Self-Stabilizing Byzantine Token Circulation

Ariel Daliot[1] and Danny Dolev[1]

School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel. {adaliot,dolev}@cs.huji.ac.il

Abstract. There is an abundance of writing about Token Circulation (or leader election). Much of the work is dedicated to self-stabilizing Token Circulation, ever since the publishing of Dijkstra's seminal paper. Few of the papers focus on the Byzantine fault model and, to the best of our knowledge, there is no self-stabilizing Token Circulation algorithm that tolerates Byzantine faults. In this paper, we present an elegant self-stabilizing Byzantine Token Circulation algorithm that has comparably good time and message complexities. It has optimal fairness and every node holds the token $1/n$ part of the time in the long run, where n is the number of nodes in the network. Our protocol is based on a tight Byzantine self-stabilizing pulse synchronization procedure. The synchronized pulses are used as events for initializing Byzantine consensus on the id of the next node to hold the token. When the system performs well the time complexity of our scheme is minimal, merely two communication rounds. When the system converges to a desired state following a chaotic situation the additional overhead is that of Byzantine consensus ($O(f')$, where f' is the actual number of faulty nodes).

1 Introduction

There are algorithms for self-stabilizing token circulation (or leader election, see [3][4]) and for fault tolerant leader election, but to the best of our knowledge there is no protocol that both tolerates Byzantine faults and which is also self-stabilizing. In [1] a self-stabilizing mutual exclusion algorithm is presented that tolerates permanent arbitrary faults. These may resemble Byzantine faults but it is assumed that correct nodes can identify faulty nodes which thus makes it a much weaker fault model than Byzantine faults. The problem with handling failures in token circulation is how to maintain the circulation of tokens, despite faults. Once self-stabilization is addressed, the effect of failures is amplified by the loss of synchronization and the need to converge to a stable state.

Token circulation is a method used to solve a variety of problems. It can be used both explicitly and implicitly. One can consider the problem facing a variety of faults model. In this model, we present the solution assuming Byzantine faults, though the general scheme can be optimized to other fault models, as well.

The idea, in a bird's-eye view of the protocol, is to have a distributed background protocol that periodically invokes a tightly synchronized "pulse" at all participating nodes. Once the pulse arrives, the correct nodes exchange the id of the expected next token holder. If they all hold the same view, the relative overhead is minimal and no change occurs in the regular token handling. Otherwise, the nodes initiate consensus on the id of the next token holder and, as long as the system remains stable, no further change will need to take place. The pulse synchronization procedure that is used in this paper is presented in [2].

Another contribution of this paper is the detailed presentation of a Byzantine Consensus protocol that works in a time driven model. The basic protocol follows closely to the Early Stopping Byzantine Agreement protocol of Toueg, Perry and Srikanth [6]. The main difference is that when presented in the timing model, the protocol converges after two rounds of information exchange among the correctly operating nodes. This, typically, is much faster than the upper bound on message delivery between a pair of correct nodes.

2 Network Model and Definitions

The network model is similar to that defined in [2], but to make this paper self-contained the relevant network model assumptions and the basic definitions are described below.

The environment is a network of n processors (nodes), $(p_0, p_2, \ldots, p_{n-1})$, that communicate by exchanging messages. The communication network does not guarantee any order on messages. Individual nodes have no access to a central clock and there is no global pulse system. The hardware clocks (referred to as the *physical timers*) of correct nodes have a bounded drift rate, ρ, from real-time. The nodes can invoke timers that proceed at their physical clock rate. When the system is unstable, the network and all the nodes can behave arbitrarily, though eventually the network performs within the assumption boundaries defined in Definition 2.

Definition 1. *A node is* **non-faulty** *at times that it complies with the following:*

1. *Obeys a global constant* $0 < \rho << 1$ *(typically* $\rho \approx 10^{-6}$*), such that for every real-time interval* $[u, v]$:

   $(1-\rho)(v-u) \leq$ *'physical timer'*$(v) -$ *'physical timer'*$(u) \leq (1+\rho)(v-u)$.
2. *Operates according to the instructed protocol.*

A node is considered **faulty** if it violates any of the above conditions. A faulty node may recover from its faulty behavior once it resumes obeying the conditions of a non-faulty node. For consistency reasons, the recovery is not immediate but rather takes a certain amount of time during which the non-faulty node is not considered correct, although it behaves correctly[1]. To simplify the exposition, as long as a non-faulty node is not considered correct, it is counted as faulty. We later specify the bound on the time it takes a recovered node to be considered correct.

Definition 2. *The* **network assumption boundaries** *are:*

1. *Message passing allowing for an authenticated identity of the senders.*
2. *At most f of the n nodes may not be correct at any given moment.*
3. *Any message sent and received by any non-faulty node will eventually reach every correct node within δ time units.*

Notice that the network assumption boundaries implicitly implies that once the system recovers, all pending messages among correctly operating nodes arrive to their destinations within δ. The reference to correct nodes eliminates the case where the adversary replaces the correctly behaving nodes with faulty nodes continuously and thus prevents the system from converging.

Basic notations:

- $d \equiv \delta + \pi$, where π is the upper bound on message processing time. Thus d is an upper bound on the elapsed real-time from the sending of a message by any correct node until it is received and processed by every correct node.
- A "*pulse*" is an internal event for the triggering of the desired recurrent algorithm, ideally every *cycle* time units. A ***cycle*** is the actual (effective) time interval length between two successive pulses that a node invokes. The cycle length has upper and lower bounds (see [2] for the exact definition of *pulse synchronization*).
- σ represents the upper bound on the real-time between the invocation of the pulses of different correct nodes (*tightness of pulse synchronization*).
- *pulse_conv* represents the convergence time of the underline pulse synchronization module.
- *agreement_duration* represents the maximal real-time required to complete the chosen Byzantine consensus/agreement procedure[2]. It is assumed that $\sigma + agreement\_duration < cycle$.

