Self-stabilizing Byzantine Agreement

Ariel Daliot
School of Engineering and Computer Science
The Hebrew University, Jerusalem, Israel
adaliot@cs.huji.ac.il

Danny Dolev*
School of Engineering and Computer Science
The Hebrew University, Jerusalem, Israel
dolev@cs.huji.ac.il

ABSTRACT

Byzantine agreement algorithms typically assume implicit initial state consistency and synchronization among the correct nodes and then operate in coordinated rounds of information exchange to reach agreement based on the input values. The implicit initial assumptions enable correct nodes to infer about the progression of the algorithm at other nodes from their local state. This paper considers a more severe fault model than permanent Byzantine failures, one in which the system can in addition be subject to severe transient failures that can temporarily throw the system out of its assumption boundaries. When the system eventually returns to behave according to the presumed assumptions it may be in an arbitrary state in which any synchronization among the nodes might be lost, and each node may be at an arbitrary state. We present a self-stabilizing Byzantine agreement algorithm that reaches agreement among the correct nodes in optimal time, by using only the assumption of bounded message transmission delay. In the process of solving the problem, two additional important and challenging building blocks were developed: a unique self-stabilizing protocol for assigning consistent relative times to protocol initialization and a Reliable Broadcast primitive that progresses at the speed of actual message delivery time.

Categories and Subject Descriptors: C.2.4 [Distributed Systems]: Distributed applications;

General Terms: Algorithms, Reliability, Theory.

Keywords: Byzantine Agreement, Self-Stabilization, Byzantine Faults, Pulse Synchronization, Transient Failures, Reliable Broadcast.

1. INTRODUCTION

The Byzantine agreement (Byzantine Generals) problem was first introduced by Pease, Shostak and Lamport [11]. It is now considered as a fundamental problem in fault-tolerant distributed computing. The task is to reach agreement in a network of n nodes in which up-to f nodes may be faulty. A distinguished node (*the General* or *the initiator*) broadcasts a value m, following which all nodes exchange messages until the non-faulty nodes agree upon the same value. If the initiator is non-faulty then all non-faulty nodes are required to agree on the same value that the initiator sent.

Standard deterministic Byzantine agreement algorithms operate in the synchronous network model in which it is assumed that all correct nodes initialize the agreement procedure (and any underlying primitives) at about the same time. By assuming concurrent initializations of the algorithm a synchronous rounds structure can be enforced and used to infer on the progression of the algorithm from the point of initialization. Moreover, there is always an implicit assumption about the consistency of the initial states of all correct nodes, or at least a quorum of them.

We consider a more severe fault-model in which in addition to the permanent presence of Byzantine failures, the system can also be subject to severe transient failures that can temporarily throw all the nodes and the communication subsystem out of the assumption boundaries. E.g. resulting in more than one third of the nodes being Byzantine or messages of non-faulty nodes getting lost or altered. This will render the whole system practically unworkable. Eventually the system must experiences a tolerable level of permanent faults for a sufficiently long period of time. Otherwise it would remain unworkable forever. When the system eventually returns to behave according to the presumed assumptions, each node may be in an arbitrary state. It makes sense to require a system to resume operation after such a major failure without the need for an outside intervention to restart the whole system from scratch or to correct it.

Classic Byzantine algorithms cannot guarantee to execute from an arbitrary state, because they are not designed with self-stabilization in mind. They typically make use of assumptions on the initial state of the system such as assuming all clocks are initially synchronized or that the initial states are initialized consistently at all correct nodes (cf. from the very first polynomial solution [8] through many others like [12]). Conversely, A *self-stabilizing* protocol converges to its goal from any state once the system behaves well again, but is typically not resilient to the permanent presence of faults.

In trying to combine both fault models, Byzantine failures present a special challenge for designing self-stabilizing distributed algorithms due to the "ambition" of malicious nodes to incessantly hamper stabilization. This difficulty may be

*Part of the research was done while visiting Cornell University.

PODC'06, July 22-26, 2006, Denver, Colorado, USA.

indicated by the remarkably few algorithms resilient to both fault models (see [4] for a review). The few published self-stabilizing Byzantine algorithms are typically complicated and sometimes converge from an arbitrary initial state only after exponential or super exponential time ([7]).

In our model correct nodes cannot assume a common reference to time or even to any common anchor in time and they cannot assume that any procedure or primitive initialize concurrently. This is the result of the possible loss of synchronization following transient faults that might corrupt any agreement or coordination among the correct nodes and alter their internal states. Thus synchronization must be restored from an arbitrary state while facing on-going Byzantine failures. This is a very tricky task considering that all current tools for containing Byzantine failures, such as [2, 12], assume that synchronization already exists and are thus preempted for use. Our protocol achieves self-stabilizing Byzantine agreement without the assumption of any existing synchrony besides bounded message delivery. In [1] it is proven to be impossible to combine self-stabilization with even crash faults without the assumption of bounded message delivery.

Note that the problem is not relaxed even in the case of a one-shot agreement, i.e. in case that it is known that the General will initiate agreement only once throughout the life of the system. Even if the General is correct and even if agreement is initiated after the system has returned to its coherent behavior following transient failures, then the correct nodes might hold corrupted variable values that might prevent the possibility to reach agreement. The nodes have no knowledge as to when the system returns to coherent behavior or when the General will initiate agreement and thus cannot target to reset their memory exactly at this critical time period. Recurrent agreement initialization by the General allows for recurrent reset of memory with the assumption that eventually all correct nodes reset their memory in a coherent state of the system and before the General initializes agreement. This introduces the problem of how nodes can know when to reset their memory in case of many ongoing concurrent invocations of the algorithm, such as in the case of a faulty General disseminating several values all the time. In such a case correct nodes might hold different sets of messages that were sent by other correct nodes as they might reset their memory at different times.

In our protocol, once the system complies with the theoretically required bound of $3f < n$ permanent Byzantine faulty nodes in a network of n nodes and messages are delivered within bounded time, following a period of transient failures, then regardless of the state of the system, the goal of Byzantine agreement is satisfied within $O(f')$ communication rounds (where $f' \leq f$ is the actual number of concurrent faults). The protocol can be executed in a one-shot mode by a single General or by recurrent agreement initializations and by different Generals. It tolerates transient failures and permanent Byzantine faults and makes no assumption on any initial synchronized activity among the nodes (such as having a common reference to time or a common event for triggering initialization).

For ease of following the arguments and proofs, the structure and logic of our SS-Byz-Agree procedure is modeled on that of [12]. The rounds in that protocol progress following elapsed time. Each round spans a constant predefined time interval. Our protocol, besides being self-stabilizing, has the additional advantage of having a message-driven rounds structure and not time-driven rounds structure. Thus the actual time for terminating the protocol depends on the actual communication network speed and not on the worst possible bound on message delivery time.

