Technische Universität Wien

DIPLOMARBEIT

Switching On. How Processes Initialize for Consistent Broadcast

carried out at the Department of Automation
on the Vienna University of Technology

under guidance of
ao. Univ. Prof. Dr. Ulrich Schmid

by

Josef Widder
Matr.Nr.: 9625114

Meidlgasse 41/4/4
1110 Wien

October 16, 2002

Abstract

Well known system models of distributed systems assume that all processes are already running when the algorithm starts. This includes the assumption that there are enough non-faulty processes in the system to run an algorithm correctly even in the presence of failures.
This diploma thesis deals with the problem of initializing a network where the processes are down at the beginning and therefore don't receive messages until they have booted. When a process is up it has to decide when it is save to start the desired service, based only on the information it gets from messages. That includes that the process has no timing information. The analysis of the algorithm is done using a partially synchronous system model where exist lower and upper bounds on the message transmission delay which are not known by the processes. We consider the problem of round synchronization. The solution is a modification of the well known *non authenticated broadcast primitive* by Srikanth and Toueg.

Zusammenfassung

Bekannte Systemmodelle von Verteilten System gehen von der Annahme aus, dass alle Prozesse bereits laufen wenn der Algorithmus startet. Das inkludiert die Annahme, dass genügend fehlerfreie Prozesse im System sind, um den Algorithmus korrekt zu exekutieren falls fehlerhafte Prozesse vorhanden sind.
Diese Diplomarbeit behandelt das Problem der Netzwerkinitialisierung bei dem Prozesse zu Beginn nicht laufen und sie deshalb keine Nachrichten erhalten bis sie gestartet sind. Basierend auf den empfangenen Nachrichten muss ein gestarteter Prozess entscheiden, wann es sicher ist einen bestimmten Dienst zu starten. Dabei hat er keinerlei Information über Zeitverhalten. Die Analyse des Algorithmus basiert auf einem partiell synchronen Systemmodell wo untere und obere Grenzen der Dauer von Nachrichtenübertragungen existieren, die allerdings den Prozessen nicht bekannt sind. Der untersuchte Dienst ist das gemeinsame Durchführen von Rundenwechseln aller korrekten Prozesse im System. Die Lösung ist eine Adaption der bekannten *non authenticated broadcast primitive* von Srikanth und Toueg.

Contents

1 Introduction

A still not sufficiently explored problem in distributed computing is initialization: Nodes in a distributed system boot at unpredictable times and have to decide locally when it is save to start a given service even in presence of faults. Nodes that start late and don't receive messages sent earlier should not be counted as faulty, however.

In synchronous systems, initialization could be done based upon a priori knowledge of maximum booting delays and is hence straight forward. The synchrony assumption is a quite strong one, however, and it is well-known that important classes of applications (like Internet-services) cannot reasonably be modeled as synchronous systems. Partially synchronous systems are an interesting alternative here.

This thesis shows a solution for the problem of initialization of a partially synchronous network. The network service considered is *round synchronization* which ensures that for any two nodes p_i and p_j with local round numbers $R_i(t)$ and $R_j(t)$, respectively, the following *synchrony condition* is satisfied at all times t :

Definition 1.1. *Synchrony Condition:* $| R_i(t) - R_j(t) | \leq D_{max}$*, where* D_{max} *denotes the maximum* round skew[1].

Note that round synchronization is also known as distributed clocks in the literature [4], which is a weak form of clock synchronization. Our type of round synchrony is weaker than the usual synchrony of the times when two correct processes switch to a given round k. More specifically the synchrony condition can be satisfied without a progress guarantee, whereas it is impossible to bound the maximum difference of round switching times without it. Still the synchrony condition from Definition 1.1 is sufficient in certain applications, as the perfect muteness detector from [6].

The basic algorithm employed in this thesis is *Consistent Broadcasting (CB)* introduced by Srikanth and Toueg in [9]. The basic problem of initialization for Consistent Broadcasting is how to decide locally when enough correct processes are booted. To conclude that the required $2f+1$ correct processes are up, a process must have received $3f+1$ messages from different nodes. When the number of processes in the system is exactly $3f+1$, however, we cannot guarantee that every correct process starts the service since it need not receive messages sent by faulty processes.

[1]analogous to the clock skew in the field of clock synchronization as found in [3]

Since we do not want to increase the required number of processes just because of initialization, we allow round synchronization to degrade during initialization. This thesis shows that a certain *modified consistent broadcast (MCB)* primitive can guarantee some D_{max} for any number of booted processes, which decreases to some D_{CB} (determined by the properties of Consistent Broadcast) if at least $2f+1$ correct processes are up. Our modified primitive is capable of both, initializing the system and performing round synchronization afterwards.

1.1 Related Work

The basic algorithm for round synchronization is the non-authenticated broadcast primitive from [9]. Algorithms for initialization and integration for CB can be found there also, but they assume enough running processes. This solution is in fact based upon ideas developed in the clock synchronization context in synchronous systems in [7]. The MCB primitive developed in this thesis employs an idea from [4, 2], which had been used for providing a distributed clock in partially synchronous systems. Related work on partially synchronous distributed systems can be found in [4, 8]. Initialization from the very beginning is also somehow related to the crash recovery model in [1].

1.2 Roadmap

The remainder of this thesis is organized as follows: Section 2 contains the description of the partially synchronous system model and some notational conventions. In Section 3 the non authenticated broadcast primitive by Srikanth and Toueg [9] is considered in the partially synchronous model, still without initialization. In Section 4 the modified broadcast primitive is introduced which guarantees a maximum round skew for an arbitrary N. Section 5 eventually contains description and analysis of the algorithms that initialize the system. The thesis closes with the concluding Section 6.

2 System Model

2.1 The Partially Synchronous System Model

This thesis considers a system of N distributed processes $p_1, ..., p_N$ that communicate through a reliable, error-free and fully connected point-to-point message system. We assume that a non-faulty receiver of a message knows the sender. The communication channels between processes need not to provide FIFO transmission.
Amongst the N processes there is a maximum of f faulty ones. Since we examine the starting of a network, processes that have not booted yet are not counted as faulty, except of course if they never boot. No assumption is made on the behaviour of faulty nodes, i.e. they are said to be Byzantine faulty [5].

The examined network is partially synchronous [4]. More specifically, we assume that there exist bounds on the duration of actions (computational and transmission delays) in the system, which are not known in advance. Therefore processes have no timing information and can only make decisions based on received messages.

