Proceedings of the Pakistan Academy of Sciences:
A. Physical and Computational Sciences 56 (1): 21–28 (2019)

ISSN: 2518-4245 (print), 2518-4253 (online)

Pakistan Academy of Sciences

Research Article

Comparative Analysis of Controller Area Network's Migration Code in a "Shared-Clock" Environment

Muhammad Amir[1*], Syed Waqar Shah[1], Bilal ur Rehman[1], and Michael J. Pont[2]

[1]Electrical Engineering Department, University of Engineering and Technology, Peshawar
[2]SafeTTy Systems Ltd, United Kingdom

Abstract: Industrial and automotive applications since long have been utilizing the Controller Area Network (CAN) protocol for their communications needs. History relating to the use of CAN suggests that although it is cost-effective and less complex; a lack of flexibility and an incomprehensive fault-management strategy makes its use scarce in safety-critical applications. This scarcity of CAN in safety related applications is primarily due the fact that this protocol was originally implemented using a bus-topology. Previously, through our research, a migration of CAN from bus to star topology was suggested. The results of which exhibited that such a migration positively impacted the flexibility and fault-management capability of CAN. Here, in this paper, a comparative analysis of the codes used for both CAN architectures (bus and star) is presented. The analysis exhibits that such a fruitful migration can be achieved through almost the same software-overhead and complexity as was in the original CANbus-based architecture.

Keywords: CAN, Topology, Migration, Code-Volume.

1. INTRODUCTION

The bus-topology-based CAN protocol was introduced in the early 1990's specifically for automotive applications [1]. Since then, it has infiltrated many field-systems that rely on control [2], [3], [4] and [5].

1.1 Problem Statement & Motivation

Nowadays, almost all systems (e.g. avionics, automotive, marine transport (including ships, ferries, boats & submarines), industrial process control systems, building elevator systems, home appliances, security systems, fire safety systems, CCTV systems, ATM systems, Automated bomb disposal systems etc) are controlled through embedded systems that are invisible to us as users. These systems are known as embedded because we as users only see the system's level of their applications. In the above examples, if for a moment, we consider avionics, automotive, industrial, fire safety & elevator systems that are prime safety critical applications of embedded systems then it is worth thinking that each day we trust these systems with our lives. It is thus required that such safety-critical systems are governed by protocols that are flexible and fault- tolerant [6] and [7].

To us, previous research on CAN suggested that although cost-effective and simple, the protocol was neither flexible in "Shared-Clock" (SC) environments [8] nor it possessed an inherent fault-management strategy [9], [10] and [11]. Keeping in mind such limitations of CAN, one can easily move towards the more complex/expensive protocols like Time-Triggered Protocol (TTP) [12] and FlexRay [13]. On the contrary, our research motivated us to progress towards making CAN more flexible & fault-managed in order to keep the simplicity/cost-effectiveness by suggesting a topology migration from bus towards star-based architectures [14], [15] and [16]. As elaborated in [8], for real-time embedded safety-critical time-triggered (TT) architectures, one of the major concerns for system designers at system's operation phase is jitter. Jitter

Received: September 2017; Accepted: March 2019
* Corresponding Author: Muhammad Amir <amir@nwfpuet.edu.pk>

is simply defined as the delay introduced in system's response time due to task context switching either in co-operative or pre-emptively scheduled TT architectures [16]. Such delays are caused due to scheduler-software-overheads and incur a negative effect on the response timing of systems where safety is the main concern. This research paper presents a comparison between CAN-bus and CAN-star implementations of industrial process control, pivoting primarily on code volume and complexity metrics.

The comparison will show that such a topology migration can be achieved with less scheduling-software-overhead in order to keep jitter levels at minimum [8]. This manuscript is constituted in a manner such that: Section 2 describes the setup on which our proposed migration was carried out. In Section 3, comparative results of our case-study based on values of code volume and complexity metrics of both topology implementations are laid out. Section 4 entails a discussion centering on the obtained metrics values. Finally, in Section 5 we present our conclusion.

