# Semisynchrony and Real Time

Extended abstract

Stephen Ponzio[1] and Ray Strong[2]

[1] MIT Laboratory for Computer Science, 545 Technology Sq., Cambridge, MA 02139,
ponzio@theory.lcs.mit.edu
[2] IBM Almaden Research Center, 650 Harry Rd., San Jose, CA 95120-6099
strong@almaden.ibm.com

**Abstract.** This paper represents the confluence of several streams of research on the real time complexity of distributed algorithms. The primary focus of our study is on two models and two problems: the timed automata model of Attiya and Lynch and the ("latency") model of approximately synchronized clocks studied by Strong et. al., and the problems of consensus and atomic broadcast. We compare these models and problems, producing new results and significant improvements of previously known bounds. In particular, we are able to significantly improve the upper bound of Strong, Dolev, and Cristian on latency for Byzantine failures, giving an algorithm that is much simpler with vastly easier analysis. For this problem, we also improve the best known lower bound on latency. We also provide certain reductions between problems and models and provide preliminary answers to some new questions in the timed automata model.

## 1 Introduction

In the interest of obtaining more accurate and useful time bounds for distributed algorithms, there recently has been much attention devoted to deriving time bounds that explicitly account for the degrees of asynchrony that exist in distributed systems. Several different models of semisynchrony have been used, capturing different concerns about issues of real time. Similar but different problems have been studied in these models, yielding quantitative results that are seemingly related. This paper attempts to present a unified view of these research efforts, summarizing the concerns addressed by each. We compare solutions to the general problem of simulating round-based synchronous algorithms and focus on specific solutions to consensus-type problems, which, as the fundamental distributed problems requiring some synchrony, are the natural candidates for the initial stages of this research. By comparing the concerns and approaches in both studies, we have sometimes been able to achieve significant and surprising improvements over existing results. Although we have found that most techniques do not carry over from one setting to the other, understanding the disparity has led to a greater appreciation of what aspects of these two problems are important to different measures of real-time performance. In addition,

our study has revealed some very natural unanwered questions regarding these problems as well as the area of clock synchronization.

For this extended abstract we consider two models of timing in distributed message-passing systems[3]; briefly (see Section 2 for complete definitions), these are

1. The TA ("**T**imed **A**utomata") model: A basic model of semisynchrony, formalized and studied by Attiya and Lynch ([AL89, ADLS90, P91]). Studies in this model focus on the effect of the maximum possible ratio of processor rates, denoted $C$. Message delay time is denoted $d$.
2. The AC ("**A**pproximately synchronized **C**locks") model: A model in which processors are assumed to have approximately synchronized clocks. Studied by Strong, Dolev, and Cristian ([CASD86, SDC90, GSTC90]). Work in this model have focused on the effect of the maximum difference $e$ between clocks ("skew" or "precision"). The maximum possible ratio of clock rates is denoted $A$ and the message delay is denoted $d_{AC}$.

In each of the models, it is easy to simulate arbitrary synchronous round-based algorithms by allowing the maximum possible time for each round. However such straightforward simulations are generally inefficient. The motivating question is

**Can one do better than directly simulating round-based algorithms?**

Natural vehicles to explore time complexities are the fundamental fault-tolerant problems of consensus and atomic broadcast. We distinguish atomic broadcast from consensus in two ways: consensus is *"multi-source"* and *one-time only*—each processor gets exactly one input value and the output is a single value; atomic broadcast is *"single-source"* and *dynamic*—each processor may get input values repeatedly and the output is a sequence of these values, which must be identical for each processor. The consensus problem has been studied in the TA model ([ADLS90, P91]) and the atomic broadcast problem has been studied in the AC model ([CASD86, SDC90, GSTC90, BGT90]).

### 1.1 Our results

We apply algorithmic techniques used for consensus (with Byzantine failures) in the TA model to obtain a greatly improved algorithm for atomic broadcast (with Byzantine failures) in the AC model. The best previous algorithm ([SDC90]) for atomic broadcast in the presence of Byzantine failures had "latency"[4] $2e+3(2+A+A^2+\cdots+A^f)d_{AC}$, where $f$ is the number of Byzantine processor failures to be tolerated. We adapt an algorithm from [P91] to obtain a vastly simpler algorithm with much simpler analysis and an improved latency of $2e+((1+2A)f+1)d_{AC}$.

[3] In the full paper, we also include comparisons with the related model studied by Herzberg and Kutten ([HK89]).

[4] A measure of time complexity defined in Section 2.4.

We also prove that, even for more benign failures (such as authenticated Byzantine or clock failures with send and receive omissions), a lower bound for latency is $2e + (f + 1)d_{AC}$. This improves on the previously best known lower bounds of $2e + 2d_{AC}$ ([SDC90]) and $e + (f + 1)d_{AC}$ ([CASD86]).

Although there is a vast literature on the problem of atomic broadcast (e.g., [CM84, BJ87, MMA90, MMA91, ADKM92]), we know of no work that focuses on the real time complexity of this problem when processors are not fully synchronous. Surprisingly, there is no simple algorithm for solving atomic broadcast in the TA model (even inefficiently). We consider implementing synchronized clocks in the TA model as one way of solving the atomic broadcast problem. Unfortunately, many important clock synchronization algorithms such as [ST87], [DHSS89] and [LL88] were designed only for systems with extremely small drift ($C \approx 1 + \epsilon$); it is not clear whether these algorithms can be extended to work for the case we are interested in, when $C$ is large. [5] We also adapt a lower bound proof of [ST87] to show that $A \geq C$ for any clocks implemented in TA.