Definition 2.1. *Transmission Delay. δ is the end-to-end computational + transmission delay of a message sent between two correct processes. δ can be different for each message sent.*

Definition 2.2. *Bounds on the Transmission Delay. $0 < \tau^- \leq \delta \leq \tau^+ < \infty$. The bounds τ^- and τ^+ are not known in advance. Since $\tau^+ < \infty$ every message sent from a correct process to another is eventually received.*

Definition 2.3. *Transmission Delay Uncertainty. $\epsilon = \tau^+ - \tau^-$. If a correct process sends a message at time t, it cannot be received by a correct process before $t + \tau^-$ and not after $t + \tau^+$.*

Definition 2.4. *Transmission Delay Ratio. $\mathcal{P} = \frac{\tau^+}{\tau^-}$. This ratio will be a central part of the analysis of the algorithms discussed in this thesis.*

Since the processes have no knowledge of τ^+, τ^- and even $\mathcal{P}$, those values cannot be used by the algorithm but are only used in the analysis. This effectively leads to asynchronous algorithms that run in synchronous systems. The timing performance of our algorithms hence depends on the underlying network properties: If τ^+ and τ^- are small, the timing behaviour will be good. A violation of the system assumption does no harm to the correctness of the algorithm, only its quality (in our case the round skew D_{max}) is getting worse than expected. The error is not in the analysis but rather in the assumptions. If, however, the system returns to the assumed behaviour, the algorithm also provides the calculated D_{max}.

What is more, even our bound on the round skew D_{max} neither includes τ^+ nor τ^- but rather $\mathcal{P}$. In terms of assumption coverage, this is an advantageous property: At heavy network load, when messages have to be queued at the network interfaces, both τ^+ and τ^- increase. Therefore it is possible that an assumption on $\mathcal{P}$ holds despite the fact that an assumption on τ^+ does not. Our algorithm would hence still provide the expected performance in this case, whereas a synchronous would fail.

2.2 Model of the Initialization Phase

At the beginning all correct processes are down, i.e. they do not send or receive messages. Every message that arrives at a process while it is down is lost. A process decides independently when it wishes to participate in the system (or is just turned on). Since faulty processes can perform Byzantine behaviour, we can assume that faulty processes are always up or at least booted before the first correct one. Since we consider a distributed system at the very beginning, we cannot assume any kind of authentication service.

During initialization correct processes go through the following modes:

1. *down*: A process is down when it is not started yet or has not completed booting.

2. *up*: A process is up if it has completed booting. To get a clean distinction of up and down we assume that a process flushes the input queues of its network interface as first action after booting is completed. Hence it gets messages only if they arrived when it was up.

3. *passive*: A running process initially performs some initialization algorithm that does not provide the required service to the application. During this phase it is said to be passive.

4. *active*: A process that has done its initialization in passive mode and started to provide the required service to the application is called active.

Definition 2.5. *Number of running processes. N_{up}. We will use N_{up} for the number of processes that are up at a given instant in time. This includes up to f faulty ones.*

3 Round Synchronization by Consistent Broadcasting

In this Section the non-authenticated consistent broadcast primitive from [9] is considered under the partially synchronous system model of Section 2. We still assume, however, that all processes ($N \geq 3f + 1$) are up right from the beginning and that the processes are initially synchronized.

Originally the broadcast primitive shown in Figure 1 was designed for a clock synchronization algorithm although it can also be used as building block for other applications [4, 6].

```
for each correct process

**if** received *(init, k)* from at least $f + 1$ distinct processes
  $\rightarrow$ send *(echo, k)* to all;
**fi**

**if** received *(echo, k)* from at least $f + 1$ distinct processes
  $\rightarrow$ send *(echo, k)* to all;
**fi**

**if** received *(echo, k)* from at least $2f + 1$ distinct processes
  $\rightarrow$ *accept (round k);*
     send *(init, k+1)* to all; /* start next round */
**fi**
```

Figure 1: Primitive for Consistent Broadcast of [9]

In our context, the algorithm reaches agreement regarding the processes local round numbers: A process switches to *round* $k + 1$ if its instance of the consistent broadcast primitive reaches *accept(round k).* The algorithm guarantees that if enough correct processes ($f+1$) decide that they want to switch to the next round, then every correct process switches to the next round within a time interval that is only determined by the end-to-end transmission delay of messages.

In the clock synchronization context of [9] the time between consecutive round switches is determined by a re-synchronization interval, which is controlled by a local clock device: After a process has switched rounds it waits for a sufficiently large time and then starts the next round by sending an *(init, k)* message. This guarantees that all processes have already switched when the next round starts. In our context the processes do not wait[2] after they switched rounds but start the next round immediately. Therefore processes that get messages fast could go ahead several rounds. We will see that the maximum round skew of any two processes is determined only by the Uncertainty Ratio $\mathcal{P}$. This property seems typical for partially synchronous systems.

The consistent broadcast primitive from Figure 1 guarantees the following properties of round synchronization. They must hold in our system model as well:

Definition 3.1. *Properties of Round Synchronization.*

- *P1. Correctness. If at least $f+1$ correct processes switch to round k by time t, then every correct process switches to round $k+1$ by time $t+\tau_{rt}$, where $\tau_{rt} = 2\tau^+$.*
- *P2. Unforgeability. If no correct process switches to round k by time t, then no correct process switches to round $k+1$ by $t+2\tau^-$ or earlier.*
- *P3. Relay. If a correct process switches round k at time t, then every correct process does so by time $t+\tau_\Delta$, where $\tau_\Delta = \epsilon + \tau^+$.*

Theorem 3.2. *The primitive from Figure 1 guarantees the properties P1, P2 and P3 from Definition 3.1 if $N \geq 3f+1$.*

Proof. Correctness. Since at least $f+1$ correct processes broadcast *(init, k)* by time t, all correct processes receive at least $f+1$ *(init, k)* messages by time $t+\tau^+$. When they do so, they send *(echo, k)* to all. By time $t+2\tau^+$ every correct process receives at least $2f+1$ *(echo, k)* messages, and then switches to round $k+1$.

Unforgeability. The proof is by contradiction. Assume that there is a process p_a that switches to round $k+1$ before instant $t+2\tau^-$. p_a switches because it has seen $2f+1$ *(echo, k)* messages, i.e. at least $f+1$ such messages sent by correct processes. These messages must have been sent before $t+\tau^-$. Correct processes only send *(echo, k)* when they have seen $f+1$ *(init, k)* or *(echo, k)* messages. That is at least one correct process that must have sent either of those messages before t. This contradicts the assumption in the unforgeability property, however.

[2]The omission of the re-synchronization interval is only introduced since it simplifies the analysis. It could be re-introduced quite easily, however, since τ^+ and τ^- include both the computational and the transmission delay. The re-synchronization interval can hence be incorporated as part of the computational delay.

