NASA/TM-2006-213951

Comments on the “Byzantine Self-Stabilizing Pulse Synchronization” Protocol: Counterexamples

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

Radu Siminiceanu
National Institute of Aerospace, Hampton, Virginia

February 2006

NASA/TM-2006-213951

Comments on the "Byzantine Self-Stabilizing Pulse Synchronization" Protocol: Counterexamples

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

Radu Siminiceanu
National Institute of Aerospace, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

February 2006

Comments on the “Byzantine Self-Stabilizing Pulse Synchronization” Protocol: Counterexamples

Mahyar R. Malekpour
NASA Langley Research Center, Hampton, VA, USA
m.r.malekpour@larc.nasa.gov
Radu Siminiceanu
National Institute of Aerospace, Hampton, VA, USA
radu@nianet.org

Abstract

Embedded distributed systems have become an integral part of many safety-critical applications. There have been many attempts to solve the self-stabilization problem of clocks across a distributed system. An analysis of one such protocol called the Byzantine Self-Stabilizing Pulse Synchronization (BSS-Pulse-Synch) protocol from a paper entitled “Linear Time Byzantine Self-Stabilizing Clock Synchronization” by Daliot et al [Daliot 03] is presented in this report. This report also includes a discussion of the complexity and pitfalls of designing self-stabilizing protocols and provides counterexamples for the claims of the above protocol.

Introduction

Synchronization and coordination algorithms are part of distributed computer systems. Clock synchronization algorithms are essential for managing the use of resources and controlling communication in a distributed system. Also, a fundamental criterion in the design of a robust distributed system is to embed the capability of tolerating and potentially recovering from failures caused by malicious behavior that are not predictable in advance. Overcoming such failures is most suitably addressed by tolerating Byzantine faults [Lamport 82]. A Byzantine fault model encompasses asymmetric failures within the limitations of the maximum number of faults at a given time. Driscoll et al. [Driscoll 03] addressed the frequency of occurrences of Byzantine faults in practice and the necessity to tolerate Byzantine faults in ultra-reliable distributed systems. A distributed system tolerating as many as f Byzantine faults requires a network size of more than $3f$ nodes. Lamport et al. [Lamport 82, Lamport 85] were the first to present the problem and show that Byzantine agreement cannot be achieved for fewer than $3f$+1processors. Dolev et al. [Dolev 84] proved that at least $3f$+1 processors are necessary for clock synchronization in the presence of f Byzantine faults.

A self-stabilizing system is able to start in any random state and to recover from transient failures after the faults dissipate. The possibility of self-stabilizing distributed computation was first presented in a classic paper by Dijkstra [Dijkstra 74]. In that paper, he asked whether it would be possible for a set of machines to stabilize their collective behavior in spite of unknown initial conditions and distributed control. The idea was that the system should be able to converge to a legitimate state within a bounded amount of time, by itself, without external intervention. The main challenges associated with self-stabilization are complexity of the design and proof of correctness of the protocol. Another difficulty is achieving efficient convergence time for the proposed self-stabilizing protocol.

A recent result in this area is the Byzantine self-stabilization pulse synchronization (BSS-Pulse-Synch) protocol developed by Daliot et al [Daliot 03]. In this paper we report a flaw in that protocol by providing explicit counterexamples.

The BSS-Pulse-Sync Protocol

The BSS-Pulse-Synch protocol as stated in [Daliot 03] is reproduced in Figure 1. Statement labels S1, S2, and S3 are added for future reference in subsequent sections. Cycle is the self-stabilization period, n is the number of nodes in the system, f the maximum number of faulty nodes, ρ the clock drift with respect to real time, and d is the bound on message transmission time.

```
BSS-Pulse-Sync(Cycle, $n$, $f$)
S1 – if (cycle_countdown_is_0) then                        /* endogenous message */
        send “Propose-Pulse” message to all;
        cycle_countdown_is_0 = ‘False”;

S2 – if received $f$+1 distinct “Propose-Pulse” messages then  /* triggered message */
        send “Propose-Pulse” message to all;

S3 – if received $n – f$ distinct “Propose-Pulse” messages then     /* pulse invocation */
        invoke “pulse” event;
        cycle_countdown = Cycle;
        flush “Propose-Pulse” message counter;
        ignore “Propose-Pulse” messages for $2d(1 + 2\rho)$ time units;
```

Figure 1. The BSS-Pulse-Synch protocol.

