number b are overestimates (each overestimate costs the transmission of $2e$ bits).

Consider first the case in which the unknown number is a positive integer in the interval $[1,M]$; i.e., the values are $v(x) \leq M$. We will see later the case when the interval is unbounded, i.e., $M = +\infty$.

1.4.2.1 Lower Bound

We want to determine the minimum number $h(M,b)$ of questions needed to correcly guess any value in $[1,M]$ with no more than b overestimates.

To do so, we consider the "converse" problem of determining the largest integer $f(t,b)$ such that any value X known to be within the interval $[1, f(t,b)]$ can be guessed using at most t questions of which at most b are overestimates. The next theorem determines $f(t,b)$.

Theorem 11. *For every $t \geq b \geq 1$,*

$$f(t,b) = \sum_{0 \leq i \leq b} \mathbf{Bin}(t,i) \tag{1.1}$$

Proof. It is easy to see that $f(t,1) = t+1$, since the only algorithm that makes at most one overestimate is "sequential search"; i.e., using the guesses $g_1 = 1, g_2 = 2, ..., g_t = t$. It also trivial to see that $f(t,t) = 2^t$ since, if any question can be an overestimates, the largest possible interval is $[1,2^t]$.

Now, for $t > b \geq 1$, suppose the first question is "is the number $> v$?". If the answer is *yes*, the unknown number is greater than v, and the player has to find it with $t-1$ questions and b overestimates; hence, the largest interval that can be

correctly searched in this case is $[v+1, v+f(t-1,b)\,]$. If the answer is *no*, then the unknown number lies in the interval $[1, v\,]$, to be searched using $t-1$ questions and at most $b-1$ overestimates. Thus, the largest value of v that allows for a correct solution in this case is $f(t-1,b-1)$. We therefore have

$$\forall b,t > b \geq 1,\ f(t,b) = f(t-1,b) + f(t-1,b-1). \tag{1.2}$$

One can show now that the unique solution to (1.2), satisfying the boundary conditions $f(t,1) = t+1$ and $f(t,t) = 2^t$, is given by (1.1). (Another approach is to determine the generating function $F(x,y) = 1/(1-z)(1-y-z*y))$.) $\square$

For future use we extend the definition of $f(t,b)$ to the case where $1 \leq t < b$, so as to satisfy (1.2) for every t and b, by

$$f(t,b) = f(t,t) \qquad for\ 1 \leq t < b. \tag{1.3}$$

We can now return to our original quest for determining a bound on the minimum number $h(M,b)$ of questions needed to correcly guess any value in $[1,M]$ with no more than b overestimates. We are now able to do so; in fact, by Theorem 11 we have:

Theorem 12. *Let* $\hat{t}(M,b) = \min\{t :\ f(t,b)\ \geq M\ \}$. *Then, for every* $b \geq 1$,

$$\hat{t}(M,b) \geq\ h(M,b)\ \geq \hat{t}(M,b) - 1 \tag{1.4}$$

Observe that, if $M = f(t,b)$, then $h(M,b)\ = \hat{t}(M,b) = t$.

There is no known closed-form expression for $h(M,b)$; however, it can be closely estimated as follows:

Lemma 5.

$$h(M,b)\ = (b\,!\,N)^{\frac{1}{b}}\ +\ \varepsilon\, b\ ,\quad \text{for some } \varepsilon = \varepsilon(N,b)\ \text{ with } -1 < \varepsilon < 1\ . \tag{1.5}$$

Proof. 1. By induction on (t,b) (using (1.2)), one can show that

$$\mathbf{Bin}(t+1,b) \leq f(t,b) \tag{1.6}$$

By (1.4) and (1.6) we have

$$\mathbf{Bin}(h,b) \leq f(h-1,b) < N$$

and hence

$$h(h-1)\cdots(h-b+1) < b\,!\,N.$$

Thus $(h-b+1)^b < b\,!\,N$ and we have

$$h < (b\,!\,N)^{\frac{1}{b}} + b\ . \tag{1.7}$$

2. By induction on (t,b) (using (1.2)), one can show that

$$f(t,b) \leq \mathbf{Bin}(t+b,b). \tag{1.8}$$

By (1.4) and (1.8) we have

$$N \leq f(h,N) \leq \mathbf{Bin}(h+b,b),$$

and hence

$$b \,!\, N \leq (h+b)(h+b-1)\cdots(h+1).$$

Thus, using the inequality between the arithmetic and geometric means:

$$\begin{aligned}(b \,!\, N)^{\frac{1}{b}} &\leq [(h+b)(h+b-1)\cdots(h+1)\,]^{\frac{1}{b}} \\ &\leq \frac{1}{b}\,[(h+b)+(h+b-1)+\cdots+(h+1)\,] = h+\frac{b+1}{2}\,.\end{aligned}$$

