Study, Design and Simulation of Time Triggered Protocol for Distributed Sensor Network

By

Sampada Bhujbal

Enrolment No. ENGG01201801008

Bhabha Atomic Research Centre, Mumbai

A thesis submitted to the

Board of Studies in Engineering Sciences

In partial fulfilment of requirements

for the Degree of

MASTER OF TECHNOLOGY

of

HOMI BHABHA NATIONAL INSTITUTE

January 2021

Homi Bhabha National Institute

Recommendations of the Thesis Examining Committee

As members of the Thesis Examining Committee, we recommend that the thesis prepared by **Sampada Bhujbal** entitled "**Study, Design and Simulation of Time Triggered Protocol for Distributed Sensor Network**" be accepted as fulfilling the thesis requirement for the Degree of Master of Technology.

	Name	**Signature**
Member-1	Dr. (Smt.) Gopika Vinod	
Member-2	Dr. Manoj Kumar	
Member-3	Shri S. K. Bahuguna	
Member-4	Dr. Ajith K. J.	
Technical Advisor	Shri Abhishek Borana	
Examiner		
Guide and Convenor	Shri D. A. Roy	
Chairman	Dr. S. Kar	

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to HBNI.

I hereby certify that I have read this thesis prepared under my direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: 16-2-2021

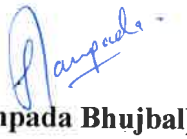
Place: Mumbai

Name of the Guide: D. A. Roy

Signature of the Guide

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institution/University.



(Sampada Bhujbal)

Homi Bhabha National Institute

Recommendations of the Thesis Examining Committee

As members of the Thesis examining Committee, we recommend that the thesis prepared by **Gagandeep Singh** entitled "**Fragility Assessment of RC Structure using Incremental Dynamic Analysis Considering Variability of Ground Motion, Damping and Soil Parameters**" be accepted as fulfilling the thesis requirement for the Degree of Master of Technology.

	Name	Signature
Member – 1	Dr. R.R. Rakesh	RRRakesh
Member – 2	Dr. Praveen Kumar	Praveen K
Examiner	-	
Guide/Convenor	Dr. Smt. Y.M. Parulekar	Y M Parulekar 22.7.2021
Technical Advisor	Shri. Sumit T Bhamare	STBhamare
Chairman	Dr. Kapilesh Bharghava	KB 22/07/2021

Final approval and acceptance of the thesis is contingent upon the candidate's submission of the final copies of the thesis to HBNI.

I hereby certify that I have read this thesis prepared under my direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: 20.7.2021

Place: RSD, BARC

Y M Parulekar

Signature of Guide

Name of Guide: Dr. Smt. Y.M.Parulekar

ACKNOWLEDGEMENTS

Immeasurable appreciation and deepest gratitude for the help and support are extended to the following people who in one way or another have contributed in making this project possible.

First and foremost, I am extremely grateful to my guide, Shri D. A. Roy, OS and Head-RRPSS, RCnD, BARC for his guidance throughout the life cycle of project. His experience, expertise in subject and support in course of project execution were of great help.

I take this opportunity to thank my Technical Advisor, Shri Abhishek Borana, RRPSS, RCnD, BARC for his constant supervision, support and advice without which it would have been difficult to complete this work.

I owe my special thanks to Shri S. T. Sonnis, SO/G and Head-Software Reliability section, RCnD, BARC for his invaluable suggestions and able guidance that helped in shaping the project.

I would also like to express my appreciation towards seniors from my division, Shri K. K. Singh, Head-INRPC, NRB and Shri L. M. Antony, INRPC for their valuable support and co-operation towards completion of the project.

I would also like express my sincere gratitude towards Shri U. W. Vaidya Head-RCnD and Shri P. P. Marathe, AD, E&I, BARC for allowing me to do the project and providing with the necessary setup required in the project execution.

I am also thankful to all the people working in Reactor Control Division and Integrated Nuclear Recycle Plant of Nuclear Recycle Board for their direct and indirect support.

Sampada Bhujbal

Contents

List of figures

List of tables

Abstract

In modern C&I applications, reliable and deterministic data communication is an essential part of distributed control systems to improve the performance & for higher availability. In time triggered communication, a time-driven static scheduling provides fixed latency and improves determinism. It can be used to provide real-time communication with determinism and fault tolerance. It is a master-less protocol with broadcast transmission and conflict free TDMA bus access as per a static schedule-table. All nodes in the network are synchronized in temporal domain with distributed clock synchronization algorithm. A network of distributed intelligent sensor nodes with time-triggered synchronized communication provides overall sensor data in real-time to control and monitoring systems. It will provide robust and fault tolerant communication for distributed sensors in a nuclear plant environment and also result in substantial savings in cabling and penetrations.

As a part of the project, study and comparison of architecture and protocols used in existing sensor level networks has been carried out. A custom deterministic and fault tolerant network architecture and protocol suitable for a sensor level network in nuclear plants and facilities has been evolved. Formulation and simulation of the two core algorithms namely clock synchronization and group membership algorithms required for protocol operation has been carried out. The main objective of the project is to design a robust sensor level network protocol based on study and simulation to verify performance and quality parameters.

CHAPTER 1: Introduction

As computing and communications become irreplaceable tools, reliability, availability, fault tolerance and fail silence are important aspects. In all safety critical systems including flight control system, railway signaling, automotive system, emergency response systems; airline flight controls; health care; and most of all, nuclear power plant safety systems, requirement of reliability and fault tolerance has increased with high-performance computing and communications. The concept of fault tolerance was formulated in 1967: "A system is fault-tolerant if its programs can be properly executed despite the occurrence of logic faults".

1.1 Motivation

In typical real-time (RT) applications, the computer system has to perform a multitude of different functions in parallel, e.g. the processing of inputs, the monitoring of system state to detect alarm conditions, display to operator etc. In distributed computer systems, these different functions are normally allocated to different nodes. In real-time systems using communication data, bounded data latency in presence of disturbing factors such as overload or faults is a fundamental requirement. A real-time system has to meet the strict timing deadlines dictated by its environment. If a real-time system misses a deadline, it has failed. But for safety critical systems, single fault tolerance is an important criterion. Fault tolerance guarantees that the systems will perform as per design specifications and also ensure trust of users. This was the motivation to work on aforementioned topic of "Study, Design and Simulation of Time Triggered Protocol for Distributed Sensor Network".

1.2 Objectives

In order to reduce cabling costs, the nodes of distributed control system are connected to each other via a network. Network systems offer the opportunity of modularity and scalability which allow a flexible design of variants. Further, (intelligent) sensors and actuators are used in a multiple way to realize high level functions and services.

There is a huge number of possible network topologies, like for instance, the currently frequently used bus communication, topologies with gateways, active stars or cascaded star structures. If the application requires fault-tolerance, the classic redundant network or a mixed redundant network which efficiently maps the system safety structure into a system architecture is utilized. Bus topology has been discussed further. Unfortunately, communication networks introduce delays. Furthermore, these delays may be varying in a random fashion, making the control system time varying.

If one is concerned to formulate the requirements of distributed control systems two different situations should be distinguished:

1. Occurrence of a critical situation, for instance, any parameter value getting beyond process range
2. The regular operation of the control system.

The former situation requires as fast as possible reaction to the asynchronous event in order to start emergency mechanisms. For that purpose, event and time-triggered communication are compared later.

During the regular (periodical) operation of distributed control systems, time-triggered

architectures seem to be ideally suited. They often lead to a higher latency compared with event-triggered architectures, but there is no jitter if all participating nodes at the network are synchronized to a global time.

As a part of the project, study and comparison of architecture and protocols used in existing sensor level networks has been carried out. In which, time-triggered architectures offer very interesting properties from the control design point of view. The main objective of the project is to design a deterministic sensor level network protocol based on study and simulation.

1.3 Scope

The work consists of design of custom time triggered protocol, formulation and simulation of algorithms for use in network of smart sensors. The scope of work is as under:

1) Literature survey and comparative study of sensor network architecture and communication protocols existing in real-time for distributed sensor networks.
2) Formulation of clock synchronization algorithm for custom time triggered protocol.
3) Formulation of membership algorithm for custom time triggered protocol.
4) Modelling and Simulation of synchronization algorithm and membership algorithm using simulation tool.
5) Validation on suitable hardware and conceptual design of sensor node**.**

1.4 Organization of report

Chapter 2 describes the important findings from the literature survey. In distributed control systems (DCS) two conceptually distinct models have been identified, named event-triggered (ET) and time-triggered (TT) which are compared in the beginning of chapter 2. Later is the

list of various communication fieldbuses present in the market. Among those, the most prevalent in the industry are described in detail with comparison of their significant features.