2. MIGRATION & CASE-STUDY RIG

The topology shift for CAN was tested in a manner that encompassed firstly an implementation of the case-study using a CANbus architecture running on a CANbus-based algorithm termed as Time-Triggered Cooperative-Shared-Clock1 (TTC-SC1) [8] and secondly, the architecture was shifted towards a CANstar design and automated/tested through a CANstar-based algorithm termed as Time-Triggered Cooperative-Shared-Clock5 (TTC-SC5) [15]. It is important to note here that the task-sets, their schedules and their execution deadlines on the Master and Slave nodes were kept identical in both implementations. It keenly suggests that only the topology of CAN architecture was shifted from bus to star. The case-study rig, depictions of topologies and their descriptions are presented in this section as following:

The "RIG" or process control apparatus used for both implementations is shown in Fig. 1. On the far right in Fig. 1 is the "Basic Process Rig" (BPR) used to simulate fluid (water/chemical) flow while on far left is the heating element known as the "Temperature Process Rig" (TPR) used to simulate the boiler heating process in an industrial setup. The module in the middle shown in Fig. 1 is the "Forced Air Cooler" (FAC) kept them in the process for emergency cooling of fluids in the setup. The entire RIG is known as "PROCON" (short for process control) apparatus and is supplied by Feedback Instruments, UK [17]. On the other hand, Fig. 2 shows the original CANbus based implementation of our case study. The Master node is in the middle accompanied by Slave nodes on left and right. The Master node is a developmental board housing an LPC2294 microcontroller [18] with 4xCAN interface support. It is supplied by Olimex [19]. Moreover, the Slave nodes are also development boards having LPC-2129 microcontrollers [18] with 2xCAN interface support [19]. The Master node here is responsible for deployment of TTC-SC1 protocol through the original CANbus topology as well as for notifying system status as shown in

Fig. 1. Process Control setup (Rig) used for topology migration

Fig. 2. CANbus-based implementation (one Master and two Slaves)

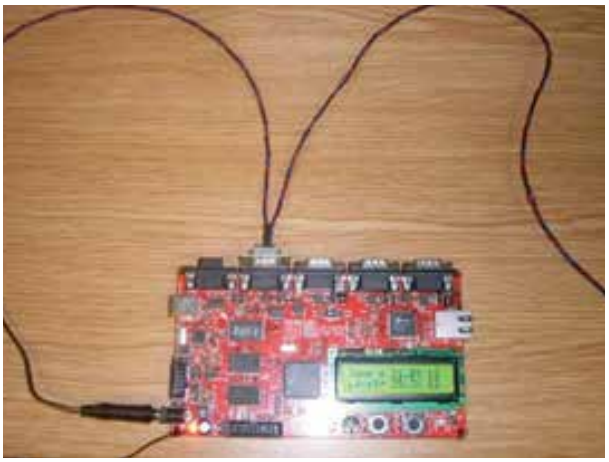

Fig. 3. CANbus-based implementation (Master node)

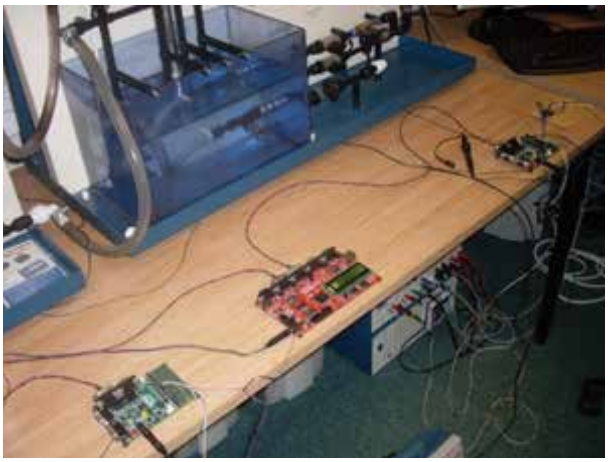

Fig. 5. CANstar-based implementation (one Master and two Slaves)

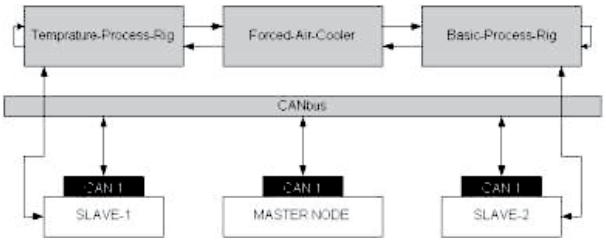

Fig. 4. Block diagram of CANbus-based implementation

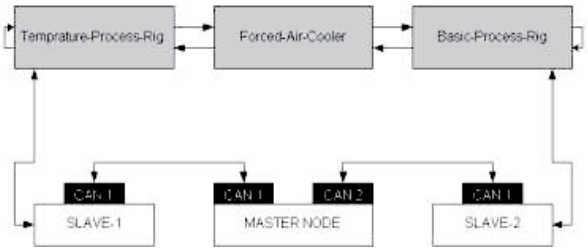

Fig. 6. Block diagram of CANstar-based implementation

Fig. 3. Furthermore, the Slave (seen on the right in Fig. 2) is responsible for controlling the flow of water inside the BPR while the Slave (seen on the left in Fig. 2) is responsible for temperature control of the heating element inside the TPR.