It is important to note that we have previously presented a distributed self-stabilizing Byzantine pulse synchronization procedure in [3]. It aims at delivering a common anchor in time to all correct nodes within a short time following transient failures and with the permanent presence of Byzantine nodes. We have also previously presented a protocol for making any Byzantine algorithm be self-stabilizing [5], assuming the existence of synchronized pulses. Byzantine agreement can easily be achieved using a pulse synchronization procedure: the pulse invocation can serve as the initialization event for round zero of the agreement protocol. Thus any existing Byzantine agreement protocol may be used, on top of the pulse synchronization procedure, to attain self-stabilizing Byzantine agreement. The current paper achieves Byzantine agreement without assuming synchronized pulses. Moreover, we show in [6] that synchronized pulses can actually be produced more efficiently atop the protocol in the current paper. This pulse synchronization procedure can in turn be used as the pulse synchronization mechanism for making any Byzantine algorithm self-stabilize, in a more efficient way and in a more general model than by using the pulse synchronization procedure in [3].

In [13] it is shown how to initialize Byzantine clock synchronization without assuming a common initialization phase. It can eventually also execute synchronized Byzantine agreement by using the synchronized clocks. The solution is not self-stabilizing as nodes are booted and thus do not initialize with arbitrary values in the memory.

In [9] consensus is reached assuming eventual synchrony. Following an unstable period with unbounded failures and message delays, eventually no node fails and messages are delivered within bounded, say d, time. At this point there is no synchrony among the correct nodes and they might hold copies of obsolete messages. This is seemingly similar to our model but the solution is not truly self-stabilizing since the nodes do not initialize with arbitrary values. Furthermore, the solution only tolerates stopping failures and no new nodes fail subsequent to stabilization. Consensus is reached within $O(d)$. That paper also argues that in their model, although with Byzantine failures, consensus cannot be reached within less than $O(f') \cdot d$ time, which is essentially identical to our time complexity. Our solution operates in a more severe fault model and thus converges in optimal time.

2. MODEL AND PROBLEM DEFINITION

The environment is a network of n nodes that communicate by exchanging messages. We assume that the message passing medium allows for an authenticated identity of the senders. The communication network does not guarantee any order on messages among different nodes. Individual nodes have no access to a central clock and there is no external pulse system. The hardware clock rate (referred to as the *physical timers*) of correct nodes has a bounded drift, ρ, from real-time rate. Ensuant to transient failures there can be an unbounded number of concurrent Byzantine nodes, the turnover rate between faulty and non-faulty nodes can be arbitrarily large and the communication network may behave arbitrarily.

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

1. (Bounded Drift) *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. (Obedience) *Operates according to the instructed protocol.*
3. (Bounded Processing Time) *Processes any message of the instructed protocol within* π *real-time units of arrival time*[1].

A node is considered **faulty** if it violates any of the above conditions. A faulty node may recover from its Byzantine behavior once it resumes obeying the conditions of a non-faulty node. In order to keep the definitions consistent, the "correction" is not immediate but rather takes a certain amount of time during which the non-faulty node is still not counted as a correct node, although it supposedly behaves "correctly"[2]. We later specify the time-length of continuous non-faulty behavior required of a recovering node to be considered **correct**.

DEFINITION 2. *The communication network is* **non-faulty** *at periods that it complies with the following:*

1. *Any message arrives at its destination node within* δ *real-time units;*
2. *The sender's identity and content of any message being received is not tampered.*

Thus, our communication network model is a "bounded-delay" communication network. We do not assume the existence of a broadcast medium. We assume that the network cannot store old messages for arbitrary long time or lose any more messages, once it becomes non-faulty[3].

We use the notation $d \equiv \delta + \pi$. Thus, when the communication network is non-faulty, d is the upper bound on the elapsed real-time from the sending of a message by a non-faulty node until it is received and processed by every non-faulty node[4].

Note that n, f and d are fixed constants and thus non-faulty nodes do not initialize with arbitrary values of these constants.

A recovering node should be considered correct only once it has been continuously non-faulty for enough time to enable it to have deleted old or spurious messages and to have exchanged information with the other nodes.

DEFINITION 3. *The communication network is* **correct** *following* Δ_{net} *real-time of continuous non-faulty behavior*[5].

DEFINITION 4. *A node is* **correct** *following* Δ_{node} *real-time of continuous non-faulty behavior during a period that the communication network is correct*[6].

DEFINITION 5. (System Coherence) *The system is said to be* **coherent** *at times that it complies with the following:*

1. (Quorum) *There are at least* $n - f$ *correct nodes*[7]*, where* f *is the upper bound on the number of potentially non-correct nodes, at steady state.*
2. (Network Correctness) *The communication network is correct.*

Hence, if the system is not coherent then there can be an unbounded number of concurrent faulty nodes; the turnover rate between the faulty and non-faulty nodes can be arbitrarily large and the communication network may deliver messages with unbounded delays, if at all. The system is considered coherent, once the communication network and a sufficient fraction of the nodes have been non-faulty for a sufficiently long time period for the pre-conditions for convergence of the protocol to hold. The assumption in this paper, as underlies any other self-stabilizing algorithm, is that the system eventually becomes coherent.

It is assumed that each node has a local timer that proceeds at the rate of real-time. The actual reading of the various timers may be arbitrarily apart, but their relative rate is bounded in our model. To simplify the presentation we will ignore the drift factor of hardware clocks. Since nodes measure only periods of time that span several d, we will assume that d is an upper bound on the sending time of messages among correct nodes, measured by each local timer. To distinguish between a real-time value and a node's local-time reading we use t for the former and τ for the latter. The function $rt(\tau_p)$ represents the real-time when the timer of p reads τ_p at the current execution.

3. THE SS-BYZ-AGREE PROTOCOL

We consider the Byzantine agreement problem in which a *General* broadcasts a value and the correct nodes agree on the value broadcasted. In our model any node can be a General. An instance of the protocol is executed per General, and a correct General is expected to send one value at a time[8]. The target is for the correct nodes to associate a local-time with the protocol initiation by the General and to agree on a specific value associated with that initiation, if they agree that such an initiation actually took place. We bound the frequency by which correct Generals may initiate agreements, though Byzantine nodes might trigger agreements on their values as frequent as they wish.

The General initiates agreement by disseminating a message $(Initiator, G, m)$ to all nodes. Upon receiving the General's message, each node invokes the SS-BYZ-AGREE protocol, which in turn invokes the INITIATOR-ACCEPT primitive. Alternatively, if a correct node concludes that enough nodes have invoked the protocol (or the primitive) it will participate by executing the various parts of the INITIATOR-ACCEPT

[1] We assume that the bounds include also the overhead of the operating system in sending and processing of messages.
[2] For example, a node may recover with arbitrary variables, which may violate the validity condition if considered correct immediately.
[3] A non-faulty network might fail to deliver messages within the bound but will be masked as a fault and accounted for in the f faults. Essentially, we assume that messages among correct nodes are delivered within the time bounds.
[4] Nodes that were not faulty when the message was sent.
[5] We assume $\Delta_{net} \geq d$.
[6] We assume $\Delta_{node} \geq 14(2f+3)d + 10d$.
[7] The result can be replaced by $2f+1$ correct nodes with some modifications to the structure of the protocol.
[8] One can expand the protocol to a number of concurrent invocations by using an index to differentiate among the concurrent invocations.

primitive, but will not invoke it. If all correct nodes invoke the protocol within a "small" time-window, as will happen if the General is a correct node, then it is ensured that the correct nodes agree on a value for the General. If all correct nodes do not invoke the ss-Byz-Agree protocol within a small time-window, as can happen if the General is faulty, then if any correct node accepts a non-null value, all correct nodes will accept and agree on that value.

