Authenticated Agreement Protocols without Explicit Clock Synchronisation

Francisco V. Brasileiro,¹ Paul D. Ezhilchelvan² and Neil A. Speirs²

¹Departamento de Sistemas e Computação
Universidade Federal da Paraíba - UFPb/Campus II
Av. Aprígio Veloso s/n, Campina Grande - Pb, 58100-000, Brazil
(on leave at University of Newcastle upon Tyne)

²Computing Science Department
University of Newcastle upon Tyne
Newcastle Upon Tyne, NE1 7RU, UK

Abstract
Replicated processing requires that replicas reach agreement on the order in which messages are to be processed. Synchronous and deterministic agreement protocols published in the literature require replicas to maintain an abstraction of clocks that are kept in bounded synchronism. We present a protocol that does not have this requirement and relies only on physical, hardware clocks with bounded drifting rates. Its performance is shown to be as good as the performance of protocols that do require synchronised clocks. A variation of the protocol is then derived for more practical, Triple Modular Redundant (TMR) systems, and is shown to have better performance than existing protocols. These protocols are well suited to perform process replication.

Keywords and Phrases
Byzantine failures, fault tolerance, N-Modular Redundancy (NMR), process replication, agreement, message ordering, message authentication, physical clocks.
1. Introduction

The synchronous and deterministic agreement protocols published in the literature require that non-faulty processors have access to *synchronised clocks* whose readings at any given instance of real time are guaranteed to differ only within a known bound. Meeting this requirement will in turn require each processor to periodically compute the amount of adjustment to be made to the reading of the local physical clock, and then to make the adjustment. The need to adjust a read-only clock leads to constructing an *abstraction* of a synchronised clock whose reading is the sum of the physical clock reading and the adjustment that is stored in a memory location (see [Dolev84] for example).

In this paper, we develop an order protocol that does not require the maintenance of synchronised clock abstraction, but relies directly on physical clocks for knowing the current time and for scheduling operations at future times. Many operating systems provide an efficient, low level scheduler that meets our requirement (which is typically in the form of • schedule an execution of process \( p \) when clock reads \( t' \)). Using only the physical clock and a scheduler, an agreement protocol is developed and its performance is analysed. We show that the protocol works at least as fast as the synchronised-time based ones that use sophisticated amortisation techniques to perform clock adjustment.

We cast the problem in the form of agreeing on the order in which messages are presented to the non-faulty processors of an \( N \)-processor *node*. All messages that are input to the node must be consumed in an identical order by non-faulty processors; i.e., non-faulty processors must *agree* on the relative order in which each input message is to be processed. We observe that the agreement protocol developed is appropriate for building \( N \)-Modular Redundant (NMR) nodes where \( N, N = 2\| +1 \), is the number of processors in the node, and \( \| \) is the maximum number of processors that can fail. We then show a modified protocol for Triple Modular Redundant (TMR) nodes \( (N = 3, \| = 1) \), that can reach agreement faster than synchronised-time based protocols.

2. Assumptions

We term the \( N \) processors trying to reach agreement a *node* and suppose that each processor \( P_i, 1 \leq i \leq N \), is directly connected to each other via reliable links, and to the rest of the distributed system through a reliable bus (or any other network architecture).

**Assumption 1**: In an \( N \)-processor node there are at most \( \| < N \) faulty processors. Hence, at least one of \( N \) processors is non-faulty and never fails; processors are uniquely numbered and ordered, and the ordering is known to all non-faulty processors.

When a processor receives an input message from the bus, it forms a message \( m \) that contains the received message in the field \( m.\mu \). It decides the processing order for \( m \) (\( m.\mu \) to be precise) by sending \( m \) to all other processors in the node using a protocol that guarantees the following conditions:

**Validity**: If a non-faulty \( P_i \) forms and sends \( m \), all non-faulty processors (including \( P_i \)) will decide on an order for \( m \), within a known and bounded real time interval \( _- \), and

**Unanimity**: If a non-faulty \( P_i \) decides on an order for a given \( m \), then every non-faulty \( P_j \) decides on the same order for \( m \).

