A MEMBERSHIP AGREEMENT ALGORITHM DETECTING AND TOLERATING ASYMMETRIC TIMING FAULTS

Håkan Sivencrona[1], Mattias Persson[2] and Jan Torin[2]
[1]SP Swedish National Testing and Research Institute, Software Electronics, Brinellgatan 4, SE-501 15 Borås, Sweden; [2]Chalmers University of Technology, Department of Computer Engineering, Rannvagan 6, SE 412 96 Göteborg, Sweden

Abstract: Our paper presents a new membership agreement algorithm that address asymmetric timing faults and includes a new tool simulating TTP/C clusters. The proposed algorithm flags deviating or slightly untimely messages to assure that single marginal transmitting faults are detected and that only the faulty node will be expelled. The tool can demonstrate the behavior of membership agreement algorithms such as the original TTP-C1 algorithm or our modified flagging algorithm. The performed simulations use experimental results from heavy-ion fault injection logged timing faults. The gathered results show the rare faults, which made a network using the original algorithm either collapse or become degraded, are detected and handled with the new algorithm without loss of more than the faulty node.

Key words: Membership agreement; Asymmetric timing faults; Fault detection.

1. INTRODUCTION

A severe type of communication faults is the infamous Byzantine fault class, which includes so-called slightly-out-of-specification faults[1], SOS, which may cause inconsistencies at the communication level in distributed systems when a number of nodes receive a message while other nodes fail to correctly receive the same message. This may affect the application with problems such as reaching application consensus.

In the FIT project (IST-1999-10748)[2], a time-triggered architecture[3] was evaluated by use of several fault injection techniques. One major finding was that the fault detection and error processing with respect to SOS faults was

insufficient[1], which had major effects on the application due to communication black out and the degraded operation of the cluster.

There are basically two approaches to design a system for a specific fault tolerance: a) to have sufficient redundancy to mask the Byzantine fault[4], b) to implement methods to identify (diagnose) and reconfigure the system before additional faults arrive[5-7]. We present a diagnosis algorithm where the basic idea is that messages are "quality stamped" with respect to the timelines they demonstrate at each receiving node. The designed algorithm differs significantly compared to the original implementation of TTP/C where no effort has been put to detect a specific node, rather the caused inconsistency, solved through minority partition reintegration.

The paper is organized as follows. First we briefly present TTP/C, especially mechanisms[8-11] vital for the development and understanding of our algorithm such as membership agreement and clique avoidance. Secondly Byzantine faults and slightly-out-of-specification faults are described. Then the membership agreement is presented with respect to asymmetric faults while the simulation tool is presented in section five. Section six contains the simulation results while section seven concludes the paper.

2. TIME-TRIGGERED PROTOCOL CLASS C

TTP/C[3] basically provides three services; clock synchronization[9], deterministic message sending, and a membership service[8]. The clock synchronization information uses the FTA clock synchronization algorithm where the time data is extracted from the arrival of the latest four messages. The two most extreme clock values are removed and the sum of the remaining clocks' values is averaged, a correction term. The nodes will adjust their clocks with the correction term and thus remain synchronized with the cluster.

The membership agreement[8,10] in TTP/C is represented by a unique identification vector, which is stored in all nodes as a local membership vector. All nodes update the membership information continuously. The membership service is closely coupled to features such as clique avoidance[10], which further improve the error handling capabilities in a distributed system.

If the membership vectors differ between sending and receiving nodes, the CRC calculation should not produce a readable message. When a CRC error is found, the receiving node raises a membership error for the sending node locally (after some internal checks such as implicit acknowledgement algorithm, see below) and the corresponding membership vector value is set

to false. If the frame is not semantically correct it is considered as an invalid frame, which could be due to a transmission error, and the membership bit is in this case set to false. If a node discovers that it is not in agreement with the majority of the active nodes in the cluster, it is not allowed to send and has to reintegrate this is guaranteed by the clique avoidance algorithm.