How TTP is a candidate protocol for our requirements of Fault-tolerant Deterministic network is described in detail in **Chapter 3**.

One of the core aspects TTP, which is clock synchronization algorithm, has been studied during literature survey. **Chapter 4** describes the two core algorithms of the protocol i.e. clock synchronization and group membership protocol. The former part of the chapter mentions the necessity of clock synchronization algorithm and its implementation details. Simulation results and conclusions are depicted in the same. The other important core aspect of TTP is Group membership. This has been described in detail along with illustrations of its simulation at various different fault scenarios in the latter part of the chapter.

Chapter 5 consists of details regarding the hardware set-up used for validation of the algorithms. The observations, after the execution of the algorithms over the test set-up, are also illustrated in this chapter.

Chapter 6 comprises of the conceptual design of the sensor node along with a performance specifications of designed system. The details of custom designed Time triggered protocol are also described in this chapter.

Chapter 7 concludes the thesis and explains the future scope of this activity.

CHAPTER 2: Literature survey

In distributed control systems (DCS) two conceptually distinct communication models have been identified, named event-triggered (ET) and time-triggered (TT). In event-triggered communication, message exchanges are initiated when a significant event occurs. They include arbitrary state changes in the computer system or the environment (e.g., interrupt from a sensor). In time-triggered systems, message exchange is derived from a clock tick according to a static schedule table. In distributed control systems (DCS) two conceptually distinct models have been identified [1], named event-triggered (ET) and time-triggered (TT) and compared below.

2.1 Event-triggered (ET) communication

In an event-triggered system all actions are triggered due to the occurrence of a significant event or state change. With respect to communication, only transmitter nodes have knowledge about the exact instant a message will be sent, that is, each node has only its own view of the physical time.

- The major advantage of an event-triggered bus is the better responsiveness to asynchronous external events (e.g. an alarm condition) at low communication load. System reaction is fast since the transmission times of messages are not under the control of the platform architecture, that is, a node is able to try for media access whenever it needs. The problem is that, depending on the bus arbitration mechanism of the protocol, collisions will have direct impact on response times of communication services.
- In addition, event-triggered distributed systems are more flexible. The total bandwidth of the channel is shared among the nodes which means that it can automatically readapt when

nodes are removed or included even if there is ongoing communication (hot-plug)[2].

- Nonetheless, in event-triggered systems the response times of data transmission deteriorates as communication load increases.
- Even at low loads the communication behavior will experiment quite large latency variations, which has negative impact on the control performance.

2.2 Time-Triggered (TT) communication

At the other opposite, a time-triggered distributed system is characterized by its regular, deterministic time behavior. All transmission times of messages are defined in a global schedule at design time, prior to deployment and the communication controller is responsible for every communication activity (transmission and reception of messages) and operates independently from the host computer.

- An important advantage is that time-triggered based protocols offer cyclical data communication services with fixed latencies.
- The bandwidth is partitioned into dedicated time slots inside which only one node can transmit at a time hereby collisions never occur under normal operation.
- If there are no collisions, the transmission of messages defined in the global schedule is never delayed. This is the Time-Division Multiple Access (TDMA) arbitration mechanism.
- Moreover, time-triggered systems have a very attractive property from the point of view of embedded system design which is named composability, that is, the possibility to develop two subsystems independently with prior guarantees that the integration will not jeopardize timing and other functional characteristics that were individually defined. Time-triggered buses are composable because communication media access is multiplexed, that is, each node has its own portion of bandwidth that will never be snatched by another node under normal conditions.

- Error detection is performed inside the controllers by verifying if every message was received within its expected time window. This facility providing intrinsic means to verify a node crash.
- A shortcoming of the time-triggered paradigm is its lack of flexibility. For every node inclusion that was not previously predicted, the whole system must be reprogrammed.

When comparing design complexity, the implementation of time-triggered systems is indeed more complicated. Within each node it must be ensured that the application layer follows the communication scheme correctly. This means that the worst-case execution time (WCET) of all tasks in a node must be carefully estimated in order to avoid inconsistencies in the values that are used by actuators to interact with the system under control. During system design, not only the communication schedule but also the schedule of all control tasks must be defined [3].

A small change in one subsystem may in general imply an entire new system design. The system design itself is very complicated and there is still a lack of adequate tools for the design process. It depends on the actual application whether a time-triggered or an event-triggered behavior is more suitable. Very safety critical systems require fault-tolerance and redundancy. The implementation of such systems probably will fail without the framework of time-triggered architectures. For large-scale problems time-triggered architectures are a good choice and offers some interesting opportunities.

Today, there are two emerging time-triggered solutions claiming being able to fulfil the rigorous demands of predictability, performance and reliability. The first one is the TTP/C protocol and its Time-Triggered Architecture [4]. The other one is the FlexRay

Communications System developed by the FlexRay Consortium formed by a group of strong companies in the automotive area [5].

It is noteworthy that the two paradigms (ET and TT) are not mutually exclusive in practice but one of them always prevails against the other. Both FlexRay and TTP/C architectures provide event-triggered services in addition to their strictly regular time-triggered communication. For instance, the communication cycle in FlexRay is divided into four different time windows: a static segment, an optional dynamic segment, a symbol window and the network idle time (NIT) [6]. All communication services are handled within the static and dynamic segments. Therefore, this protocol prioritizes time-triggered traffic as the static segment is mandatory and the dynamic is optional. The former segment is used for time-triggered traffic while the other is shared among the nodes for asynchronous transmissions. Reaction to external events can be quite slow depending on the length of the static segment. Consider for example that an asynchronous transmission request arrives at the end of the idle time. In this case, the transmission will be delayed at least by the entire duration of the static segment. In TTP/C, however, there is no time window reserved to handle asynchronous requests, event-triggered services are implemented over the time-triggered scheduled using empty slots (if any). Having all that in mind, it is clear that in an inherent time-triggered system, the cyclical traffic is always prioritized.

Event-triggered and time-triggered concepts do not have a direct relationship with synchronous and asynchronous communication model present in traditional distributed system theory. An asynchronous model refers to the application processing in a distributed environment which has no notion of time at all whereas the event-triggered node in the DCS has its own notion of time. Research on embedded DCS has recognized the time-triggered communication model as

the most adequate design paradigm for safety-critical applications [7]. Bus architectures derived from the time-triggered model have some desirable properties like low communication jitter, predictable transmission delays, composability and efficient support for fault-tolerant techniques.

2.3 Communication protocols in distributed systems

An industrial distributed control system (such as manufacturing assembly line) consists of an organized hierarchy of controllers. In this hierarchy, there is usually a Human Machine Interface (HMI) at the top, where an operator can monitor or operate the system. This is typically linked to a middle layer of programmable logic controllers (PLC) via a non-time-critical communications system (e.g. Ethernet). At the bottom of the control chain is the fieldbus. It links the PLCs to the components that actually do the work, such as sensors, actuators, electric motors, console lights, switches, valves and contactors. Fieldbus is the name of a family of industrial computer network protocols used for real-time distributed control of field devices, standardized as IEC 61158.

2.3.1 Challenges in Fieldbus

Users see the following as some of the challenges

- Selecting the best combination of bus technologies for the application
- Integration options and solutions
- Technology training & development at all levels of the organization
- Testing and interoperability issues

2.4 Prominent communication protocols for fieldbus

The prominently used protocols (in an industrial environment) are described further briefly:

2.4.1 TTP/C

The Time-Triggered Protocol (TTP) is a communication protocol for distributed fault-tolerant real-time systems [8]. It is designed for applications with stringent requirements concerning safety and availability, such as avionic, automotive, industrial control and railway systems. TTP was initially named TTP/C and later renamed TTP. The initial name of the communication protocol originated from the classification of communication protocols of the Society of Automotive Engineers (SAE), which distinguishes four classes of in-vehicle (refer Table 2.1) networks based on the performance TTP/C satisfies the highest performance requirements in this classification of in-vehicle networks and is suitable for network classes C and above.

TTP provides a consistent distributed computing base in order to ease the construction of reliable distributed applications. Given the assumptions of the fault hypothesis, TTP guarantees that all correct nodes perceive messages consistently in the value and time domains. In addition, TTP provides consistent information about the operational state of all nodes in the cluster.