Each processor can sign the messages it sends, and can authenticate the signed messages it receives. The following assumptions are made on the signature and authentication capabilities of non-faulty processors [Rivest78].
Assumption 2 A non-faulty processor's signature for a given message is unique and cannot be generated by any other processor.

Assumption 3: Any attempt to alter the contents of a non-faulty processor's signed message is detected by any other non-faulty processor.

In this paper we adopt the style of writing real time values in Greek or italicised upper case Roman letters, and clock time values in italicised lower case Roman letters; the term 'clock' will always refer to a processor's hardware clock.

Assumption 4: If a non-faulty $P_1$ sends $m$ to set of processors in the node at real time $T_1$, every non-faulty destination $P_j$ will receive $m$ at real time $T_j$, $T_1 - T_j < T_1 + \epsilon$, where $\epsilon > 0$, is known.

Assumption 5: A non-faulty $P_1$'s clock measures a clock time interval $\mu$ in a real time interval $x(1+p_1)$, where $|p_1| - \rho$ and $\rho > 0$, is known.

3. The protocol

The development of the protocol mainly involves implementing (i) message diffusion to ensure that non-faulty processors exchange an identical set of messages between them, and (ii) timeliness checks to enable a non-faulty processor to assess whether a received message is timely or not. First we will present the implementation of message diffusion.

Message diffusion

Each processor $P_i$ within the node maintains a message counter, denoted by $MC_i$, whose value is an integer that never decreases and is initialised to 1 when the node is first started. Before $P_i$ sends the message $m$ it formed, it performs the following actions on $m$: first it time-stamps $m$ by setting the message time-stamp field $m.TS = MC_i$ and increments $MC_i$ by 1 - thus ensuring that any message it later forms gets a time-stamp larger than $m.TS$; then it sets the message originator field $m.O$ to its identifier, $i$; and finally it generates a signature for $m$ which is put in the signature field $m.S$. $P_i$ then sends $m$ to all other processors in the node, and enters a copy of $m$ in a message list called $accepted_i$.

Whenever $P_i$ receives a message $m$ from another processor, it accepts $m$ only if $m$ has been properly signed and is timely. If $m$ is accepted, then it is entered in the message list $accepted_i$ and $MC_i$ is set to the maximum of $\{MC_i, m.TS+1\}$. If the accepted $m$ has been signed by less than $\frac{\#}{2}+1$ processors, $P_i$ generates a signature for $m$ and appends the generated signature to the signatures already in $m.S$; $m$ is then sent to all other processors that have not signed it. We use the notation $|m.S|$ to denote the number of signatures contained in $m$; thus, any accepted $m$ with $|m.S| < \frac{\#}{2}+1$ is diffused. As there are at most $\frac{\#}{2}$ faulty processor, this diffusion ensures that if $m$ enters $accepted_i$ of any non-faulty $P_i$, then some $m'$ such that $m'.\mu = m.\mu$, $m'.O = m.O$ and $m'.TS = m.TS$, is received by every other non-faulty $P_j$ in the node. For any given message $m$, we define an $equiv(m)$ as any $m'$ such that $m'.\mu = m.\mu$, $m'.O = m.O$ and $m'.TS = m.TS$. That is, an $equiv(m)$ is either identical to $m$ or differ from $m$ only in the signature field.

Lemma 1 (message diffusion): If a non-faulty $P_j$ accepts $m$ at real time $T_j$, then every non-faulty $P_k$ receives an $equiv(m)$ at real time $T_k$ such that $|T_j - T_k| < \epsilon$.

Proof: It follows from the message diffusion mechanism described above and from assumption 4.

To present the protocol, we assume the availability of the primitives $receive(m)$ and $send(m)$. These two primitives implement message diffusion. The $receive(m)$ primitive returns a message $m$ that has been received from an internal link and has been authenticated. Hence, any message $m$ received by a non-faulty processor by invoking the $receive(m)$ primitive is
authentic, and further development of the protocol need only be concerned with the timeliness of $m$. The send($m$) primitive generates the sender’s signature for $m$ if $m$ is eligible for diffusion (i.e., $\lceil m.S \rceil < \#+1$), and appends the generated signature to the signature sequence already in $m.S$ (if any). The resulting message is then transmitted to all other processors in the node that have not signed $m$.