Starting with its own sending slot all nodes in a cluster counts all incorrect nodes in a fail counter, *FC* and correct nodes in an accept counter, *AC* during one TDMA round. $AC + FC = n$, where n is the number of nodes in the system before failure (or operating). Null frames are not counted by either *FC* or *AC* and it is assumed that all nodes detect Null frames thus n is decreased by detected Null frames. If $FC > 1$ the cluster is considered a partitioned cluster otherwise the acknowledgement algorithm is enough to retain a consistent cluster.

A node may transmit if $FC < n/2$ if n is even and $(n+1)/2$ if n is odd which results in that *FC* must be $< n/2$. An in depth explanation can be found in[10].

When a node, lets say *A*, has sent a message it usually (can be a sending error) increases the *AC* counter. If it detects a CRC error during the two succeeding TDMA slots it does not passively await the resolution of the upcoming situation. TTP/C has an algorithm to address this situation, the implicit acknowledgement algorithm.

The algorithm introduces the denotation of first and second successor. If the first succeeding node *B* does not have the same membership list, *A* decides to preliminary remove *B* from its local membership list and increases its *FC*. If it was *B* that was receive-faulty the situation will be solved by a second successor node, *C*. If *C* sends a syntactically correct message, e.g. with B removed from the membership, node *A* is acknowledged. If not, *B* is probably acknowledged and *A* has been removed from *C*s membership, *A* is thus not acknowledged. *A* was as a result faulty. A removes itself from the membership list but adds both *B* and *C* to the membership list and updates *AC* and *FC* correspondingly and then reintegrates. The implicit acknowledgment algorithm is completed.

3. BYZANTINE FAULTS

Since its initial presentation the Byzantine Generals problem[12] has been the subject of intense academic scrutiny, leading to the development of numerous Byzantine-tolerant algorithms and architectures[6,7,13]. A sub-class of Byzantine faults are slightly-out-of-specification faults[1] which occur in the transition between the analog and discrete world. In the time domains these faults occur when entities get different views of the time, a marginal

transmission. A message that is timely at a certain node may be considered as untimely at another node due to different drifts of the local clocks or internal errors.

In TTP/C all nodes have a start of frame window where an expected message must be received (with respect to start of frame transmission) to be processed further by the node and application. This window is usually so small that a feasibly low jitter can be achieved. When the receive window has opened up for message reception a counter is incremented every microtic, from for example -40 microtics before expected arrival time and up to 40 microtics after expected arrival time, a span of 81 microtics. Within this span the design specific window is specified, for example -20 < acceptable reception time < 20 microtics. Settings for this window depend on the size of the TDMA slots and the time for a TDMA round.

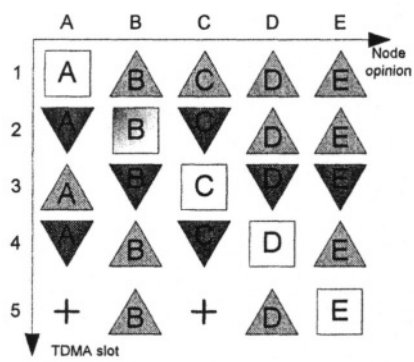

Figure 1. TTP/C membership agreement and clique avoidance handling an Asymmetric fault (node B sending in slot two) + = nodes resetting

In Figure 1, an asymmetric fault scenario is shown. Triangles pointing upward mean agreement with the sent message while triangles that point down is in disagreement while squares mean sending node. When *B* transmits, second row, node *A* and *C* raise an error. *D* and *E* on the other hand accept the message.

In the third TDMA slot (when *C* sends its opposing opinion to the system) the inconsistency is known to parts of the system (node *B, D* and *E*) while *A* still regards the upcoming situation as a normal fault (non-asymmetric).