Therefore

$$h \geq (b \,!\, N)^{\frac{1}{b}} - \frac{b+1}{2}\,. \tag{1.9}$$

The lemma follows from (1.7) and (1.9). □

1.4.2.2 Optimal Protocol

The optimal guessing strategy follows directly from the proof of Theorem 11:

Optimal Guessing Strategy To optimally search in $[1,M]$ with at most k overestimates:

1. Use as a guess $p = h(q-1,k-1)$, where $q \geq k$ is the smallest integer such that $M \leq h(q,k)$.
2. If p is an underestimate, then optimally search in $[p+1,M]$ with k overestimates.
3. If it is an overestimate, then optimally search in $[1,p]$ with $k-1$ overestimates.

This means that

Theorem 13. *The number guessing game in a bounded interval $[1,M]$ can be solved with b bits and $\hat{t}(M,b)$ questions.*

Worst case optimality follows from Theorem 12.

We must still consider solving the guessing game when the interval in unbounded; i.e., $M = +\infty$. A solution strategy could be to first determine a bounded

interval containing X using $b' < b$ overestimates, and then use the Optimal Guessing Strategy above to find X in this interval with at most $b - b'$ overestimates.

To determine an interval containing X, we find an upperbound on X by using a monotonically increasing integer function g and proceeding through a sequence of questions "*is the number* $> g(i)$?" ($i = 1, 2, \cdots$), until we determine the value j such that $g(j-1) < X \leq g(j)$. This approach requires exactly j questions and one overestimate.

Once this is done, we are left to determine X in an interval of size $\Delta(j) = g(j) - g(j-1)$ with only $b-1$ overestimates; this can be done using the Optimal Guessing Strategy above with at most $h(\Delta(j), b-1)$ questions.

The entire process will thus require at most $j + h(\Delta(j), b-1)$ questions. In other words:

Theorem 14. *The number guessing game in a unbounded interval can be solved with b overestimates using at most* $2h(X,b) - 1$ *questions, where X is the unknown number.*

Proof. Choose $g(i) = f(i,b)$, for $i \geq 1$. Let $t = h(X,b)$ (i.e., the smallest integer i such that $f(i,b) \geq X$); then, following the above procedure, we stop when $j = t$. From this follows that $\Delta(j) = g(t) - g(t-1) = f(t,b) - f(t-1,b) = f(t-1,b-1)$; that is, $t-1$ questions will suffice to solve the resulting guessing game in an interval of size $\Delta($ j with $b-1$ overestimates. Altogether, we determined the unknown X with a total of at most $2t-1$ questions and b overestimates. □

1.4.3 Improved Bounds for Distributed Problems

Using the correspondence between guessing games and minimum-finding, the results of the previous section will now be reinterpreted in the context of distributed computations. First observe that, in the distributed problem, no upper-bound is assumed on the range of the values among which the minimum must be found. Further observe that each solution strategy for the number guessing game with k overestimates corresponds to a minimum-finding algorithm requiring the transmission of $O(k \cdot e)$ bits (Theorem 10). Let C_k denote the class of such minimum-finding algorithms.

Theorem 15. *The minimum value* v_{min} *in a synchronous anonymous network can be determined using at most* $O(k \cdot e)$ *bits in time* $O(k \cdot n \cdot v_{min}^{\frac{1}{k}})$ *for any integer* $k > 0$*, provided n is known to the entities, and the entities start simultaneously. For every value of the integer k, this bound is optimal among all algorithms in* C_k*.*

Proof. Let t be the smallest integer such that $f(t,k) \geq v_{min}$; by Theorem 13 it follows that v_{min} can be guessed using at most $2t-1$ questions. Thus, by Theorem 10, the minimum value v_{min} can be determined using at most $(4t-2) \cdot n$ time and $2 \cdot k \cdot e$ bits. By Lemma 5, $t < (k! \cdot i)^{\frac{1}{k}} + k$, which is approximately $\frac{i^{\frac{1}{k}} k}{e} + k$ (using Stirling's

approximation), from which the bound follows. By Theorem 11 and Lemma 5, any algorithm in C_k requires at least $\Omega((\frac{k!\cdot v_{min}}{2})^{\frac{1}{k}})$ questions, from which optimality follows. □

In a similar way, the following theorem can be proved.