The primary claim of that paper, as stated in Theorem 2, is that the protocol self-stabilizes in the presence of at most f Byzantine nodes where $n \geq 3f + 1$. Theorem 2 is restated here for ease of reference. Another claim is that the nodes will converge to within a precision of $2d(1 + 2\rho)$ time units of each other.

Theorem 2 [Daliot 03]. *BSS-Pulse-Sync solves the Self-Stabilization Pule Synchronization Problem in the presence of at most f Byzantine nodes, $n \geq 3f + 1$.*

Interpretation of the BSS-Pulse-Sync Protocol

The BSS-Pulse-Synch protocol is a slight variation of the Srikanth and Toueg [Srikanth 87] protocol. In the context of clock synchronization, it is understood that all statements of the protocol are concurrently executed. Furthermore, the protocol is executed continuously at every local clock tick, unless stated otherwise.

Due to the ambiguity of the description of the protocol as described in [Daliot 03] various interpretations are possible. To avoid unintended interpretations, the authors of the paper were contacted and some of the issues were clarified. The following is our understanding of the intended protocol.

- The protocol is executed continuously.
- All statements are executed at every tick. However, S2 sends a "Propose-Pulse" once and only when it reaches the threshold value of f+1 as opposed to repeatedly at and after reaching the threshold.
- A good node counts its own message.

The Counterexamples

In the counterexamples presented here we show that the BSS-Pulse-Synch protocol [Daliot 03] does not converge. Table 1 is an execution trace of a system with parameters $n = 4$, $f = 1$, Cycle = C, with no clock drift, $\rho = 0$, i.e. $\lceil 2d(1 + 2\rho)\rceil = 2d$, all clocks starting in phase, and $d = 1$ tick. Node 4 is the faulty node while nodes 1, 2, and 3 are good nodes. Table 2 is another execution trace of a system with parameters $n = 4$, $f = 1$, Cycle = C, with clock drift $\rho \geq 0$, i.e. $\lceil 2d(1 + 2\rho)\rceil = 2d$ or $3d$, all clocks starting in phase, and $d = 1$ tick. The state of each node is represented by $(C - t)$, in time units since the last pulse event, with the stored propose-pulse message as superscripts. Symbol 'x' represents a received message and symbol '–' represents no message received from the corresponding node, 4 positions, one for each node. The types of faults considered are symmetric and asymmetric (a.k.a. Byzantine).

The tables have four columns, one for time reference and one for each good node. A row of the table depicts activities of all good nodes, in their corresponding columns, for that time tick. As is shown in Table 1 the system starts from a random state where the nodes are $4d$ apart and reaches the same state within 5 ticks. This process repeats indefinitely. The faulty node in this counterexample is symmetric. The symmetrically faulty node transmits its message to all nodes at t+0, t+2, and t+5. At t+1, nodes 1 and 2 ignore that message while node 3 accepts it.

Table 1. The counterexample for $\rho = 0$, $2d(1 + 2\rho) = 2d$, and symmetric fault.

Time	Node 1	Node 2	Node 3
t + 0	$(\text{C-5})^{\text{xxx-}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-1})^{\text{----}}$, ignore	$(\text{C-3})^{\text{x---}}$
t + 1	$(\text{C-1})^{\text{----}}$, ignore	$(\text{C-2})^{\text{----}}$, ignore	$(\text{C-4})^{\text{x--x}}$, send
t + 2	$(\text{C-2})^{\text{----}}$, ignore	$(\text{C-3})^{\text{--x-}}$	$(\text{C-5})^{\text{x-xx}} \Rightarrow \text{C}^{\text{----}}$
t + 3	$(\text{C-3})^{\text{---x}}$	$(\text{C-4})^{\text{--xx}}$, send	$(\text{C-1})^{\text{----}}$, ignore
t + 4	$(\text{C-4})^{\text{-x-x}}$, send	$(\text{C-5})^{\text{-xxx}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-2})^{\text{----}}$, ignore
t + 5	$(\text{C-5})^{\text{xxx-}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-1})^{\text{----}}$, ignore	$(\text{C-3})^{\text{x---}}$

In Table 2 the system starts from a random state where the nodes are $4d$ apart and reaches the same state within 6 ticks. This process repeats indefinitely. Therefore, the system does not converge and the precision remains $4d$, $2d$ more than the expected $2d$ value. This table can be viewed from many angles. For instance, if $1 \gg \rho > 0$ and $\lceil 2d(1 + 2\rho) \rceil = 3d$ time units, then the type of faulty node that results in this counterexample is symmetric. The symmetric faulty node transmits its message to all nodes at t+0, t+2, t+4, and t+6. At t+1, nodes 1 and 2 ignore that message while node 3 accepts it. However, if $\rho = 0$ and $2d(1 + 2\rho) = 2d$ time units, the faulty node in this counterexample is asymmetric. The asymmetrically faulty node transmits its message to one node at a time, specifically, at t+0 to node 3, at t+2 to node 2, and at t+4 to node 1.