Relay. Since a correct process p switches to round $k+1$ at time t, it has received at least $2f+1$ *(echo, k)* messages, i.e. at least $f+1$ *(echo, k)* messages from correct processes. Every other correct process must receive these *(echo, k)* messages by time $t+\epsilon$, and therefore broadcast *(echo, k)*. By time $t+\epsilon+\tau^+$ every correct process receives at least $2f+1$ *(echo, k)* messages and thus switches to round $k+1$. □

The following simple Lemma 3.3 follows immediately from the property P2. It is hence true for any implementation of round synchronization that respect unforgeability and will be used frequently in our proofs.

Lemma 3.3. *Let p_f be the first correct process that switches to some round k at time t. Then no correct process can switch to a larger round k' before $t+2\tau^-(k'-k)$.*

Proof. By induction on $l = k' - k$. For $l = 1$ Lemma 3.3 is identical to unforgeability and therefore true. Assume that no correct process could have switched to round $k+l$ before $t+2\tau^- l$ for some l. Thus no processes may enter round $k+l+1$ before $t+2\tau^- l + 2\tau^- = t + 2\tau^-(l+1)$ following unforgeability. Thus Lemma 3.3 is true for $l+1$. □

We will now show how the results of Theorem 3.2 and Lemma 3.3 translate into a result on the maximum round skew according to Definition 1.1 in our partially synchronous model. Note carefully, however, that this result does not apply to the initialization phase since all processes must already be up.

Theorem 3.4. *For the primitive in Figure 1 there exists a constant D_{CB} such that $\mid R_i(t) - R_j(t) \mid \leq D_{CB}$ for all processes p_i and p_j with their local round numbers $R_i(t)$ and $R_j(t)$ at every time t. $D_{CB} = \lfloor \mathcal{P} + \frac{1}{2} \rfloor$.*

Proof. The relay property P3 states that the maximum difference in the time of two correct processes to switch to a round is $\epsilon+\tau^+$. Let p_f be the first and p_l be the last correct process to switch to a given round k. Let p_f do so at instant t. According to Lemma 3.3, the maximum number of rounds any correct process could switch to, before the last correct process switches to k is $\lfloor \frac{\epsilon+\tau^+}{2\tau^-} \rfloor = \lfloor \frac{2\tau^+ - \tau^-}{2\tau^-} \rfloor = \lfloor \mathcal{P} - \frac{1}{2} \rfloor$. Since at time t there is already a difference of 1 in the rounds of p_f and p_l, the maximum round skew is $\lfloor \mathcal{P} + \frac{1}{2} \rfloor$. □

Remark Note that for the primitive from Figure 1 all round numbers k have to be observed concurrently.

4 Round Synchronization by Modified Consistent Broadcasting

This sections primitive (Figure 2) is the core of the initialization algorithm of Section 5.2 since it can be used during the initialization phase i.e. when $N_{up} < 3f + 1$. In the analysis below, we will assume a system of $N \geq 3f + 1$ processes, with $N_{up} < 3f + 1$ up and running initially. All N_{up} processes, except the at most f faulty ones among those, are initially synchronized. Note that considering a fixed N_{up} allows us to show how our primitive works with such a small number of participating processes.

```
for each correct process
VAR k : integer := 0;

if received (init, k) from at least $f + 1$ distinct processes
  → send (echo, k) to all;
fi

if received (echo, k) from at least $f + 1$ distinct processes
  → send (echo, k) to all;
fi

if received (echo, k) from at least $2f + 1$ distinct processes
  → accept (round k);
    $k := k + 1$;
    send (init, k) to all; /* start next round */
fi

if received (echo, l) from at least $f + 1$ distinct processes where $l > k + 1$
  → accept (round l-2);
    for $i := k$ to $l - 2$
       → send (echo, i) to all;
    $k := l - 1$; /* jump to new round */
    send (echo, k) to all;
fi
```

Figure 2: The MCB Primitive

We will see that the modified consistent broadcast (MCB) primitive of Figure 2 guarantees that two correct active processes are never too far apart from each other in terms of round numbers. We cannot guarantee the same properties of round synchronization as we could if we had $N_{up} \geq 3f + 1$, however. In particular MCB cannot guarantee progress.

The MCB primitive of Figure 2 differs from the CB primitive of Figure 1 in two respects:

1. *The* 4^{th} *if*[3]. Without the 4^{th} *if* it could be that, when $N_{up} < 3f + 1$, two processes' round numbers could be apart arbitrarily. The faulty processes could help $f + 1$ processes to switch rounds arbitrarily often, while they let other processes behind, which would not receive the needed $2f + 1$ *(echo)* messages because there are not enough correct processes in the system yet. The fourth *if* allows such processes to catch up. When a process uses the fourth *if* to switch rounds the loop guarantees that *(echo)* messages are sent for all rounds smaller than the round the process has switched to.

2. *The built-in round number.* The MCB primitive has to distinguish between *(echo)* messages from its current round and messages from rounds ahead. Therefore the primitive must be aware of the current local round number.

Since we have less processes in the system than the original consistent broadcast primitive needs, we can only guarantee weaker properties of round synchronization when $N_{up} < 3f + 1$:

Definition 4.1. *Properties of Weak Round Synchronization.*

- *P1W. Weak Correctness. If at least $f+1$ correct processes switch to round k by time t, then every correct process switches to round $k-1$ by time $t+2\tau^+$.*

- *P2. Unforgeability. If no correct process switches to round k by time t, then no correct process switches to round $k+1$ by $t + 2\tau^-$ or earlier.*

- *P3W. Weak Relay. If a correct process switches to round k at time t, then every correct process switches to round $k-2$ by time $t + \epsilon$.*

The following Theorem 4.2 shows that MCB satisfies P1W, P2 and P3W. Note that it is also valid for $N_{up} < 2f + 1$, although it is trivial to see from Figure 2 that no correct process can make any progress in this case.

[3]Note that a similar construct is used in a clock synchronization algorithm for partially synchronous systems in [4]. In [2] it is used in a phase protocol which is part of a consensus algorithm.

Theorem 4.2. *The broadcast primitive from Figure 2 achieves the properties P1W, P2 and P3W for $N \geq 3f+1$ with any N_{up}.*

Proof. Weak Correctness. Processes can switch rounds by the third and the fourth *if.* If round k is the maximum round in the system all $f+1$ correct processes must have switched using the third if, and therefore have sent *(init, k)* by time t. These $f+1$ correct processes receive those messages by time $t+\tau^+$ and therefore send *(echo, k)* to all. All correct processes must receive those $f+1$ *(echo, k)* messages by time $t+2\tau^+$ and therefore switch to round $k-1$ by the fourth *if*, if they have not already done so.
If round k is not the maximum round, processes could switch to round k using the third or the fourth if. A correct process enters round k using the fourth *if* if it has seen at least $f+1$ *(echo, $k+1$)* messages. Thus at least one correct process p is in round $k+1$. That is at least one correct process has switched to round $k+1$ by the third *if.* Therefore *(echo, k')* for all $k' \leq k$ was sent by at least $f+1$ correct processes before t and must be received by every correct process by time $t+\tau^+$. Among those messages are at least $f+1$ *(echo, k)* messages which are received by all correct processes, so all correct processes will enter round $k-1$ by time $t+\tau^+$.