D is the next node to transmit and must now decide which opinion it should have (Assume for a while that the message sent by *C* can be viewed by *D,* in TTP/C this would have caused a CRC error). In TTP/C it does not accept the message from *C*. We now have a situation where two nodes out of five have expressed their opposing view. *D* uses, in this case, the clique avoidance algorithm to decide. The upcoming situation is solved as

described in Figure 1. Node *A* and *C* will cease their transmission because their fail counters will be larger than their accept counters. The faulty node is on the other hand still synchronized and remains undetected.

4. FLAGGING ALGORITHM

The proposed algorithm uses some assumptions. Only one node is faulty during one TDMA round. The algorithm is assumed to be a transparent layer on top of the ordinary TTP/C mechanisms, not voiding any of the original properties.

Following assumptions have been used with respect to the receive window, see Figure 2:

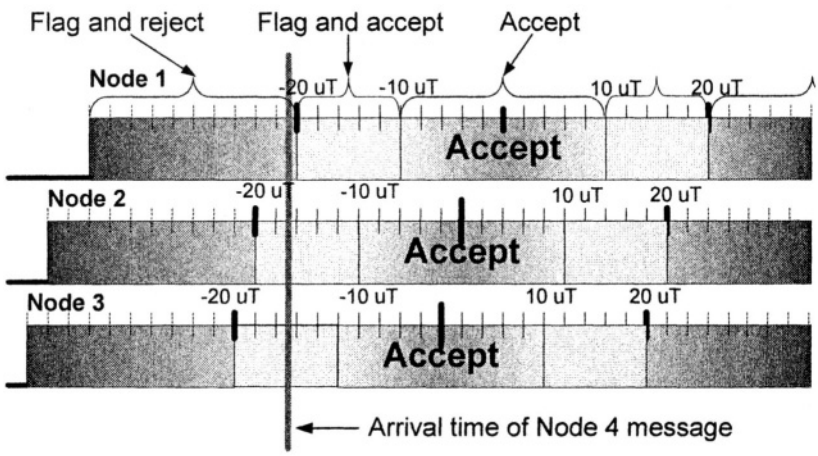

Figure 2. The modified receive window where node 4 is received early, too early by node 1

• A message (node) is declared invalid and removed from membership list (if it arrives later than 20 microtics or earlier than 20 microtics from expected arrival time. –20 < expected arrival < 20, a window of 40 microtics. A message received outside this window of 40 microtics is also flagged.

• A node is not allowed to transmit if the own clock synchronization calculation results in a clock correction term that is larger than 10 microtics

• A message that arrives within 10 to 20 microtics from expected time is declared as a message possible SOS-message. The message is flagged in an internal register by the communication controller.

• A node that has a non-empty flag register will not immediately remove disagreeing nodes without an extra check with respect to flag position

If a received message, *M*, is declared invalid due to a faulty CRC calculation (non-readable message) the receiving node *R* will check its flag register for flags, since voting message *M* as invalid at this state could result in that the majority regards *R* as a faulty node. *R* will thus assume that the flag corresponds to a node *F* that the sending node *S* has removed from its membership, thus causing the CRC error. *R* changes the corresponding bit in the membership vector and recalculates the CRC. Following this assumption node *R* tries once more to access message *M* and if this succeeds it will accept the message as valid assuming that it was because of a timing fault *S* had removed *F* which *R* did only flag. All other nodes that have the same view about this node and flagged the same will update their membership vectors correspondingly. All nodes will thus have removed node F that was flagged by everyone but the faulty and removed from membership by at least one. In cases when not all nodes, except the faulty, have flagged the node the original algorithm will solve the situation. All nodes will then recount their accepted messages and update their fail counters and accept counters accordingly. This way all nodes should, in case of a single fault, have the same consistent view of operating nodes within one TDMA round. Any shortage of the algorithm will be solved within a second TDMA round, see Figure 3 for a corresponding situation.