**Timeliness checks**

These checks enable a processor to determine the timeliness of a received $m$ and ensure that $m$ is found timely by any non-faulty receiver if the immediate sender of $m$ is non-faulty. Before presenting the principles behind these checks, we will define a clock time interval $d$ such that by measuring $d$ in its local clock a non-faulty processor is guaranteed to measure a real time interval of at least $l$ duration; $d$ is known to all non-faulty processors of the node. To allow for the compensation of the inaccuracy of the clocks of non-faulty processors whilst ordering a particular $m$, $d$ must be chosen such that $d > l/(1-(2\#+1)\rho)$ (see [Brasil95]). We will assume, for simplicity, that a processor takes zero time to execute any instruction of the protocol and the send($m$) and receive($m$) primitives. (To realise this assumption in practice, it is necessary to increase the value of $d$, which is possible as the protocol does not impose any upper bound on the value of $d$.)

Supposing that $P_i$ receives a message $m$ at its local clock time $t_i$, we will define the checks that $P_i$ must carry out to determine whether $m$ is timely or not. In a synchronised-time based protocol, $m.TS$ will indicate the time that $m$ was sent according to the sender’s (i.e., $m.O$) synchronised clock, and $P_i$ must consider $m$ timely only if it receives $m$ before an interval of length $l(m)$ expires, after its own synchronised clock has read $m.TS$ (i.e. before its synchronised clock reads $m.TS+l(m)$), where $l(m) > 0$, depends on $\lceil m.S \rceil$ and the clock synchronisation precision. In our protocol $P_i$ will consider $m$ timely only if $m$ is received before a time interval of $l(m)$ elapses after the timing instance when $MC_i$ first became larger than $m.TS$; also, $l(m)$ will depend only on $\lceil m.S \rceil$ and $\rho$. Note that when $m$ was received, if $MC_i > m.TS$ then $m$ is either a ‘present’ or a ‘future’ message with respect to $MC_i$, and must be considered as timely; if, on the other hand, $MC_i$ is already larger than $m.TS$ when $m$ is being received, then $m$ is a ‘past’ message with respect to $MC_i$, and its timeliness will depend whether or not an interval of length $l(m)$ has elapsed since $MC_i$ first became larger than $m.TS$.

To establish timeliness checks for a non-faulty $P_i$, we will suppose that $MC_i$ was already larger than $m.TS$ when $P_i$ received $m$.

Recall that $MC_i$ is incremented whenever a message is either formed and sent by $P_i$ or accepted by $P_i$. We will suppose that $m'$ is the message whose sending or acceptance by $P_i$ at local clock time $t_i'$ made $MC_i > m.TS$ for the first time. Obviously, $m'.TS > m.TS$ and $MC_i > m.TS$ just before $P_i$'s clock read $t_i'$. Each non-faulty $P_i$ executes a protocol which employs the following check:

**C-N**: A received $m$, $m.TS > m'.TS$ and $\lceil m.S \rceil = s$, is timely only if it is received before local clock time $t_i'+2sd$, for every $s, 1 < s < (\#+1)$.

The reasoning behind this timeliness check is as follows: from lemma 1, every non-faulty $P_k$, $k \neq i$, will set $MC_k > m'.TS$ at latest $d$ units of clock time after $m'$ is accepted by $P_i$, thus timely single-signed $m$, $m.TS > m'.TS$ and $m.O = k$, must be received by $P_i$ within $2d$ clock time interval after $P_i$ has accepted $m'$. Also, from lemma 1, every non-faulty $P_j$, $j \neq i$, will accept equiv($m'$) at most $d$ units of clock time after $P_i$ and will perform a similar timeliness check onto single-signed $m$ they receive; thus, since a relayed $m$ takes at most $d$ units of clock time to be transmitted from a non-faulty processor to another, the timeliness checks for two equiv($m$)
with \( s \) and \( s+1 \) signatures, \( \lfloor 1 - s \rfloor \), must be at least \( 2|d| \) apart; in this way if a non-faulty \( P_i \)
accepts \( m \), \( |m.S| = s \), then the \( \text{equiv}(m) \), \( |\text{equiv}(m).S| = s+1 \), relayed by \( P_i \) to the other processors
that have not signed \( m \) is always found timely by any non-faulty receiver.