Finally, we derive several simple reductions between the problems and models. They relate latency of atomic broadcast with the real time required to achieve consensus. We first show that if there is an atomic broadcast protocol with latency $L$, then there is a consensus protocol for the AC model that requires at most real time $R \approx (L + 2e)/\sqrt{A}$. We also show that if there is a consensus protocol that requires at most real time $R$, then there is an atomic broadcast protocol with latency $L \approx \sqrt{A}R + d_{AC} + e$.

## 2 Models, problems, and discussion

Consensus-type problems are the most natural candidate to study in a semisynchronous model: their time complexity is well understood in the case of synchronous round-based computation; they are well known to be impossible for completely asynchronous systems, and the necessary degrees of synchrony have been thoroughly studied ([DDS87]). However, the different models of semisynchrony have inspired the study of different versions of the consensus problem. We begin by describing the two models in more detail. In both models, processors are completely connected by reliable message links and all parameters are known to the processors. We consider the standard failure modes; unless otherwise stated, "omission" failures refers "send-omission" failures only.

### 2.1 Model TA: A basic model of semisynchrony

A basic model of semisynchrony is developed in the work of Attiya and Lynch ([AL89]), based on the timed automaton model ([MMT90]). Conceptually, the model is very simple: successive steps of a nonfaulty process are separated by at least time $c_1$ and at most $c_2$ and all messages sent are delivered within time $d$.

[5] The question of whether $e$ and $A$ can be simultaneously minimized—$e \approx d$ and $A = C$—is a long-standing open question in the area of clock synchronization.

Although these constants are common knowledge among the processors, a processor cannot directly determine the exact time between any two particular steps. A "step" of a process consists of performing some local computation and sending messages to other processors. Messages may be delivered to a processor between two of its steps. Processors are assumed to obey the timing constraints if they suffer omission failures but not if they suffer Byzantine failures. (An interesting but unstudied alternative model of failure is "timing" failures, where processors act correctly except that they may violate the step-time constraint. The most efficient algorithm known for this class of failures is the algorithm for Byzantine failures.)

In this model, the ratio $c_2/c_1$ is used as a measure of the timing uncertainty and denoted simply $C = c_2/c_1$. This parameter measures the rate of drift between processors. Processor steps are typically much faster than message transmission, so we usually consider $c_2 \ll d$ and make approximations appropriately. An essential factor in the running time of a round-based simulation is the time required to timeout the message of another processor. We therefore first outline why a timeout may take up to time $Cd + d$ in this model. Suppose processors implement fault-detection by continuously sending "I'm alive" messages to each other, so that $d$ is approximately an upper bound on the time between the delivery of any two successive messages. If $q$ fails to send a message to $p$ at time $t$, $p$ will begin to notice an absence of messages at time $t + d$. Processor $p$ concludes that $q$ has failed when it is sure that time $d$ has elapsed since the last message received. It can only conclude that time $d$ has elapsed by waiting for $d/c_1$ steps, which may take up to time $t + d + c_2(d/c_1) = t + d + Cd$. Thus we see that although it takes only time $d$ to receive a message, it may take up to time $Cd + d$ to *detect the absence* of a message.

Any round-based algorithm may then be simulated despite stopping or omission failures by continuously performing this timeout protocol between every pair of processors. Each processor simulates round $i$ by waiting until for each processor $q$, $p$ has either received a round $i - 1$ message from $q$ or has detected the failure of $q$. Each round then takes approximately time $Cd + d$ to simulate. The goal of the work of [ADLS90, P91] is to quantify the effect of semisynchrony on the real-time complexity of distributed computing problems: given a system with parameters $c_1, c_2, d$, what are tight upper and lower bounds on the *real time* required for these problems?

## 2.2 Model AC: Approximately synchronized clocks

A higher-level model of semisynchrony in the spirit of the work on clock synchronization has been studied by Dolev, Strong, and Cristian ([SDC90]). Each processor has a clock that stays within a linear envelope of real time: there exist positive constants $a_1 \leq 1 \leq a_2$ and $a_3$ such that for each clock of a correct processor and for all real times $t_1 < t_2$,

$$a_1(t_2 - t_1) - a_3 \leq Clock(t_2) - Clock(t_1) \leq a_2(t_2 - t_1) + a_3.$$

Clocks of correct processors never differ by more than $e$. This is a discretized version of the standard model of clocks that has been used throughout the literature on clock synchronization (e.g., [DHSS89, DHS86, LM85]). Processors are interrupt-driven: they may be caused to take a step either by the arrival of a message or by its clock reaching a prespecified time. As in the TA model, a processor may send messages to several other processors during one step. Also, processors are assumed to obey the timing constraints if they suffer omission failures but not if they suffer Byzantine failures.

In the AC model, the maximum delay of any message is defined in terms of clock time: the interval between the sending and delivery of any message measures at most $d_{AC}$ on the *clock* of *any* correct processor. We use the subscript $AC$ to distinguish this term from $d$, defined in the TA model, which we retain to denote maximum *real time* between sending and delivery. To put the drift assumption into a more usable form, we first note that if an interval is timed to be of length $t$ on the clock of a correct processor, then it measures at most $(a_2/a_1)t+2a_3 \approx At$ on the clock of any other correct processor (take $t_2-t_1 = t/a_1$ in the above definition). As with the TA model, we will generally assume that the granularity of the clocks is much less than message delay—$a_3 \ll d_{AC}$—and make appropriate approximations. We denote $A = a_2/a_1$ and call this quantity the *relative accuracy*[6].