Table 2. The counterexample for $\rho > 0$, $\lceil 2d(1 + 2\rho) \rceil = 3d$, and symmetric fault and for $\rho = 0$, $2d(1 + 2\rho) = 2d$, and asymmetric faulty.

Time	Node 1	Node 2	Node 3
t + 0	$(\text{C-6})^{\text{xxx-}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-2})^{\text{----}}$	$(\text{C-4})^{\text{x---}}$
t + 1	$(\text{C-1})^{\text{----}}$	$(\text{C-3})^{\text{----}}$	$(\text{C-5})^{\text{x--x}}$
t + 2	$(\text{C-2})^{\text{----}}$	$(\text{C-4})^{\text{--x-}}$	$(\text{C-6})^{\text{x-xx}} \Rightarrow \text{C}^{\text{----}}$
t + 3	$(\text{C-3})^{\text{----}}$	$(\text{C-5})^{\text{--xx}}$	$(\text{C-1})^{\text{----}}$
t + 4	$(\text{C-4})^{\text{-x--}}$	$(\text{C-6})^{\text{-xxx}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-2})^{\text{----}}$
t + 5	$(\text{C-5})^{\text{-x-x}}$	$(\text{C-1})^{\text{----}}$	$(\text{C-3})^{\text{----}}$
t + 6	$(\text{C-6})^{\text{xxx-}} \Rightarrow \text{C}^{\text{----}}$	$(\text{C-2})^{\text{----}}$	$(\text{C-4})^{\text{x---}}$

Discussion

There is a vast literature on the topic of clock self-stabilization. Although there is no definitive guideline for a design of a distributed protocol, there are some invaluable points to keep in mind in the design process. Some of these points are well known within the community while others are not so obvious. In particular, below are a couple of points from [Kopetz 97].

- *If two nodes broadcast their messages at the same time, there is no guarantee in the order of arrival of the messages at other nodes.*
- *A consistent delivery order of a set of events in a distributed system does not necessarily reflect the temporal or causal order of the events.*

In addition to the above points, our research has resulted in a number of other key points that are essential for the design of a protocol. Some of the pertinent findings are stated here along with an observation. These remarks are in the context of a distributed system with $n \geq 3f + 1$ and all good nodes actively participating in the self-stabilization process.

- *At random start up, a good node could be observed asymmetrically.*
- *A faulty node can increase the likelihood of a good node being observed asymmetrically.*

We have also observed that a good node should be responsive to the incoming messages at all times. In other words, a blank rejection of all messages from other nodes for any length of time and during the self-stabilization process is to be avoided. The counterexamples provided here are a direct result of violation of this observation. Consequently, a hand simulation resulted in the compact counterexamples reported here. To verify the counterexamples, we modeled the protocol in Stochastic Model checking Analyzer for Reliability and Timing (SMART) [Ciardo 03] and reproduced the counterexamples. In addition, the model checker explored all various startup scenarios that eventually lead to the failure of the protocol. All traces produced by the model checker are variations of the above counterexamples. Most traces took many cycles to reach the given state. Other traces show that the protocol falls in and out of such states over many cycles depending on the behavior of the faulty node. The counterexamples presented here are the most compact scenarios that capture the essence of the flaw in the proposed protocol. These counterexample reveal the repetition cycle is significantly less than the Cycle as specified by the BSS-Pulse-Synch protocol.

Also, due to the ambiguity of the description of the protocol, different variations of the protocol were modeled. In particular, we wondered if it mattered whether a good node counted its own message or not. Also, does it matter whether statement S2 were to send a message even after crossing the (f+1) threshold. All variations of the protocol suffered from the same flaw and, thus, resulted in similar counterexamples.

The fundamental flaw in the design of the BSS-Pulse-Synch protocol is that it failed to consider that good nodes might be observed asymmetrically. In order for a distributed system to converge, it is essential that eventually all good nodes reach a point where they all have a consistent view of each other, but the system cannot be assumed to start in such a state.