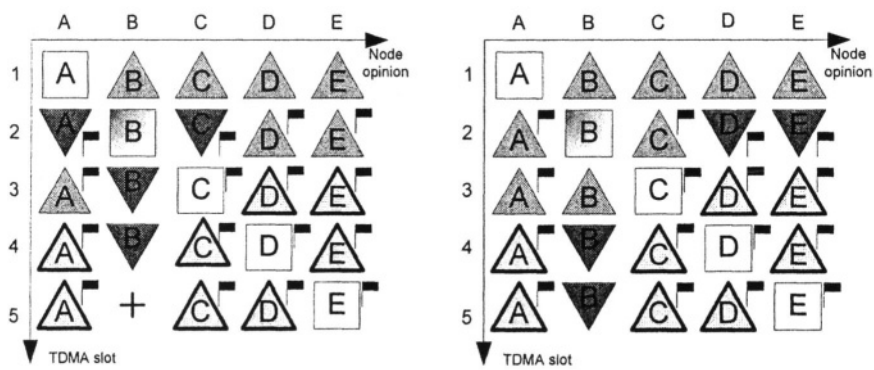

Figure 3. Membership behavior under two fault scenarios

One variation of an SOS faults shows up if the third node, *C,* flags message *B* and transmits this knowledge. *C* has thus not removed *B* from membership while the successor *D* has already done so, see Figure 3 (right). This means that node *D* will receive a message that could differ on two places with respect to the own membership list (*D* has already removed *B* from membership and has a pending membership change on *C*, meaning it would have removed *C* from the membership list using the original algorithm). But *C* and *D* do only disagree about their opinion concerning *B*.

Node *D* will still keep the opinion about *B* and raise a CRC error on *B,* but accept node *C*. *D* will be notified about this agreement about *B* upon inverting the membership bit in question (flagged position). This will cause a transient inconsistency in the system, which is a complex state, meaning we disagree over one message but accepts each other, which is against the prerequisite of the original membership agreement protocol utilization but a prerequisite for the accurate operation of the flagging algorithm.

Figure 3 furthermore shows that node *D* and *E* did not accept the message in TDMA slot 2 and that node *C* must change its already public membership view. *D* and *E* will *win* while *A* and *C* will adapt to this view.

The algorithm is interpreted in Figure 4 which shows the flowchart of the algorithm which is then implemented and tested using our tool.