Protocol outline

To implement check C-N, each \( P_i \) maintains timing counters, denoted as \( C_i[k, s] \), for every \( P_k \),
\( k \neq i \), and for every \( s \), \( 1 \leq s \leq |\lfloor 1 + \rfloor| \). These counters have integer values which are initialised
to zero and never decrease. A timing counter \( C_i[k, s] \) is set to \( ts \), to indicate that any \( m, m.TS < ts \),
\( |m.S| = s \) and \( m.O = k \), received thereafter must not be considered timely. Check C-N indicate
that setting \( C_i[k, s] \) and \( C_i[k, s+1] \) to a given \( ts \) must be separated by a clock time interval \( 2|d| \).
A scheduler is used to execute a process, named \( \text{update} \), at appropriate times. An execution by
a non-faulty \( P_i \) of \( \text{update}(k, s, ts) \), \( k \neq i \) and \( 1 \leq s \leq |\lfloor 1 + \rfloor|+1 \), will set \( C_i[k, s] = \max \{C_i[k, s], ts\} \),
and if \( s < |\lfloor 1 + \rfloor|+1 \), \( p \) will also schedule itself to be executed after \( 2|d| \) time. Thus, if at local clock time \( t_i \),
\( P_i \) forms and sends, or accepts \( m \), then executions of \( \text{update}(k, 1, m.TS) \) are
scheduled to set \( C_i[k, 1] = \max \{C_i[k, 1], m.TS\} \) for every \( P_k \), \( k \neq i \), at local clock

time \( t_i+2|d| \).

We define \( C_i, \min = \min \{C_i[k, |\lfloor 1 + \rfloor|] | 1 \leq k \leq N \land k \neq i \} \). Any \( m \), \( m'.TS < ts \), that is
received after \( C_i, \min \) has become larger than or equal to \( ts \), will be considered late and will
not enter \( \text{accepted}_i \), i.e., the message list \( \text{accepted}_i \) becomes closed for such \( m' \) after \( C_i, \min \)
\( ts \). So, every \( m, m.TS < ts \), that is already in \( \text{accepted}_i \) can be safely ordered after \( C_i, \min \)
\( ts \). Such \( m \) are removed from \( \text{accepted}_i \) and put into another list called \( \text{stable}_i \) for ordering.
Messages \( m \) and \( m' \) that are in \( \text{stable}_i \), are said to be \emph{spurious} if \( m.O = m'.O \) and \( m.TS = m'.TS \)
\( m.mu \neq m'.mu \). Spurious messages must have been formed by a faulty processor that failed by
giving the same time-stamp to distinct messages \( m \) and \( m' \). They are removed from \( \text{stable}_i \)
before message ordering. An ordering relation « for distinct \( m \) and \( m' \) is defined: \( m \ll m' \), if
\( (m.TS < m'.TS) \) or \( m.TS = m'.TS \) and \( m.O < m'.O \). The protocol for processor \( P_i \) is
presented below:

```plaintext
do /* when receiving a message from the bus */
    get an input message from the bus;
    \( m.mu = \text{input-message} \); \( m.TS = NC_i \); \( m.O = i \);
    \( MC_i = MC_i + 1 \);
    \( \text{send}(m) \);
    \( \text{accepted}_i = \text{accepted}_i \upharpoonright \{ m \} \);
    \( t = \text{clock}_i \);
    \( \text{for every } P_k, 1 \leq k \leq N \land k \neq i \text{ schedule } \text{update}(k, 1, m.TS) \text{ at } t+2|d|; \text{end for}; \)
end do

∥
do /* when receiving a message from a link */
    receive(m);
    \( \text{if } (m.TS < C_i[m.O, |m.S|] \text{ or an equivalent } \text{accepted}_i) \text{ then } \text{discard}(m) \)
    \( \text{else} \)
    \( \text{send}(m) \);
    \( C_i = \max \{C_i, m.TS+1\} \);
    \( \text{accepted}_i = \text{accepted}_i \upharpoonright \{ m \} \);
    \( t = \text{clock}_i \);
    \( \text{for every } P_k, 1 \leq k \leq N \land k \neq i \text{ schedule } \text{update}(k, 1, m.TS) \text{ at } t+2|d|; \text{end for}; \)
    \( \text{end if} \)
end do
```
do /* ordering and delivering messages */
if \( C_{i,\text{min}} \leq \text{minimum of } \{ C_i[k, \rho+1] \mid 1 \leq k \leq N \text{ and } k \not= i \} \) then
  \( C_{i,\text{min}} = \text{minimum of } \{ C_i[k, \rho+1] \mid 1 \leq k \leq N \text{ and } k \not= i \} \)
  \( \text{stable}_i = \{ m \mid m \_ \text{ accepted}_i \text{ and } m \_ \text{TS} \_ C_i,\text{min} \} \)
  \( \text{accepted}_i = \text{accepted}_i \text{- stable}_i \)
  \( \text{spurious}_i = \{ m, m' \_ \text{ stable}_i \mid m' \_ \text{O} = m \_ \text{O} \text{ and } m' \_ \text{TS} = m \_ \text{TS} \text{ and } m' \_ \mu \not= m \_ \mu \} \)
  order and deliver each \( m, m \_ \text{ stable}_i \)-spurious\( _i \) according to « relation;
end if
end do