For ease of following the arguments and the logic of our ss-Byz-Agree protocol, we chose to follow the building-block structure of [12]. The equivalent of the broadcast primitive that simulates authentication in [12] is the primitive msgd-Broadcast presented in Section 5. The main differences between the original synchronous broadcast primitive and msgd-Broadcast are two-folds: first, the latter executes rounds that are anchored at some agreed event whose local-time is supplied to the primitive through a parameter; second, the conditions to be satisfied at each round at the latter, need to be satisfied by some time span that is a function of the round number and need not be executed only during the round itself. This allows nodes to rush through the protocol in the typical case when messages happen to be delivered faster than the worse case round span.

The ss-Byz-Agree protocol needs to take into consideration that correct nodes may invoke the agreement procedure at arbitrary times and with no knowledge as to when other correct nodes may have invoked the procedure. A mechanism is thus needed to make all correct nodes attain some common notion as to when and what value the General has sent. The differences of the real-time representations of the different nodes' estimations should be bounded. This mechanism is satisfied by the Initiator-Accept primitive defined in Section 4.

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

- Let Φ be the duration of time equal to $(\tau^G_{skew} + 2d)$ local-time units on a correct node's timer, where $\tau^G_{skew} = 5d$ in the context of this paper. Intuitively, Φ is the duration of a "phase" on a correct node's timer.
- Δ will be equal to $(2f+3)\cdot\Phi$, the upper bound on the time it takes to run the agreement protocol.
- $\perp$ denotes a null value.
- In the Initiator-Accept primitive:
  - A I-*accept*[9] is issued on values sent by G.
  - τ^G_q denotes the local-time estimate, at node q, as to when the General have sent a value that has been I-accept in Initiator-Accept by node q.

In the context of this paper we assume that a correct node will not initiate agreement on a new value at least $6d$ time units subsequent to termination of its previous agreement.

Definition 6. *We say:*

A node p ***decides*** *at time τ if it stops at that local-time and returns value $\neq\perp$.*

A node p ***aborts*** *if it stops and returns $\perp$.*

A node p ***returns*** *a value if it either aborts or decides.*

[9]An *accept* is issued within msgd-Broadcast.

The ss-Byz-Agree protocol is presented (see Figure 1) in a somewhat different style than the original protocol in [12]. Each round has a precondition associated with it: if the local timer value associated with the initialization by the General is defined and the precondition holds then the step is to be executed. It is assumed that the primitives instances invoked as a result of the ss-Byz-Agree protocol are implicitly associated with the agreement instance that invoked them. A node stops participating in the procedures and the invoked primitives $3d$ time units after it returns a value.

The ss-Byz-Agree protocol satisfies the following typical properties:
Agreement: The protocol returns the same value ($\neq\perp$) at all correct nodes;
Validity: If all correct nodes are triggered to invoke the ss-Byz-Agree protocol by a value sent by a correct General G, then the all correct nodes return that value;
Termination: The protocol terminates in a finite time.

It also satisfies the following properties:

Timeliness:

1. (agreement) For every two correct nodes q and q' that decide on (G, m) at τ_q and $\tau_{q'}$, respectively:
   (a) $|rt(\tau_q) - rt(\tau_{q'})| \leq 3d$, and if validity holds, then $|rt(\tau_q) - rt(\tau_{q'})| \leq 2d$.
   (b) $|rt(\tau^G_q) - rt(\tau^G_{q'})| \leq 5d$.
   (c) $rt(\tau^G_q), rt(\tau^G_{q'}) \in [t_1 - 2d, t_2]$, where $[t_1, t_2]$ is the interval within which all correct nodes that actually invoked the ss-Byz-Agree (G, m) did so.
   (d) $rt(\tau^G_q) \leq rt(\tau_q)$ and $rt(\tau_q) - rt(\tau^G_q) \leq \Delta$ for every correct node q.
2. (validity) If all correct nodes invoked the protocol in an interval $[t_0, t_0 + d]$, as a result of some value m sent by a correct General G that spaced the sending by at least $6d$ from the completion of the last agreement on its value, then for every correct node q, the decision time τ_q, satisfies $t_0 - d \leq rt(\tau^G_q) \leq rt(\tau_q) \leq t_0 + 3d$.
3. (termination) The protocol terminates within Δ time units of invocation, and within $\Delta + 7d$ in case it was not invoked explicitly.
4. (separation) Let q be any correct node that decided on any two agreements regarding p, then $t_2 + 5d < \bar{t}_1$ and $rt(\tau_q) + 5d < \bar{t}_1 < rt(\bar{\tau}_q)$, where t_2 is the latest time at which a correct node invoked ss-Byz-Agree in the earlier agreement and $\bar{t}_1$ is the earliest ss-Byz-Agree invoked by a correct node in the later agreement.

Note that the bounds in the above property is with respect to d, the bound on message transmission time among correct nodes and not the worse case deviation represented by Φ.

Observe that since there is no prior notion of the possibility that a value may be sent, it might be that some nodes associate a $\perp$ with a faulty sending and others may not notice the sending at all.

The proof that the ss-Byz-Agree protocol meets its properties appears in Section 6.3.

```
Protocol ss-Byz-Agree on (G, m)        /* Executed at node q. τ_q is the local-time at q. */
                                       /* Block Q is executed only when (and if) invoked. */
                                       /* Each block is executed at most once, when the precondition holds. */
                                       /* Executed as a result of I-accept at Line R1, or when τ_q^G is defined. */
                                       /* Invoked at node q upon arrival of a message (Initiator, G, m) from node G. */

Q.  Initiator-Accept (G, m).           /* determines τ_q^G and a value m' for node G */

R1.  if I-accept ⟨G, m', τ_q^G⟩ and τ_q − τ_q^G ≤ 4d then
R2.        value := ⟨G, m'⟩;
R3.        msgd-Broadcast (q, value, 1);
R4.        stop and return ⟨value, τ_q^G⟩.

S1.   if by τ_q (where τ_q ≤ τ_q^G + (2r + 1)·Φ)
         accepted r distinct messages (p_i, ⟨G, m''⟩, τ_i, i)
         where ∀i, j  1 ≤ i ≤ r, and p_i ≠ p_j ≠ G then
S2.        value := ⟨G, m''⟩;
S3.        msgd-Broadcast (q, value, r + 1);
S4.        stop and return ⟨value, τ_q^G⟩.

T1.   if by τ_q (where τ_q > τ_q^G + (2r + 1)·Φ) |broadcasters| < r − 1 then
T2.        stop and return ⟨⊥, τ_q^G⟩.

U1.   if τ_q > τ_q^G + (2f + 3)·Φ then
U2.        stop and return ⟨⊥, τ_q^G⟩.

cleanup:
  3d after returning a value reset the related Initiator-Accept and msgd-Broadcast;
  Remove any value or message older than (2f + 3)·Φ + 3d time units.
```

Figure 1: The ss-Byz-Agree protocol

4. THE Initiator-Accept PRIMITIVE

In the protocol in [12] a General that wants to send some value broadcasts it in a specific round (round 0 of the protocol). From the various assumptions on synchrony all correct nodes can check whether a value was indeed sent at the specified round and whether multiple (faulty) values were sent. In the transient fault model no such round number can be automatically adjoined with the broadcast. Thus a faulty General has more power in trying to fool the correct nodes by sending its values at completely different times to whichever nodes it decides.