It is important to mention here that part of the TTC-SC1 protocol (specific to Slaves) is run by the abovementioned Slaves. The block diagram of our CANbus based setup shown in Fig. 2 is given in Fig. 4. It is evident from the block diagram that Master and Slaves are sharing the same CANbus for communication while deploying the TTC-SC1 protocol. Fig. 5 on the other hand shows our migrated CANstar based implementation of the case study. For this migrated setup, the Master and Slave nodes are kept the same with exactly the same task-sets. Only difference here is that the Master is now responsible for running the TTC-SC5 protocol on the migrated CANstar topology.

The block diagram of our migrated CANstar based setup shown in Fig. 5 is depicted in Fig. 6. The block diagram portrays that the Master and Slave nodes are now using dedicated CAN interfaces due to the CANstar topology for deployment of TTC-SC5 protocol [15]. So with this described setup we were able to achieve the mentioned topology shift and were able to perform a comparative case-study while keeping identical task-sets on TPR (Slave-1) and BPR (Slave-2) for both implementations. The following section presents results acquired from the source codes of both topologies. Source codes included protocol software on Master as well as Slaves in both topologies.

3. COMPARATIVE OBSERVATIONS

After running the case-study-rig on both topologies we were able to obtain an identical system behavior and response [15]. It is important to note that our main concern here was to keep an eye on the number of linearly-independent-paths (LIPs) through each code that are going to be followed by the system during run-time. More LIPs mean more complexity and consequently more task jitter [20], [21], and [22]. The definitions of code metrics examined in such an architectural code comparison are described in Table 1. The code comparison metrics for both topologies were obtained through

Table 1. Code Metrics and their descriptions

Metric	Description
"AvgCyclomatic"	"Average-cyclomatic-complexity-for-all-nested-functions-or-methods"
"MaxCyclomatic"	"Maximum-cyclomatic-complexity-of-all-nested-functions-or-methods"
"MaxNesting"	"Maximum-nesting-level-of-control-constructs (if, while, for, switch, etc.) in-the-function"
"CountPath"	"Number-of-unique-paths-though-a-body-of-code, not-counting-abnormal-exits-or-gotos"
"SumCyclomatic"	"Sum-of-cyclomatic-complexity-of-all-nested-functions-or-methods"
"SumEssential"	"Sum-of-essential-complexity-of-all-nested-functions-or-methods"
"CountLineCodeDecl"	"Number-of-lines-containing-declarative-source-code. Note-that-a-line-can-be-declarative-and-executable (e.g. int i = 0)"
"CountLineCodeExe"	"Number-of-lines-containing-executable-source-code"
"CountDeclFileCode"	"Number-of-code-files"
"CountDeclFileHeader"	"Number-of-header-files"

the visualization software Understand™ 2.0 [23] freely available from scitools ™ [24]. They work with companies/organizations like BOEING, Adobe, Apple, IBM, NASA, SIEMENS, BMW, GENERAL DYNAMICS & TOYOTA.

3.1 Case-Study Results

The comparison graphs and their corresponding software metrics values for CANbus & CANstar implementations on the PROCON apparatus are presented in this sub-section as following.

The code volume comparison of both topologies is given in Fig. 7 while the corresponding metrics values are given in Table 2. From these two depictions it is evident that in terms of declarative and executable lines of code (LoC), on File-System's level, the original CANbus implementation requires fewer LoCs when compared with the CANstar implementation. The reason for this is obvious as CANstar implementation provides flexibility and fault-management at a very higher level [16]. Even if one looks at it the difference is not that substantial. Fig. 8 and Table 3 present the file volume comparison between both implementations. From Fig. 8 it is clear that on File-System's level the CANstar's implementations volume on embedded level is marginally better than its counterpart. What the above means is that during implementation the CANstar code for Master and Slave nodes will require less memory on the microcontrollers than for CANbus implementation.