Theorem 16. *In a synchronous network with distinct values, election and spanning-tree construction can be performed using at most $O(k \cdot e)$ bits in time $O(k \cdot n \cdot v_{min}^{\frac{1}{k}})$ for any k, where v_{min} is the smallest value in the network, provided n is known and the entities start simultaneously.*

Again, by choosing k to be any constant > 1, the theorem yields an improvement in the time complexity of using waiting (Theorem 8) without increasing the order of magnitude of the bit complexity.

1.5 Waking Up in Complete Networks

A basic activity in distributed computing systems is that of *WAKE-UP*: initially all entities are *asleep*; one or more entities, called *initiators*, independently wake-up and send WAKEUP messages to some neighbours, starting a process to ensure that within finite time all entities become *awake*. Since the WAKEUP message contains no other information, the wakeup process is a prototypical microscopic computation.

The wake-up process is used in a variety of situations, including *initialization*, *notification*, and *reset*, e.g., to ensure that every entity in the system becomes aware of the start (or termination) of a computation. Called also *weak unison* and *distributed reset*, this process can be carried out by totally anonymous entities. It offers an interesting trade-off between time (the difference between the time the first entity wakes up and the time that all entities are awakened) and communication (number of WAKEUP messages sent in that interval) in synchronous networks.

This is indeed the case in *complete networks*. For simplicity, let denote the set of nodes by $\{0,1,\ldots,n-1\}$, and for nodes x,y, the label of the edge $\{x,y\}$ is the integer $(y-x) \bmod n$.

The obvious solution is to "flood": an initiator sends a WAKEUP message to all its neighbours. Since we are in a complete graph, this protocol requires only a single round. The number of messages will however be $k(n-1)$, where k is the number of initiators; this means that, in the worst case, n^2-n messages will be transmitted. The only way to keep down the number of messages is for initiators to send only to a subset of their neighbours. For example, if each node x sends only to node $(x+1) \bmod n$ then only n messages will be sent in total, regardless of the number of initiators. However, the time to complete wakeup will be the maximum distance between successive initiators; this means that, in the worst case (i.e., with a single initiator), the wakeup requires $n-1$ time units.

In this type of situation, to measure the complexity of a protocol, the integrated cost measure $time \times bits$ (*TB*) is used, i.e., the number of messages times the num-

ber of steps required in the worst case for the completion of the algorithm, over all possible choice and schedule of the initiators. Notice that, for the two algorithms described above, the TB-complexity is the same: $O(n^2)$. The quest is for more efficient wakeup protocols.

1.5.1 Oblivious Protocols

We consider a special class of protocols, those where an entity sends the WAKEUP message to the same set of neighbours both when it is an initiator and when it receives a WAKEUP message while asleep. This class of protocols is called *oblivious* (because the set of neighbours does not depend on the state of the entity).

Notice that, since the network is anonymous, an oblivious protocol P can be seen as specifying a subset S of integers modulo n and requiring every entity x to send a WAKEUP message only to the subset $\{(x+j) mod n : j \in S(P)\}$ of its neighbours. It is assumed that $1 \in S$ while $0 \notin S$.

Further notice that the set $S = \{d_0, d_1, d_2, \ldots, d_{k-1}\}$, where $d_0 = 1$ and $d_{i-1} < d_i < n$ for $1 \leq i < n$, defines a graph $R_n[d_1, d_2, \ldots, d_{k-1}]$ where the nodes are $0, 1, \ldots, n-1$ and there is an edge between nodes x, y if and only if $(x-y) \bmod n \in S$. The class of graphs so defined are known as *chordal ring*.

Example 1. For $n \leq 2^k$ the chordal ring $R_n[2, 4, \ldots, 2^{k-1}]$ has diameter $\leq 2 \log n$ and degree $\log n$. This chordal ring is also called *double-cube*.