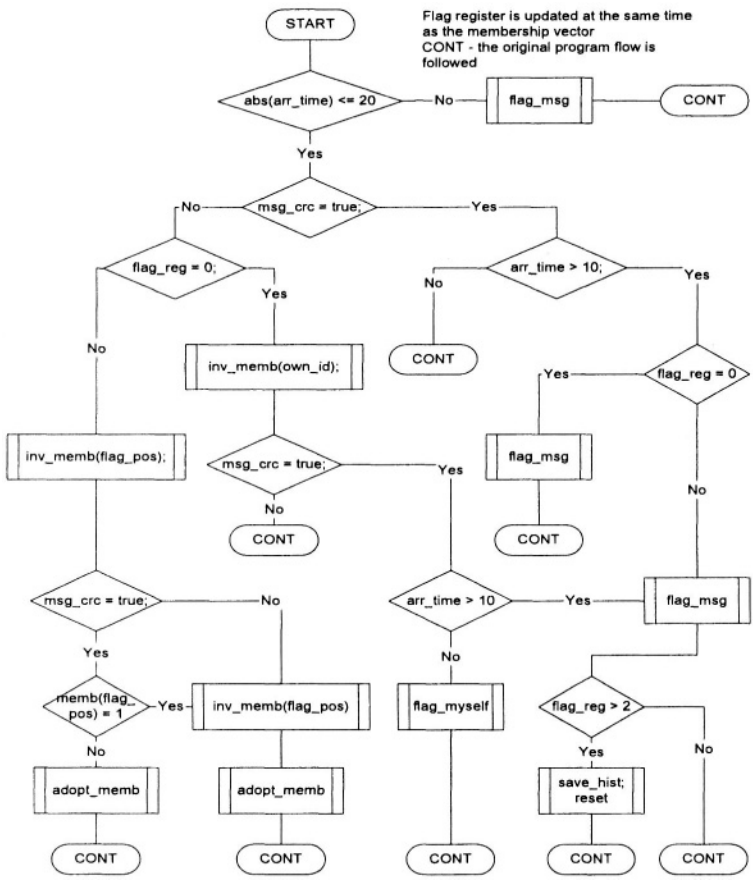

Figure 4. A flowchart of the algorithm

5. SIMULATION SETUP

The purpose of the TTP/C algorithm simulator is to provide a simple simulation environment for a TTP/C network. It simulates a network of *n+1* nodes, where *n* is the number of nodes for which experimental log files exists, where one node is assumed to be the fault-injected without any logged data available, see Figure 5.

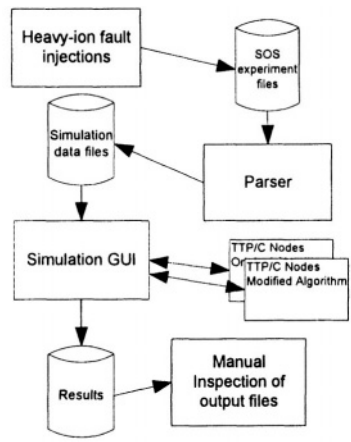

Figure 5. The basic work flow of the simulation tool environment

The application mainly consists of three parts; the Simulator GUI which controls the flow of the application, the Parser which parses the experiment files and the TTP/C Node, simulating the behavior of a real TTP/C node. Using the object orientation principle, all significant data structures such as the TTP/C Message and the Membership Vector are represented by objects.

Through the Simulator GUI the user controls the simulation. After invoking the Parser, the Simulator sets up the simulation and calls the TTP/C nodes. Because not all experiment logs are identical, the Parser is actually a Java interface. This allows the user to tailor-make one Parser per log file type, as long as it contains a specified method to parse a set of log files. When running, the parser creates a scenario from n-1 files, where n is the size of the cluster.

Finally, the TTP/C Nodes consists of TTP/C protocol implementations. Utilizing the same versatility as the Parser, the protocol is also an interface. Simplifying the program flow, each simulated node actually processes all messages received from one TDMA round at the same time. The TTP/C message is distributed to all “nodes” in the network, using the Membership vector calculated by the protocol implementation and the time drift obtained from the same slot in the logged scenario.

6. SIMULATION RESULTS

The SOS scenarios were first executed with the original algorithm. Figure 6 shows the last part of the printout of one experiment. When the same object file was executed using the flagging algorithm the cluster remained synchronized but the faulty node was detected and expelled from the membership at all nodes.

One type of scenario was not solved perfectly. The flagging algorithm did not resolve situations when more than one node was badly synchronized, meaning this node did not flag the SOS-node. But the cluster remained synchronized in throughout al test cases but in some cases with two nodes expelled.

```
Node 3: Message from node 0;
Sender 0, NULLFRAME
Node 3 thinks node 0 is NOK
Node 3: Message from node 1;
Sender 1, NULLFRAME
Node 3 thinks node 1 is NOK
Node 3: Message from node 2;
Sender 2, NULLFRAME
Node 3 thinks node 2 is NOK
Node 3 Membership: 0x0
Performing acknowledgement algorithm,
my memb is 0x0 and the 1st succ's
message is Sender 0, NULLFRAME
Performing check 2A because 1st Succ
sent a null frame
Node 3 NOT acknowledged
Round 4, Slot 19, Got message Sender 3,
NULLFRAME
ALL NODES KICKED OUT
Simulation Aborted
  ***Simulation complete!***
Node 0gets drift 5
Node 1gets drift 9
Node 2gets drift 2
Exiting...
```

```
Node 3: Message from node 0;
Sender 0, Drift = 0, Membership = 0x7
Node 3 thinks node 0 is OK!
Node 3: Message from node 1;
Sender 1, Drift = 0, Membership = 0x7
Node 3 thinks node 1 is OK!
Node 3: Message from node 2;
Sender 2, Drift = 0, Membership = 0x7
Node 3 thinks node 2 is OK!
Node 3 Membership: 0x7
Performing acknowledgement algorithm,
my memb is 0x7 and the 1st succ's
message is Sender 0, Drift = 0, Membership = 0x7
Performing check 2A because 1st Succ thinks I'm out
Node 3: Check 2B passed!
Node 3 NOT acknowledged
Round 9, Slot 39, Got message Sender 3,
NULLFRAME
Node 0gets drift 5
Node 1gets drift 9
Node 2gets drift 2
  ***Simulation complete!***
```