How can synchronous round-based algorithms be simulated in this model? Because clocks are available, timing out other processors is a simple matter: if $p$ is supposed to send a message to $q$ at time $t$ on its clock, then $q$ knows that the message should be sent no later than time $t + e$ on its own clock and therefore should be received no later than time $t + e + d_{AC}$ on its clock. To simulate a round-based algorithm starting at clock time $t$, each processor waits until time $t + i(d_{AC} + e)$ on its clock to receive round $i$ messages and then sends its round $i + 1$ message.

### 2.3 Consensus

This version was studied in [ADLS90, P91]. It is the standard classical binary consensus problem: each processor has a one-bit input and all correct processors must agree on a one-bit output which is equal to the input if all inputs are equal.

Because each processor is supposed to receive an input for the problem, it makes the most sense to assume that these inputs arrive within some known time interval. (In the synchronous round-based model, processors are assumed to begin executing the algorithm at the same time.) We therefore introduce a parameter $x$ to denote the length of the interval of real time in which all processors receive their inputs. We measure running time as the difference between the real time at which the last correct processor decides on a value and the real time at which the first correct processor gets its input. Note that this definition applies to all failure models. (The algorithms from [ADLS90] and [P91] work for $x > 0$ with little or no modifications.)

[6] This ratio is equivalent to "$(1+\rho)^2$" in [ST87, LL88, DHSS89] and "$1+\rho$" in [SDC90].

## 2.4 Atomic broadcast

This version was studied in [CASD86, SDC90, GSTC90]. It is a dynamic problem in the sense that inputs arrive repeatedly and asynchronously. At any time, a processor may receive a binary input which must be broadcast to all other processors. Processors must output a sequence of values such that (1) all correct processors output the same sequence of values, and (2) the input sequence of each correct processor appears as a distinct subsequence of this sequence. Note that this definition allows for the possibility that processors may agree on a value different from the sender's input even if the sender suffers only stopping or omission failure. When a processor (irreversibly) adds a message to its list, it is said to *deliver* the message.

A natural definition of real-time complexity for this problem is to measure the difference between the time that a processor gets an input and the time the last correct processor delivers the message. This definition is workable for omission failures, but it is not meaningful if a Byzantine processor delays acting on its input and then correctly executes the broadcast algorithm on that input; in this case the time cannot be bounded.

In [CASD86], this difficulty is resolved for the AC model by defining a time complexity measure called the *latency*. This measurement requires as part of the *problem statement* that when a processor initiates a message, it should attach its local time to the message.

**Definition 1.** The *latency* of an algorithm for atomic broadcast is the maximum difference (over all executions, processors, and messages) between the *local clock time* that a correct processor delivers the message and the *timestamp* on that message.

Thus the algorithms developed for atomic broadcast in the AC model are concerned not with minimizing the elapsed real time, but with minimizing the age of any message (as defined by its timestamp) that must be accepted by a processor, relative to its current clock time. Although this measure may seem unnatural at first, it does have the advantage of being directly observable by processors. Note that for stopping or omission failures, the latency is equal to $e$ plus the maximum time that can elapse on a processor's clock between the real time of the input and the real time that the processor delivers the corresponding message. Of course, this is not true for Byzantine failures, as clocks of faulty processors need not be within $e$ of each other.

## 2.5 Previous work

Work on consensus in the TA model has focused on the extent to which the drift, or timing uncertainty, $C$, affects the real-time complexity. A straightforward rounds simulation (for omission failures) requires time approximately $Cd$ per round. Interesting new algorithms were developed with running times of $2fd+Cd$ for stopping failures ([ADLS90]) and $4(f+1)d+Cd$ for omission failures and $(2f+1)Cd+fd$ for Byzantine failures ([P91]).

Work on atomic broadcast in the AC model, however, has focused on the extent to which the clock skew $e$ affects the running time; the effect of drift ($A$) has not been a primary concern of this research. A straightforward rounds simulation may require clock time $d_{AC} + e$ per round. It is easy to show that this is not optimal for stopping and omission failures; a simple message-diffusion algorithm gives a total latency of $(f+1)d_{AC} + e$ ([CASD86]). However, a great deal of effort was needed to achieve a latency of $3(2 + \sum_{i=0}^{f} A^i)d_{AC} + 2e$ for Byzantine failures ([SDC90]).

From looking at the results above, it is tempting to infer some kind of relationship between the additive factor of $Cd$ which the TA bounds minimize and the additive factor of $e$ which the AC bounds minimize. However, we shall see that no such relationship exists.

# 3 Improved latency bounds for atomic broadcast

## 3.1 Round simulations

We first consider the general problem of simulating synchronous rounds. For simplicity, we will assume that all processors begin a TA rounds simulation at the same real time and an AC rounds simulation at the same clock time. We saw in Section 2 that rounds may be simulated in the AC model at a cost of $d_{AC}+e$ elapsed clock time per round (for all failure models) and in the TA model at a cost of $Cd + d$ real time per round (for stopping or omission failures) or $(2C+1)d$ per round (for Byzantine failures). In neither model does the respective round simulation yield an efficient algorithm for consensus or atomic broadcast (except for the Byzantine consensus algorithm in the TA model, which is not known to be suboptimal—i.e., it is not known whether $O(fd) + Cd$ is sufficient or if $\Omega(fCd)$ is required).

We consider adapting rounds-simulation algorithms of the TA model to work in the AC model. The algorithms for the TA model use the bounds on step time exclusively for deriving upper bounds on elapsed time—for instance, counting enough steps to ensure that time $d$ has passed after sending a message in order to be sure that it has been delivered. These algorithms can thus be used in the AC model with little change by instead using the clocks to give such guarantees—for example, if a processor waits for time $d_{AC}$ on its clock after sending a message, it ensures that the message must be delivered because at most time $d_{AC}$ can elapse on *any* correct clock while the message is in transit. Thus the clocks are used as "timers" to measure the length of intervals, and their synchronization–that they are within $e$ of each other—is ignored.