Using the above notations, a recovered node can be considered **correct** once it goes through a complete synchronization process. This happens once the node invokes a pulse and completes the consensus algorithm.

Definition 3. *A node is* **correct** *following* $pulse\_conv+\sigma+agreement\_duration$ *time of continuous non-faulty behavior.*

The system converges once it is within the network assumption boundaries for some period of time, as specified below. Once the system converges, it is considered *stable* as long as it remains within the network assumption boundaries.

Definition 4. *The system is said to be* **stable** *following continuous adherence to the network assumption boundaries for* $pulse\_conv+\sigma+agreement\_duration$ *time.*

[1] For example, a node may recover with arbitrary variables, which may violate the validity condition if considered correct immediately.

[2] We differentiate between *consensus* on an initial value held by all nodes and *agreement* on an initial value sent by a specific node.

The token circulation concept addresses the problem of determining which node holds the token at a given time. The definition of the token circulation problem in the current environment needs to address the issue of synchronization and the potential existence of faulty nodes. There is no assumption of any fault detection ability and, therefore, the token circulation requirements can be limited only to the well behavior of correct nodes. The above holds for any type of faulty behavior, not only Byzantine.

Since there is no assumption on any external method to synchronize the nodes, the token circulation needs to be driven by the local timers of the nodes and by the messages they exchange. Since there are limits on how close nodes can be synchronized, and there are uncertainties concerning message delivery time, the definition cannot assume instantaneous transfer of the token responsibility. The self-stabilizing Byzantine Token Circulation problem is defined as follows:

Basic definitions:

- $\mathbf{token_q(t)} \in \{\mathbf{p_i}\}_{\mathbf{i=0}}^{\mathbf{n-1}}$ is the node holding the token according to node q's view at time t.
- The **token_state** of the system at real-time t is given by: $token\_state(t) \equiv (token_{p_0}(t), \ldots, token_{p_{n-1}}(t))$.
- A system is in an **agreed token_state** at real-time t if $\forall$ *correct* $p_i, p_j, \quad token_{p_i}(t) = token_{p_j}(t)$.
- Let G be the set of all possible token_states of a system S.
- $s \in G$ is an **eventually-agreed token_state** of the system at real-time t if the system is in an agreed token_state at some real-time t_{agree} in the interval $[t, t + \hat{\sigma}]$, where $\hat{\sigma}$ is some small constant.

In the context of this paper we choose $\hat{\sigma} = \sigma$.

Definition 5. The Self-Stabilizing Token Circulation Problem
As long as the system is stable:
Convergence: *Starting from an arbitrary state, s, the system reaches an eventually-agreed token_state after a finite time.*
Closure: *If s is an eventually-agreed token_state of the system at real-time* t_0 *then* $\forall$ *real time* $t \geq t_0$,

1. *token_state(t) is an eventually-agreed token_state,*
2. *"Fairness": if* $t \longrightarrow \infty$ *then* $\forall i, p_i$ *holds the token an infinite number of times.*

The fairness validity requirement is used as a method for defying triviality such as setting $Token := q$ for all value of time t.

3 Self-stabilizing Byzantine Token Circulation

Self-stabilizing protocols have inherent recurrence or infinite execution in them. This is due to the fact that there is never an agreed t_0 at which all the correct nodes could check the system state concurrently and decide whether to perform a system-wide operation or a reset. This is because the system could enter a brief illegal state just before time t_0 which could render a disagreement on t_0, thus the system would never be able to exit the illegal state. This is especially severe when facing Byzantine faults as otherwise it could be possible for every node to send a "system-state verification" message every time period which could be marked as some t_0. Byzantine nodes can cause such a protocol to continuously reset itself. The recurrence raises the issue of consistency on who holds the token during

the time period when one correct node has decided on the new token while the other correct nodes have not yet done so and still hold the value of the previous tokens. At this brief period there is an alleged violation of agreement. This is addressed by the definition of the eventually-agreed token_state. Whereas a non-stabilizing protocol could stay in an agreed token_state forever, stabilizing protocols will always have to address the issue of agreement state change and the consequent short-lived violation of agreement.

The Self-stabilizing Byzantine Token-Passing algorithm (Figure 1) resembles the basic clock synchronization algorithm in [2], though it uses a time-explicit consensus primitive to enable the protocol to complete faster, when conditions allow for that. Another difference is that instead of agreeing on the Expected Time of the next pulse, the agreement is on the index of the node that will hold the token following the next pulse.

Note that when one considers a different type of fault, the consensus primitive can either be used as is, or can be replaced by a similar one that is better optimized to the specific fault model. The early stopping property can be maintained in other schemes, as well.

Intuitively, once a pulse arrives, the node resets all variables, other than the index to the next token holder. It starts a timer and waits some time to make sure that all nodes have received the pulse. Notice that the pulse synchronization implies that this takes at most $\sigma(1+\rho)$ time units on its clock. The $(1+\rho)$ factor counts for the drift between the local timer readings and real time.