Moving along, comparison of the first three metrics relating to cyclomatic complexity [20] of both implementations is presented in Fig. 9 along with their values given in Table 4. From Fig. 9 and Table 4, it can be seen that the first three metrics represented by Cyclomatic-Complexity-1 here have values almost identical to one another for both implementations. Such identical values suggest that code for the migrated topology setup is as simple as the original setup. It is pivotal to note here that

Table 2. Code volume of CANbus & CANstar implementations (Metrics-values)

	CountLineCode Decl (CANbus)	CountLineCode Decl (CANstar)	CountLineCodeExe (CANbus)	CountLineCodeExe (CANstar)
File-System	538	544	1563	1637
Flow-Rig-Code	157	155	433	434
Master-Code	224	234	702	774
Temp-Rig-Code	157	155	428	429

Table 3. File volume of CANbus & CANstar implementations (Metrics-values)

	CountLineCode Decl (CANbus)	CountLineCode Decl (CANstar)	CountLineCodeExe (CANbus)	CountLineCodeExe (CANstar)
File-System	29	27	29	27
Flow-Rig-Code	10	9	10	9
Master-Code	9	9	9	9
Temp-Rig-Code	10	9	10	9

Table 4. Cyclomatic-Complexity-1 (Metrics-values) both implementations

	Avg Cyclomatic (CANbus)	Avg Cyclomatic (CANstar)	Max Cyclomatic (CANbus)	Max Cyclomatic (CANstar)	MaxNesting (CANbus)	MaxNesting (CANstar)
File-System	3.34	3.35	22	22	5	5
Flow-Rig-Code	2.97	3.03	9	9	5	5
Master-Code	3.74	3.67	22	22	5	5
Temp-Rig-Code	2.95	3	8	8	5	5

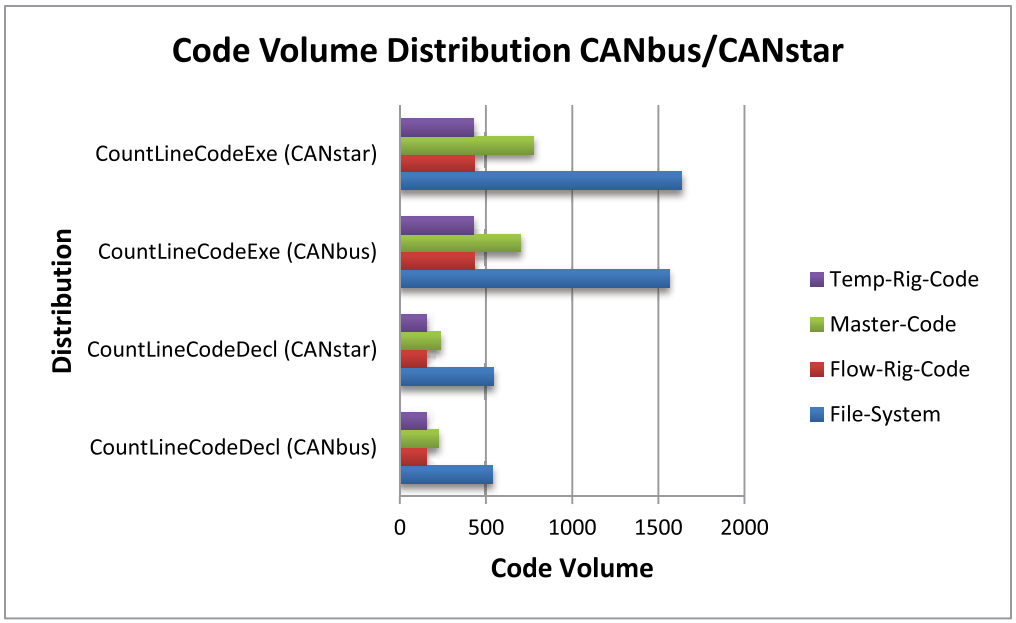

Fig. 7. Code volume comparison of CANbus & CANstar implementations

the original CANbus based setup is revered for its simplicity thus making it more reliable and predictable [8], [14], [15] and [16]. The last three metrics for both implementations represented by Cyclomatic-Complexity-2 are compared side by side in Fig. 10 and their values are given in Table 5. A visible difference in this comparison of both implementations emerges when one looks at the Path Counts (first two columns of Table 5). It is clear from it that the proposed migrated topology has fewer linearly-independent-paths (LIPs) through its body of code making it less complex, more reliable and more predictable than the original CANbus topology code [20]. Rest of the columns in Table 5 does not show a major difference in complexity between the two implementations.