Table 2.1: SAE Network classes

Network Class	Examples of Protocols	Bandwidth	Typical Latencies	Automotive Applications
Class A	Local Interconnect Network (LIN)	< 10 kbps	10-100ms	sensor/actuator access
Class B	Controller Area Network (CAN)	10kbps-125kbps	10-100ms	comfort domain
Class C	Controller Area Network (CAN)	125kbps-1Mbps	5ms	powertrain domain
Class D	Time-Triggered Protocol (TTP), FlexRay	> 1 Mbps	5ms	multimedia, X-by-wire

2.4.2 TTP/A

The TTP/A protocol is the low-cost field-bus protocol that is harmonized with the fault-tolerant system bus TTP/C of the time-triggered architecture (TTA). It is intended for the connection of smart sensors and actuators in embedded real-time systems in different application domains, e.g., industrial, automotive, etc. It is the objective of TTP/A to provide all services needed by a smart sensor, including timely communication, remote online diagnostics, and plug-and-play capability.

TTP/A implements the Object Management Group (OMG) Smart Transducer Interface (STI) standard. The STI standard defines a smart transducer system as a system comprising several clusters with transducer nodes connected to a bus. Via a master node, each cluster is connected

to a gateway. It is possible to monitor the smart transducer system via the interface without disturbing the real-time traffic.

In TTP/A, the master nodes of each cluster share a synchronized time that supports coordinated actions (e. g., synchronized measurements) over transducer nodes in several clusters. Each cluster can address up to 250 smart transducers that communicate via a cluster-wide broadcast communication channel. There may be redundant shadow masters to support fault tolerance. One active master controls the communication within a cluster.

2.4.3 FlexRay

It is fault tolerant and deterministic, and is aimed at advanced applications such as x-by-wire. Data is transmitted on the FlexRay bus in both timed and event driven manner. Each message is divided into static segment and the dynamic segment. The static segment is defined during the configuration of the application and transmits the data on a TDMA basis. The dynamic segment of the message handles data on an event triggered basis. The protocol defines parts of the physical layer, data link layer, presentation layer and application layer of the OSI model in the context of a FlexRay communication controller [6].

FlexRay have different time slots for messages i.e., scheduled by static cyclic and fixed priority. Static cyclic scheduling got the advantages that it is easy to calculate response times and is very predictable. However, since it needs to be scheduled before runtime it got some limitations, like how to schedule very important messages that rarely needs to be sent. In worst case that kind of message may have to wait for a whole cycle if its slot has passed, or multiple slots have to be assigned to that message but then these slots will pass empty most of the time. Making a static schedule may not be that easy for complex systems. Using one static section and one

dynamic section like FlexRay may be an advantage since frequent real-time messages can be sent in the static segment, while uncommon high priority messages and lower prioritized messages can use the dynamic segment. Data rate is up to 10Mbps over dual redundant channel.

2.4.4 Controller Area Network

The CAN protocol is internationally standardized in ISO 11898-1 and comprises the data link layer and components of the physical layer of the 7-layer ISO-OSI reference model [9]. The main principles of CAN are:

- Multi-master: Any node may send if the bus is idle.
- Guarantee of latency times: It is possible to calculate the worst case time for a message to be sent and reached at the destination node.
- Configuration flexibility: Nodes can be added to the network without change in hardware or software.
- Prioritization of messages: Conflicts are avoided by prioritizing the messages.
- Broadcast communication: Every transmitted message is received in all nodes

CAN use Fixed Priority scheduling, which means that each message got a set priority, and higher prioritized messages will get access to the bus before lower prioritized messages. The advantage with this system is that high prioritized messages will have a quick response time, however low prioritized messages may have to wait a long time before they get bus access. Another disadvantage with fixed priority scheduling is that it is not very predictable. However, it is possible to calculate worst case response times for every message. Maximum baud rate is 1 Mbps and overheads are more than 50 percent.

2.4.5 TTCAN

Time-Triggered CAN (TTCAN) [10] uses the concept of cyclic communication, divided into slots to implement time-triggered behavior. The standard requires that all activity assigned to one slot (including interrupt handling) is finished until the next slot starts, whereas one message can be assigned to several slots.

It is important to note that TTCAN adds an additional layer before the existing standard CAN layers. The physical layer and data link layer of CAN are kept unchanged. Mapped to the ISO-OSI model, TTCAN resides on layer 5, the session layer. Within this layer, TTCAN is divided into two different modes of operation. While level 1 and level 2 both support time-triggered behavior, the capabilities of these levels differ in key properties.
The most outstanding difference is the notion of a global time, that only exists in level 2, and allows a much more fine grained synchronization of the nodes in a TTCAN network.

2.4.6 TTEthernet

Time-Triggered Ethernet (TTEthernet) [11] expands classical Ethernet use with powerful services (SAE AS6802) to meet the new requirements of reliable, real-time data delivery in advanced integrated systems. With TTEthernet, critical control systems, audio/video and standard LAN applications can share one network. TTEthernet facilitates design of mixed criticality systems and system-of systems integration.

Time-Triggered Ethernet network devices are Ethernet devices which at least implement:
SAE AS6802 synchronization services for advanced integrated architectures, fail-operational and safety-critical systems time-triggered traffic flow control with traffic scheduling per-flow policing of packet timing for time-triggered traffic robust internal architecture with traffic

partitioning TTEthernet (or TTE) network devices from TTTech are standard Ethernet devices with additional capability to configure and establish robust synchronization, synchronous packet switching, traffic scheduling and bandwidth partitioning, as described in SAE AS6802. If no time-triggered traffic capability is configured or used, rugged TTEthernet devices operate as full duplex switched Ethernet devices compliant with IEEE802.3 and IEEE802.1 standard.

2.4.7 HART

The HART Communication Protocol (Highway Addressable Remote Transducer) is a hybrid (analog+digital) communication standard. The purpose of the HART standard was to create a way for instruments to digitally communicate with one another over the same two wires used to convey a 4-20 mA analog instrument signal. So, HART is a hybrid communication standard, with one variable (channel) of information communicated by the analog value of a 4-20 mA DC signal, and another channel for digital communication whereby many other variables could be communicated using pulses of current to represent binary bit values of 0 and 1. Those digital current pulses are superimposed upon the analog DC current signal, such that the same two wires carry both analog and digital data simultaneously.

HART communication over 4-20 mA signal wires is a legacy technology. HART protocol is the most popular form of wired digital field instrument communication in industrial use. In applications where speed is not a concern, HART communication is a very practical solution for acquiring multiple channels of data from one instrument over a single pair of wires. However, more modern digital standards such as Profibus and FOUNDATION Fieldbus deliver all the benefits of HART technology and more. It seems that wired-HART will remain in wide industrial use for many years to come, but it is really just the beginning of digital field instrument technology and does not represent the state of the art.

2.4.8 MODBUS

Modbus is a protocol designed specifically for exchanging process data between industrial control devices. The Modbus standard does not specify any details of physical networking, and thus may be deployed on many different types of physical networks. In other words, Modbus primarily falls within layer 7 of the OSI Reference Model (the so-called "Application Layer") and therefore is compatible with any lower-level communication protocols including EIA/TIA-232, EIA/TIA-485, Ethernet (the latter via TCP/IP), and a special token-passing network also developed by Modicon called Modbus Plus.

The Modbus standard primarily defines the meaning of various Modbus commands, the addressing scheme used to place data within devices, and the formatting of the data. Modbus consists of a set of standardized digital codes intended to read data from and write data to industrial devices. A Modbus-compliant industrial device has been programmed to understand these codes and respond to them appropriately when received.

The channel arbitration mechanism is master/slave, where one PLC functions as the master Modbus device and all other devices function as Modbus slaves. Even when Modbus commands are communicated via networks with their own differing arbitration methods, same mechanism is followed. For example, Modbus commands communicated over Ethernet still reference "slave" addresses even though the Ethernet network those messages are sent over uses CSMA/CD arbitration. In other words, there is a hint of OSI layer 2 embedded within Modbus messages that still dictates which Modbus devices may issue commands and which must obey commands.

2.4.9 Foundation Fieldbus (FF)

Foundation Fieldbus is an all-digital, serial, two-way communications system that serves as the base-level network in a plant or factory automation environment. It is an open architecture targeted for applications using basic and advanced regulatory control, and for much of the discrete control associated with those functions. Foundation Fieldbus technology is mostly used in process industries, but has recently been implemented in power plants.

Two related implementations of Foundation Fieldbus have been introduced to meet different needs within the process automation environment. These two implementations use different physical media and communication speeds:

- FF H1 - Operates at 31.25 kbit/s and is generally used to connect to field devices and host systems. It provides communication and power over standard stranded twisted-pair wiring in both conventional and intrinsic safety applications. H1 is currently the most common implementation.

- FF HSE (High-speed Ethernet) - Operates at 100/1000 Mbit/s and generally connects input/output subsystems, host systems, linking devices and gateways.