After that pause the nodes invoke Byz-Consensus to agree on the index of the token holder in the following round. When the system is in a stable state, this results in the index each correct node expects. If the resulting index differs from what a node expects it will adjust its index. Note that the Byz-Consensus may return $\perp$. We identify that value with the default value, say p_0.

When the system is in a stable state, the Byz-Consensus completes after two rounds of information exchange among the correct nodes, which may be completed extremely fast. Due to this, the actual time it takes for messages to travel may be much faster than the expected upper bound d. The protocol presents an option to invoke the next pulse after a period of time when the current node holds the token, resulting in a pretty fast token exchange. Otherwise, the token holder changes every *cycle*.

A recovered node will be considered correct after it receives a pulse and completes the Byzantine Consensus that follows it. The proof below shows that this is also the time it takes the system to converge.

Theorem 1. *SS-BYZ-Token solves the Self-Stabilizing Token Circulation Problem in the presence of at most f Byzantine nodes, where $n > 3f$.*

Proof.
Convergence: Similar to the proof in [2]. Once the system is stable all messages among non-faulty nodes arrive within the pre-specified bound. The pulse is invoked within a σ time unit at all non-faulty nodes. Immediately following the pulse a correct node sets the token holder (**Token** variable) to the next agreed token holder. Following the Byz-Consensus, after all nodes have stopped and decided on the same value, the token is reset in order to get a fast token agreement, if the correct nodes have initiated the consensus with different token values (can only happen following system initialization or following an illegal state). Thus, the system converges once it goes through a pulse and completes the Byzantine Consensus. At this point in time, nodes that recovered prior to the pulse can be considered correct.

```
SS-BYZ-Token
at ''pulse'' event                    /* received the internal pulse event */
begin
   Token := p_next;        /* next is the index of the next token holder*/
   Timer := 0;
   Abort any other running instance of SS-BYZ-Token
                and reset all the procedures it had invoked;
   Wait until Timer = σ(1 + ρ) time units;
   p_next := Byz-Consensus(p_((next+1) mod n), Timer);
   Token := p_((next−1) mod n);                        /* posterior adj. */
   Option: Wait a predefined clock time and
           invoke the next pulse.
end
```

Fig. 1. The self stabilizing token distribution protocol

Closure: If $\hat{\sigma} = \sigma$, then when the first node to execute a pulse sets its new token holder, there is a time period, bounded by σ, until the last node to execute its pulse will also set its token to the same value. Therefore, for up to σ time, the system will not be in agreed token_state but rather be in an eventually-agreed token_state, following which it will return to an agreed token_state. Note that when all correct nodes enter the Byz-Consensus algorithm with identical values than the posterior adjustment makes no change. The faulty nodes cannot affect the choice of the next token holder and, therefore, the fairness also holds. □

The Byzantine Consensus procedure of Section 4 possesses "early-stopping" (ES-1 and ES-2) features:

Theorem 2. *SS-BYZ-Token possesses the two early stopping features (ES-1 and ES-2) of the Byzantine Consensus procedure in Section 4:*

1. *When the system is in a stable state, the protocol stops after two rounds of information exchange among correct nodes.*
2. *Should the actual number of Byzantine faults be $f' \leq f$, then the time to reach consensus in the worst case is $O(f')$.*

Proof. The proof follows immediately from the proof of Theorem 4

□

4 The Byzantine Consensus Procedure

The Byzantine Consensus module can use many of the classical Byzantine Consensus algorithms. In order to consider the whole variety of potential problems that may arise when the system is stabilizing, the algorithm is presented in the context of real time assumptions on clocks and timing difference of relative clocks.

We assume that timers of correct nodes are always within $\bar{\sigma}$ of each other. More specifically, we assume that nodes have timers that reset periodically, say at intervals $\leq cycle'$. Let $T_i(t)$ be the reading of the timer at node p_i at real time t. We thus assume that there exists a bound such that for every time t, when the system is stable,

$$\forall i, j \text{ if } \bar{\sigma} < T_i(t), T_j(t) < cycle' - \bar{\sigma} \text{ then } |T_i(t) - T_j(t)| < \bar{\sigma} .$$

The bound $\bar{\sigma}$ includes all drift factors that may occur among the timers of correct nodes during that period. When the timers are reset to zero it might be that, for a short period of time, the timers may be further apart. The pulse synchronization algorithm [2] satisfies the above assumptions and implies that $\bar{\sigma} \geq d$.

The self-stabilization requirement and the deviation that may arise from any synchronization assumption imply that any consensus protocol must be carefully specified. The consensus algorithm will function properly if it is invoked when the timers of correct nodes are within $\bar{\sigma}$ of each other. The only delicate point is to make sure that an arbitrary initialization of the procedure cannot cause the nodes to block or deadlock. Below we show how to update the early stopping Byzantine Agreement Algorithm of Toueg, Perry and Srikanth [6] to overcome self-stabilization and to make it into a general consensus procedure. The resulting algorithm meets the requirements of the Token Circulation algorithm.

In the Token Circulation algorithm the procedure is invoked at a specific timer value. The procedure below is presented as one that can be invoked at any time. The node invokes the procedure with the value to agree on and the timer value at which the procedure is invoked. When nodes invoke the procedure they also consider all messages accumulated in their buffers that were accepted prior to the invocation, if they are relevant to the invoked procedure.

We use the following notations in the description of the consensus procedure:

- Let $\bar{d}$ be the duration of time equal to $(\bar{\sigma} + d) \cdot (1 + \rho)$ time units on a correct node's timer. Intuitively, $\bar{d}$ can be assumed to be a duration of a "phase" on a correct node's timer.
- The *broadcast* primitive is the primitive defined in Section 5 and is an adaptation of the one described in [6]. Note that an *accept* is issued within the broadcast primitive.