The rounds simulation for omission failures in the TA model described in Section 2.1 uses real time $(C+1)d$ per round. To analyze the adapted algorithm in the AC model, we must ask how much any clock may advance during a "round". As with the analysis in the TA model, the worst case is when a single processor fails just before sending its round message; this causes the other processors to wait for $d/c_1$ steps, or $Cd$ time, before concluding that a failure has occurred. In

the AC model, this failure leads to a worst-case latency if a processor with a fast clock quickly concludes that a failure has occurred while another processor with a slow clock takes longer to reach that conclusion; on the slower clock, time $d_{AC}$ elapses while on the faster clock, time $A \cdot d_{AC}$ elapses. By a similar argument as in Section 2.1, we see that some clock may advance $(A+1)d_{AC}$ each round. We note that the worst-case execution in the TA bound is with all processors going fast, whereas in the AC bound, it is with one clock going slow and another going fast.

Thus the adapted simulation is successful in avoiding an additive $e$ with each round, improving in that respect on the first simulation described for the AC model. It suffers, however, from the $A+1$ factor of $d_{AC}$. Suppose we use this round simulation to run a standard atomic broadcast algorithm for omission failures (assuming a common clock start time). Simulating $f+1$ rounds gives a total latency of about $(f+1)(A+1)d_{AC}+e$, as a faster clock may have started out $e$ ahead of that of the processor receiving (and timestamping) the input. This fails to improve the latency of the simple message-diffusion algorithm of [CASD86].

Indeed, even if we translate the efficient consensus algorithm of [ADLS90], the resulting latency is $2fd_{AC}+Ad_{AC}+e$, which is also worse than [CASD86]. We remark that the algorithm of [ADLS90] can be viewed as an optimized simulation of a synchronous early-stopping consensus algorithm with a special property regarding the circumstances under which a processors must advance to successive rounds (see [P91]). This suggests that it may be possible to identify a class of efficiently simulatable synchronous algorithms whose simulations need incur neither the $e$ per round nor the $A+1$ factor of $d_{AC}$.

The algorithm for consensus with Byzantine failures in the TA model [P91] works by simulating synchronous rounds efficiently (relative to naive strategies). In Section 3.2 below, we show that this simulation, which uses time $(2C+1)d$ per round, may be adapted to the AC model so that any clock advances at most $(2A+1)d_{AC}$ per round. However, because the clocks of Byzantine processors may differ from correct clocks by more than $e$, the total latency for simulating $f+1$ rounds turns out to be $(f+1)(2A+1)d_{AC}+2e$ (instead of plus $1e$). Surprisingly, this is far better than the latency bound of [SDC90] (modulo the synchronized start assumption).

We see that except for this assumption the adaptation of the TA rounds-simulation improves the atomic broadcast latency bound for Byzantine failures but not for stopping or omission failures. We can now see that the differing factor of $e$ for omission and Byzantine failures (1 and 2, respectively) is precisely due to the difference on the clocks at the beginning of the algorithm.

### 3.2 The algorithm

The following algorithm simulates synchronous rounds despite Byzantine failures, under the assumption that it is common knowledge that the input message should be timestamped $T$. The algorithm uses the synchronized clocks to wait

for round one messages (this is where the additive $2e$ is incurred) and then relies only on the rates of the clocks for the rest of the algorithm. All times are measured on local clocks.

---

1a. **Wait** until time $T + d_{AC} + e$ **or** until $f + 1$ round 2 messages received
1b. **Send** round 2 message

2a. **Wait** until $2f + 1$ round 2 messages received
2b. **Wait** for time $2d_{AC}$ **or** until $f + 1$ round 3 messages received
2c. **Send** round 3 message

$\vdots$

$(i-1)$a. **Wait** until $2f + 1$ round $i - 1$ messages received
$(i-1)$b. **Wait** for time $2d_{AC}$ **or** until $f + 1$ round $i$ messages received
$(i-1)$c. **Send** round $i$ message

$\vdots$

$r$a. **Wait** until $2f + 1$ round $r$ messages received ;the last round
$r$b. **Wait** for time $2d_{AC}$
**END**

---

**Theorem 2. (Correctness)** *For $n \geq 3f+1$, the above simulation ensures that each correct processor receives round $i-1$ messages from all correct processors before sending its round $i$ message.*

*Proof.* First note that because $n \geq 3f+1$, processors will eventually advance to all rounds of the simulation. It is clear that a processor does not send its round 2 message before receiving a round 1 message from all correct processors: consider the first correct processor to send a round 2 message. It cannot receive $f + 1$ round 2 messages before it sends, so it must wait until $T + d_{AC} + e$ on its clock before sending. Clearly, all round 1 messages of correct processors are delivered by this time. All other correct processors send their round 2 messages later.