Based on the above observations, Section 4 that follows presents a discussion relating to the achievement of such a commercial off-the-shelf

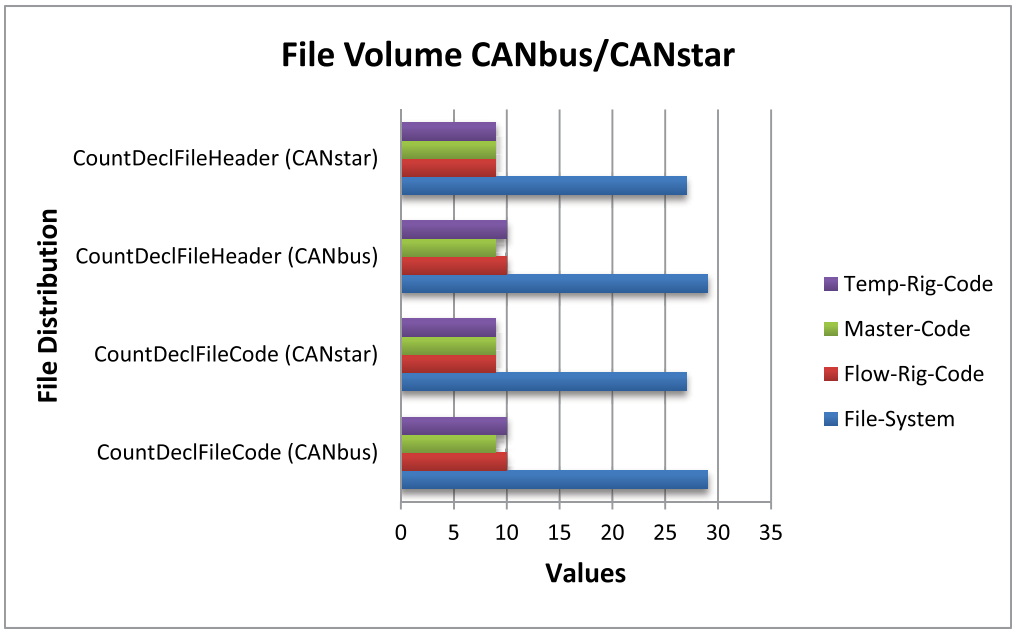

Fig. 8. File volume comparison of CANbus & CANstar implementations

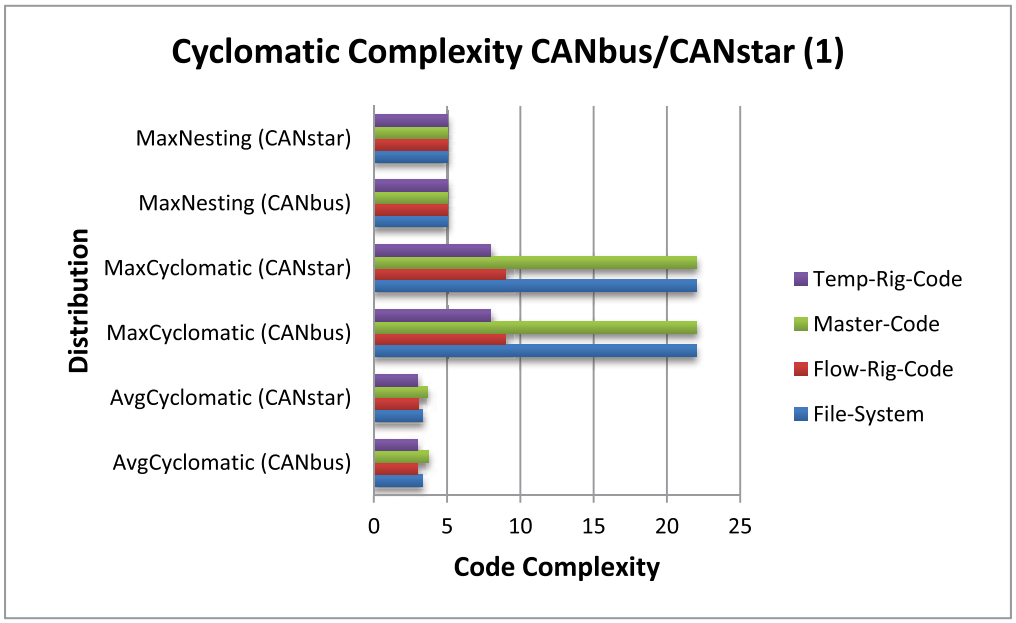

Fig. 9. CANbus & CANstar side by side comparison (Cyclomatic-Complexity-1)