Example 2. For $n < k!$ the chordal ring $R_n[2!, 3!, \ldots, (k-1)!]$ has diameter $O(k^2)$ (use the fact that every $x < n$ can be represented in the mixed basis $1!, 2!, \ldots, (k-1)!$ as $x = x_1 + x_2 2! + \cdots + x_{k-1}(k-1)!$, with $0 \leq x_i \leq i$, for $i \geq 1$) and degree $n \leq \log n / \log \log n$.

Summarizing, the edges on which messages are sent during the execution of an oblivious wakeup protocol P form a chordal ring R. The message and the time complexity of P will be the number of edges and the diameter of R, respectively.

We will use this observation to derive an optimal wakeup protocol.

1.5.2 Lower Bounds for Oblivious Protocols

We first derive a lower bound on the TB-complexity of oblivious wakeup protocols. The derivation of this lower bound is based on the following bound for chordal rings.

Lemma 6. *Let chordal ring $R_n[d_1, d_2, \ldots, d_{k-1}]$ have diameter ψ. Then*

$$k \cdot \psi = \Omega\left(\log^2 n\right)$$

As a consequence

Theorem 17. *Any oblivious wakeup protocol has $\Omega(n\log^2 n)$ TB-complexity.*

Proof. This follows easily from Lemma 6. Since the protocol is oblivious every entity transmits a fixed number of messages in each iteration of the wakeup protocol, say k. The graph resulting from such a protocol is the chordal ring $R_n[S]$, where S is a set of size k. The time required for the WAKEUP message to reach all the entities is at least the diameter ψ of the chordal ring $R_n[S]$. Eventually all n entities are awakened. Since the protocol is oblivious every entity that receives a WAKEUP message must transmit to all its k neighbours. Hence the number of messages transmitted during the execution of the protocol is nk. It follows that the complexity is at least $nk\psi = \Omega(n\log^2 n)$. □

1.5.3 Optimal Oblivious Wakeup

In this section we describe an optimal oblivious wakeup algorithms. To derive it, first observe that the k-dimensional mesh can be viewed as a chordal ring $R_n[S]$, for some set S of links. For example, the 2-dimensional mesh is the chordal ring $R_n[\sqrt{n}]$, while the k-dimensional mesh is the chordal ring $R_n[n^{1/k}, n^{2/k}, \ldots, n^{(k-1)/k}]$.

For each point $p = (p_1, p_2, \ldots, p_k)$ in the k-dimensional mesh, let

$$S_p^i = \{(p_1, \ldots, p_{i-1}, x_i, p_{i+1}, \ldots, p_k) : x_i < n^{1/k}\}.$$

If we define $S^i = \{(0, \ldots, 0, x_i, 0, \ldots, 0) : 0 \le x_i < n^{1/k}\}$ then we see easily that $S_p^i = p + S^i$. Let $S_p = S_p^1 \cup \cdots \cup S_p^k$, and $S = S^1 \cup \cdots \cup S^k$.

This indicates a way to design an efficient oblivious wakeup algorithm for the complete graph which terminates in k steps. In this protocol, entity p sends wakeup messages to all and only the entities in the set $p + S$.

Oblivious Chordal Wakeup (for entity p)

1. If p is an initiator then it sends a WAKEUP message to all its neighbors in the set $p + S$ and becomes awake.
2. If p receives a WAKEUP message from another entity and is not awake then it sends WAKEUP messages to all entities in the set $p + S$ and and becomes awake.

Theorem 18. *For any k, if $n = q^k$ for some q, then protocol Oblivious Chordal Wakeup has TB-complexity $O(k^2 n^{(k+1)/k})$*

Proof. Let $n = q^k$; then in the execution of protocol Oblivious Chordal Wakeup, the size of each transmission is $kn^{1/k}$. A transmission from entity $p = (p_1, \ldots, p_k)$ will reach all entities of the form $p' = (p_1, \ldots, p_{i-1}, p_i', p_{i+1}, \ldots, p_k)$, where $0 \le p_i' < n, i = 1, 2, \ldots, k$. Therefore every entity will be reached after k steps. The complexity is easily seen to be as claimed. □

More generally, by carefully choosing S, we have

Theorem 19. *For any k and any $m \leq n^{1/k}$, if $n = q^k$ for some q, then protocol Oblivious Chordal Wakeup is a t-step wakeup protocol, where $t = \frac{k}{m} n^{1/k}$, and its TB-complexity is $O(m\ k\ t\ n) = O(k^2 n^{(k+1)/k})$.*

Proof. Let $n = q^k$, $m \leq n^{1/k}$, and $t = \frac{d}{m} n^{1/k}$. In the execution of Oblivious Chordal Wakeup, each entity p transmits to the set $p + S$, where $S = S_1 \cup \cdots \cup S_k$ and $S_i = \{(0,\ldots,0,x_i,0,\ldots,0) : 0 \leq x_i < m\}$. □

We now have all the ingredients to make Oblivious Chordal Wakeup optimal.