In subsequent rounds, when the first correct processor $p$ sends its round $i$ message, the round $i-1$ messages of all correct processors have been delivered: Because $p$ is the first correct processor to send its round $i$ message at $(i-1)$c, it could not have received $f + 1$ round $i$ messages before then and therefore must have waited for a period of $2d_{AC}$ on its clock after it received $2f+1$ round $i-1$ messages. After $p$ has waited $d_{AC}$, all correct processors have received at least $f + 1$ of those messages and therefore, by the code, they must have sent round $i-1$ messages (they must already be at least to $(i-2)$b, since they have each sent a round $i-2$ message to $p$ by the induction hypothesis and then advanced to $(i-2)$a, and subsequently received at least $2f+1$ round $i-2$ messages from each other). These round $i-1$ messages are received by all processors within another time $d_{AC}$ on $p$'s clock, which is when $p$ sends its round $i$ message. □

To tolerate Byzantine failures with authentication and $n \geq 2f + 1$, simply change "**Wait** until $2f+1$ round $i-1$ messages received" to "**Relay** $f+1$ round

$i-1$ messages." Thus a processor $p$ executing this statement ensures that all other correct processors will send their $i-1$ messages within $2d_{AC}$ because the signed relayed messages satisfy $(i-2)$b.

**Theorem 3. (Latency)** *The latency for simulating a synchronous algorithm of $f+1$ rounds in the presence of Byzantine failures is $((1+2A)f+1)d_{AC}+2e$.*

*Proof.* For a given execution define
- $C_p(t)$ = the value of processor $p$'s clock at real time $t$,
- $t_i$ = the latest real time at which a correct processor sends a round $i$ message,
- $t_i^{del}$ = the latest real time at which the round $i$ message of any correct processor is delivered.

So we have $C_p(t_2) \leq T + d_{AC} + 2e$ for all correct $p$, since every correct processor sends a round 2 message by time $T + d_{AC} + e$ on its clock, at which time the clock of any other correct processor reads at most $(T + d_{AC} + e) + e$.

By induction on the round number $i \geq 2$, we show $C_p(t_{i+1}) - C_p(t_i) \leq d_{AC} + 2d_{AC}A$: First note that $C_p(t_i^{del}) - C_p(t_i) \leq d_{AC}$ for all correct $p$, by the definition of $d_{AC}$. Now consider the last correct processor $q$ to send a round $i+1$ message (at time $t_{i+1}$). It receives a round $i$ message from each correct processor by real time $t_i^{del}$ and sends its round $i+1$ message no more than $2d_{AC}$ on its clock thereafter, so we have $C_q(t_{i+1}) - C_q(t_i^{del}) \leq 2d_{AC}$. As we showed in Section 2.2, this implies that $C_p(t_{i+1}) - C_p(t_i^{del}) \leq A(2d_{AC})$ for all correct $p$.

Summing over rounds 2 through $f+2$ (the processors END at "$t_{f+2}$"), we have $C_p(t_{f+2}) - C_p(t_2) \leq f(1+2A)d_{AC}$ for all correct processors $p$.

Because a processor knows the initial message was scheduled to be sent at time $T$, it need not deliver a message with timestamp older than $T$, and the latency of the algorithm is

$$C_p(t_{f+2}) - T \leq C_p(t_{f+2}) - (C_p(t_2) - d_{AC} - 2e) \leq (1+2A)fd_{AC} + d_{AC} + 2e$$

□

**Removing the synchronized start assumption.** A simple but message- and computation-inefficient way to remove the assumption that processors know the starting time is for the processors to execute the broadcasts as if an input were known to be received every $\epsilon$ time on their clocks, using "dummy" messages if they have received no input. When a processor gets an input, it sends the initial message with the beginning of the next scheduled execution of the simulation. The total latency is then $\epsilon + ((1+2A)f+1)d_{AC} + 2e$, for any $\epsilon$.

A less wasteful way to remove this assumption is to use a clever protocol developed in [BGT90] to synchronize the starting round of an agreement algorithm.[7] This protocol adds $3(e+d)$ to the latency. When a processor receives an input $m$ at local time $t$, it broadcasts a message "start: $t$" announcing that an execution of the broadcast algorithm will begin at clock time $t + 3(d_{AC} + e)$. At that time, the processors will execute $n$ atomic broadcast algorithms in parallel

[7] One could use the "firing squad" algorithms for this purpose, but they require $f+3$ rounds, whereas this technique requires only an additional three.

as each processor broadcasts a vote (the original processor broadcasts $m$ along with its vote). That is, the broadcast algorithms are executed by each processor as if it knew that every processor were scheduled to receive an input (which is actually its vote) at clock time $t + 3(d_{AC} + e)$. The vector of votes produced by the broadcasts determines whether $m$ is delivered by the processors. Any processor that receives a "start: $t$" message by $t + d_{AC} + e$ on its clock and relays the message to everyone and participates in the broadcasts with a vote of YES. Any processor that receives a (possibly relayed) "start: $t$" message by time $t + 2(d_{AC} + e)$ on its clock (but not by $t + d_{AC} + e$) relays the message to everyone and participates in the broacasts with a vote of NO. In either case, if the broadcasts produce a vector of votes with at least $f + 1$ YES's, then these processors deliver $m$ iff they would deliver $m$ according to the atomic broadcast algorithm corresponding to the originator. However, a processor that does not receive a relayed message by time $t+2(d_{AC}+e)$ participates in the broacasts (to the best of its ability—depending upon when it first hears about the broadcasts) but does *not* deliver $m$ as a result of the broadcasts. The claim is that despite the fact that the original atomic broadcast algorithm is guaranteed to work only if all correct processors participate, with the addition of this protocol a correct processor delivers $m$ if and only if all correct processors deliver $m$.