Fieldbus H1 network communication may be divided into two broad categories: scheduled (cyclic) and unscheduled (acyclic). Scheduled communication events are reserved for exchanging critical control data such as process variable measurements, cascaded setpoints, and valve position commands. These scheduled communications happen on a regular, timed schedule so that loop determinism is guaranteed. Unscheduled communications, by contrast, are the way in which all other data is communicated along an H1 segment. Manual setpoint

changes, configuration updates, alarms, and other data transfers of lesser importance are exchanged between devices in the times between scheduled communication events. The Fieldbus Foundation recommends new H1 segments be configured for no more than 30% scheduled communications during each macrocycle (70% unscheduled time). This should leave plenty of "free time" for all necessary acyclic communications to take place without having to routinely wait multiple macrocycles.

Foundation Fieldbus was originally intended as a replacement for the 4-20 mA standard, and today it coexists alongside other technologies such as Modbus, Profibus, and Industrial Ethernet. Foundation Fieldbus today enjoys a growing installed base in many heavy process applications such as refining, petrochemicals, power generation, and even food and beverage, pharmaceuticals, and nuclear applications.

The International Electrotechnical Commission (IEC) standard on field bus, including Foundation Fieldbus, is IEC 61158. Type 1 is Foundation Fieldbus H1, while Type 5 is Foundation Fieldbus HSE.

2.4.10 Profibus

PROFIBUS (Process Field Bus) is a standard for fieldbus communication in automation technology. PROFIBUS is openly published as part of IEC 61158. Profibus DP (Decentralised Peripherals) is a protocol made for (deterministic) communication between Profibus masters and their remote I/O slaves. There are two variations of PROFIBUS in use today; the most commonly used PROFIBUS DP, and the lesser used, application specific, PROFIBUS PA:

PROFIBUS DP is used to operate sensors and actuators via a centralized controller in

production (factory) automation applications. The many standard diagnostic options, in particular, are focused in PROFIBUS DP.

PROFIBUS PA (Process Automation) is used to monitor measuring equipment via a process control system in process automation applications. This variant is designed for use in explosion/hazardous areas (Ex-zone 0 and 1). The Physical Layer (i.e. the cable) conforms to IEC 61158-2, which allows power to be delivered over the bus to field instruments, while limiting current flows so that explosive conditions are not created, even if a malfunction occurs. The number of devices attached to a PA segment is limited by this feature. PA has a data transmission rate of 31.25 kbit/s. However, PA uses the same protocol as DP, and can be linked to a DP network using a coupler device.

The much faster DP acts as a backbone network for transmitting process signals to the controller. This means that DP and PA can work tightly together, especially in hybrid applications where process and factory automation networks operate side by side. Moreover, the physical layer of Foundation Fieldbus happens to be identical to that of Profibus-PA, further simplifying installation by allowing the use of common network validation tools and connection hardware.

2.5 Comparison among the protocols

The Open Systems Interconnection (OSI) model is a conceptual model that characterizes and standardizes the communication functions of a telecommunication system without regard to its underlying internal structure and technology. The physical layer of the OSI model is responsible for the transmission and reception of data between a device and a physical medium. The physical layer specifications of the protocols are compared in the Table 2.2 given below:

Table 2.2: Physical layer comparison:

Bus Technology	Standards	Comm. Type	Comm. Speed	Max Distance	# devices
FF H1	IEC 61158, ISA SP50	All Digital	31.25 Kbs	1.9km, 9.5 km	32 per seg
Profibus PA	IEC 61158	All Digital	31.25 Kbs	1.9km, 9.5 km	32 per seg
FF HSE	IEC 8802, IEEE 802.3	All Digital	100 Mbs, 1 Gbs	100 m	Unlimited
ProfiNet	IEC 8802, IEEE 802.3	All Digital	100 Mbs, 1 Gbs	100 m	Unlimited
MODBUS	IEEE 1451.2, TIA-485	All Digital	9.6 Kbs – 12 Mbs	1512 m	247 per seg
Profibus DP	IEEE 1451.2, TIA-485	All Digital	9.6 Kbs – 12 Mbs	1512 m	127 per seg
HART	Bell 202, 4-20mA	Digital over analog	1.2 Kps – 9.6 Kps	3.0 km	64 in multidrop
TTP/C	SAE AS6003	All Digital	25 MB/s	It will depend on the physical layer.	64

The data link layer provides node-to-node data transfer—a link between two directly connected nodes. It detects and possibly corrects errors that may occur in the physical layer. It defines the protocol to establish and terminate a connection between two physically connected devices. It also defines the protocol for flow control between them.

The Data link layer specifications of the protocols described before are compared in the Table 2.3 given below.

Table 2.3: Data link layer comparison:

Bus Technology	Standards	Data Link Type	Error Detection	Deterministic	Comm Relationships
FF H1	IEC 61158, ISA SP50	Token Passing	16-bit CRC	Yes	Client/server, pub/sub, sink/source
Profibus PA	IEC 61158	Token Passing	16-bit CRC	Yes	Master/slave
FF HSE	IEC 8802	Token Passing	16-bit CRC	No	Client/server, pub/sub, sink/source
ProfiNet	IEC 8802	Token Passing	16-bit CRC	No	Master/slave
MODBUS	None	master/slave address scheme	1-bit	No	Master/slave
Profibus DP	IEC 61158	master/slave address scheme	1-bit	No	Master/slave, pub/sub
HART	None	Flat addressing	CRC	No	Master/slave
TTP	SAE AS6003	TDMA	24- bit CRC	Yes	Masterless

The application layer is the OSI layer closest to the end user, which means both the OSI application layer and the user interact directly with the software application. Application-layer functions typically include identifying communication partners, determining resource availability, and synchronizing communication.

The Application layer specifications of the protocols described before are compared in the Table 2.4 given below.

Table 2.4: Application layer Comparison:

Bus Technology	Standards	Data Transfer	Supports Control in the Field	Peer to Peer Comm	Alerts and Trends in Devices
FF H1	IEC 61158, ISA SP50, Function block application based on IEC 61804 (Draft)	AI, AO, DI, DO, PID, PD, CS, MIO, many more	Yes	Yes	Yes
Profibus PA	IEC 61158	AI, AO, DI, DO	No	No	Yes
FF HSE	IEC 61158	Same as H1	Yes	Yes	Yes
ProfiNet	IEC 61158	Same as DP	No	No	Yes
MODBUS	IEC 61158	Registers	No	No	No
Profibus DP	IEC 61158	AI, AO, DI, DO	No	No	No
HART	IEC 61158	Commands	Yes	No	No

For a reliable data communication in distributed systems, determinism and fault tolerance are important aspects. For this reason, Time Triggered Architecture (TTA) have been used and proposed to be used in various safety-critical applications. Conflict free time-triggered medium access provides deterministic behavior (constant latency with guaranteed transmission), since all nodes know which node is next to transmit at all times.

Table 2.5 lists the differences between the features of Event-triggered and Time-triggered architectures briefly.

Table 2.5: **Comparison between the ET and TT paradigms**

	Event-triggered (ET)	**Time-triggered (TT)**
Control traffic latencies	Variable	Fixed
Channel bandwidth	Shared	Dedicated
Temporal behavior	-	Predictable
Fault tolerance	May fail under heavy load	Good
Latency	-	Constant
Verification required	Extensive system tests	-
Resource utilization	Better	-
Reaction to external events	Fast	Slow
Flexibility	Average	Low
Design complexity	Low	Average

Time-triggered architectures support features such as 1) Single failure criterion, 2) Fail silence and 3) Determinism to provide a predictable service. Hence TTP has been explored further.

2.6 Clock synchronization algorithm in Time Triggered Architecture

The control signals for the CPU originate from a physical oscillator, a quartz crystal. Because the mechanical dimensions of any two physical quartz crystals are slightly different, no two physical oscillators have the same drift. Since any clock drifts, the clock times of an ensemble of clocks will drift apart if they are not periodically re-synchronized with respect to each other. The clocks of a subsystem are called local clocks. Clock synchronization is concerned with bringing the time of local clocks in close relation with respect to each other.

Approaches to clock synchronization are refined for the employment in large distributed systems [12]. Rushby and von Henke have formally verified clock synchronization algorithms [13]. Schedl simulated several clock synchronization algorithms [14].

Most of the presented clock synchronization algorithms use replication as fault tolerance strategy. Dolev, Welch and Papatriantafilou present approaches that achieve fault tolerance using self-stabilization [15][16][17]. Analyses of self-stabilizing clock synchronization algorithms can be found in [18][19]. Measurement and control applications are increasingly using distributed system technologies such as network communication, local computing and distributed objects. As these measurement and control applications are based on distributed embedded systems, clock synchronization has become an important area for standardization.