The Initiator-Accept primitive aims at making the correct nodes associate a relative time to the invocation of the protocol by (the possibly faulty) General, and to converge to a single candidate value for the agreement to come. Since the full invocation of the protocol by a faulty General might be questionable, there may be cases in which some correct nodes will return a $\perp$ value and others will not identify the invocation as valid. If any correct node returns a value $\neq\perp$, all will return the same value.

Each correct node records the local-time at which it first received messages associated with the invocation of the protocol and produces an estimate to its (relative) local-time at which the protocol may have been invoked. The primitive guarantees that all correct nodes' estimates are within bounded real-time of each other.

We say that a node does an **I-accept** of a value sent by the General if it accepts this value as the General's initial value, and τ_q^G is the estimated local-time at q associated with the invocation of the protocol by the General.

The nodes maintain a vector *initiator*$[G, \_]$ for the possible values sent by the General G, where each non-empty entry is a local-time associated with the entry value. We will consider the data structures of a node *fresh* if up to d units of time ago *initiator*$[G, \_]$ did not contain any value and *latest_ accept* was $\perp$.

Nodes decay old messages and reset the data structures shortly after completion of the primitive, as defined below. It is assumed that correct nodes will not invoke the Initiator-Accept primitive when the data structures are not fresh.

The Initiator-Accept primitive satisfies the following properties:

IA-1 *(Correctness)* If all correct nodes invoke Initiator-Accept (G, m), with *fresh* data structures, within some real-time interval $[t_0, t_0 + d]$, then:

- **1A** All correct nodes I-accept $\langle G, m, \tau^G\rangle$ within $2d$ time units of the time the last correct node invokes the primitive Initiator-Accept(G, m).
- **1B** All correct nodes I-accept $\langle G, m, \tau^G\rangle$ within $2d$ time units of each other.
- **1C** For every pair of correct nodes q and q' that I-accepts $\langle G, m, \tau_q^G\rangle$ and $\langle G, m, \tau_{q'}^G\rangle$, respectively: $|rt(\tau_{q'}^G) - rt(\tau_q^G)| \leq d$.
- **1D** For each correct node q that I-accepts $\langle G, m, \tau_q^G\rangle$ at τ_q, $t_0 - d \leq rt(\tau_q^G) \leq rt(\tau_q) \leq t_0 + 3d$.

IA-2 *(Unforgeability)* If no correct node invokes Initiator-Accept (G, m), then no correct node I-accepts $\langle G, m, \tau^G\rangle$.

```
Primitive INITIATOR-ACCEPT (G, m)
                                        /* Executed at node q. τ_q is the local-time at q. */
                                  /* Lines L1 and L2 are repeatedly executed until I-accept. */
                            /* The rest are executed at most once, when the precondition holds. */
                 /* Block K is executed only when (and if) the primitive is explicitly invoked. */

K1. if τ_q − last_τ_q > 7d and if at τ_q − d initiator[G, _] =⊥ then      /* allow recent entries */
K2.    send (support, G, m) to all;                                       /* for a single m ever */
K3.    set initiator[G, m] := τ_q − d;                                          /* recording time */

L1. if received (support, G, m) from ≥ n − 2f distinct nodes
          within a window of α ≤ 4d time units of each other then
L2.    initiator[G, m] := max[initiator[G, m], (τ_q − α − 2d)];                 /* recording time */
L3. if received (support, G, m) from ≥ n − f distinct nodes
          within a window of 2d time units of each other then
L4.    send (ready, G, m) to all;

M1. if received (ready, G, m) from ≥ n − 2f distinct nodes then
M2.    send (ready, G, m) to all;
M3. if received (ready, G, m) from ≥ n − f distinct nodes then
M4.    τ_q^G := initiator[G, m]; I-accept ⟨G, m, τ_q^G⟩; last_τ_q := τ_q;

cleanup:
   Remove any value or message older than Δ + 7d time units.
   If last_τ_q > τ_q then last_τ_q :=⊥ .
```

Figure 2: The INITIATOR-ACCEPT primitive that yields a common notion of protocol invocation

IA-3 *(Δ-Relay)* If a correct node q I-accepts $\langle G, m, \tau_q^G\rangle$ at real-time t, such that $0 \le t - rt(\tau_q^G) \le \Delta$, then:

- **3A** Every correct node q' I-accepts $\langle G, m, \tau_{q'}^G\rangle$, at some real-time t', with $|t - t'| \le 2d$ and $|rt(\tau_q^G) - rt(\tau_{q'}^G)| \le 5d$.
- **3B** Moreover, $rt(\tau_q^G), rt(\tau_{q'}^G) \in [t_1 - 2d, t_2]$, where $[t_1, t_2]$ is the interval within which all correct nodes that actually invoked SS-BYZ-AGREE (G, m) did so.
- **3C** For every correct node q', $rt(\tau_{q'}^G) \le rt(\tau_{q'})$ and $rt(\tau_{q'}) - rt(\tau_{q'}^G) \le \Delta + 7d$.

IA-4 *(Uniqueness)* If a correct node q I-accepts $\langle G, m, \tau_q^G\rangle$, then no correct node I-accepts $\langle G, m', \tau_p^G\rangle$ for $m \ne m'$, for $|rt(\tau_q^G) - rt(\tau_p^G)| \le 5d$.

Each node maintains in addition to *initiator*$[G, \_]$ a data structure in which the latest message from each partner regarding a possible value sent by the General is kept. The data structure records as a time stamp the local-time at which each message is received. If the data structure contains illegal values or future time stamps (due to transient faults) the messages are removed. The protocol also requires the knowledge of the state of the vector *initiator*$[G, \_]$ d time units in the past. It is assumed that the data structure reflects that information.

When the primitive is explicitly invoked the node executes block K. A node may receive messages related to the primitive, even in case that it did not explicitly invoke the primitive. In this case it executes the rest of the blocks of the primitive, if the appropriate preconditions hold. A correct node repeatedly executes Line L1 and Line L2, whenever the precondition holds. The rest are executed at most once, when the precondition holds for the first time.

Following the completion of SS-BYZ-AGREE, the data structures of the related INITIATOR-ACCEPT instance are reset.

The proof that the INITIATOR-ACCEPT primitive satisfies the [IA-*] properties, under the assumption that $n > 3f$, appears in Section 6.1.