Unforgeability. Switching rounds can be done at a process by two rules. The proof for both is by contradiction.
Assume that there is a process p_a that switches to round $k+1$ before instant $t+2\tau^-$ using the third *if.* p_a switches because it has seen $2f+1$ *(echo, k)* messages. That are at least $f+1$ *(echo, k)* messages sent by correct processes among those. These messages must have been sent before $t+\tau^-$. Correct processes only send *(echo, k)* when they have seen $f+1$ *(init, k)* or *(echo, l)* messages for a $l \geq k$, however. That is, at least one correct process must have sent a message for round l with $l \geq k$ before t. By assumption no round k messages is sent before t. Since all round l messages for $l > k$ are sent after round k messages at a process no such message has been sent by t which provides the required contradiction.
Assume that there is a process p_s that switches to round k+1 before instant $t+2\tau^-$ using the fourth *if.* p_s accepts because it has seen $f+1$ *(echo, l)* messages for some $l > k+1$. That is, at least one *(echo, l)* message must have been sent by a correct process p_c before $t+\tau^-$. p_c has sent it, because it has seen at least $f+1$ messages for round x with $x \geq l$. These messages must have been sent before t. Since $x > k$, messages for round x are sent by a process only after round k messages. But by P2's assumption no round k message was sent before t, which again provides the contradiction.

Weak Relay. Assume k is the maximum round in the system. A process must switch to round k using the third *if* because it has received at least $2f+1$ *(echo, $k-1$)* messages. Among those are at least $f+1$ *(echo, $k-1$)* messages sent by

correct processes. Those messages must be received by all correct processes by time $t+\epsilon$ and therefore they switch to round $k-2$ (using the fourth *if*).
If k is not the maximum round in the system, at least one correct process has already switched to a round $k' > k$ at t' with $t' \leq t$ using the third *if*. We have shown in the previous paragraph that all correct processes must switch to round $k'-2$ by time $t'+\epsilon$ so all correct processes must also switch to round $k-2$ by $t+\epsilon$. □

Now we will see how P3W (weak relay) and Lemma 3.3 can be used to give a maximum round skew for the MCB primitive so that the synchrony condition from Definition 1.1 is satisfied even with $N_{up} < 3f+1$.

Theorem 4.3. *For the primitive in Figure 2 there exists a constant D_{MCB} such that $\mid R_i(t) - R_j(t) \mid \leq D_{MCB}$ for all processes p_i and p_j at every time t. $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$.*

Proof. Let p_f be the first process to switch the maximum round in the system at instant t and let $R_f(t) = k+2$. Weak relay guarantees that every correct process will enter round k by time $t' = t+\epsilon$. Following Lemma 3.3 the maximum round a correct process could have switched to by time t' is round $k + D_{MCB}$ iff $\epsilon \geq 2\tau^-(D_{MCB}-2)$, thus $D_{MCB} \leq \frac{\epsilon}{2\tau^-} + 2 = \frac{1}{2}\mathcal{P} + \frac{3}{2}$. □

Remark Theorem 4.3 shows that the *Synchrony Condition* from Definition 1.1 is satisfied even if there are less than $3f+1$ processes running. One might conclude that clock synchronization or similar problems can be solved using this primitive with $N \leq 3f$. This is not true of course: After all, in such a system there is no progress guarantee. Progress depends on the behaviour of the faulty processes, since the 3^{rd} *if* needs $2f+1$ messages of distinct processes. Byzantine nodes could make the system switch several rounds and then stay in a certain configuration for an arbitrary long time.
The 4^{th} *if* only guarantees that every process reaches round $x-2$ if one process reaches round x. Therefore it is not possible to bound the maximum difference ($\Delta_{\sigma_{max}}$) of the times when two processes switch to a given round when not enough correct processes are booted. If at least $2f+1$ correct processes are booted and synchronized, $\Delta_{\sigma_{max}}$ can of course be bounded. But a process cannot always determine locally that $2f+1$ correct processes are synchronized, and so this property cannot be used in an application.

Remark Theorem 4.3 statisfies the synchrony condition from Definition 1.1 bounded by D_{MCB} for any number of running processes N_{up}. For $N_{up} \geq 3f+1$ the result for the classic CB primitive from Section 3 applies also to MCB and we get two bounds for the synchrony condition: $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$ and $D_{CB} = \lfloor \mathcal{P} + \frac{1}{2} \rfloor$. Note that we have a MCB' primitive with $D_{MCB'} = \lfloor \frac{1}{2}\mathcal{P} + \frac{1}{2} \rfloor$ which is not shown here, since we have no initialization solution for it yet.

Lemma 4.4 is required in Section 5.2.3 to show the quality of an event. That event is based on messages that were sent by $N - f$ distinct processes. With the knowledge that $f + 1$ processes are in front we can be sure that at least one of the $N - f$ messages was sent by a process from the maximum rounds.

Lemma 4.4. *For the primitive in Figure 2 there are always at least $f + 1$ correct processes within the 2 largest rounds.*

Proof. Let p_f be the first correct process that enters the maximum round at instant t. $R_f(t)$ is the maximum round in the system at t. It has entered this round, because it has seen at least $2f + 1$ *(echo, round $R_f(t) - 1$)* messages, i.e. at least $f + 1$ sent by correct processes. Correct processes never send *(echo)* messages for round numbers larger than their local ones. Therefore at least $f + 1$ processes must be in round $R_f(t)$ or $R_f(t) - 1$ at time t. □

For the initialization solution of Section 5.2 it has to be guaranteed that at least $f + 1$ *(init, k)* messages are sent for every round k. For this purpose we need the following Lemma 4.5.