In 2001, the IEEE standard organization started a standardization process for clock synchronization in measurement and control applications. In 2002, the standardization committee published a draft standard for clock synchronization, called “Precision Time Protocol” (IEEE P1588). This standard was revised and extended in 2008 and is called “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems” [20]. This standard addresses the needs of measurement and control systems:

microsecond to submicrosecond accuracy; administration free; and most importantly, accessible for both high-end devices and low-cost, low-end devices. This standard decouples the application and communication task by introducing an application and a communication layer and an isolation layer that isolates application activities from communication activities and provides time provision services. The communication layer of the protocol requires all communicating nodes to follow a time-division multiple access (TDMA) strategy for the communication medium in order to achieve low latency jitter.

2.6.1 Convergence Functions

In the literature, clock synchronization approaches often differentiate themselves via the convergence function. The convergence function uses the different remote clock estimates and calculates a single new clock value which the local clock uses to correct itself to. Schneider and Anceaume and Puaut [21][22] list different convergence functions with the following being the most often referenced ones.

A. Convergence Averaging Techniques

In the following, some kind of averaging on clock estimates characterizes the convergence functions. Some functions tolerate potentially faulty clock estimates, while others don't. For the overview, $f(p_i,x_1, ...,x_n)$ identifies a convergence function, where *pi* is the processor (or node) requesting the convergence function and $x_1, ...,x_n$ are the estimated clock values. For the convergence function f_{mm} and f_{im}, *xi* is an interval otherwise an integer value.

a. **Interactive convergence function**. E*gocentric average function* denoted by f_e. $f_e(p_i,x_1, ...,x_n)$ returns the average of the arguments modified in the following: x_j, $1 \leq j \leq n$, stays $x\,j$ if $|x_j - x_i| < \omega^-$ (i.e., if x_j is not further than ω^- away from x_i) otherwise x_j is replaced

with x_i. ω^- has to be chosen appropriately, but be at least $\omega^- > \pi$, π being the precision. For f_e no sorting algorithm needs to be applied to the remote clock estimates, which is an advantage.

b. **Fast convergence function**. f_{fc} returns the average of all arguments x_1 to x_n that are within ω^- of at least $n - f$ other arguments. f_{fc} yields a high-quality precision, but is computationally quite complex.

c. **Fault-tolerant midpoint function**. Denoted f_{ftm} and used in [23] returns the midpoint of the range of values spanned by arguments x_1 to x_n after the f highest and f lowest values have been discarded.

d. **Differential fault-tolerant midpoint**. Denoted $f_d\ f_{tm}$ and used in [24][25] is optimal with respect to the best precision achievable and best drift rate achievable for logical clocks. $f_d\ f_{tm}$ is defined as $(min(T-\Theta, x_l)+max(T+\Theta, x_u))\ /\ 2$,

Where $x_l = x_{hf}+1$,$x_u = x_{hn}-f$ with $x_{h1} \leq x_{h2} \leq x_{hn}$, $h_p \neq h_q$; $1 \leq h_p\ h_q \leq n$ where
T is p's logical time and Θ is the maximum reading error of a remote clock.

e. **Sliding window function**. This function selects a fixed size window w that contains the larger number of clock estimates. This function is proposed in [26] and proposed to convergence functions differing by the way a window is chosen when multiple windows contain the same number of clock estimates and differing by the way the correction term is computed once the window has been identified. The first function, *f det mean*, chooses the first window and returns the mean of the clock values contained

in the window instance. The second function, *fmedian*, chooses the window containing clock estimates having the smallest variance and returns the median of all clock estimates within the selected window. The main interest of sliding window convergence functions is that logical clocks closeness degrades gracefully when more failures are assumed to occur.

f. **Minimization of the maximum error interval-based function.** Denoted *fmm* and takes for each *xi* an interval $[L_{pi}(t)-e_{pi}(t), L_{pi}(t)+e_{pi}(t)]$, where $e_{pi}(t)$ is the maximum error on p_i's clock estimate and returns an interval for the corrected clock value. f_{mm} is used in [27].

g. **Intersection of the maximum error interval-based function.** Denoted *fim* and also used in [28] is similar to f_{mm} in the sense that it also takes intervals representing clock estimates as arguments. f_{im}, however, returns an intersection of the intervals of the clock estimates.

B. Convergence Non-Averaging Techniques

Convergence non-averaging techniques compute a new clock value based on the fact that a fixed number of estimates of remote clocks have been received to compute a new clock value. The number of expected clock estimates depends on the type and number of tolerated failures. When all required clock estimates are received, the local clock is corrected to the value that is computed using the respective algorithm. Reference [28] uses only one estimate as only performance failures are tolerated and the logical clock is corrected with kR, where k is the round number and R is the round duration.

2.6.2 Classifications of Clock Synchronization Algorithms

Clock synchronization algorithms can be classified by the clocks used as reference clocks. If the clocks used as reference are not part of the synchronizing ensemble, this is commonly called *external clock synchronization*. Examples of external clock synchronization algorithms are [27][29]. If the clocks used as reference are also synchronizing clocks, this is called *internal clock synchronization.*

2.6.3 Hardware Versus Software Implementation

Clock synchronization algorithms that are implemented in hardware use specialized hardware components and achieve tight synchronization. On the other side, implementations in software, do not achieve synchronization as tight as hardware algorithms, but use commercial-off-the-shelf components. Lately, the trend towards support of some critical clock synchronization elements being supported by hardware enables tighter precision values for clock synchronization. An example of this trend is driven by the standardization activities around IEEE 1588 [20] effectively requiring some hardware support to be efficiently possible.

2.6.4 Performance of Clock Synchronization Algorithms and its limitations

As the remote clock readings are estimates of local clocks by other nodes and contain some unknown variances, clock synchronization can never be perfect. In other words, local clocks cannot be re-synchronized perfectly, so that at a given time the local clock readings are equal at all clocks. For a symmetric clock synchronization algorithm and a given uncertainty ε and n local clocks, the bound that a symmetric clock synchronization algorithm can achieve immediately after resynchronization cannot be smaller than $\varepsilon(1-1/n)$. For asymmetrical clock synchronization and one master, the bound is ε, the uncertainty of the remote clock reading of the master reference clock. Krause et al. show in [30] that there is a relationship between

precision and drift of the global time. The more frequently an algorithm synchronizes, the tighter the precision, but frequent resynchronization may lead to a larger drift of the global time. This is easily explained by the fact that any error terms in remote clock readings are integrated and increase the drift.

Also, the problem of reaching a common input data or time reference value in spite of faults is an interesting investigation subject [31][32]. It is called "interactive consistency" or "source congruency". It appears that it is theoretically impossible for three computers to agree on a common value, and that at least 4 are required for this. The proof for it is known as the "Byzantine Generals' problem".

The situations which can occur are shown in Fig. 2.1. This way, one can see that to solve this problem you need at least 3t + 1 participants to cope with t traitors.

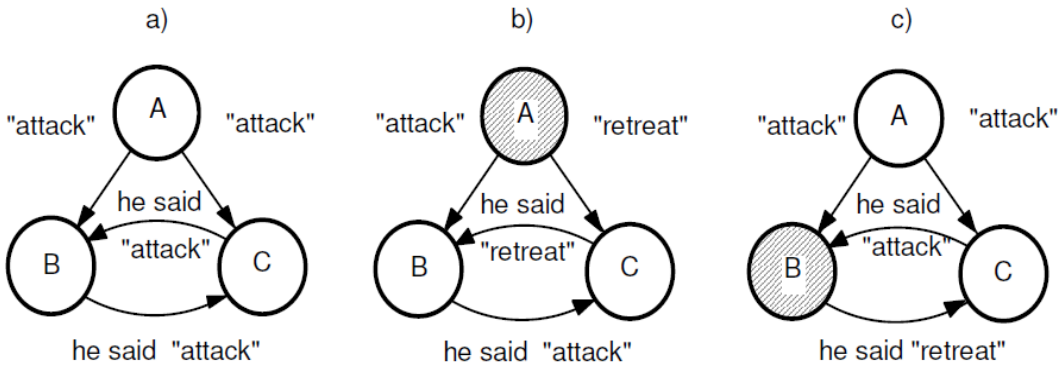


Fig. 2.1 The Byzantine Generals problem

There are however two ways to solve this situation with only 3t participants:

1. Encryption. General A should encrypt his message such that B cannot falsify it, for instance, by writing the command on security paper with his seal and signature. Situation 3 can no longer occur, since B cannot falsify the command it received from A. The worst he can do is not deliver it. So, if situation 2 occurs, C knows that A is the traitor.