5. THE MSGD-BROADCAST PRIMITIVE

This section presents the MSGD-BROADCAST (a message driven broadcast) primitive, which **accept**s messages being **broadcast**ed by executing it. The primitive is invoked by the SS-BYZ-AGREE protocol presented in Section 3. The primitive follows the broadcast primitive of Toueg, Perry, and Srikanth [12]. In the original synchronous model, nodes advance according to rounds that are divided into phases. This intuitive lock-step process clarifies the presentation and simplifies the proofs. The primitive MSGD-BROADCAST is presented without any explicit or implicit reference to time, rather an anchor to the potential initialization point of the protocol is passed as a parameter by the calling procedure. The properties of the INITIATOR-ACCEPT primitive guarantee a bound between the real-time of the anchors of the correct nodes. Thus a general notion of a common round structure can be implemented by measuring the elapsed time units since the local-time represented by the passed anchor.

In the broadcast primitive of [12] messages associated with a certain round must be sent by correct nodes at that round and will be received, the latest, at the end of that round by all correct nodes. In MSGD-BROADCAST, on the other hand, the rounds progress with the arrival of the anticipated messages. Thus for example, if a node receives some required messages before the end of the round it may send next round's messages. The length of a round only imposes an upper bound on the acceptance criteria. Thus the protocol can progress at the speed of message delivery, which may be significantly faster than that of the protocol in [12].

```
Primitive msgd-Broadcast (p, m, k)
                                                    /* Executed per such triplet at node q. */
                                                /* Nodes send specific messages only once. */
                                          /* Nodes execute the blocks only when τ^G is defined. */
                                     /* Nodes log messages until they are able to process them. */
                                 /* Multiple messages sent by an individual node are ignored. */

    At node q = p:                          /* if node q is node p that invoked the primitive */
V.    node p sends (init, p, m, k) to all nodes;

W1. At time τ_q : τ_q ≤ τ_q^G + 2k·Φ
W2.   if received (init, p, m, k) from p then
W3.       send (echo, p, m, k) to all;

X1. At time τ_q : τ_q ≤ τ_q^G + (2k − 1)·Φ
X2.   if received (echo, p, m, k) from ≥ n − 2f distinct nodes then
X3.       send (init', p, m, k) to all;
X4.   if received (echo, p, m, k) messages from ≥ n − f distinct nodes then
X5.       accept (p, m, k);

Y1. At time τ_q : τ_q ≤ τ_q^G + (2k + 2)·Φ
Y2.   if received (init', p, m, k) from ≥ n − 2f then
Y3.       broadcasters := broadcasters ∪ {p};
Y4.   if received (init', p, m, k) from ≥ n − f distinct nodes then
Y5.       send (echo', p, m, k) to all;

Z1. At any time:
Z2.   if received (echo', p, m, k) from ≥ n − 2f distinct nodes then
Z3.       send (echo', p, m, k) to all;
Z4.   if received (echo', p, m, k) from ≥ n − f distinct nodes then
Z5.       accept (p, m, k);                                          /* accept only once */

cleanup:
  Remove any value or message older than (2f + 3)·Φ time units.
```

Figure 3: The msgd-Broadcast primitive with message-driven round structure

Note that when a node invokes the primitive it evaluates all the messages in its buffer that are relevant to the primitive. The msgd-Broadcast primitive is executed in the context of some initiator G that invoked ss-Byz-Agree, which makes use of the msgd-Broadcast primitive. No correct node will execute the msgd-Broadcast primitive without first producing the reference (anchor), τ^G, on its local timer to the time estimate at which G supposedly invoked the original agreement. By IA-3A this happens within $2d$ of the other correct nodes.

The synchronous Reliable Broadcast procedure of [12] assumes a round model in which within each phase all message exchange among correct nodes take place. The equivalent notion of a round in our context will be Φ defined to be: $\Phi := t^G_{skew} + 2d$.

The msgd-Broadcast primitive satisfies the following [TPS-*] properties of Toueg, Perry and Srikanth [12], which are phrased in our system model.

TPS-1 (*Correctness*) If a correct node p msgd-Broadcast(p, m, k) at τ_p, $\tau_p \leq \tau_p^G + (2k-1)\cdot\Phi$, on its timer, then each correct node q accepts (p, m, k) at some τ_q, $\tau_q \leq \tau_q^G + (2k + 1)\cdot\Phi$, on its timer and $|rt(\tau_p) - rt(\tau_q)| \leq 3d$.

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

TPS-3 (*Relay*) If a correct node q_1 accepts (p, m, k) at τ_1, $\tau_1 \leq \tau_1^G + r\cdot\Phi$ on its timer then any other correct node q_2 accepts (p, m, k) at some τ_2, $\tau_2 \leq \tau_2^G + (r + 2)\cdot\Phi$, on its timer.

TPS-4 (*Detection of broadcasters*) If a correct node accepts (p, m, k) then every correct node q has $p \in broadcasters$ at some τ_q, $\tau_q \leq \tau_q^G + (2k+2)\cdot\Phi$, on its timer. Furthermore, if a correct node p does not msgd-Broadcast any message, then a correct node can never have $p \in broadcasters$.

Note that the bounds in [TPS-1] are with respect to d, the bound on message transmission time among correct nodes.

The msgd-Broadcast primitive satisfies the [TPS-*] properties, under the assumption that $n > 3f$. The proofs that appear in Section 6.2 follow closely the original proofs of [12], in order to make it easier for readers that are familiar with the original proofs.

6. PROOFS

Note that all the definitions, theorems and lemmata in this paper hold only from the moment, and as long as, the system is coherent.

6.1 Proof of the INITIATOR-ACCEPT Properties

THEOREM 1. *The* INITIATOR-ACCEPT *primitive presented in Figure 2 satisfies properties [IA-1] through [IA-4], assuming that a correct node that invokes the primitive invokes it with fresh data structures.*

PROOF.

Correctness: Assume that within d of each other all correct nodes invoke INITIATOR-ACCEPT (G, m). Let t_1 be the real-time at which the first correct node invokes the INITIATOR-ACCEPT and t_2 be the time the last one did so. Since all data structures are *fresh*, then no value $\{G, m'\}$ appeared in *broadcasters* d time units before that, thus Line K1 will hold for all correct nodes. Therefore, every correct node sends $(support, G, m)$. Each such message reaches all other correct nodes within d. Thus, between t_1 and $t_2 + d$ every correct node receives $(support, G, m)$ from $n - f$ distinct nodes and sends $(ready, G, m)$ and by $t_2 + 2d$ I-accepts $\langle G, m, \tau' \rangle$, for some τ', thus, proving [IA-1A].

To prove [IA-1B], let q be the first to I-accept after executing Line M4. Within d all correct nodes will execute Line M2, and within $2d$ all will I-accept.