**Claim 1** *For any atomic broadcast algorithm that is correct when the clock time of input is common knowledge, the protocol above ensures that even without this common knowledge a correct processor delivers $m$ if and only if all correct processors deliver $m$.*

*Proof.* Suppose a correct processor $p$ delivers $m$. Then $p$ must have received a "start: $t$" message by time $t+2(d_{AC}+e)$ on its clock and therefore all processors received a "start: $t$" message by time $t + 3(d_{AC} + e)$ on their clocks. Thus, all correct processors participated in the entire broadcasts and the same vector of votes is therefore produced at each processor. In particular, all correct processors agree on whether or not $m$ was received. Now, for $p$ to deliver $m$, at least $f+1$ of those votes must be YES, so some correct processor received a "start: $t$" message by time $t + d_{AC} + e$ on its clock. It follows that each processor receives a "start: $t$" message by time $t + 2(d_{AC} + e)$ on its clock and therefore delivers $m$ as a result of seeing $f + 1$ YES's. □

### 3.3 Optimal precision

In this section, we prove a lower bound on the latency of the atomic broadcast problem of $(f + 1)d_{AC} + 2e$ if processors fail by omitting messages and having clocks that differ from correct clocks by more than $e$. By improving over the previously best known lower bounds of $2e + 2d_{AC}$ ([SDC90]) and $e + (f + 1)d_{AC}$ ([CASD86]), we obtain the first lower bound that is tight (with the Byzantine upper bound) both precisely in the factor of $e$ and to within a "constant" factor in its coefficient of $d_{AC}$. The "constant" factor is in fact equal to about twice the drift rate $A$; it remains a major open question in this area to obtain lower bounds that depend substantially on the drift ($A$ or $C$).

**Theorem 4.** *Any algorithm for atomic broadcast in the AC model tolerating send and receive omission failures and clock failures has latency at least* $(f + 1)d_{AC} + 2e$.

*Proof.* Let $Q = \{2, \ldots, n - f\}$ and $R = \{n - f + 1, \ldots, n\}$ be subsets of the processors. For the purposes of the proof, the rate of clocks and all absolute readings are unimportant. We will assume throughout the proof all clocks run at the rate of real time, with 1's clock displaying real time exactly, $Q$'s clocks reading $e$ greater than real time, and $R$'s clocks reading $2e$ greater than real time. Processor 1 receives an input message $x$ at real time 0. All times and intervals in the proof refer to real time unless otherwise specified. We use $d = d_{AC}$.

Let $E_0$ be an execution in which (1) for all $k$, any message sent in the interval $[(k-1)d, kd)$ is delivered at time $kd$, and (2) processor 1 acts as if it has done everything correctly, but it omits $x$ to $R$. By delivering messages only at multiples of $d$, we can identify each interval $[(k-1)d, kd)$ with a "round" in the natural way. Although processor 1 is clearly faulty in $E_0$ from the point of view $R$, to $Q$ it is equally possible that all processors in $R$ (which number $f$) received $x$ but are claiming otherwise. Note that $Q$ may also be able to discern that either 1 or $R$ must be faulty by discovering that their clocks differ by more than $e$. Now, if 1 sends follows the algorithm with respect to $Q$, then processors in $Q$ must deliver it, despite what 1 says to $R$. To ensure agreement, correct processors in $R$ must deliver $x$ if those in $Q$ deliver it. Thus, in $E_0$, all correct processors deliver $x$.

We can now mimic the argument of the synchronous lower bound ([DS83, DM86, M85, CD86]) of creating a chain of executions $E_0, \ldots, E''$ such that in $E''$ no input is received by processor 1. Each pair of successive executions is indistinguishable to some correct processor in $R$ before time $T + (f+1)d + 2e$ on its clock. All correct processors must deliver $x$ in $E_0$ but not in $E''$, so some pair of executions must be distinguishable to all correct processors by time $T$ plus the latency on their clocks. Thus, the latency must be at least $(f+1)d+2e$.

Each successive pair of executions differ only in the existence of a single message. Clearly, if a message $m$ is sent in the interval $[(f-1)d, fd)$, then only the recipient can tell by time $fd$ if $m$ has been sent or not. Since subsequent messages sent by the recipient are not delivered until time $(f+1)d$, processors in $R$ cannot tell before time $(f+1)d+2e$ on their clocks if $m$ has been sent or not. Thus, only one processor in $R$ (if it is the recipient of $m$) can distinguish before local time $(f+1)d+2e$ between two executions that differ only in whether or not $m$ is sent in the interval $[(f-1)d, fd)$.

Starting with execution $E_0$, for any processor $p$ we may construct a sequence of executions $E_0, \ldots, E'$ such that $p$ sends no messages at all in the interval $[(f-1)d, fd)$ of $E'$ and each pair of successive executions is indistinguishable to some correct processor in $R$. This is done by removing one at a time each message sent by $p$ in the interval. Another execution created by removing a message sent by $p'$ to $p$ in the interval $[(f-2)d, (f-1)d)$ is then clearly indistinguishable from $E'$ to all processors but $p$ (since $p$ sends no messages in $E'$ after time $(f-1)d$). Now, by adding back one at a time the messages sent by $p$ in the interval $[(f-1)d, fd)$, we can continue the sequence to arrive at an execution that differs

from $E_0$ only in the message from $p'$ to $p$. In this manner, we can remove any messages of up to $f-1$ processors in addition to processor 1. This is easily proved formally with a (standard) recursive proof (see [DS83, CD86, M85, DM86]). We finally arrive at an execution $E''$ in which processor 1 omits $m$ to all processors, completing the proof.