The main challenges associated with self-stabilization are complexity of the design and proof of correctness of the protocol. As is evident here, although a mathematical hand proof for the BSS-Pulse-Synch protocol was provided in [Daliot 03], that proof was found to be flawed. Because self-stabilization is a notoriously subtle and difficult problem, it is recommended that mathematical proofs of proposed solutions be rigorously examined using formal methods. One way of accomplishing such goal is mechanical verification of the proofs via theorem proving, i.e. HOL, PVS, SAL, etc., or model checking, i.e. SMART, SMV/NuSMV, SPIN, etc. Mechanically checked proofs are the only way we can have strong assurance that all possible scenarios are covered.

The authors of the BSS-Pulse-Synch protocol have acknowledged the flaw and have since proposed other solutions to the problem [Daliot 05]. However, these newly proposed solutions are yet to be analyzed.

Acknowledgment

This work was supported by NASA's Vehicle Systems Program. We'd also like to thank Wilfredo Torres-Pomales, Paul Miner, Victor Carreno, Cesar Munoz, and Ricky Butler for their invaluable comments.

References:

[Ciardo 03]	Ciardo, Gianfranco; Siminiceanu, Radu: Structural Symbolic CTL Model Checking, CAV 2003, Boulder, Colorado, LNCS 2725, pp.40-53, July 2003.
[Daliot 03]	Daliot, Ariel; Dolev, Danny; Parnas, Hanna: Linear Time Byzantine Self-Stabilizing Clock Synchronization, Proceedings of 7th International Conference on Principles of Distributed Systems (OPODIS-2003), La Martinique, France, December 2003.
[Daliot 05]	Daliot, Ariel; Dolev, Danny: Self-stabilizing Byzantine Pulse Synchronization, Technical Report TR2005-84, School of Engineering and Computer Science, The Hebrew University of Jerusalem, Aug. 2005.
[Dijkstra 74]	Dijkstra, E.W.: Self stabilizing systems in spite of distributed control, Commun. ACM 17,643-644m 1974.
[Dolev 84]	Dolev, Danny; Halpern, J.Y.; Strong, R.: On the Possibility and Impossibility of Achieving Clock Synchronization. In proceedings of the 16th Annual ACM STOC (Washington D.C., Apr.). ACM, New York, 1984, pp. 504-511. (Also appear in J. Comput. Syst. Sci.)
[Driscoll 03]	Driscoll, Kevin; Hall, Brendan; Sivencronam, Hakan; Zumsteg, Phil: Byzantine Fault Tolerance, from Theory to Reality: Computer Safety, Reliability, and Security, Springer-Verlag Heidelberg, ISBN: 3-540-20126-2, Volume 2788 / 2003, October 2003, pp. 235 - 248
[Lamport 82]	Lamport, Leslie; Shostak, Robert.; Pease, Marshall: The Byzantine General Problem, ACM Transactions on Programming Languages and Systems, 4(3), pp. 382-401, July 1982.
[Lamport 85]	Lamport, L; Melliar-Smith, P. M.: Synchronizing clocks in the presence of faults, J. ACM, vol. 32, no. 1, pp. 52-78, 1985.
[Kopetz 97]	Kopetz, H: Real-Time Systems, Design Principles for Distributed Embedded Applications, Kluwar Academic Publishers, ISBN 0-7923-9894-7, 1997.
[Srikanth 87]	Srikanth, T. K.; Toueg, Sam: Optimal Clock Synchronization. Journal of the Association for Computing Machinery, Vol. 34, No. 3, July 1987, pp. 626-645.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 02 - 2006	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Comments on the "Byzantine Self-Stabilizing Pulse Synchronization" Protocol: Counterexamples	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Malekpour, Mahyar R.; and Siminiceanu, Radu	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 23-645-84-06
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-19203
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM-2006-213951

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 62
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov

14. ABSTRACT

Embedded distributed systems have become an integral part of many safety-critical applications. There have been many attempts to solve the self-stabilization problem of clocks across a distributed system. An analysis of one such protocol called the Byzantine Self-Stabilizing Pulse Synchronization (BSS-Pulse-Synch) protocol from a paper entitled “Linear Time Byzantine Self-Stabilizing Clock Synchronization” by Daliot, et al., is presented in this report. This report also includes a discussion of the complexity and pitfalls of designing self-stabilizing protocols and provides counterexamples for the claims of the above protocol.

15. SUBJECT TERMS
Byzantine Fault; Byzantine Pulse Synchronization; Clock Synchronization; Counterexample; Formal Verification; Self-Stabilization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	12	(301) 621-0390