The main differences from the original protocol of [6] are:

- Instead of the General in the original protocol we use a virtual (faulty) "General" notion of a virtual node whose value is the assumed value of all correct nodes at a correct execution. It is the value with which the individual nodes invoke the procedure. Thus, every correct node does a Consensus-broadcast of its initial Val in contrast to the original protocol in which only the General does this. If all correct nodes initiate with the same value and at the same timer time this will be the agreed value.
- The Consensus-broadcast primitive has been modified by omitting the code dealing with the *init* messages. All correct nodes send an echo of their initial values as though they previously received the *init* message from the virtual General.
- It is assumed that the broadcast and Consensus-broadcast primitives are implicitly initiated when a corresponding message arrives.

The Byz-Consensus algorithm is presented in a somewhat different style. Each step has a condition attached to it, if the condition holds and the timer value assumption holds, then the step is to be executed. Notice that only the step needs to take place at a specific timer value.

The Byz-Consensus algorithm satisfies the following typical properties:

Termination: The protocol terminates in a finite time;
Agreement: The protocol returns the same value at all correct nodes;
Validity: If all correct nodes invoke the protocol with the same value and time, then the protocol returns that value;

```
Procedure Byz-Consensus(Val, T)
                              /* invoked at node p with timer value T */

broadcasters := ∅;  value =⊥;

Consensus-broadcast(General, Val, T, 1);
by time (T + 2d̄) :
   if accepted (General, v, T, 1) then
         value := v;
by time (T + (2f + 4)d̄)
   if value ≠⊥ then
         broadcast (p, value, T, ⌊(T_i − T)/(2d̄)⌋ + 1);
         stop and return value.
at time (T + 2rd̄)
   if (|broadcasters| < r − 1) then
         stop and return value.
by time (T + 2rd̄)
   if accepted (General, v', T, 1) and r − 1 distinct messages (q_i, v', T, i)
                  where ∀i, j  2 ≤ i ≤ r, and q_i ≠ q_j then
         value := v';
```

Fig. 2. The Byz-Consensus primitive

It also satisfies the following **early stopping** properties:

ES-1 If all correct nodes invoke the protocol with the same consensus value and with the same timer value, then they all stop within two "rounds" of information exchange among correct nodes.

ES-2 If the actual number of faults is $f' \leq f$ then the algorithm terminates by $\min[T + (2f' + 6)\bar{d},\ T + (2f + 4)\bar{d}]$ on the timer of each correct node.

Notice that ES-1 takes in practice significantly less time than the specified upper bound on the message delivery time.

We first prove the properties of the Consensus-broadcast primitive and later we prove the correctness of the Byz-Consensus protocol. The Consensus-broadcast primitive and the Broadcast primitive (defined in the Appendix) satisfy the following *TPS* properties of Toueg, Perry and Srikanth [6], which are phrased in our system model.

TPS-1 *(Correctness)* If a correct node p broadcasts (p, m, τ, k) by $\tau + (2k - 2)\bar{d}$ on its timer, then every correct node accepts (p, m, τ, k) by $\tau + 2k\bar{d}$ on its timer.

TPS-2 *(Unforgeability)* If a correct node p does not broadcast (p, m, τ, k), then no correct node accepts (p, m, τ, k).

TPS-3 *(Relay)* If a correct node accepts (p, m, τ, k) by $\tau + 2r\bar{d}$, for $r \geq k$, on its timer then every other correct node accepts (p, m, τ, k) by $\tau + (2r + 2)\bar{d}$ on its timer.

TPS-4 *(Detection of broadcasters)* If a correct node accepts (p, m, τ, k) by $\tau + 2r\bar{d}$, on its timer then every correct node has $p \in broadcasters$ by $\tau + (2k+1)\bar{d}$ on its timer. Furthermore, if a correct node p does not broadcast any message, then a correct node can never have $p \in broadcasters$.

In addition the Consensus-Broadcast primitive also satisfies:

```
Procedure Consensus-broadcast(General, v, τ, 1)
                    /* invoking a broadcast simulating the General */
          /* nodes send specific message with the same τ only once */
        /* multiple messages sent by an individual node are ignored*/

send (echo, General, v, τ, 1) to all;
by time (τ + d̄)
   if received (echo, General, v, τ, 1) from ≥ n − 2f distinct nodes then
        broadcasters := broadcasters ⋃ {General} ;
   if received (echo, General, v, τ, 1) from ≥ n − f distinct nodes q then
        send (echo′, General, v, τ, 1) to all;
at any time
   if received (echo′, General, v, τ, 1) from ≥ n − 2f distinct nodes then
        send (echo′, General, v, τ, 1) to all;
   if received (echo′, General, v, τ, 1) from ≥ n − f distinct nodes then
        accept (General, v, τ, 1);
```

Fig. 3. Consensus-broadcast

TPS-5 *(Uniqueness)* If a correct node accepts $(General, m, \tau, 1)$, then no correct node ever accepts $(General, m', \tau, 1)$ with $m' \neq m$.

Notice the differences from the original properties. The detection property does not require having $r \geq k$. In general, the relay property holds even earlier than $r \geq k$. The condition $r \geq k$ of when the property can be guaranteed is used to simplify the possible cases. At $r < k$, if an accept takes place as a result of getting $n - f$ echo messages, the adversary may cause the relay to take $3\bar{d}$ by rushing messages to one correct node and delay messages to and from others.