2. Atomic broadcast. If the message system is such that the command is transmitted

simultaneously to all participants, then A cannot send a different message to C and B, and situation 2) cannot occur any more. In addition, since C and B broadcast their commands, the traitor will be immediately uncovered. This transmission requires that the message is either received identically or not at all by all participants. Atomic broadcast requires that the message be encoded with an error detecting code such as a CRC. Then, the probability is extremely small that any participant recognizes a false message as good.

2.6.5 Interfacing to Time-Triggered Communication Systems

A paper discusses different interfacing techniques between a computing component (CC) and a time-triggered communication network [33]. The results show that best predictability in time and value domain is achieved by synchronous interfacing, where the execution of activities at the CC is synchronous with the message communication over the time-triggered network. To achieve this, the local clock of the CC has to be synchronized to the global clock of the time-triggered network. With asynchronous access, no clock synchronization is required, but it comes at the cost of unpredictability in the value domain, as it is not ensured which message instance is obtained by a read access.

2.6.6 Global synchronization in the context of the sensor network paradigm

Many emerging sensor network applications require that the sensors in the network agree on the time. A sensor system with global clock will be capable of coordinated operation and data synthesis for future predictions [32]. Consider, for example, a vehicle tracking application. Each sensor may know the time when a vehicle is approaching. By matching the sensor location and sensing time, the sensor system may predict the vehicle moving direction and speed. Without a global agreement on time, the data from different sensors cannot be matched up. Most applications that require the coordination of locally sensed data (e.g., environment

monitoring) or coordination of mobile nodes (e.g., localization in the presence of mobility) are facilitated by the ability of the system to achieve global clock synchronization.

Clock synchronization has been a seminal topic in distributed systems but extending these results and designing clock synchronization algorithms in the context of a sensor network is challenging for several reasons. First, traditional distributed systems assume that all the nodes in a network can communicate directly with each other. A sensor network, however, is subject to spatial constraints. Communication between two remote nodes is accomplished by message relay using intermediate nodes. Second, nodes in a sensor network generally relies on less information about the system than more traditional distributed systems, where nodes have access to the clock values of all the other members of the system, including the faulty nodes. Third, a sensor node has only limited processing capability. The computation intensive signature algorithms, such as RSA, are not suitable for sensor networks. Instead, some light-weight algorithms (such as using a one-way key chain or a key management scheme) are more suitable. The spatial constraints, the communication cost and delay, and the diminished computational capability are key reasons why localized algorithms that involve lightweight computations are preferred for sensor networks.

Among the various methods discussed in [34], the ‘all-node based’ and ‘cluster based’ methods require a node to initiate the global synchronization, which is neither fault tolerant nor localized. In the ‘diffusion-based’ method, each node can perform its operation locally, but still achieve the global clock value over the whole network. There are two implementations of the clock diffusion: synchronous and asynchronous. The synchronous method assumes all the nodes perform their local operations in a set order, while the asynchronous method relaxes the constraint by allowing each node to perform its operation at random. The ‘fault-tolerant

diffusion-based' protocol goes one step further in assuming the presence of malicious nodes that exhibit Byzantine faults. These algorithms can be extended to other sensor network applications, such as data aggregation.

2.6.7 Interaction of clock synchronization and membership algorithm

TTA is able to tolerate more than a single fault by reconfiguring to exclude nodes that are detected to be faulty. This is accomplished by the group membership algorithm of TTA, which is discussed in the following section. The four clocks considered for synchronization are chosen from the members of the current membership; it is therefore essential that group membership have the property that all non-faulty nodes have the same members at all times. A node whose clock loses synchronization will suffer send and/or receive faults and will therefore be detected and excluded by the group membership algorithm. The TTA algorithm is intended to operate in networks where there are at least four good clocks, and it is able to mask any single fault in this circumstance [35].

2.7 Group Membership algorithm

The clock synchronization algorithm tolerates only a single (arbitrary) fault. Additional faults are tolerated by diagnosing the faulty node and reconfiguring to exclude it. This diagnosis and reconfiguration is performed by the group membership algorithm, which ensures that each node has a record of which nodes are currently considered non-faulty.

In addition to supporting the internal fault tolerance, membership information is made available as a service to applications; this supports the construction of strategies for tolerating faults at the application level. For example, in an automobile brake-by-wire application, the node at each wheel can adjust its braking force to compensate for the failure (as indicated in the

membership information) of the node or brake at another wheel. For such strategies to work, it is obviously necessary that the membership information should be reliable, and that the application state of nonmembers should be predictable.

Group membership is a distributed algorithm: each node maintains a private membership list, which records all the nodes that it believes to be non-faulty. The group membership algorithm has to fulfill three major correctness requirements [35]:

- **Validity:** At all times, non-faulty processors should have all and only the non-faulty processors in their membership sets, while faulty processors should have removed themselves from their sets. This requirement is, however, impossible to satisfy as it may take some time to diagnose the faultiness of a processor. We therefore must allow a single faulty processor to be included in the membership sets of non-faulty processors, while faulty processors may have (a subset of) the non-faulty processors plus themselves in their sets.
- **Agreement:** All non-faulty processors should have the same membership sets.
- **Self-diagnosis**: A faulty processor should eventually diagnose its fault and remove itself from its own membership set.

These properties are subject to two additional assumptions that constitute the fault hypothesis. First, as processors will be able to diagnose a fault only if no new fault occurs during that process, the specification of the Time-Triggered Protocol requires the membership algorithm to work properly only if two successive failures occur at least two TDMA rounds apart. More frequent fault arrivals are dealt with by other protocol mechanisms of TTP.

The goal of group membership is to maintain a consistent record of those processors that communicate reliably and execute the protocol. A group membership protocol need not tolerate all the types of faults that may afflict the complete system. For example, redundant broadcast

buses, and strong CRCs (checksums), effectively eliminate message corruption and reduce message loss to a very low level. Clock synchronization ensures that all non-faulty processors share a common notion of time, and "bus guardians" with independent knowledge of the broadcast schedule prevent faulty processors from speaking out of turn.

Send fault: a processor fails to broadcast when its slot is reached.

Receive fault: a processor fails to receive a broadcast.

Other types of faults can be ignored because they are separately detected by other elements of the total protocol suite and then manifest themselves as either send or receive faults. For example, a transient internal fault can lead to a processor shutting down and thus exhibiting a send fault when its slot is next reached.

The requirements mentioned before can be satisfied only under restricted fault hypotheses. For example, validity cannot be satisfied if new faults arrive too rapidly, and it is provably impossible to diagnose an arbitrary-faulty node with certainty. When unable to maintain accurate membership, the best recourse is to maintain agreement, but sacrifice validity. This weakened requirement is called clique avoidance.

2.7.1 Clique Avoidance mechanism

Nodes that suffer receive faults have their local membership lists differ from those of non-faulty nodes, so their next broadcast will be rejected by both their successors. However, TTP employs a slightly different mechanism that is also used to avoid the formation of disjoint cliques at the same time. A clique is a group of processors where agreement on the current state is reached only within the group.

Each node maintains **accept** and **reject** counters that are initialized to **1** and **0**, respectively,

following its own broadcast. Incoming messages that indicate a membership matching that of the receiver cause the receiver to increment it's accept count; others (and missing messages) cause it to increment its reject count. Before broadcasting, each node compares it's accept and reject counts and shuts down unless the former is greater than the latter.

TTA operates as a broadcast bus (even though the recent versions are stars topologically); the global schedule executes as a repetitive series of rounds, and each node is allocated one or more broadcast slots in round. The fault hypothesis of the membership algorithm is a benign one: faults must arrive two or more rounds apart, and must be symmetric in their manifestations: either all or exactly one node may fail to receive a broadcast message (the former called a send fault, the latter a receive fault). The membership requirements would become relatively easy to satisfy if each node were to attach a copy of its membership list to each message that it broadcasts. Unfortunately, since messages are typically very short, this would use rather a lot of bandwidth (and bandwidth was a precious commodity in early implementations of TTA), so it operates with less explicit information and nodes must infer the state and membership of other nodes through indirect means.

2.7.2 Implicit Acknowledgement mechanism in TTP/C

Frames are broadcast over the bus to all stations but they are not explicitly acknowledged. TTP has implicit acknowledgment. Implicit acknowledgment in TTP/C contains an additional feature involving first and second successors. Transmission faults are detected as follows: each broadcaster listens for the message from its first successor (roughly speaking, this will be the next node to broadcast) to check whether it suffered a transmission fault: this will be indicated by its exclusion from the membership list of the message from its first successor. However, this indication is ambiguous: it could be the result of a transmission fault by the original

broadcaster, or of a receive fault by the successor. Nodes use the local membership carried by the message from their second successor to resolve this ambiguity: a membership that excludes the original broadcaster but includes the first successor indicates a transmission fault by the original broadcaster, and one that includes the original broadcaster but excludes the first successor indicates a receive fault by the first successor.