The key fact is that this sequence of executions leading to $E''$ has the property that each consecutive pair is *indistinguishable to some correct processor in* $R$. This is because the recursion requires that for each execution, at most $i$ processors fail in the first $i$ rounds. Since processors in $R$ don't receive $m$ directly from 1 by time $d$, there is no need to remove any messages sent by processors in $R$ before time $2d$; therefore at most $f-2$ processors in $R$ are faulty in any pair of successive executions. At most one processor can distinguish between any pair of successive executions, leaving us at least one in $R$ that cannot. □

### 3.4 Real-time bounds for atomic broadcast

In addressing the atomic broadcast problem in the TA model, we encounter two problems. The first is that new techniques are needed to establish a common ordering of messages—atomic broadcast algorithms for the AC model establish this ordering by delivering messages in timestamped order, making critical use of the synchronized clocks. The second problem, discussed in Section 2.4, is with defining the running time for Byzantine failures. In this section, we avoid Byzantine failures altogether and take solve the ordering problem by simply implementing synchronized clocks in the TA model.

The extensive literature on clock synchronization has thoroughly studied almost exactly this problem of implementing clocks—given "hardware" clocks that drift from real time at some rate bounded by a constant, implement logical "software" clocks that drift from real time as little as possible and are also within some constant ("skew") of each other. Unfortunately, to the best of our knowledge, all clock synchronization algorithms (e.g., [ST87], [DHSS89], [LL88]) were designed only for assume extremely small rates of drift in the "hardware" clocks ($C \approx 1+\epsilon$).

We can implement a "hardware clock" as a counter that increments by $\sqrt{c_1 c_2}$ at each step. An interval with $k$ steps is measured to be of length $\ell = k\sqrt{c_1 c_2}$ on the hardware clock but its real time may be as little as $kc_1 = \ell/\sqrt{C}$ or as great as $kc_2 = \sqrt{C}\ell$. Thus we essentially get a drift of $\sqrt{C}$ relative to real time; the relative accuracy is $C$. The clock synchronization algorithm of Srikanth and Toueg ([ST87]) preserves this drift in the logical clocks and gives a worst-case skew of $\sqrt{C}(2C+1)d$ (c.f. their expression for $D_{\max}$ on p. 631, with their $(1+\rho)$ equal to $\sqrt{C}$ and their $d_{\min} = 2t_{\text{del}}$ equal to our $2d$).

Using the synchronized clocks, we can run the message diffusion algorithm ([CASD86]) with latency $L = (f+1)d_{AC} + e$. Because an interval of $t$ units of real time may be measured as $\sqrt{C}t$ on a fast clock, we conclude that the delay $d$ of any message measures at most $\sqrt{C}d$ on the clock of any correct processor; this is "$d_{AC}$". Because any clock reads at least $T-e$ when the input is received at time $t$, the maximum elapsed clock time from $t$ to the last delivery

is $L + e$. The maximum elapsed real time is at most a factor of $\sqrt{C}$ greater: $\sqrt{C}\left[(f+1)\sqrt{C}d + 2\left(\sqrt{C}(2C+1)d\right)\right] = (f+1)Cd + 2C(2C+1)d$. This gives

**Theorem 5.** *For sufficiently small values of $C$, there is an algorithm for atomic broadcast tolerating omission failures with real time complexity at most $(f+1)Cd + 2C(2C+1)d$.*

It is not obvious how to achieve such a running time without explicitly implementing clocks. Algorithms are known to solve atomic broadcast without clocks; it would be interesting to investigate their real time complexities.

As might be expected, the relative accuracy of $C$ achieved above is optimal; it is not difficult to adapt a lower bound of Srikanth and Toueg ([ST87]) to show

**Theorem 6.** *For any logical clocks implemented in the TA model, the relative accuracy $A$ must be at least $C$.*

This theorem shows that blind syntactic translation of maximum elapsed clock time to maximum elapsed real time will always incur a factor of $C$. Clocks measure message delay to be at most $a_2 d + a_3 = d_{AC}$ and if elapsed clock is expressed as $g(n, f, e, A)d_{AC}$ for some function $g$ then an upper bound on the elapsed real time is $\approx \frac{1}{a_1} g(n, f, e, A)a_2 d = Ag(n, f, e, A) \geq Cg(n, f, e, A)$. However, it may be that the elapsed real time can be better bounded by more carefully examining the possible executions of an algorithm.

### 3.5 Reductions

In this section, we work exclusively within the AC model, converting back and forth between real time bounds for consensus and latency bounds for atomic broadcast.

**Theorem 7.** *If there is an atomic broadcast protocol with latency $L$, then for each $x \geq 0$ there is a consensus protocol for the AC model with start interval length $x$ and real time from start to finish bounded above by $\approx x + (L + 2(e + x_{AC}))/a_1$, where $x_{AC} \approx a_2 x$.*

Of course, the idea is to have each processor use the atomic broadcast algorithm to send its value to all other processors. Note that a trivial algorithm of merely adopting the first message delivered is incorrect by our definition of atomic broadcast: if the sender is faulty, the atomic broadcast algorithm may cause processors to deliver a value different from the sender's input, possibly violating the validity condition of consensus. For $n \geq 3f + 1$, it is sufficient for processors to wait for the first $2f + 1$ messages delivered and decide on the majority of those; this gives a real time of $(L+e)/a_1$. Our theorem is interesting for $f < n \leq 3f$. The problem is how to ensure that all processors resolve the vector of delivered messages in the same way. In particular, we need to ensure that if a faulty processor starts its broadcast too late, then all correct processors either include that value in their vector or not. To solve this problem, we use a technique developed in studies of the AC model, of reducing the vector of delivered

messages to a subset that are "believable" in the sense that for each one, there are enough other messages with timestamps inside a small enough interval.

Finally, we have the following simple theorem which we state without proof.