Table 5. Cyclomatic-Complexity-2 (Metrics-values) both implementations

	Avg Cyclomatic (CANbus)	Avg Cyclomatic (CANstar)	Max Cyclomatic (CANbus)	Max Cyclomatic (CANstar)	MaxNesting (CANbus)	MaxNesting (CANstar)
File-System	878	679	395	393	169	174
Flow-Rig-Code	170	169	110	109	52	51
Master-Code	551	354	176	176	65	72
Temp-Rig-Code	169	168	109	108	52	51

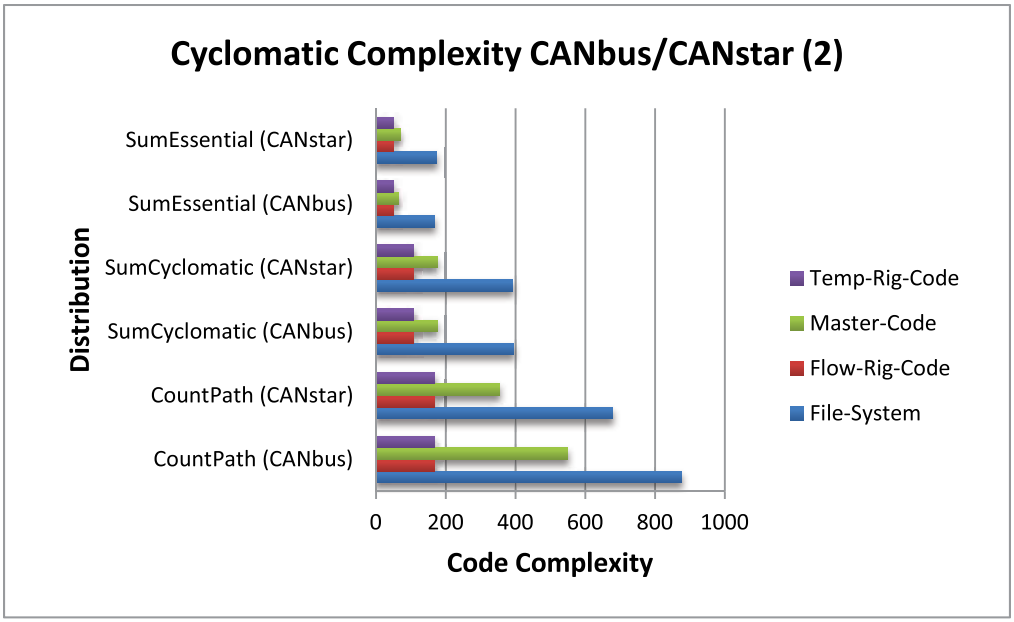

Fig. 10. CANbus & CANstar side by side comparison (Cyclomatic-Complexity-2)

migration without drastically increasing code-volume and the consequent jitter due to scheduler overheads on the Master and Slave nodes.

4. DISCUSSION

As we know, viscosity of code complexity is a measure of linearly-independent-paths (LIPs) throughout the structure of the code [20]. Meaning, high number of LIPs lead towards an increase in complexity which consequently causes the system to waste time on verifying corresponding conditions for a single input variable. Such wastage of time leads toward a delay (i.e. jitter) in the control action of any safety-critical system.

By looking at the comparative analysis given in Section 3, in Table 2, it is evident that population of executable-lines-of-code (eLOC) in CANstar-based implementation is 74 lines more than the CANbus-based implementation. These eLOC are essential for supporting the topology migration and are executed at the initialization stage of the system. Moreover, they do not represent any LIPs in the topology code.

On the other hand, Table 5 (relating to number of LIPs), one can see that the entire file system of CANbus-based implementation has 199 LIPs more when compared with the CANstar-based architecture. This greater number of LIPs in CANbus is due to scheduling code overhead on the Master node that constitutes a bus-based shared-clock environment. This scheduling overhead causes eventually causes task jitter in turn making CANbus based implementations unsuitable for safety-critical applications. The above comparison exhibits simplicity in the migrated topology code projecting it as reliable & predictable for safety-critical applications.