Theorem 20. *Oblivious wakeup is possible with optimal TB-complexity $\Theta(n \log^2 n)$.*

Proof. Consider protocol Oblivious Chordal Wakeup when the set of neighbours to which an entity p sends the WAKEUP messages defines the double-cube, i.e., the chordal ring $R_n[2, 2^2, \ldots, 2^{k-1}]$ described in Example 1. By Theorem 19, the claimed complexity follows. Optimality follows from the lower bound for oblivious protocols of Theorem 17. □

1.6 Further Reading

The aim of this chapter has been to introduce the basics of synchronous computing at the microscopic level, describing some simple but powerful computational and analytical tools.

Notice that, even though everything has been expressed in terms of message-passing, the validity of what we said is independent of whether message transmission to a neighbour and its reception are implemented by using physical communication channels, or by writing to and reading from a predesigned shared register; in the latter case, the microscopic view examines synchronous computations when the size of the registers is limited by a system constant.

The reader interested in knowing more about the microscopic nature of synchrony is referred to the overview material in Chapter 6 of [37], as well as to the significant amount of investigations on the subject.

These investigations cover a wide spectrum of problems and topics, including *election* (e.g., [8, 12, 14, 24, 26, 34, 38, 41]), *extrema finding* (e.g., [1, 24, 36]), *symmetry breaking* (e.g., [13, 15, 20, 27]), *consensus* (e.g., [6]), *communicators* and their use (e.g., [5, 7, 31, 32, 39]), *shortest paths* (e.g., [29]), *unison, firing squad* and *wake-up* (e.g., [4, 11, 16, 19, 30, 33]), *matching* (e.g., [18, 42]). Also relevant are the results in the more powerful CONGEST model, on problem such as *minimum dominating sets* (e.g. [21, 23]), *coloring* and *independent sets* (e.g. [2, 15, 22, 17, 28, 40]), and *minimum-spanning-tree construction* (e.g. [9, 10, 25, 35]). For a recent investigation in a different application area see [43].

Acknowledgements This work has been supported in part by the Natural Sciences and Engineering Research Council (Canada) under the Discovery Grant program.

References

1. Paola Alimonti, Paola Flocchini, and Nicola Santoro. Finding the extrema of a distributed multiset of values. *Journal of Parallel and Distributed Computing*, 37:123–133, 1996.
2. Noga Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4): 567583, 1986.
3. Dana Angluin. Local and global properties in networks of processes. *Proceedings of the 12th ACM Symposium on Theory of Computing (STOC)*, 82-93, 1980.
4. Anish Arora, Shlomi Dolev, and Mohamed Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1(1):11–18, 1991.
5. Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, 1988.
6. Amotz Bar-Noy and Danny Dolev. Consensus algorithms with one-bit messages. *Distributed Computing*, 4(3), 105-110, 1991.
7. Amotz Bar-Noi, Joseph Naor, and Moni Naor. One bit algorithms. *Distributed Computing*, 4(1): 3–8, 1990.
8. Hans L. Bodlaender and Gerard Tel. Bit-optimal election in synchronous rings, *Information Processing Letters* 36(1): 53-64, 1990.
9. Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72 (8): 1282–1308, 2006.
10. Juan Garay, Shay Kutten, and David Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27: 302–316, 1998.
11. Shimon Even and Sergio Rajsbaum. Unison, canon and sluggish clocks in networks controlled by a synchronizer. *Mathematical System Theory*, 28:421–435, 1995.
12. Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring, *Journal of the ACM*, 34: 95-115, 1987.
13. Greg N. Frederickson and Nicola Santoro. Breaking symmetry in synchronous networks, *Proceedings of 2nd International Workshop on Parallel Computing and VLSI (now SPAA)*, 26-33, 1986.
14. Eli Gafni. Improvements in the time complexity of two message-optimal election algorithms, *Proceedings of the 4th Conference on Principles of Distributed Computing (PODC)*, 175-185, 1985.
15. Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4): 434446, 1988.
16. Mohamed Gouda and Ted Herman. Stabilizing unison. *Information Processing Letters*, 35(4):171–175, 1990.
17. Magnús M. Halldórsson and Christian Konrad. Distributed large independent sets in one round on bounded-independence graphs. *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, 559-572, 2015.
18. Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22: 77–80, 1986.
19. Amos Israeli, Evangelos Kranakis, Danny Krizanc, and Nicola Santoro. Time-messages tradeoffs for the weak unison problem. *Nordic Journal of Computing* 4: 317-329, 1997.
20. Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks, *Information and Computation*, 88(1): 60871, 1990.
21. Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4): 193–205, 2002.