Lemma 4.5. *For the MCB primitive of Figure 2 it is guaranteed that for every round $k > 0$ and smaller than the maximum round in the system (init, k) messages were sent by at least $f + 1$ correct processes.*

Proof. Processes can switch rounds by the third and the fourth *if*. If a process switches to the maximum round k in the system it must use the third *if* and therefore must send *(init, k)*. Then it executes the MCB primitive an waits for $2f + 1$ *(echo, k)* messages. Processes only send *(echo)* messages for rounds less or equal their local round, thus at least $f + 1$ correct processes must switch to round k before the first process switches to round $k + 1$. These $f + 1$ correct processes must have switched to k by the third *if* and therefore $f + 1$ correct processes must have sent *(init, k)*. □

4.1 MCB using Perception Vectors

Although MCB can deal with arbitrary N_{up}, it works only if all processes are up and initially synchronized right from the start. If processes can join the system when it already made some progress, however, the algorithm cannot guarantee D_{MCB}: A late joiner will start its initialization algorithm by asking the other processes about their round numbers. Since we can only guarantee that their round numbers are within D_{MCB} the answers an initializing process gets can be from different rounds. We must hence find a fault-tolerant algorithm that uses this information to guarantee that synchronization is reached within a given time bound. In this Section we will see a primitive (shown in Figure 3) that guarantees the properties P1W, P2 and P3W and is capable of dealing with diverse round numbers.

```
for each correct process
GLOBAL VAR k : integer;

if received (init, x) from $p_s$
  → $vInit_s := max(x, vInit_s)$;
    $vEcho_s := max(x - 1, vEcho_s)$;
    decide();
fi

if received (echo, x) from $p_s$
  → $vEcho_s := max(x, vEcho_s)$;
    $vInit_s := max(x, vInit_s)$;
    decide();
fi
```

Figure 3: MCB with Perception Vectors

This section's algorithm uses an array, or perception vector, *vEcho* that stores, for every process, the maximum round number it has sent via an *(echo)* message. If the last message received from another process was an *(init, x)* message, $x - 1$ is written into that vector. This information is sufficient, since if a process sends an *(init, x)* message it has accepted round $x - 1$ and therefore must have sent *(echo,* $x - 1$*)*. The array *vInit* stores the similar information for *(init)* messages.

After each message it receives, the primitive in Figure 3 decides what to do based on the information it has received from the other processes. The *decide()* function shown in Figure 4 sends the necessary messages. Note that the reception of *(echo, k)* also implies the reception of all *(echo, x)* messages for every $x \leq k$.

In the following formal definitions of perception vectors and the functions that need them as input parameter we use sets instead of arrays because this simplifies the presentation.

Definition 4.6. *Perception Vector:* $vMessage_q = \{(p, r) \mid r$ *is the maximum round process* p *has sent a message for to process* $q\}$*, and* $M_{i,q} = \{(p, r) \mid (p, r) \in vMessage_q \wedge r = i\}$*.*

Two special cases of a perception vector used in our algorithm are defined in Definitions 4.7 and 4.8.

Definition 4.7. $vEcho_q = \{(p, r) \mid r$ *is the maximum round process* p *has sent an (echo) for to process* $q\}$*, and* $E_{i,q} = \{(p, r) \mid (p, r) \in vEcho_q \wedge r = i\}$*.*

Definition 4.8. $vInit_q = \{(p, r) \mid r$ *is the maximum round process* p *has sent an (init) for to process* $q\}$*, and* $I_{i,q} = \{(p, r) \mid (p, r) \in vInit_q \wedge r = i\}$*.*

The Function *maxRound()* of Definition 4.9 is used to find the maximum round we can be sure a correct process has sent a message for. It is used in conjunction with *vInit* for deciding what *(echo)* to send (corresponding to the first *if* in Figure 2). Applying *maxRound()* to *vEcho* is used to implement the second and fourth *if*. From now on we will omit the index q since the functions are used locally by processes that have no knowledge on the perception vectors of other processes .

Definition 4.9. *maxRound(vMessage):*

$$maxRound(vMessage) = \begin{cases} \text{the largest } i \text{ that} & \\ \sum_{i \le j} \mid M_j \mid \geq f + 1 & \text{if } \exists i \\ & \\ INVALID\_ROUND & \text{if } \neg\exists i \end{cases}$$

The Function *newRound(vMessage)* of Definition 4.10 will be used with *vEcho* to find out when a process can start a new round based on the third *if*. We only consider three adjacent rounds instead of all rounds to guarantee that *(init)* messages are not sent for arbitrary small round numbers (this is needed during initialization when *(init)* messages have additional semantics). On the other hand weak relay (P3W) guarantees that every correct process switches to round $k - 2$ if a correct process has switched to round k. For liveness the use of these three adjacent rounds in the function is sufficient.

Definition 4.10. *newRound(vMessage):*

$$newRound(vMessage) = \begin{cases} \text{the largest } i \text{ that} & \\ \mid M_i \mid + \mid M_{i+1} \mid + \mid M_{i+2} \mid \geq 2f + 1 & \text{if } \exists i \\ & \\ INVALID\_ROUND & \text{if } \neg\exists i \end{cases}$$

```
procedure decide()
begin
    /* this block corresponds to the modified 4th if */
    $r := maxRound(vEcho) - 1$;
    if $r \neq INVALID\_ROUND$ and $r \geq k$
      → send (echo, r) to all;
        if $r > k$
          → accept (round $r - 1$);
            $k := r$;
        fi
    fi

    /* this block corresponds to the 1st if */
    $r := maxRound(vInit)$;
    if $r \geq k$
      → send (echo, k) to all;
    fi

    /* this block corresponds to the 2nd if */
    $r := maxRound(vEcho)$;
    if $r \geq k$
      → send (echo, k) to all;
    fi

    /* this block corresponds to the 3rd if */
    $r := newRound(vEcho)$;
    if $r \neq INVALID\_ROUND$ and $r \geq k$
      → accept (round r);
        $k := r + 1$;
        send (init, k) to all;
    fi
end;
```

Figure 4: The *decide()* function

The algorithm of Figure 3 only writes the information it gets from an incoming message into its perception vectors *vEcho* and *vInit* and lets the *decide()* function of Figure 4 do the sending of messages and the accepting of rounds.

5 Initialization for Consistent Broadcasting

The initialization scenario in [9] is very simple: All processes are assumed to be running and listening to the network right from the beginning. In this case the start of round 0 is quite simple: The processes decide independently when they want to start round 0 and then send an *(init, 0)* message. Accepting this message is done by the consistent broadcast primitive, which guarantees that after the $f + 1^{th}$ correct process has sent *(init, round* 0*)* every correct process decides to switch to round 1.

This scenario, however, is not appropriate for a network start, since we cannot assume that every correct process is started when the first correct process sends its *(init, 0)* message. In fact, when the first started process sends its *(init, 0)* message, no other process could be listening. In common system models this would be described as $N - 1$ omission faulty processes.

If the processes have a priori knowledge about the maximum difference of starting times of any two processes, a simple solution to this problem would be to use timeouts to assure that all processes are booted. A disadvantage of this solution is that if the timing assumption is violated the algorithm fails. Therefore our goal is a timer free initialization algorithm that allows us to start a CB primitive after booting.

The above described initialization algorithm from [9] works because the knowledge that all processes are up is implicit at every process. Section 5.1 will introduce an initialization algorithm that starts round synchronization without any assumption on the other processes but learns during initialization when enough processes are up to start the classic CB primitive.