**Theorem 8.** *If there is a consensus protocol in AC with* $x \geq (e + a_3)/a_1$ *and with real time from start to finish bounded above by* $R$*, then there is an atomic broadcast protocol in AC with latency bounded above by* $L \approx a_2 R + e + d_{AC}$.

## 4 Directions for further research

- Is there an algorithm for multi-source consensus with Byzantine failures in the TA model, assuming synchronized start, that runs in time $o(fCd)$? $O(fd)$? Same question but for 'timing" failures (see Section 2.1)?
- What are good bounds for the real time complexity of atomic broadcast in the TA model?
- How well can clocks be synchronized for very inaccurate "hardware" clocks ($C > 3/2$)?
- Can the algorithm for atomic broadcast in the AC model presented in section 3 be generalized for authenticated Byzantine failures with $n \leq 2f$ to give an algorithm running in time with $\approx \frac{2n}{n-f}$ as the coefficient of $e$? (See [SDC90].)

## 5 Acknowledgments

We thank Faith Fich for her comments.

## References

[ADKM92] Y. Amir, D. Dolev, S. Kramer and D. Malki. Total ordering of messages in broadcast domains. Manuscript.

[ADLS90] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. MIT/LCS/TM-435, November 1990. Also: STOC 1991.

[AL89] H. Attiya and N. A. Lynch. Time bounds for real-time process control in the presence of timing uncertainty. *Proc. 10th IEEE Real-Time Systems Symposium,* 1989, pp. 268–284. Also: MIT/LCS/TM-403, July 1989.

[BJ87] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM TOCS*, Vol. 5, No. 1 (February 1987), pp. 47–76.

[BGT90] N. Budhiraja, A. Gopal and S. Toueg. Early-stopping distributed bidding with applications. *Proc. 4th Int'l. WDAG* 1990.

[CASD86] F. Cristian, H. Aghili, R. Strong and D. Dolev. Atomic broadcast: from simple message diffusion to Byzantine agreement. *Proc. 15th Int. Conf. on Fault Tolerant Computing,* 1985, pp. 1–7. Also: IBM Research Report RJ5244, revised October 1989.

[CM84] J. M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM TOCS*, Vol. 2, No. 3 (August 1984), pp. 251–273.

[CD86] B. A. Coan and C. Dwork. Simultaneity is harder than agreement. *Information and Computation* Vol. 91, No. 2, 1991.

[DDS87] D. Dolev, C. Dwork and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *JACM*, Vol. 34, No. 1 (1987), pp. 77–97.

[DHS86] D. Dolev, J. Y. Halpern and R. Strong. On the possibility and impossibility of achieving clock synchronization. *JCSS*, Vol. 32, No. 2, 1986, pp. 230–250.

[DHSS89] D. Dolev, J. Halpern, R. Stong and B. Simons. Dynamic fault-tolerant clock synchronization. IBM Research Report RJ 6722, March 1989. Also: Fault-tolerant clock synchronization. *Proc. 3rd ACM PODC* 1984, pp. 89–102.

[DS83] D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Computing*, Vol. 12, No. 3 (November 1983), pp. 656–666.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *JACM*, Vol. 35 (1988), pp. 288–323.

[DM86] C. Dwork and Y. Moses. Knowledge and common knowledge in Byzantine environments I: crash failures. *Information and Computation*, Vol. 88, No. 2 (1990), pp. 156–186.

[DS91] C. Dwork and L. Stockmeyer. Bounds on the time to reach agreement as a function of message delay. IBM Research Report RJ8181, June 1991.

[FL82] M. Fischer and N. Lynch. A lower bound for the time to assure interactive consistency. *IPL*, Vol. 14, No. 4 (June 1982), pp. 183–186.

[FLP85] M. Fischer, N. Lynch and M. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, Vol. 32, No. 2 (1985), pp. 374–382.

[GSTC90] A. Gopal, R. Strong, S. Toueg and F. Cristian. Early-delivery atomic broadcast. *Proc. 9th ACM PODC*, 1990, pp. 297–309.

[HK89] A. Herzberg and S. Kutten. Efficient Detection of Message Forwarding Faults. *Proc. 8th ACM PODC*, 1989, pp. 339–353.

[LM85] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *JACM*, Vol. 32, No. 1 (January 1985), pp. 52–78.

[LSP82] L. Lamport, R. Shostak and M. Pease. The Byzantine generals problem. *ACM TOPLAS*, Vol. 4, No. 3 (1982), pp. 382–401.

[LL84] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, Vol. 62, Nos. 2/3 (1984), pp. 190–204.

[LL88] J. L. Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, Vol. 77, No. 1, (1988), pp. 1–36.

[MMA90] P. M. Melliar-Smith, L. Moser and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Trans. on Parallel and Dist. Systems*, Vol. 1, No. 1 (January 1990), pp. 17–25.

[MMA91] L. Moser, P. M. Melliar-Smith and V. Agrawala. Asynchronous fault-tolerant total ordering algorithms. Manuscript.

[M85] M. Merritt. Notes on the Dolev-Strong lower bound for Byzantine agreement. Unpublished manuscript, 1985.

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

[P91] S. Ponzio. Consensus in the presence of timing uncertainty: omission and Byzantine failures. *Proc. 10th ACM PODC*, 1991, pp. 125–138. Also: MIT SM Thesis, June 1991. MIT/LCS/TR–518, October 1991.

[ST87] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *JACM*, Vol. 34, No. 3, July 1987, pp. 626–645.

[SDC90] R. Strong, D. Dolev and F. Cristian. New latency bounds for atomic broadcast. *Proc. 11th IEEE Real-Time Systems Symposium*, 1990.