Note that for every pair of correct nodes q and q', the associated initial recording times τ and τ' satisfy $|\tau - \tau'| \leq d$. Line K3 implies that the recording times of correct nodes can not be earlier than $t_1 - d$. Some correct node may see $n - 2f$, with the help of faulty nodes as late as $t_2 + 2d$. All such windows should contain a *support* from a correct node, so should include real-time $t_2 + d$, resulting in a recording time of $t_2 - d$. Recall that $t_2 \leq t_1 + d$, proving [IA-1C].

To prove [IA-1D] notice that the fastest node may set τ' to be $t_1 - d$, but may I-accept only by $t_2 + 2d \leq t_1 + 3d$.

Unforgeability:
If no correct node invokes INITIATOR-ACCEPT and will not send $(support, G, m)$, then no correct node will ever execute L4 and will not send $(ready, G, m)$. Thus, no correct node can accumulate $n - f$ $(ready, G, m)$ messages and therefore will not I-accept $\langle G, m \rangle$.

Δ-Relay:
Let q be a correct node that I-accepts $\langle G, m, \tau_q^G \rangle$ at real-time t, such that $0 \leq t - rt(\tau_q^G) \leq \Delta$. It did so as a result of executing Line M4. Let X be the set of correct nodes whose $(ready, G, m)$ were used by q in executing Line M4. Either there exists in X a correct node sending it as a result of executing Line L4, or at least one of the nodes in X have heard from such a node (otherwise, it heard from other $f + 1$ distinct nodes and there will be at least $n - f + f + 1 > n$ distinct nodes in total, a contradiction).

Let $\bar{q}$ be the first correct node to execute Line L4, and assume it took place at some real-time t''. Note that $t'' \leq t$. Node $\bar{q}$ collected $n - f$ *support* messages, with at least $n - 2f$ from correct nodes[10]. Let t_1 be the time at which the $(n - 2f)^{th}$ *support* message sent by a correct node was received. Since $\bar{q}$ executed Line L4, all these messages should have been received in the interval $[t_1 - 2d, t_1]$. Node $\bar{q}$ should have set a recording time $\tau \geq t_1 - 4d$ as a result of (maybe repeating) the execution of Line L2.

[10]Ignore for a moment decaying of messages (we will prove below that no correct node decays these messages at that stage).

Every other correct node should have received this set of $(n - 2f)$ *support* messages sent by correct nodes in the interval $[t_1 - 3d, t_1 + d]$ and should have set the recording time after (maybe repeatedly) executing Line L2, since this window satisfies the precondition of Line L1. Thus, eventually all recording times are $\geq t_1 - 5d$.

Some correct node may send a *support* message, by executing Line K2, at most d time units later (just before receiving these $n - 2f$ messages). This can not take place later than $t_1 + d$, resulting in a recording time of t_1, though earlier than its time of sending the *support* message. This *support* message (with the possible help of faulty nodes) can cause some correct node to execute Line L2 at some later time. The window within which the *support* messages at that node are collected should include the real-time $t_1 + 2d$, the latest time any *support* from any correct node could have been received. Any such execution will result in a recording time that is $\leq t_1 + 2d - 2d = t_1$. Thus the range of recording times for all correct nodes (including q) are $[t_1 - 5d, t_1]$. Proving the second part of [IA-3A].

Since we assumed that $0 \leq t - rt(\tau_q^G) \leq \Delta$, all messages above are within the decaying window of all correct nodes and none of these messages will be decayed, proving that the result holds. For the same reason, all correct messages collected by q will not be decayed by other correct nodes. By time $t + d$ all correct nodes will be able to execute Line M2, and by $t + 2d$ each correct node q' will execute Line M4 to I-accept $\langle G, m, \tau_{\bar{q}} \rangle$. Proving the first part of [IA-3A].

To prove [IA-3B] notice that any range of values in Line L2 includes a *support* of a correct node. The resulting recording time will never be later than the sending time of the *support* message by that correct node, and thus by some correct node. To prove the second part of [IA-3B] consider again node $\bar{q}$ from the proof of the $\Delta - relay$ property. It collected $n - 2f$ *support* messages from correct nodes in some interval $[\bar{t}_1, t_1]$, where $\bar{t}_1 \geq t_1 - 2d$. These messages, when received by any correct node will be within an interval of $4d$, with the first message in it from a correct node. These messages will trigger a possible update of the recording time in Line L2. Thus, the resulting recording time of any correct node cannot be earlier than some $2d$ of receiving a *support* message from a correct node, thus not earlier than $2d$ of sending such a message.

The first part of [IA-3C] is immediate from Line L2 and Line K3. For the second part observe that for every other correct node q', $rt(\tau_{q'}) \leq rt(\tau_q) + 2d$ and $rt(\tau_{q'}^G) \geq rt(\tau_q^G) - 5d$. Thus, $rt(\tau_{q'}) - rt(\tau_{q'}^G) \leq rt(\tau_q) - rt(\tau_q^G) + 7d \leq \Delta + 7d$.

To prove [IA-4] observe that each node sent a *support* for a single m. In order to I-accept, some correct node needs to send *ready* after receiving $n - f$ *support* messages. That can happen for at most a single value of m. What is left to prove, is that future invocations of the primitive will not violate [IA-4]. Observe that by [IA-3B], once a correct node sends a *ready* message, all recording times are within $2d$ of the reception time of some *support* message from a correct node. Moreover, by [IA-3C], this is always prior to the current time at any node that sets the recording time. Let q be the latest correct node to I-accept, at some time τ_q on its clock with some $last\_\tau_q$ as the returned recording time. Let p be the first correct node to send a *support* following that, at some local-time $\bar{\tau}_p$. We will denote by τ timings in the former invocation and by $\bar{\tau}$ timings in the later one.

Observe that $\tau_q^G \leq last\_\tau_q$ and that $rt(last\_\tau_q) - 2d \leq rt(last\_\tau_p)$. When p sends its *support*, $rt(\bar{\tau}_p) - rt(last\_\tau_p) > 9d$ implying that $rt(\bar{\tau}_p) - rt(last\_\tau_q) > 7d$. Once any correct node will send *ready* in the later invocation the resulting recording time of all correct nodes, including q will satisfy $rt(\bar{\tau}_q^G) \geq rt(\bar{\tau}_p) - 2d$, which implies $rt(\tau_q^G) \leq rt(last\_\tau_q) < rt(\bar{\tau}_p) - 7d \leq rt(\bar{\tau}_q^G) - 5d$. □

6.2 Proof of the MSGD-BROADCAST Properties

For lack of space we do not present all the proofs. The proofs essentially follow the arguments in the original paper [12].