We are, however, equipped with the MCB primitive which satisfies the synchrony condition from Definition 1.1 for any number of processes. We only have to introduce a solution that guarantees the synchrony condition also during initialization, when N_{up} is not constant, as in the analysis of Section 4, but increasing. A solution can be found in Section 5.2.

5.1 A $4f$ solution

The above mentioned trivial initialization algorithm from [9] works only if there are at least $2f + 1$ correct running processes. With only one adaptation this algorithm can be used to devise a solution that works in our system model provided that $N \geq 4f + 1$. A process does not decide independently when to send an *(init, 0)* message, but sends it when it has received messages from $3f + 1$ different processes. This means that at least $2f + 1$ correct processes are started and listening. The algorithm can be seen in Figure 5. It starts with sending a *(ping)* message. Every received *(ping)* is answered with a *(pong)*. To make sure that a pair of processes don't miss each other, a process must send its *(ping)* after it has started listening to the network.

```
send (ping) to all;

if received (ping) from processor $p_s$ then
  → send (pong) to $p_s$; /* tell $p_s$ that I am not started yet */
fi

if received (ping) or (pong) from at least $N - f$ distinct processes
  → send (init, round 0) to all;
    execute Consistent Broadcast Primitive;
fi
```

Figure 5: Initialization Algorithm for $N \geq 4f + 1$

A liveness property of this algorithm would be that *round 0* has to be started eventually. Since the processes wait for $N - f \geq 3f + 1$ messages, the only way to guarantee that they eventually receive them is that there are $3f + 1$ correct processes in the system, which means the number of processes in the network must be $N \geq 4f + 1$, if there are f faulty processes. Since consistent broadcasting needs only $N \geq 3f + 1$ this means that our solution adds a penalty of f additional processes for handling initialization properly. Still it is an instance of an initialization algorithm that does not use timeouts or repeated sending of messages (heartbeats).

Remark Note that in the initialization algorithm of Figure 5 for starting the CB primitive a process waits for $N - f$ *(ping)* or *(pong)* messages. $N - f$ is preferable to $3f + 1$ because for large N it is possible that cliques are built, if processes wait for $3f + 1$.
Assume that the Consecutive Consistent Broadcast is already running and there are $3f + 1$ late starters, all sending a *(ping)* at time t. They could receive *(ping)* from the others at $t + \tau^-$, while the *(echo, x)* from advanced processes can be received as late as $t + 2\tau^+$. The clique of late starters now has a time interval of $2\tau^+ - \tau^-$ to switch rounds. Since it takes at least $2\tau^-$ for a round switch, the clique could come to a round number of $\frac{2\tau^+ - \tau^-}{2\tau^-} = \mathcal{P} - \frac{1}{2}$ before it gets synchronized. Such a behaviour cannot satisfy the synchrony condition from Definition 1.1.

Correct late starters that do not receive $N - f$ *(ping)* or *(pong)* messages can initialize themselves by the inactive integration described in [9]. A late starter starts with listening to the network to learn about system state. Then it waits for one round and joins the system. This is guaranteed to work since we guarantee progress by providing $N \geq 4f + 1$.

We will not work out a complete round synchronization solution based upon combining this simple initialization algorithm with original CB, however, since one can do better: In the following Section 5.2, we will introduce an initialization algorithm that needs only $N \geq 3f + 1$ and works in conjunction with MCB.

5.2 Initialization with $N \geq 3f + 1$ processes

By now we are equipped with the MCB primitive that guarantees the synchrony condition from Definition 1.1 for any N_{up} and with a timer-free initialization algorithm. We can now bring it all together to introduce our solution for initialization. A process starts initialization with the sending of an *(echo, 0)* message, let p_i be that process. The receiving processes will recognize that it was the first message p_i has sent them, and will answer with some *(echo, k)*, telling p_i about their local round number. This protocol can be seen as adaptation of the ping-pong protocol of the previous Section 5.1. So it is guaranteed that any two correct processes never miss each other.

If the *(echo, k)* messages from the answering processes are from *round 0* no round switching has taken place so far, and initialization is very simple: The $2f + 1^{st}$ correct running process will bring liveness into the system and round synchronization is initialized. If, however, faulty processes have forced several round switches, the answers p_i receives can be from different rounds. Our MCB algorithm based upon perception vectors can handle such messages. A process uses it to get a better view of the system with every message it receives. After p_i has received $f + 1$ messages it can calculate a round number for the first time. These messages, however, can include messages sent by faulty processes and therefore the difference of p_i's round number to the others cannot be guaranteed to satisfy the synchrony condition. Therefore we introduce the passive and active mode.

A running process starts in passive mode. It delivers no service to the application until it is ready to do so. In our case in passive mode no rounds are accepted (telling the application the current round number). The process, however, sends all messages required for MCB. Such processes cannot violate the synchrony condition since they have no local round numbers yet. But eventually they must accept a round number. A process in passive mode can do so, when it is sure that its calculated round number (which it did not tell the application) is good enough.

Our solution will switch to active mode, when it has received $f+1$ *(init)* messages for a round. Recall the MCB using perception vectors. We accept rounds, and therefore generate *(init)* messages, based on messages from adjacent rounds. We will show that at least one of those adjacent rounds will be near the maximum round in the system, and therefore we will be able to guarantee the quality of *(init)* messages and therefore calculate a maximum round skew D_{max} that satisfies the synchrony condition during initialization. Note that Lemma 4.5 guarantees that at least $f+1$ *(init, k)* messages are sent for every round k.

5.2.1 The Active Broadcast Primitive

The active primitive presented in this Section as seen in Figure 6 is based on MCB with perception vectors from Section 4.1.

```
for each correct process
GLOBAL VAR k : integer;

if received (init, round x) from $p_s$
  → $vEcho_s := max(x-1, vEcho_s)$;
    $vInit_s := max(x, vInit_s)$;
    decide();
fi

if received (echo, round x) from $p_s$
  → $vEcho_s := max(x, vEcho_s)$;
    $vInit_s := max(x, vInit_s)$;
    decide();
fi

/* *** for initialization of new processes: *** */
if received a message from a new process $p_s$
  → send last (echo) message sent to $p_s$; /* (echo, $k$ or $k-1$) */
fi
```

Figure 6: Active Primitive

To satisfy the requirements of initialization we have to make some adaptations to MCB: We need a rule to answer messages from processes that have sent messages the first time (ping-pong principle). We also have to avoid the clique problem explained in Section 5.1. So we change $2f+1$ to $N-f$ in the function *newRound()* (which is equivalent to the third *if*).