Figure 6. Scenarios using old algorithm (left) and the flagging algorithm (right) where node 3 is the faulty node

7. CONCLUSION

We have presented an algorithm for increasing the tolerance against asymmetric timing faults in a time-triggered protocol (TTP-C1). The major conclusion is that any single untimely node will be disclosed and that a global agreement can be reached about the system state, including time within two TDMA rounds.

The algorithm does not guarantee that all SOS faults are detected, at least not with respect to the chosen parameters. The success depends on the ratio

between the different windows, e.g. the width of the flag window compared to accept window.

We have furthermore provided and presented an uncomplicated simulator GUI for a TTP/C network. The TTP/C Simulator can mimic a TTP/C network of *n+1* nodes, where *n* is the number of nodes for which log files exists, where one node is assumed to be a fault-injected node without any logged data available, as was the case when a single TTP/C communication controller was injected with heavy-ions.

REFERENCES

1. H. Sivencrona, P. Johannessen, M. Persson and J. Torin, Heavy-ion Fault Injection in the Time-triggered Communication Protocol. Proc. First Latin American Symposium on Dependable Computing (LADC03), São Paulo, Brazil, October (2003).
2. FIT-project at http://www.cordis.lu/ist/projects/99-10748.htm, (2002).
3. H. Kopetz, TTP/C Protocol, Available at http://www.ttpforum.org. (1999).
4. K. Driscoll, B. Hall, H. Sivencrona and P. Zumsteg, Byzantine Fault Tolerance, from Theory to Reality. Proc. 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP03), pp. 235-248, Edinburgh, Scotland, UK, October 2003.
5. J. Rushby, Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms, IEEE Transactions on Software Engineering, Volume 25, Number 5, September, pp: 651-660, (1999).
6. A. Ademaj, H. Sivencrona, G. Bauer and J. Torin, Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology. Proc. International Conference on Dependable Systems and Networks (DSN 2003), pp. 123-132, San Francisco, USA, (2003).
7. K. Hoyme and K. Driscoll, SAFEbus. Proc. Digital Avionics Systems Conference (AIAA-11), pp. 68 -73, Seattle, WA, USA, (1992).
8. H. Kopetz, G. Grünsteidl and J. Reisinger, Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System, in Dependable Computing for Critical Applications, pp. 411- 429, Springer-Verlag, Vienna, Austria, (1991).
9. H. Kopetz and W. Ochsenreiter, Clock Synchronization in Distributed Real-Time Systems, IEEE Transactions on Computers. Vol. 36, Nr. 8, pp. 933-940, (1987).
10. G. Bauer and M. Paulitsch, An Investigation of Membership and Clique Avoidance in TTP/C, Proc. of the 19th IEEE Symposium on Reliable Distributed Systems, pp. 118-124, Nuremberg, Germany, (2000).
11. A. Merceron, Proving “no Cliques” in a Protocol, Computer Science Conference, (ACSC 2001), Proc. 24th Australasian, pp: 134 –139, (2001).
12. L. Lamport, R. Shostak and M. Pease, The Byzantine generals problem, ACM Transactions on Programming Languages and Systems, vol. 4 issue 3, pp. 382-401, (1982).
13. L. Gong, P. Lincoln and J. Rushby, Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults. Proc. Dependable Computing for Critical Applications (DCCA-5), volume 10 of Dependable Computing and Fault Tolerant Systems, pp. 139-157. IEEE Computer Society, (1995).