LEMMA 1. *If a correct node p_i sends a message at local-time τ_i, $\tau_i \leq \tau_i^G + r \cdot \Phi$ on p_i's timer it will be received and processed by each correct node p_j at some local-time τ_j, $\tau_j \leq \tau_j^G + (r+1) \cdot \Phi$, on p_j's timer.*

LEMMA 2. *If a correct node ever sends $(echo', p, m, k)$ then at least one correct node, say q', must have sent $(echo', p, m, k)$ at some local-time $\tau_{q'}$, $\tau_{q'} \leq \tau_{q'}^G + (2k+2) \cdot \Phi$.*

LEMMA 3. *If a correct node ever sends $(echo', p, m, k)$ then p's message $(init, p, m, k)$ must have been received by at least one correct node, say q', at some time $\tau_{q'}$, $\tau_{q'} \leq \tau_{q'}^G + 2k \cdot \Phi$.*

LEMMA 4. *If a correct node p invokes the primitive* MSGD-BROADCAST *(p, m, k) at real-time t_p, then each correct node q accepts (p, m, k) at some real-time t_q, such that $|t_p - t_q)| \leq 3d$.*

THEOREM 2. *The* MSGD-BROADCAST *primitive presented in Figure 3 satisfies properties [TSP-1] through [TSP-4].*

PROOF. For lack of space and the essential similarity to the original proofs we will prove only the relay part.

Relay: The delicate point is when a correct node issues an accept as a result of getting echo messages. So assume that q_1 accepts (p, m, k) at $t_1 = rt(\tau_1)$ as a result of executing Line X5. By that time it must have received $(echo, p, m, k)$ from $n - f$ nodes, at least $n - 2f$ of them sent by correct nodes. Since every correct node among these has sent its message by $\tau^G + 2k \cdot \Phi$ on its timer, by Lemma 1, all those messages should have arrived to every correct node q_i by $\tau_i \leq \tau_i^G + (2k+1) \cdot \Phi$ on its timer. Thus, every correct node q_i should have sent $(init', p, m, k)$ at some τ_i, $\tau_i \leq \tau_i^G + (2k+1) \cdot \Phi$, on its timer. As a result, every correct node will receive $n - f$ such messages by some $\bar{\tau}$, $\bar{\tau} \leq \tau^G + (2k+2) \cdot \Phi$ on its timer and will send $(echo', p, m, k)$ at that time, which will lead each correct node to accept (p, m, k) at a local-time τ_i.

Now observe that all $n-2f$ $(echo, p, m, k)$ were sent before time t_1. By $t_1 + d$ they arrive to all correct nodes. By $t_1 + 2d$ all will have their τ^G defined and will process them. By $t_1 + 3d$ their $(init', p, m, k)$ will arrive to all correct nodes, which will lead all correct nodes to send $(echo', p, m, k)$. Thus, all correct nodes will accept (p, m, k) at time $\tau_i \leq t_1 + 4d$.

By assumption, $t_1 = rt(\tau_1) \leq rt(\tau_1^G) + r \cdot \Phi$. By IA-3A, $rt(\tau_1^G) \leq rt(\tau_i^G) + t_{skew}^G$. Therefore we conclude: $rt(\tau_i) \leq rt(\tau_1) + 4d \leq rt(\tau_1^G) + r \cdot \Phi + 4d \leq rt(\tau_i^G) + t_{skew}^G + r \cdot \Phi + 4d \leq rt(\tau_i^G) + (r+2) \cdot \Phi$.

The case that the accept is a result of executing Line Z5 is a special case of the above arguments. □

6.3 Proof of the ss-Byz-Agree Properties

THEOREM 3. *(Convergence) Once the system is coherent, any invocation of* ss-Byz-Agree *presented in Figure 1 satisfies the Termination property. When $n > 3f$, it also satisfies the Agreement and Validity properties.*

PROOF. Notice that the General G itself is one of the nodes, so if it is faulty then there are only $f - 1$ potentially faulty nodes. We do not use that fact in the proof since the version of ss-Byz-Agree presented does not refer explicitly to the General. One can adapt the proof and reduce Δ by $2 \cdot \Phi$ when specifically handling that case.

Let $\hat{t}$ be the real-time by which the network is correct and there are at least $n - f$ non-faulty nodes. These nodes may be in an arbitrary state at that time. If G does not send any $(\text{Initiator}, G, M)$ message for $\Delta + 7d$, all spurious invocations of the primitives and the protocol will be reset by all correct nodes. If G sends such an Initiator message, then within $\Delta + 3d$ of the time that any non-faulty node invokes the protocol, either a decision will take place (by all non-faulty nodes) or all will reset the protocol and its primitives. Beyond that time, any future invocation will happen when all data structures are reset at all non-faulty nodes. Note, that before that time a non-faulty G will not send the Initiator message again.

Thus, by time $\hat{t} + 2\Delta + 10d$, when the system becomes coherent, any invocation of the protocol will take place with empty (fresh) data structures and will follow the protocol as stated.

LEMMA 5. *If a correct node p aborts at local-time τ_p, $\tau_p > \tau_p^G + (2r+1) \cdot \Phi$, on its timer, then no correct node q decides at a time τ_q, $\tau_q \geq \tau_q^G + (2r+1) \cdot \Phi$, on its timer.*

PROOF. Let p be a correct node that aborts at time τ_p, $\tau_p > \tau_p^G + (2r+1) \cdot \Phi$. In this case it should have identified at most $r - 2$ broadcasters by that time. By the detection of the broadcasters property [TPS-4], no correct node will ever accept $\langle G, m' \rangle$ and $r-1$ distinct messages (q_i, m', i) for $1 \leq i \leq r - 1$, since that would have caused each correct node, including p, to hold $r - 1$ broadcasters by some time τ, $\tau \leq \tau^G + (2(r-1)+2) \cdot \Phi$ on its timer. Thus, no correct node, say q, can decide at a time $\tau_q \geq \tau_q^G + (2r+1) \cdot \Phi$ on its timer. □

LEMMA 6. *If a correct node p decides at time τ_p, $\tau_p \leq \tau_p^G + (2r+1) \cdot \Phi$, on its timer, then each correct node, say q, decides by some time τ_q, $\tau_q \leq \tau_q^G + (2r+3) \cdot \Phi$ on its timer.*

PROOF. Let p be a correct node that decides at local-time τ_p, $\tau_p \leq \tau_p^G + (2r+1) \cdot \Phi$. We consider the following cases:

1. $r = 0$: No correct node can abort by a time τ, $\tau \leq \tau^G + (2r+1) \cdot \Phi$, since the inequality will not hold. Assume that node p have accepted $\langle G, m' \rangle$ by $\tau_p \leq \tau_p^G + 4d \leq \tau_p^G + \Phi$. By the relay property [TPS-3] each correct node will accept $\langle G, m' \rangle$ by some time τ, $\tau \leq \tau^G + 3 \cdot \Phi$ on its timer. Moreover, p invokes MSGD-BROADCAST $(p, m', 1)$, by the Correctness property [TPS-1] it will be accepted by each correct node by time τ, $\tau \leq \tau^G + 3 \cdot \Phi$, on its timer. Thus, all correct nodes will have *value* $\neq \perp$ and will broadcast and stop by time $\tau^G + 3 \cdot \Phi$ on their timers.