Definition 5.1.

$$newRound(vMessage) = \begin{cases} \textit{the largest } i \textit{ that} & \\ \mid M_i \mid + \mid M_{i+1} \mid + \mid M_{i+2} \mid \geq N - f & \textit{if } \exists i \\ & \\ INVALID\_ROUND & \textit{if } \neg\exists i \end{cases}$$

Remark That *newRound(vMessage)* checks for $N-f$ messages is also required in Lemma 5.4 to guarantee the quality of *(init)* messages.

5.2.2 The Passive Broadcast Primitive

When a process boots, it immediately starts passive mode and executes the passive primitive from Figure 7. First a process sends an *(echo, 0)* to all. Other processes that are already up will recognize it as the first message of that process and answer with *(echo)* messages. The information of the messages the initializing process receives is written into its perception vectors. Still *(init)* messages have to be stored separately too, to find out when $f+1$ *(init)* messages of one round have been received. When this happens the process changes to active mode.

The perception vectors are processed in the function *passive_decide()* which is identical to the function *decide()* from the active primitive except that it does not tell the application the round number (by accepting). As long as $N_{up} < N-f$ no round switch will happen since there are not enough processes to send messages. If $N-f \leq N_{up} < N$ we have a system state where progress is possible but not guaranteed since it depends on the behaviour of faulty processes. The synchrony condition, however, is guaranteed for any N_{up} if we are able to show that the round numbers an initializing process reports to its application is not too far apart from the round number of any active process. Will we do this in the next Section 5.2.3.

It is guaranteed that the $N-f^{th}$ correct running process brings the progress guarantee into the system, since it sends the same message as active processes. The first round numbers (which are not accepted) of passive processes can be arbitrarily small. Weak relay also applies to passive processes (only the time bound need not hold since the *(echo)* messages could be retransmitted). So a passive process will reach rounds in front. If the system was not making progress when the $N-f^{th}$ correct process has become running the processes will be able to send the required *(echo)* messages to get progress into the system in bounded time. This also will be shown in Section 5.2.3.

```
for each correct process
**GLOBAL VAR** k : integer := 0;

send *(echo, 0)* to all;

**if** received *(init, round x)* from $p_s$
  $\rightarrow vInit_s := max(x, vInit_s)$;
     $vEcho_s := max(x-1, vEcho_s)$;
     *passive_decide()*;
     **if** received (init, x) from at least $f+1$ distinct processes
       $\rightarrow$ unblock *accept (round k)*;
          start active primitive;
     **fi**
**fi**

**if** received *(echo, round x)* from $p_s$
  $\rightarrow vEcho_s := max(x, vEcho_s)$;
     $vInit_s := max(x, vInit_s)$;
     *passive_decide()*;
**fi**

/* *** for initialization of new processes: *** */
**if** received a message from a new process $p_s$
  $\rightarrow$ send last *(echo)* message sent to $p_s$; /* *(echo, k or k-1)* */
**fi**
```

Figure 7: Passive Primitive

5.2.3 Analysis

Definition 5.2. *Maximum local round: We define $R_{max}(t)$ as the maximum local round any correct running process is in at instant t.*

The following Lemmas 5.3 and 5.4 will be used in Lemma 5.5 that gives bounds of the first accepted round number at a running process in respect to $R_{max}(t)$, when t is the instant the process accepts.

Lemma 5.3. *SaveRound(vEcho) always has a result smaller or equal R_{max} when called in the context of the passive_decide() function at a passive process.*

Proof. The result is based on $f + 1$ messages from other processes. At least one of those must be correct and has sent an *(echo, $R_{max}(t)$)* message at instant t. The largest result of *SaveRound(vMessage)* occurs, if the passive process receives f faulty messages that contain round numbers greater the $R_{max}(t)$. Assume the calculation is done at instant t'. Then the result would be $R_{max}(t)$ which must be smaller or equal than any $R_{max}(t'), t' \geq t$ since $R_{max}(t)$ is monotonically increasing. $\square$

In the following Lemma 5.4 we show the quality of *(init)* messages. The result will be a central part of the analysis of our D_{max}, the bound of the synchrony condition.

Lemma 5.4. *A process that sends an (init, round k) at instant t does so for* $k \geq R_{max}(t) - \frac{1}{2}\mathcal{P} - 3$

Proof. Lemma 4.4 states that there must be always $f + 1$ processes within the rounds $R_{max}(t)$ and $R_{max}(t) - 1$ at any instant t.
If a process p_s sends an *(init)* message, it does so because it has seen $N - f$ *(echo)* messages from one of 3 adjacent rounds. At least 1 of these $N - f$ messages must be from a process within the two maximum rounds; the worst *(echo)* such a process can send is *(echo, $R_{max}(t) - 1$)*. It does so at instant t. If the system has not made progress since t and faulty and initializing processes had sent *(echo)* messages for the rounds $R_{max}(t) - 2$ and $R_{max}(t) - 3$. Hence $k \geq R_{max}(t) - 3$.
If the system made progress, a message from the front could take as long as τ^+ to be transfered and change the perception of p_s and $R_{max}(t+\tau^+)$ could be $\frac{\tau^+}{2\tau^-} = \frac{1}{2}\mathcal{P}$ larger than $R_{max}(t)$ (see Lemma 3.3: Shortest round switching time)
If p_s sends its *(init, k)* shortly before instant $t + \tau^+$ then $k \geq R_{max}(t + \tau^+) - \frac{1}{2}\mathcal{P} - 3$. $\square$

Remark Note that the worst *(init, x)* message correct processes can send are the same for active and passive processes.

Since we now have bounded the quality of *(init)* messages (which are used to change from passive to acitve mode) we can now go on with finding a D_{max} that satsifies the synchrony condition from Definition 1.1. First we give bounds for the first accepted round number in relation to $R_{max}(t)$.

Lemma 5.5. *If a process accepts a round number for the first time, its local round number lies between* $\lfloor R_{max} - \frac{3}{2}\mathcal{P} - 4 \rfloor$ *and* R_{max}.

Proof. Passive processes only accept once: When they have received $f + 1$ *(init, round x)* messages. When such a message is sent at intant t at a correct process $R_{max}(t) \leq x + \frac{1}{2}\mathcal{P} + 3$ following Lemma 5.4. This message could take as long as τ^+ to be transfered to a passive process p_i. p_i could be up since shortly before $t+\tau^+$. Therefore processes in front could have made progress of $\frac{\tau^+}{2\tau^-} = \frac{1}{2}\mathcal{P}$ rounds (following Lemma 3.3) and all the messages are not received by p_i because it was down. Therefore $R_{max}(t + \tau^+) \leq x + \mathcal{P} + 3$.
Processes in front came to a round number of $R_{max}(t+\tau^+)$ without p_i has received messages sent by them. If they keep making progress p_i becomes aware of this by time $t + 2\tau^+$. The systems maximum round number again could increase by $\frac{1}{2}\mathcal{P}$. At this instant p_i's perception gets much better caused by the fourth *if.* We must assume now that p_i receives f *(init, x)* messages shortly before $t + 2\tau^+$ which causes p_i to accept *round* $k = x - 1$. $R_{max}(t + 2\tau^+) \leq x + \frac{3}{2}\mathcal{P} + 3$. And therefore p_i's local accepted round number is $k \geq R_{max}(t + 2\tau^+) - \frac{3}{2}\mathcal{P} - 4$

From Lemma 5.3 follows that the local round number is always smaller or equal R_{max}. □

The following Theorem 5.6 guarantees that the synchrony condition from Definition 1.1 is satisfied for the whole system life-cycle, including initialization beginning with 0 correct running processes. The maximum round skew D_{max} is introduced by the initialization. When enough correct processes are up we can guarantee the round skews from CB and MCB.