22. Kishore Kothapalli, Melih Onus, Christian Scheideler, and Christian Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
23. Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. *Journal of Algorithms*, 28 : 40–66, 1998.
24. Jan van Leeuwen, Nicola Santoro, Jorge Urrutia, and Shmuel Zaks. Guessing games and distributed computations in anonymous networks. *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP)*, 347-356, 1987.
25. Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov and David Peleg. Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.
26. Alberto Marchetti-Spaccamela. New protocols for the election of a leader in a ring. *Theoretical Computer Science* 54(1): 53-64, 1987.
27. Alain Mayer, Yoram Ofek, Rafail Ostrovsky, and Moti Yung. Self-stabilizing symmetry breaking in constant-space. *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, 667-678, 1992.
28. Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, and Akka Zemmari. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing* 23(5-6): 331340, 2011.
29. Yves Métivier, John Michael Robson, and Akka Zemmari. A distributed enumeration algorithm and applications to all pairs shortest paths, diameter. *Information and Computation*, 247: 141151, 2016
30. Yasuaki Nishitani and Namio Honda. The firing squad synchronization problem for graphs *Theoretical Computer Science*, 14: 39-61, 198I.
31. Una-May O'Reilly and Nicola Santoro. Asynchronous to synchronous transformations. In *Proceedings of the 4th International Conference on Principles of Distributed Systems (OPODIS)*, 265–282, 2000.
32. Una-May O'Reilly and Nicola Santoro. Tight bound for synchronous communication of information using bits and silence. *Discrete Applied Mathematics* 129: 195-209, 2003.
33. Rafail Ostrovsky and Daniel S. Wilkersan. Faster computation on directed networks of automata. *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing (PODC)*, 38-46, 1995.
34. Mark Overmars and Nicola Santoro. Improved bounds for electing a leader in a synchronous ring. *Algorithmica* 18: 246-262, 1997.
35. David Peleg. *Distributed Computing: A Locality-Sensitive Approach*, SIAM, 2000.
36. Nicola Santoro. Computing with time: Temporal dimensions in distributed computing. *Proceedings of the 28th Allerton Conference on Communication, Control and Computing*, 558-567, 1990.
37. Nicola Santoro, *Design and Analysis of Distributed Algorithms*, Wiley, 2007.
38. Nicola Santoro and Doron Rotem. On the complexity of distributed elections in synchronous graphs. *Proceedings of the 11th International Workshop on Graphtheoretic Concepts in Computer Science (WG)*, 337-346, June 1985,
39. Bernd Schmeltz. Optimal tradeoffs between time and bit complexity in synchronous rings. *Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science (STACS)*, 1990, 275-284.
40. Johannes Schneider and Roger Wattenhofer Trading bit, message, and time complexity of distributed algorithms. *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, 51-65, 2011.
41. Paul Spirakis and Basil Tampakas. Efficient distributed algorithms by using the Archimedean time assumption. *Informatique Théorique et Applications*, 23(1): 113-128, 1989.
42. Mirjam Wattenhofer and Roger Wattenhofer. Distributed weighted matching. *Proceedings of the 18th International Symposium on Distributed Computing (DISC)*, 335348, 2004.
43. Lei Xu and Peter Jeavons. Simple algorithms for distributed leader election in anonymous synchronous rings and complete networks inspired by neural development fruit flies. *International Journal of Neural Systems*, 25(7), 2015.