This operates as follows: Each active TTA node maintains a membership list of those nodes (including itself) that it believes to be active and operating correctly. Each node listens for messages from other nodes and updates its membership list according to the information that it receives. The time-triggered nature of the protocol means that each node knows when to expect a message from another node, and it can therefore detect the absence of such a message. Each message carries a CRC checksum that encodes information about its sender's C-State (State of distributed system consisting of membership vector, global time and the current slot position), which includes its local membership list. To infer the local membership of the sender of a message, receivers must append their estimate of that membership (and other C-state information) to the message and then check whether the calculated CRC matches that sent with the message. If it is not feasible to try all possible memberships, receivers perform the check against just their own local membership, and one or two variants.

2.7.3 Low overhead membership algorithm

In [36], a protocol for synchronous group membership that, driven by practical considerations, trades a very restrictive fault model in return for very low communications overhead--just one bit per message has been described and proved. Despite the paucity of information carried by each message, the protocol allows rapid and accurate identification and elimination of faulty processors. However, it has the following limitations:

- The fault arrival rate assumed in the fault model is at most one new faulty processor in any consecutive n + 1 slots. This is clearly tight, since if n were used in place of n + 1, the algorithm fails. Consider a scenario with a receive fault of the processor just before the broadcaster, followed n steps later by a send fault of that same broadcaster. Since the receive-faulty processor will self-diagnose and fall silent in its slot just before the subsequent send fault, all non-faulty processors will not receive two consecutive expected broadcasts. They will all then incorrectly remove themselves from their local membership sets.
- The low overhead group membership protocol has no provision for readmitting previously-faulty processors that now appear to be working correctly again. Simple extensions, such as allowing a repaired processor to just "speak up" when its slot comes by, are inadequate. (A processor that has a receive fault just as the new member speaks up will not be aware of the fact and its local membership set will diverge from that of the other processors; a second fault can then provoke catastrophic failure of the entire system.).

So, if we consider that each sending node piggybacks ‘k’ flags to its message so as to confirm or refute having received the messages from its predecessors, increasing k makes the protocol resilient to a greater number of simultaneous or near-coincident failures but imposes a higher tax on the communication bandwidth. For this reason, the balance between protocol resilience and overhead can be adjusted, at design time, for each system.

Since the bandwidth requirements of a sensor network are not very intensive (specially if there are less no. of nodes), overhead cannot be considered as an issue. Group membership algorithm based on majority voting logic, suitable for smart sensor network, has been formulated and explained in details in section 4.5

CHAPTER 3: Design of Time-Triggered Protocol

3.1 Introduction

The Time-Triggered Protocol (TTP) is intended for use in distributed real-time control applications that require a high dependability and guaranteed timeliness. It integrates all services that are required in the design of a fault-tolerant real-time system, such as predictable message transmission, message acknowledgment in group communication, clock synchronization, membership, rapid mode changes, redundancy management, and temporary blackout handling. It supports fault-tolerant configurations with replicated nodes and replicated communication channels. TTP provides these services with a small overhead so it can be wed efficiently on twisted pair channels as well as on fiber optic networks.

The objective during the design of the Time-Triggered Protocol (TTP) was to develop an integrated protocol that provides all services needed in a fault-tolerant real-time application. In an automotive context, TTP is intended for real-time control systems requiring guaranteed timeliness and fault tolerance.

3.2 Goals for design of protocol

The following goals were considered in the design of the protocol:

1. Predictable low latency:

In a real-time system the temporal accuracy of information is affected by the duration of the protocol execution. A good real-time protocol must have a low maximum execution time and a small variability of the execution time under all specified load and fault conditions.

2. Fault tolerance:

A real-time computer system for safety-critical applications must be fault tolerant. The protocol must tolerate all node and channel failures that are listed in the fault hypothesis without violating the functional or temporal specification. Standard communication protocols provide error detection at the sender's site. In real-time applications communication errors that cannot be masked by redundancy must be detected at the receiving site as well as at the sending site with minimal error detection latency.

3. Temporary blackout handling:

A temporary blackout is the temporary interference of some powerful external disturbance with the operation of the control system. The protocol must detect and handle temporary blackouts promptly.

4. Clock synchronization:

The establishment of a global time base with known precision is one of the basic services that must be provided to distributed real-time applications.

5. Implicit acknowledgement and Membership service:

A membership service provides a consistent view about the health of all nodes. In TTP the membership service is the basis for the implementation of atomic multicast protocols and redundancy management protocols. It is also needed to detect incoming and outgoing link failures. Such a failure detection is required for the implementation of the fail-silent abstraction of nodes.

6. Distributed redundancy management:

The removal of failed nodes and the reintegration of spare nodes and repaired nodes has to be controlled by the redundancy management protocol. In a distributed system the redundancy management itself has to be distributed in order to avoid a single point of failure.

7. Support for rapid mode change:

In many real-time applications a set of different operational modes can be distinguished, e.g., start-up, normal operation, emergency, etc.. The protocol should support the consistent and rapid change from one mode to another mode.

8. Minimal overhead:

The protocol should provide the specified service with minimal overhead, both in message length and in the number of messages.

9. Flexibility, Scalability with determinism:

Flexibility and predictability are competing goals. The protocol should provide utmost flexibility as long as determinism can be maintained. The protocol should be scalable to high data rates. It should operate efficiently on twisted wires as well as on optical fibers.

Existing time triggered protocol specification require further formulation for implementation and poses dependability issues, since the algorithms are not available in open domain. Hence, it cannot be directly utilized in our custom protocol design for use in nuclear plans and facilities. To implement a custom time triggered sensor network protocol, algorithm formulation, simulation and conceptual design has to be carried out. A custom protocol targeted for proven physical layer devices will be provide robustness and determinism and can be qualified for use in distributed sensor level networks of future nuclear plants and facilities.

TTP is an integrated protocol that provides the services listed previously without the strict separation of concerns proposed in the layered OS1 model. The OS1 model is considered an excellent conceptual model for reasoning about the different design issues. But the OS1 model is not a good implementation model for the development of time-critical protocols, since timeliness was not a goal in the development of the OS1 model.

Major sub-systems:

A real-time application can be decomposed into a set of subsystems called clusters eg a controlled object i.e. the machine that is to be controlled and the controlling computer system.

Computational Clusters: The controlling computer system consists of at least one computational cluster. Such a computational cluster comprises a set of self-contained computers nodes which communicate via a broadcast bus using the TTP Protocol. An approximate global time base is established throughout the cluster by synchronizing the clocks located within the nodes. Each node is considered to be fail silent i.e. only crash failures and omission failures can occur. On the cluster level node failures and communication failures can be masked by replicating the nodes and grouping them into fault-tolerant units (FTUs). Message transmission is replicated in both the space domain by using two buses and the time domain by sending the messages twice on each bus.

Within a computational cluster the communication subsystem manages the global concern of providing reliable real time message transmission. The host subsystem comprises the host CPUs of each node computer which execute the local real time application. The interface between these two subsystems the communication network interface is called the Message Base Interface MBI providing host CPUs with a memory area for submitting and receiving messages and for obtaining status and control information about the real time network.

The system wide partitioning into host subsystem and communication subsystem is reflected by the design of the node computer hardware. The host subsystem executes the local part of a distributed real time application. The Interface is implemented with a dual-ported memory and represents the interface to the communication Subsystem; which executes the real time

communication protocol TTP. The protocol code as well as static configuration data is stored in a ROM device. The TTP controller is supported by two bus guardians (BGs). Each channel is protected by one of these devices which protect the bus from being monopolized by a faulty node sending at arbitrary points in time (babbling idiot failure).

3.3 TDMA Bus Access

Each node on bus is assigned a time slot, in which exactly one node is allowed to transmit information on the bus. Thus, latency of all messages can be estimated, which ensures determinism of all real-time messages. TDMA schedule of packet transmission is statically designed and programmed in the schedule table of the controller. The network cycle is a recurring sequence of one or more TDMA cycles. The communication is organized into network cycle as shown in Figure 3.1. Each TDMA round is divided into slots and each slot can be of varying time duration. Each active node in the communication system has exactly one “sending slot” to send a frame in a TDMA round. Different messages can be transmitted in different TDMA rounds but the pattern will repeat in each network cycle.