Theorem 3. *The Consensus-broadcast primitive satisfies the Uniqueness property in addition to the four TPS properties.*

Proof.
Correctness: If all correct nodes broadcast $(echo, General, v, \tau, 1)$ at time τ on their timers, then by Lemma 3 every correct node accepts $(General, v, \tau, 1)$ from $n - f$ correct nodes by $\tau + \bar{d}$ on its timer. Thus each correct node sends $(echo, General, v, \tau, 1)$ by that time and will accept $(General, v, \tau, 1)$ by $\tau + 2\bar{d}$ on their timers.

Unforgeability: If all correct nodes hold the same initial value v then no correct node will broadcast $(echo, General, v', 1)$, thus no correct node will receive $n - f$ distinct $(echo, General, v', 1)$ messages. Therefore, no correct node will broadcast $(echo', General, v', 1)$, and no correct node will ever receive $n - 2f$ or $n - f$ distinct $(echo', General, v', 1)$ messages. Thus, no correct node can accept $(General, v', 1)$.

Relay: If a correct node accepts $(General, v, \tau, 1)$ by $\tau + 2r\bar{d}$ on its timer, then it received $n - f$ distinct $(echo', General, v, \tau, 1)$ message by that time. $n - 2f$ of these were sent by correct nodes and by Lemma 3 all of them will reach all correct nodes by $\tau + (2r + 1)\bar{d}$. As a result, all such correct nodes will send $(echo', General, v, \tau, 1)$, which will be received by all correct nodes. Hence, by $\tau + (2r + 2)\bar{d}$ on their timers, all correct nodes will hold $n - f$ distinct

$(echo', General, v, \tau, 1)$ messages and will thus accept $(General, v, \tau, 1)$.

Detection of broadcasters: If a correct node q' accepts $(General, v, \tau, 1)$ by time $\tau + 2r\bar{d}$ on its timer, then node q' should have received at least $n - f$ distinct $(echo', General, v, \tau, 1)$ messages, at least $n-2f$ of which are from correct nodes. Let q be the first correct node to ever send $(echo', General, v, \tau, 1)$. If q sent it as a result of receiving $n - f$ such messages, then q is not the first to send. Therefore, it should have sent it as a result of receiving $n - f$ $(echo, General, v, \tau, 1)$ messages by time $\tau + \bar{d}$. Thus, at least $n-2f$ such messages were sent by correct nodes by time τ on their timers and would arrive at all correct nodes by time $\tau + \bar{d}$ on their timers. As a result, all will have $General \in broadcasters$.

Uniqueness: Notice that if a correct node sends $(echo', General, v, \tau, 1)$ by time $\tau + \bar{d}$, then no correct node sends $(echo', General, v', 1)$ at any later time. Otherwise, similarly to the arguments in proving the previous property we get that at least $n - f$ nodes sent $(echo, General, v, \tau, 1)$ and $n - f$ nodes sent $(echo, General, v', 1)$. Since $n > 3f$, this implies that at least one correct node sent both $(echo, General, v, \tau, 1)$ and $(echo, General, v', 1)$, and this is not allowed.

Also note that if a correct node accepts $(General, v, \tau, 1)$, than at least one correct node sends $(echo', General, v, \tau, 1)$, which yields the proof of the *Uniqueness* property. □

Nodes stop participating in the Byz-Consensus protocol when they are instructed to do so. They stop participating in the broadcast primitive $2\bar{d}$ after they stop the Byz-Consensus.

Definition 6.
A node returned *a value m if it has stopped and returned value* $= m$.
A node p decides *if it stops at that timer time and returns a value* $\neq \perp$.
A node p aborts *if it stops and returns* $\perp$.

Theorem 4. *The Byz-Consensus satisfies the Termination property. When* $n > 3f$*, it also satisfies Agreement, Validity and the two early stopping conditions.*

Proof. We prove the five properties of the theorem. We build up the proof through the following arguments.

Lemma 1. *If a correct node aborts at time* $T + 2r\bar{d}$ *on its timer, then no correct node decides at a time* $T + 2r'\bar{d} \geq T + 2r\bar{d}$ *on its timer.*

Proof.
Let p be a correct node that aborts at time $T + 2r\bar{d}$. In this case it should have identified exactly $r - 2$ broadcasters by that time. By the detection of broadcasters property (TPS-4) no correct node will ever accept $(General, v, T, 1)$ and $r - 2$ distinct messages (q_i, v, T, i) for $2 \leq i \leq r - 1$, since that would have caused all correct nodes to hold $r - 1$ broadcasters by time $T + (2r - 1)\bar{d}$ on their timers. Thus, no correct node can decide at a time $T + 2r'\bar{d} \geq T + 2r\bar{d}$ on its timer. □

Lemma 2. *If a correct node decides by time* $T + 2r\bar{d}$ *on its timer, then every correct node decides by time* $T + 2(r + 1)\bar{d}$ *on its timer.*

Proof. Let p be a correct node that decides by time $T + 2r\bar{d}$ on its timer. We consider the following cases:

1. $r = 1$: No correct node can abort by time $T + 2\bar{d}$, since the inequality will not hold. Node p must have accepted $(General, v, T, 1)$ by $T + 2\bar{d}$. By the relay property (TPS-3) all correct nodes will accept $(General, v, T, 1)$ by $T + 4\bar{d}$ on their timers. Moreover, p invokes **broadcast**$(p, v, T, 2)$, by which the correctness property (TPS-1) will be accepted by all correct nodes by time $T + 4\bar{d}$ on their timers. Thus, all correct nodes will have $value \neq \perp$ and will broadcast and stop by time $T + 4\bar{d}$ on their timers.
2. $2 \leq r \leq f+1$. Node p must have accepted $(General, v, T, 1)$ and also accepted $r - 1$ distinct (q_i, v, T, i) messages for all $i, 2 \leq i \leq r$, by time $T + 2r\bar{d}$ on its timer. By Lemma 1, no correct aborts by that time. By Relay property (TPS-3) each (q_i, v, T, i) message will be accepted by all correct nodes by time $T + (2r + 2)\bar{d}$ on their timers. Node p broadcasts $(p, v, T, r + 1)$ before stopping. By the correctness property, this message will be accepted by all correct nodes by time $T + (2r + 2)\bar{d}$ on their timers. Thus, no correct node will abort by $T + (2r + 2)\bar{d}$ and all correct nodes will have $value \neq \perp$ and will decide and stop by that time.
3. $r = f + 2$. Node p must have accepted (q_i, v, T, i) messages for all $i, 2 \leq i \leq f + 2$, by $T + (2f + 4)\bar{d}$ on its timer, where the $f + 1$ q_i's are distinct. At least one of these $f + 1$ nodes, say q_j, must be correct. By the Unforgeability property (TPS-2) q_j, invoked broadcast (q_j, v, T, j) by time $T + (2j)\bar{d}$ on its timer, and decided. Since $j \leq f + 1$ the above arguments imply that by $T + (2f + 4)\bar{d}$ on their timers all correct will decide.

□

Lemma 2 implies that if a correct node decides at time $T + 2r\bar{d}$ on its timer, then no correct node aborts at round $T + 2r'\bar{d}$. Lemma 1 implies the other direction.

Termination: Lemma 2 implies that if any correct node decides, all decide and stop. Assume that no correct node decides. In this case, no correct node ever invokes a broadcast $(q, v, T, \_)$. By detection of broadcasters property (TPS-4), no correct node will ever be considered as broadcaster. Therefore, by time $T + ((2f + 4)\bar{d}$ on their timers, all correct nodes will have at most f broadcasters and will abort and stop. □

Agreement: If no correct node decides, then all abort, and return to the same value. Otherwise, let p be the first correct node to decide. Therefore, no correct node aborts. The value returned by p is the value v of the accepted $(General, v, 1)$ message. By Properties TPS-3 and TPS-5 all correct nodes accept $(General, v, T, 1)$ and no correct node accepts $(General, v', T, 1)$ for $v \neq v'$. Thus all correct nodes return the same value. □

Validity: Let all the correct nodes begin with the same value v' and invoke the protocol with the same timer time (T). Then, by time $T + \bar{d}$ on their timers, all nodes receive at least $n - 2f$ distinct $(echo, General, v', T, 1)$ messages via the Consensus-broadcast primitive and send $(echo', General, v', T, 1)$ messages to all. Hence, all nodes receive at least $n - f$ distinct $(echo', General, v', T, 1)$ messages by $T + 2\bar{d}$ on their timers and thus accept $(General, v', T, 1)$. Hence in the agreement procedure all correct nodes set their value to v'. By $T + 2\bar{d}$ on their timers, all correct nodes will stop and return v'. □

Early-stopping: The first early stopping property (ES-1) is directly implied from the proof of the validity property. Correct nodes proceed once they receive messages from $n - f$ nodes, thus it is enough to receive messages from all correct nodes. The second Early-stopping property is identical to the proof of the termination property. By time $T + (2f' + 4)\bar{d}$ all will abort unless any correct broadcasts by that time on its timer. This implies that by $T + (2f' + 6)\bar{d}$ on their timers all correct nodes will always terminate, if the actual number of faults f' is less than f. □

Thus the proof of the theorem is concluded. □

4.1 Complexity Analysis of SS-BYZ-Token

The time complexity is the sum of the convergence time of the pulse synchronization procedure and the time complexity of the Byzantine agreement procedure. σ is of the order of $2d$ and $\bar{d}$ is of order $3d$. Therefore, the convergence time is always bounded by $O(f)cycle + 2d + 3(2f + 6)d$ from any arbitrary state (The $O(f)$ is contributed by the convergence of the pulse synchronization module). The accuracy, or the time at which correct nodes are not in an agreed token_state, equals $\sigma = 2d$. The Byz-Consensus procedure has the two early-stopping features presented above. Thus, the time complexity when the system is stable is less than $2d$, the upper bound on the time it takes to have two rounds of information exchange among the correct nodes. The message complexity is $O(nf^2)$. Token Circulation/Leader Election algorithms have a proven message complexity, lower bounds, of $\Omega(nlogn)$ and typically have a time complexity of $o(n)$. Self-stabilizing protocols usually have a higher convergence time. The self-stabilizing non-fault tolerant leader election in [4] stabilizes in $O(n^2)$ steps. In [5] a time-optimal, self-stabilizing, randomized Token Circulation algorithm for anonymous rings that tolerates only transient faults is presented. It converges in an expected $O(n^3)$ steps. Thus SS-BYZ-Token is of comparable complexity to non-stabilizing or non-fault tolerant Token Circulation/Leader Election algorithms while being much more robust.