The protocol is implemented by three concurrent processes, which respectively deal with: input messages received from outside the node; internal messages received from an internal link; and message ordering. Note that the second process enters an accepted \( m \) into accepted\( _i \), only if an equiv\( (m) \) is not already in the list; thus, after removing any spurious messages from stable\( _i \), the third process can order the remaining messages according to the « relation. We state without proof the following theorem (correctness proofs can be seen in [Brasil95]):

**Theorem 1**: The protocol guarantees unanimity and validity conditions within a real time interval \( -\alpha \_ -\alpha \_ 2d(\rho+1)(1+\rho)+\_ \), if \( d \leq f/(1-(2\rho+1)\rho) \). The protocol can guarantee ordering agreement in finite time only if \( \rho < (1-\rho)/2\rho \).

**Protocol performance**

We will compare the performance of our protocol and the synchronised-time based protocols, assuming the best possible clock environment for the latter. (We will ignore the early-stopping protocols that gain speed at the expense of increased message cost [Gopal90, Brasil95], and also those that assume redundant broadcast networks [Babao85]). We will define the protocol's ordering delay, \( -\alpha \), for a message \( m \) sent by a non-faulty processor, as the real time interval elapsed between the instance when \( m \) was formed and sent, and the latest instance when a non-faulty processor in the node ordered \( m \). (Note that \( -\alpha \) is not defined for \( m \), if the sender of \( m \) is faulty.) For our protocol, the maximum ordering delay is \( -\alpha = 2d(\rho+1)(1+\rho)+\_ \), with \( d = f/(1-(2\rho+1)\rho) \).

Let us denote the maximum ordering delay for a synchronised-time based protocol as \( -\Delta \). These protocols require at most \( \rho+1 \) rounds of message exchange between processors. Thus, if a non-faulty \( P_1 \) forms and sends \( m \) at its synchronised clock time \( t_1 \), then every other non-faulty \( P_k \) will order \( m \) at its respective synchronised clock time \( t_1+(\rho+1)(d_\Delta+\epsilon) \), and at or before \( P_1 \)'s synchronised clock time \( t_1+(\rho+1)(d_\Delta+\epsilon)+\epsilon \), where \( \epsilon \) is the upper bound on clock synchronisation precision, and \( d_\Delta = f/(1-\rho) \) (see [Cri85] for example). So, \( -\Delta = (\rho+1)(d_\Delta+\epsilon)(1+\rho') \), where \( 1+\rho' \) is the maximum running rate of a synchronised clock. As clocks are synchronised by periodically adjusting the physical clock reading, the readings of a synchronised clock will not be continuous, unless special arrangements are made. To make the comparison simple, we will assume the use of amortisation techniques [Schmuck90] to ensure continuity in the readings of synchronised clocks and also that \( \rho' = \rho \) (see [Srikanth85]). We will consider (for simplicity and in favour of synchronised-time based protocols) \( \epsilon \) to be the maximum clock difference immediately after adjustments were made; i.e., we ignore a component of \( \epsilon \) that accounts for clock drift until next adjustment. Table 1 below shows the difference \( -\Delta \) when \( \epsilon = f/(1+\rho) \) [Halp84] and \( \epsilon = (1+5\rho)/f(1+\rho) \) [Srikanth85]. (We have neglected the higher order terms - \( O(\rho^2) \) - of \( \rho \).)