5. CONCLUSION

The comparative observations discussed in the above section exhibit that a migration of CAN protocol from bus to star topology is tremendously easy, bears lesser complexity and is more fruitful. By fruitfulness here, we mean that, the migrated topology setup is more flexible and fault-manageable as shown through our previous research referenced herein as [16].

6. REFRENCES

1. Bosch, R. Controller Area Network Specifications 2.0. Postfach, Stuttgart, Germany, (1991).
2. Farsi, M. & M. Barbosa. CANopen Implementation: Application to Industrial Networks. UK Research Studies Press, Ltd. (2000).
3. Fredriksson, L. B. Controller Area Networks and

the protocol CAN for machine control systems. *Mechatronics.* 4(2): 59-192 (1994).
4. Etschberger, K. Controller Area Network: Basic Protocols, Chips and Applications. IXXAT Automation GmbH, (2001).
5. Pazul, K. Controller Area Network (CAN) Basics. Microchip Technology Inc, Preliminary DS00713A, Page-1 AN713, (1999).
6. Kelkar, S. & R. Kamal. Adaptive Fault Diagnosis Algorithm for Controller Area Network. *IEEE Transactions on Industrial Electronics*. 61(10): 5527-5537 (2014).
7. Mary, G. I., A. C. Zachariah, & J. Lawrence. Reliability Analysis of Controller Area Network Based Systems – A Review. *International Journal of Communications, Networks and System Sciences*. 6 (4): 155-166 (2013).
8. Ayavoo, D., M. J. Pont, M. J. Short, & S. Parker, Two novel shared-clock scheduling algorithms for use with 'Controller Area Network' and related protocols. *Journal of Microprocessors and Microsystems.* 31 (5): 326-334 (2007).
9. Giuseppe, B., P. Juan, & Z. Alberto. Overcoming babbling-idiot failures in CAN networks: a simple and effective Bus Guardian solution for the FlexCAN architecture. *IEEE Transactions on Industrial Informatics.* 3 (3): 225-233 (2007).
10. Short, M. J. & M. J. Pont. Fault-tolerant time-triggered communication using CAN. *IEEE Transactions on Industrial Informatics*. 3(2): 131-142 (2007).
11. Manuel, B. P. Julian, N. Guillermo, & A. Luis. An active star topology for improving fault confinement in CAN networks. *IEEE Transactions on Industrial Informatics*. 2 (2): (2006).
12. TTA-Group. Time-Triggered Protocol TTP/C High-Level Specification Document. Protocol Version. 1.1, 1.4.3 ed. Vienna, Austria, TTTECH. (2003).
13. FlexRay. FlexRay Communication System Protocol Specification Version 2.0. FlexRay Consortium. (2004).
14. Amir, M. Ayavoo, D. & Pont, M. J. A novel shared-clock scheduling protocol for fault-confinement in CAN-based distributed systems. Proceedings of the 5th IEEE International Conference on System of Systems, University of Loughborough, UK, pp. 1-6, 22nd-24th June, (2010).
15. Amir, M. & M. J. Pont. A time-triggered communication protocol for CAN-based networks with a fault-tolerant star topology. International Symposium on Advanced Topics on Embedded Systems and Applications (ESA2010) in conjunction with the 7th IEEE International Conference on Embedded Software & Systems, University of Bradford, UK, 29th June-July 1st, 2010.
16. Amir, M. & M. J. Pont. Improving flexibility and fault-management in CAN-based "Shared-Clock" architectures. *Journal of Microprocessors and Microsystems*. 37: 9-23 (2013).
17. http://www.feedback-instruments.com/products/education/process_control website accessed: 09/04/2019.
18. Philips. LPC2119/2129/2194/2292/2294 Microcontroller User Manual. Philips Semiconductor, 2004.
19. https://www.olimex.com/Products/ website accessed: 09/04/2019.
20. McCabe, T. J. A complexity measure. IEEE Transactions on Software Engineering. 2 (4): (1976).
21. Elaine, J. W. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9): (1988).
22. Geoffrey, K. G. & F. K. Chris. Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*. 17(12): (1991).
23. https://scitools.com/feature-category/metrics-reports/ website accessed: 09/04/2019.
24. https://scitools.com/ website accessed: 09/04/2019.