References

1. R. W. Buskens and R. P. Bianchini, "*Self-stabilizing mutual exclusion in the presence of faulty nodes*", In FTCS95 Proceedings of the 25th International Symposium on Fault-Tolerant Computing Systems, pages 144–153, 1995.
2. A. Daliot, D. Dolev and H. Parnas, "*Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks*", Ariel Daliot, Danny Dolev, and Hanna Parnas. In Proceedings of the Sixth Symposium on Self-Stabilizing Systems, DSN SSS '03, San Francisco, June 2003. See also LNCS 2704.
3. S. Dolev, A. Israeli and S. Moran, " Uniform Dynamic Self-Stabilizing Leader Election", IEEE Trans. Parallel Distrib. Syst. 8(4): 424-440 (1997)
4. F. Fich and C. Johnen, "*A Space Optimal, Deterministic, Self- Stabilizing, Leader Election Algorithm for Unidirectional Rings*", Proc. of the 15th International Symposium on Distributed Computing (DISC'01). Lisbon, October, 2001. also in LNCS 2180, pages 224-239.
5. C. Johnen, "*Bounded Service Time and Memory Space Optimal Self-Stabilizing Token Circulation Protocol on unidirectional rings*", Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004.
6. Sam Toueg, Kenneth J. Perry, T. K. Srikanth, "*Fast Distributed Agreement*", SIAM Journal on Computing, 16(3):445-457, June 1987.

5 Appendix - The Broadcast Primitive

The appendix presents the **Broadcast** (and accept) primitive that is used by the Byz-Consensus procedure presented earlier, in Section 4. The primitive follows the primitive of of Toueg, Perry, and Srikanth [6], though here it is presented in a real-time model.

In the original synchronous model, nodes advance according to phases. This intuitive lock-step process clarifies the presentation and simplifies the proofs. In this section, the discussion carefully considers the various time consideration and proves that nodes can rush through the protocol and do not to need to wait for a completion of a "phase" in order to move to the next step of the protocol.

Note that when a node invokes the procedure it evaluates all the messages in its buffer that are relevant to the procedure.

```
Procedure Broadcast(p, m, τ, k)
                                        /* executed per such quadruple */
          /* nodes send specific message with the same τ only once */
       /* multiple messages sent by an individual node are ignored */

node p sends (init, p, m, τ, k) to all nodes;

by time (τ + (2k − 1)d̄)
   if (received (init, p, m, τ, k) from p then
        send (echo, p, m, τ, k) to all;

by time (τ + 2kd̄)
   if (received (echo, p, m, τ, k) from ≥ n − 2f distinct nodes q then
        send (init′, p, m, τ, k) to all;
   if (received (echo, p, m, τ, k) msgs from ≥ n − f distinct nodes then
        accept (p, m, τ, k);

by time (τ + (2k + 1)d̄)
   if (received (init′, p, m, τ, k) from ≥ n − 2f then
        broadcasters := broadcasters ⋃ {p};
   if (received (init′, p, m, τ, k) from ≥ n − f distinct nodes then
        send (echo′, p, m, τ, k) to all;

at any time
   if (received (echo′, p, m, τ, k) from ≥ n − 2f distinct nodes then
        send (echo′, p, m, τ, k) to all;
   if (received (echo′, p, m, τ, k) from ≥ n − f distinct nodes) then
        accept (p, m, τ, k);
end
```

Fig. 4. Regular Broadcast

The Broadcast primitive satisfies the 4 TPS properties, under the assumption that $n > 3f$. The proofs below follow closely to the original proofs of [6], in order to make it easier for readers that are familiar with the original proofs.

Lemma 3. *If a correct node p_i sends a message at timer time $T_i \leq \tau + r\bar{d}$ on p_i's timer it will be recieved by each correct node p_j by timer time $\tau + (r+1)\bar{d}$ on p_j's timer.*

Proof. Assume that node p_i sends a message at real time t with timer time $T_i(t) \leq \tau + r\bar{d}$. Thus, $T_i(t) \leq \tau + r(\bar{\sigma} + d)(1+\rho)$. It should arrive at every correct timer p_j within $d(1+\rho)$ on any correct node's timer. Recall that $|T_i(t) - T_j(t)| < \bar{\sigma}(1+\rho)$. If $T_j \geq T_i$ we are done. Otherwise,

$$T_j(t) \leq T_i(t) + \bar{\sigma}(1+\rho) \leq \tau + r(\bar{\sigma} + d)(1+\rho) + \bar{\sigma}(1+\rho) \; .$$

By the time (say t') that the message arrives to p_j we get

$$T_j(t') \leq \tau + r(\bar{\sigma} + d)(1+\rho) + \bar{\sigma}(1+\rho) + d(1+\rho) \leq \tau + (r+1)\bar{d} \; .$$

□

Lemma 4. *If a correct node ever sends $(echo', p, m, \tau, k)$ then at least one correct node must have sent $(echo', p, m, \tau, k)$ by timer time $\tau + (2k+1)\bar{d}$.*

Proof. Let t be the earliest timer time by which any correct node q sends the message $(echo', p, m, \tau, k)$. If $t > \tau + (2k+1)\bar{d}$, node q should have received $(echo', p, m, \tau, k)$ from $n - 2f$ distinct nodes, at least one of which from a correct node that was sent prior to timer time $\tau + (2k+1)\bar{d}$. □

Lemma 5. *If a correct node ever sends $(echo', p, m, \tau, k)$ then p's $(init, p, m, \tau, k)$ must have been received by at least one correct node by time $\tau + (2k-1)\bar{d}$.*

Proof. By Lemma 4, if a correct node ever sends $(echo', p, m, \tau, k)$, then some correct node q should send it by time timer $\tau + (2k+1)\bar{d}$. By the procedure, q have received $(init', p, m, \tau, k)$ from at least $n - f$ nodes by timer time $\tau + (2k+1)\bar{d}$. At least one of them is correct who have received $n - 2f$ $(echo, p, m, \tau, k)$ by timer time $\tau + 2k\bar{d}$. One of which was sent by correct node that should have received $(init, p, m, \tau, k)$ before sending $(echo, p, m, \tau, k)$ by timer time $\tau + (2k-1)\bar{d}$. □

Theorem 5. *The broadcast primitive presented in Figure 4 satisfies properties TSP-1 through TSP-4.*

Proof.