Theorem 5.6. *For the described system there exists a constant* D_{max} *such that* $| R_i(t) - R_j(t) | \leq D_{max}$ *for all active processes* p_i *and* p_j *at every time* t. $D_{max} = \lceil \frac{3}{2}\mathcal{P} + 4 \rceil$.

Proof. At instant $t + 2\tau^+$ from the previous Proof p_i receives the messages from the processes in front and its local round number increases immedeately from $R_{max} - D_{max}$ to $R_{max} - D_{MCB}$ (caused by the fourth *if*), where $D_{MCB} < D_{max}$. D_{MCB} is the maximum round skew of the MCB primitive of Theorem 4.3. From now on p_i executes MCB as active process. Therefore P1W, P2 and P3W also are guaranteed for p_i and p_i's round number stays within the calculated bounds from time $t + 2\tau^+$ on. □

We now give a bound for the maximum time needed to get progress into the system after the $N-f^{th}$ correct processes has become running. The worst case scenario here is different from that of the worst difference in round numbers. Whereas for the biggest difference in round numbers, the initialization must happen very quickly, so that the initializing process has very few time to gain information on the system state, we now must consider that the processes are not making progress in the worst possible position and it needs the messages by the initializing process to make them switch rounds, so that the necessary *(init)* messages are sent. We start with a definition of the needed time interval.

Definition 5.7. *Initialization Time. Δ_{init} is the time needed to reach synchrony after $N-f$ correct processes are running.*

Since at least one process is in $R_{max}(t)$ when t is the instant the $N-f^{th}$ correct process becomes running, at least one *(init, $R_{max}(t)$)* could be missed by the initializing process. The first round for that it is guaranteed that our initializing process receives at least $f+1$ *(init)* messages for is round $R_{max}(t)+1$. The following Theorem 5.8 gives the latest possible instant when those *(init, $R_{max}(t)+1$)* messages are received at the initializing processes.

Theorem 5.8. *If the $N-f^{th}$ correct process has booted at time t, Consecutive Consistent Broadcast is running synchronously on all correct processes by time $t+\Delta_{init}$ with progress guarantee, where $\Delta_{init}=8\tau^{+}$.*

Proof. Let p_i be the $N-f^{th}$ correct passive process that sends its first *(echo, 0)* message at time t. If there is progress in the system p_i will be initialized very quickly because it receives the necessary *(init)* messages τ^{+} after they were sent.
If the processes in front are not making progress the local round number k of p_i at time $t+2\tau^{+}$ is $k \geq R_{max}(t)-2$ (and is reached using the fourth *if*). p_i sends *(echo, $R_{max}(t)-2$)*, so that by time $t+3\tau^{+}$ every running correct process must have received $N-f$ *(echo, $R_{max}(t)-2$)* messages.
Every correct process now switches to round $R_{max}(t)-1$ and sends an *(init, $R_{max}(t)-1$)* message. It could be that p_i does not receive $f+1$ *(init, $R_{max}(t)-1$)* messages, because there were to much processes already in round $R_{max}(t)-1$. So p_i not necessarily switches to the active primitive.
By time $t+5\tau^{+}$ all correct processes switch to round $R_{max}(t)$ and by time $t+7\tau^{+}$ to round $R_{max}(t)+1$. Therefore p_i and all other correct processes receive at least $f+1$ *(init, $R_{max}+1$)* messages by time $t+8\tau^{+}$ or earlier. At this instant at least $N-f$ correct processes run the active primitive and therefore consecutive consistent broadcast is running properly in the system. $\square$

6 Conclusions

In this thesis we have seen a solution for an initialization of a network where the processes provide round synchronization satisfying the *Synchrony Condition*: $\mid R_i(t) - R_j(t) \mid \leq D_{max}$ during the whole system life-cycle. The algorithm bases on the non-authenticated broadcast primitive by Srikanth and Toueg [9].

First we have seen that the non-authenticated broadcast primitive has a fixed maximum round skew D_{CB} when executed on a partially synchronous system with $N \geq 3f + 1$. We have shown that $D_{CB} = \lfloor \mathcal{P} + \frac{1}{2} \rfloor$.

Then we have seen that a small adaptation of the primitive (MCB) guarantees a maximum round skew for any N_{up}. The MCB primitive has a $D_{MCB} = \lfloor \frac{1}{2}\mathcal{P} + \frac{3}{2} \rfloor$. For $N_{up} < 3f + 1$, however, MCB cannot guarantee progress. This modified primitive is the core of the solution for initialization.

Finally we have collected our results to introduce a solution for network initialization. We also have analysed its properties: Only during initialization we have a D_{max} that is larger than D_{CB} and D_{MCB}. When enough correct processes are booted, the performance of our algorithm increases automatically to D_{CB} for a small $\mathcal{P}$ or to D_{MCB} if $\mathcal{P}$ is larger.

Future work on our topic could be the transformation of MCB to a Hybrid Failure Model [6] that has advantageous properties compared to the process failure model. A transformation to a model similar to the Crash-Recovery Model from [1] could explore the properties of our algorithm when N_{up} decreases again after the system was in a stable state.

References

[1] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. In *International Symposium on Distributed Computing*, pages 231–245, 1998.

[2] Hagit Attiya, Danny Dolev, and Joseph Gil. Asynchronous byzantine consensus. *Proceedings of the 3rd ACM Symposium of Distributed Computing*, pages 119–133, August 1984.

[3] Hagit Attiya and Jennifer Welch. *DISTRIBUTED Computing. Fundamentals, Simulations and Advanced Topics.* McGraw-Hill, 1998.

[4] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35:288–323, April 1988.

[5] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, July 1982.

[6] Gérard Le Lann and Ulrich Schmid. How to implement muteness detectors in partially synchronous systems with byzantine processors and link faults. *Technical Report 183/1-XXX, Dept. of Automation, TU Vienna (DRAFT)*, February 2002.

[7] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, 1984.

[8] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[9] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34:626–645, July 1987.