| synchronisation precision | \( -\Delta \) for \( \rho = 2 \) | \( -\Delta \) for \( \rho = 1 \) |
\[ \varepsilon = \frac{1}{1+p} \]  
\[ \varepsilon = \frac{1+5p}{1+p} \]

<table>
<thead>
<tr>
<th>( \Delta )</th>
<th>( \Delta ) for different synchronisation precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>( \frac{1}{1+p} )</td>
</tr>
<tr>
<td>( \rho )</td>
<td>( \rho )</td>
</tr>
<tr>
<td>( \Delta )</td>
<td>( \Delta )</td>
</tr>
<tr>
<td>( \pi )</td>
<td>( \pi )</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>( \sigma )</td>
</tr>
</tbody>
</table>

Table 1: \( \Delta \) for different synchronisation precision

\( \Delta \) is \( O(\rho \pi^2) \) in both cases. In practice, for small values of \( \pi \), the difference is negligible. It must be pointed out that this comparison was made with conditions that only favour the synchronised-time based protocols. Despite this, in practice, our protocol can be seen to be equally fast and does not involve the overhead for constructing synchronised clock abstractions.

### 4. Improving the protocol performance for TMR nodes

We improve the protocol performance for the important case of 3-processor (TMR) nodes \((N = 3, \pi = 1)\) by employing more precise timeliness checks which are derived by exploiting the fact that each processor has only two other processors to reach agreement with. Each non-faulty \( P_i \) of a TMR node executes a protocol which employs the following timeliness checks:

- **C-TMR-1**: A received \( m, m.TS \_ m.TS \) at local clock time \( t_i' \), \( P_i \) caused \( MC_i > m.TS \) for the first time, then:
  
  If the reception and acceptance by \( P_i \) of \( m, m.TS \_ m.TS \), at local clock time \( t_i' \), made \( MC_i > m.TS \) for the first time, then:
  
  **C-TMR-2**: A received \( m, m.TS \_ m.TS, m.O = m'.O \) and \( |m.S| = 1 \), is timely only if it is received before clock time \( t_i' + d \).
  
  **C-TMR-3**: A received \( m, m.TS \_ m.TS, m.O \neq m'.O \) and \( |m.S| = 2 \), is timely only if it is received before clock time \( t_i' + 3d \).

These checks have the effect of reducing the maximum ordering delay at non-faulty receivers, from \( +4d(1+p) \) to \( +3d(1+p) \). Since \( d(1+p) \) is the maximum ordering delay for this modified protocol is given by the time a message takes to be ordered by the sender (i.e. \( 4d(1+p) \)), instead of \( +4d(1+p) \), as per the protocol in the previous section for \( \pi = 1 \). We again state without proof the following theorem (correctness proofs can be seen in [Brasil95]):

**Theorem 2**: The protocol guarantees *unanimity* and *validity* conditions within a real time interval \( -a, -a -4d(1+p) \), if \( d \_ \_/(1-5p) \).

### 5. Conclusion

The protocols described here make an important contribution towards building fault-tolerant replicated nodes, and achieving message ordering without requiring the clocks of a node to be maintained in bounded synchronism. The protocol indicates that when clocks are not explicitly synchronised, reaching agreement in a synchronous environment can become an asynchronous problem, if the number of faults to be tolerated exceeds a certain value (i.e. when \( \pi \_ (1-\rho)/2\pi \)). Thus, there are situations where deterministic agreement cannot be guaranteed even in the presence of one benign, let alone Byzantine, failure. For the important practical case of TMR nodes, the protocol given in section 4 achieves better performance than synchronised-time based ones.

**Acknowledgements**: This work has been supported in part by grants from CNPq/Brazil and ESPRIT BROADCAST. Comments from Santosh Shrivastava were helpful.

**References**