Correctness: Assume that a correct node p **broadcast**s (p, m, τ, k) by $\tau + (2k-2)\bar{d}$ on its timer. Every correct node receives $(init, p, m, \tau, k)$ and sends $(echo, p, m, \tau, k)$ by $\tau + (2k-1)\bar{d}$ on its timer. Thus, every correct node receives $n - f$ $(echo, p, m, \tau, k)$ from distinct nodes by $\tau + (2k-1)\bar{d}$ on its timer and accepts (p, m, τ, k).

Unforgeability: If a correct node p does not broadcast (p, m, τ, k), it does not send $(init, p, m, \tau, k)$, and no correct node will send $(echo, p, m, \tau, k)$ by $\tau + (2k-1)\bar{d}$ on its timer. Thus, no correct node accepts (p, m, τ, k) by $\tau + 2k\bar{d}$ on its timer. If a correct node would have accepted (p, m, τ, k) at a later time it can be only as a result of receiving $n - f$ $(echo', p, m, \tau, k)$ distinct messages, some of which must be from correct nodes. By Lemma 5, p should have sent $(init, p, m, \tau, k)$, a contradiction.

Relay: Notice that $r \geq k$, ths (EK note-I don't know what word this should be, thus or this, or if it is code) even if nodes issue an accept at earlier time, the claim holds for the specified times.

The delicate point is when a correct node issues an accept as a result of getting echo messages. If $r = k$ and the correct node, say q, have received $(echo, p, m, \tau, k)$ from $n - f$ nodes by $\tau + 2k\bar{d}$ on its timer. At least $n - 2f$ of them were sent by correct nodes. Since every correct node among these has sent its message by $\tau + (2k - 1)\bar{d}$, all those messages should have arrived to every correct node by $\tau + 2k\bar{d}$ on its timer. Thus, every correct node should have sent $(init', p, m, \tau, k)$ by $\tau + 2k\bar{d}$ on its timer. As a result, every correct node will receive $n - f$ such messages by $\tau + (2k + 1)\bar{d}$ on its timer and will send $(echo', p, m, \tau, k)$ by that time, which will lead all correct nodes to accept (p, m, τ, k) by $\tau + (2r + 2)\bar{d}$ on its timer.

Otherwise, the correct node, say q, accepts (p, m, τ, k) by $\tau + 2r\bar{d}$ on its timer as a result of receiving $n - f$ $(echo', p, m, \tau, k)$ by that timer time. Since $n - f$ of these are from correct nodes, they should arrive at any correct node by $\tau + (2r + 1)\bar{d}$ on their timers. As a result, by $\tau + (2r + 1)\bar{d}$, all correct nodes would send $(echo', p, m, \tau, k)$ and by $\tau + (2r + 2)\bar{d}$ on their timers all will accept (p, m, τ, k).

Detection of broadcasters: As in the original proof, we first argue the second part. Assume that a correct node q adds node p to $broadcasters$. It should have received $n-2f$ $(init', p, m, \tau, k)$ messages. Thus, at least one correct node has sent $(init', p, m, \tau, k)$ as a result of receiving $n - 2f$ $(echo, p, m, \tau, k)$ messages. One of these should be from a correct node that has received the original broadcast message of p.

To prove the first part, we consider two similar cases to support the the Relay property. If $r = k$ and the correct node, say q, accepts (p, m, τ, k) as a result of receiving $(echo, p, m, \tau, k)$ from $n - f$ nodes by $\tau + 2k\bar{d}$ on its timer. At least $n-2f$ of them were sent by correct nodes. Since every correct node among these has sent its message by $\tau + (2k - 1)\bar{d}$, all those messages should have arrived to every correct node by $\tau + 2k\bar{d}$ on its timer. Thus, every correct node should have sent $(init', p, m, \tau, k)$ by $\tau + 2k\bar{d}$ on its timer. As a result, every correct node will receive $n - f$ such messages by $\tau + (2k + 1)\bar{d}$ on its timer and will add p to $broadcasters$.

Otherwise, q accepts (p, m, τ, k) as a result of receiving $(echo', p, m, \tau, k)$ from $n - f$ nodes by $\tau + 2r\bar{d}$ (for $r \geq k$) on its timer. By Lemma 4 a correct node sent $(echo', p, m, \tau, k)$ by $\tau + (2k + 1)\bar{d}$. It should have received $n - f$ $(init', p, m, \tau, k)$ messages by that time. All such messages that were sent by correct nodes were sent by $\tau + 2k\bar{d}$ on their timers and should arrive at every correct node by $\tau + (2k + 1)\bar{d}$ on its timer. Since there are at least $n - 2f$ such messages, all will add p to $broadcasters$ by $\tau + (2k + 1)\bar{d}$ on their timers. □