2. $1 \leq r \leq f$: Node p must have accepted $\langle G, m' \rangle$ and also accepted r distinct (q_i, m', i) messages for all $i, 2 \leq i \leq r$, by time τ, $\tau \leq \tau^G + (2r+1) \cdot \Phi$, on its timer. By Lemma 5, no correct node aborts by that time. By Relay property [TPS-3] each (q_i, m', i) message will be accepted by each correct node by some time τ, $\tau \leq \tau^G + (2r+3) \cdot \Phi$, on its timer. Node p broadcasts $(p, m', r+1)$ before stopping. By the Correctness property, [TPS-1], this message will be accepted by every correct node at some time τ, $\tau \leq \tau^G + (2r+3) \cdot \Phi$, on its timer. Thus, no correct node will abort by time τ, $\tau \leq \tau^G + (2r+3) \cdot \Phi$, and all correct nodes will have $value \neq \perp$ and will thus decide by that time.

3. $r = f+1$: Node p must have accepted a (q_i, m', i) message for all $i, 1 \leq i \leq f$, by τ_p, $\tau_p \leq \tau_p^G + (2f+3) \cdot \Phi$, 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], node q_j invoked msgd-Broadcast (q_j, m', j) by some local-time τ, $\tau \leq \tau^G + (2j+1) \Phi$ and decided. Since $j \leq f+1$ the above arguments imply that by some local-time τ, $\tau \leq \tau^G + (2f+3) \cdot \Phi$, each correct node will decide. □

Lemma 6 implies that if a correct node decides at time τ, $\tau \leq \tau^G + (2r+1) \cdot \Phi$, on its timer, then no correct node p aborts at time τ_p, $\tau_p > \tau_p^G + (2r+1) \cdot \Phi$. Lemma 5 implies the other direction.

Termination: Each correct node either terminates the protocol by returning a value, or by time $(2f+3) \cdot \Phi + 3d$ all entries will be reset, which is a termination of the protocol.

Agreement: If no correct node decides, then all correct nodes that execute the protocol abort, and return a $\perp$ value. Otherwise, let q be the first correct node to decide. Therefore, no correct node aborts. The value returned by q is the value m' of the accepted $(p, m', 1)$ message. By [IA-4] if any correct node I-accepts, all correct nodes I-accept with a single value. Thus all correct nodes return the same value.

Validity: Since all the correct nodes invoke the primitive ss-Byz-Agree as a result of a value sent by a correct G, they will all invoke Initiator-Accept within d of each other with fresh data structure, hence [IA-1] implies validity.

Timeliness:

1. (agreement) For every two correct nodes q and q' that decide on (G, m) at τ_q and $\tau_{q'}$, respectively:
   (a) If validity hold, then $|rt(\tau_q) - rt(\tau_{q'})| \leq 2d$, by [IA-3A]; Otherwise, $|rt(\tau_q) - rt(\tau_{q'})| \leq 3d$, by [TPS-1].
   (b) $|rt(\tau_q^G) - rt(\tau_{q'}^G)| \leq 5d$ by [IA-3A].
   (c) $rt(\tau_q^G), rt(\tau_{q'}^G) \in [t_1 - 2d, t_2]$ by [IA-3B].
   (d) $rt(\tau_r^G) \leq rt(\tau_r)$, by [IA-3C], and if the inequality $rt(\tau_r) - rt(\tau_r^G) \leq \Delta$ would not hold, the node would abort right away.

2. (validity) If all correct nodes invoked the protocol in an interval $[t_0, t_0 + d]$, as a result of (Initiator, G, m) sent by a correct G that spaced the sending by $6d$ from its last agreement, then for every correct node q that may have decided $3d$ later than G, the new invocation will still happen with fresh data structures, since they are reset $3d$ after decision. By that time it already reset the data structures (including *latest_ accept*) of the last execution, and the new decision time τ_q, satisfies $t_0 - d \leq rt(\tau_q^G) \leq rt(\tau_q) \leq t_0 + 3d$ as implied by [IA-1D].

3. (separation) By [IA-4] the real-times of the I-accepts satisfy the requirements. Since a node will not reset its data structures before terminating the protocol, it will not send a *support* before completing the previous protocol execution. Therefore, the protocol itself can only increase the time difference between agreements. Thus, the minimal difference is achieved when a decision takes place right after the termination of the Initiator-Accept primitive. □

7. ACKNOWLEDGEMENTS

We wish to thank Ittai Abraham and Ezra Hoch for discussing some of the fine points of the model and the proofs. This research was supported in part by grants from ISF, NSF, CCR, and AFOSR.

8. REFERENCES

[1] J. Beauquier, S. Kekkonen-Moneta, "*Fault-tolerance and Self-stabilization: Impossibility Results and Solutions Using Failure Detectors*", Int. J of Systems Science, Vol. 28(11) pp. 1177-1187, 1997.

[2] B. Coan, D. Dolev, C. Dwork and L. Stockmeyer, "*The distributed firing squad problem*", Proc. of the 7th Annual ACM Symposium on Theory of Computing, pp. 335-345, Providence, Rhode Island, May 1985.

[3] A. Daliot, D. Dolev and H. Parnas, "*Self-stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks*", Proc. of the 6th Symposium on Self-Stabilizing Systems (SSS'03 San-Francisco), pp. 32-48, 2003.

[4] A. Daliot and D. Dolev, "*Self-stabilization of Byzantine Protocols*", Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS'05 Barcelona), pp. 48-67, 2005.

[5] A. Daliot, D. Dolev and H. Parnas, "*Linear Time Byzantine Self-Stabilizing Clock Synchronization*", Proc. of 7th Int. Conference on Principles of Distributed Systems (OPODIS'03 La Martinique), France, Dec. 2003.

[6] A. Daliot and D. Dolev, "*Making Order in Chaos: Self-stabilizing Byzantine Pulse Synchronization*", unpublished manuscript July 2006.

[7] S. Dolev, and J. L. Welch, "*Self-Stabilizing Clock Synchronization in the presence of Byzantine faults*", Journal of the ACM, Vol. 51, Issue 5, pp. 780 - 799, 2004.

[8] D. Dolev, H. R. Strong, "*Polynomial Algorithms for Multiple Processor Agreement*", In Proceedings, the 14th ACM SIGACT Symposium on Theory of Computing (STOC-82), pp. 401-407, May 1982.

[9] P. Dutta, R. Guerraoui, L. Lamport, "*How Fast Can Eventual Synchrony Lead to Consensus?*", Proc. of the 2005 Int. Conf. on Dependable Systems and Networks (DSN'05 Yokohama), Japan, June 2005.

[10] L. Lamport, R. Shostak, M. Pease, "*The Byzantine Generals Problem*", ACM Transactions on Programming Languages and Systems, 4(3):382-301, 1982.

[11] M. Pease, R. Shostak, L. Lamport, "*Reaching Agreement in the Presence of Faults*", Journal of the ACM, Vol. 27, No. 2. pp. 228-234, Apr. 1980.

[12] S. Toueg, K. J. Perry, T. K. Srikanth, "*Fast Distributed Agreement*", SIAM Journal on Computing, 16(3):445-457, June 1987.

[13] J. Widder, "*Booting clock synchronization in partially synchronous systems*", In Proc. the 17th Int. Symposium on Distributed Computing (DISC'03 Sorrento), Oct. 2003.