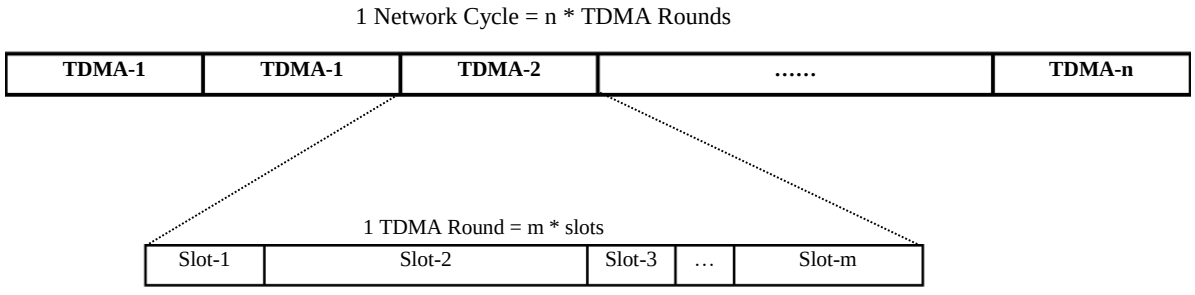


Fig. 3.1 TDMA bus access scheme

3.4 Timing and Synchronization

Synchronized clocks and a global schedule ensure that non-faulty nodes broadcast their

messages in disjoint time slots: messages sent by non-faulty nodes are guaranteed not to collide on the bus. A faulty node, however, could broadcast at any time—it could even broadcast constantly (the babbling failure mode). This fault is countered by use of a separate fault containment unit called a guardian that has independent knowledge of the time and the schedule: a message sent by one node will reach others only if the guardian agrees that it is indeed scheduled for that time.

Now, the sending node, the guardian, and each receiving node have synchronized clocks, but there must be some slack in the time window they assign to each slot so that good messages are not truncated or rejected due to clock skew within the bounds guaranteed by the synchronization algorithm. The design rules used are as follows, where Π is the maximum clock skew between synchronized components.

- The receive window extends from the beginning of the slot to 4Π beyond its allotted duration.
- Transmission begins 2Π units after the beginning of the slot and should last no longer than the allotted duration.
- The bus guardian for a transmitter opens its window Π units after the beginning of the slot and closes it 3Π beyond its allotted duration.

These rules are intended to ensure the following requirements.

- ***Agreement***: If any non-faulty node accepts a transmission, then all non-faulty nodes do.
- ***Validity***: If any non-faulty node transmits a message, then all non-faulty nodes will accept the transmission.
- ***Separation:*** messages sent by non-faulty nodes or passed by non-faulty guardians do not arrive before other components have finished the previous slot, nor after they have started the following one.

CHAPTER 4: Formulation and Simulation of Clock synchronization algorithm and Group membership algorithm

4.1 Introduction to Clock Synchronization

The common notion of time is used to sequence events and to trigger actions. Hence, each node in a time-triggered system maintains a local clock. It is not possible to have oscillators with exactly same nominal frequency in all nodes. So, the local clocks within each node will diverge. A uniform time base among all nodes should be generated within sufficient precision. One of the strategy is to periodically execute a clock synchronization algorithm in each node to bring their local clock close to a common value and to establish a system-wide global time base. Precision is a quality parameter for this global time base which is the maximum deviation between any two clocks in the system. The achievable precision will depend on message transmission jitter caused by the communication system and by clock drifts. With low-quality oscillators, the algorithm should be executed more frequently to keep the clocks of all nodes within the defined precision.

4.2 Details of Clock Synchronization algorithm

Each node maintains a local clock, triggered by a crystal oscillator. This clock drift apart but the algorithm computes correction term and applied on the local clock periodically to keep the time in agreement with other nodes. The corrected clock is used by the nodes during operation. Every receiving node knows the deviation between the sender's clock and its own clock by comparing expected known frame reception time with actual frame reception time. This correction term is used by fault tolerant average (FTA) algorithm. Clock synchronization can be describe using following steps:

1) For each valid message received, determine the difference between expected and observed arrival time for the incoming message at every receiving slot.
2) If a valid message has been received, store four such difference values in a stack.
3) When the stack is full, calculate a new correction term by taking the average of the middle values after ignoring the highest & lowest values.
4) Correction is not applied when the term is very small or beyond tolerance. Discarding the minimum & maximum values ensure that the algorithm works with one faulty node.
5) The propagation delay is subtracted from the correction term to get the final offset value. In this way adjustment is applied. Propagation delay values (depend on the distance between the nodes) will be available as a configuration in static schedule table for every slot.
6) In this way, adjustment is applied to the local time periodically in order to keep time in agreement with all other nodes.

Thus, global time base and static TDMA schedule together provide collision-free communication on a broadcast channel. The above algorithm will work correctly under following conditions:

1) The clocks of all non-faulty nodes must be within a linear envelope of real-time. That is, the drift of hardware clock of all nodes is bounded by some constant ρ

   $1 - \rho \leq d(LC(t))/dt \leq 1 + \rho$

   Where LC(t) is the local clock value at any time t.
2) The number of nodes (n) is known in the whole system and each node can send messages to all other nodes. (each message is broadcasted by every node)

3) To tolerate f number of faulty clocks there is a constraint on the number of operating nodes: $n \geq 3f + 1$. (To tolerate single fault, $n \geq 4$ because $f=1$)

4) All clocks are synchronized in the beginning:

$$| LC_i(t_0) - LC_j(t_0) | \leq \Pi / 2$$

Where, $LC_i(t_0)$ and $LC_j(t_0)$ are the local clock value of i^{th} and j^{th} node at initial time (t_0) respectively. Π is the precision value of global time base. The frame delivery delay is limited. A message is delivered with a finite jitter. In simulation it is assumed that the frame delivery delay is constant with no jitter. The algorithm runs in rounds, resynchronizing after certain time interval to correct the clocks drifting out of synchrony, and using a fault-tolerant averaging function.

Most other clock synchronization algorithms that run in rounds eg. Welch-Lynch algorithm, they differ in the step-2, in which the calculation term is calculated from the difference-values stored in the stack. For a stack of size S, after sorting of stack-values, WLA takes the average of two values: $(f+1)^{th}$ stack entry and $(n-f)^{th}$ stack entry, FTA takes the average of all values between $(f+1)^{th}$ stack entry and $(n-f)^{th}$ stack entry. Based on simulation results, stack size has been taken as four and the average of 2^{nd} and 3^{rd} stack entry is calculated.

4.3 Simulation of Clock Synchronization in Scilab

Simulation is a reasonable and powerful means for gaining insight into the functionality of systems. It is important to simulate clock synchronization algorithm to verify correctness and to tune parameters used for implementation since complete information and implementation aspects are not available in literature like the effect of stack size, effect of large drift in faulty node, number of iterations required for initial convergence etc. Simulation also verified the fault tolerance capability of the algorithm and the effect of applying weighted correction value

has been analyzed.

The algorithm was modeled and simulated to verify the correctness and convergence in Scilab. Scilab is an open source high-level, numerically oriented programming language. It provides an interpreted programming environment, with matrices as main data type. The core of Scilab is based on linear algebraic libraries. Scilab is a powerful and accessible tool for education all over the world.

Following elements were defined in Scilab program:

1) Number of nodes (N): taken as input from user.
2) TimeOffsetMatrix: It stores time offset of each node with respect to real time. Each column corresponds to a particular node. The elements of each row is updated with every iteration of algorithm execution.
3) Number of iterations: Iterations for which algorithm will execute. After some iterations of algorithm execution, Offset value converge (same value within precision).
4) TimeDiffMatrix: A collection of stack maintained by each node. Each stack has a depth 4 or a configurable stack size (S).

Following are the steps used for simulation:

1) Initial offset values for all nodes are generated and stored in first row of TimeOffsetMatrix. For each slot, the time difference as seen by a particular node is calculated and stored in TimeDiffMatrix.
2) After the collection of S (stack size) values, algorithm computes correction term. It is applied by changing the offset value of the corresponding node (say i^{th} node) and is stored

as a new time value in the next row of i^{th} column (TimeOffsetMatrix). This is one-iteration.

3) Drift for each node is added.

4) Steps from 2^{nd} to 4^{th} are repeated for 10 iterations.

5) In step 4, instead of adding the drift term directly, the correction term is divided by a value which has been named as 'weighing factor (WF)' and now this drift term is added.

4.4 Observations

Graph were plotted to see how the time offset / local clock in each node change while applying the correction term as per the existing algorithm. Figure 4.1 to Figure 4.3 show graph for varying number of nodes. Time values of all nodes (N) converge to a single value when $N \geq 3$. But, for N = 2, the time value does not converge.

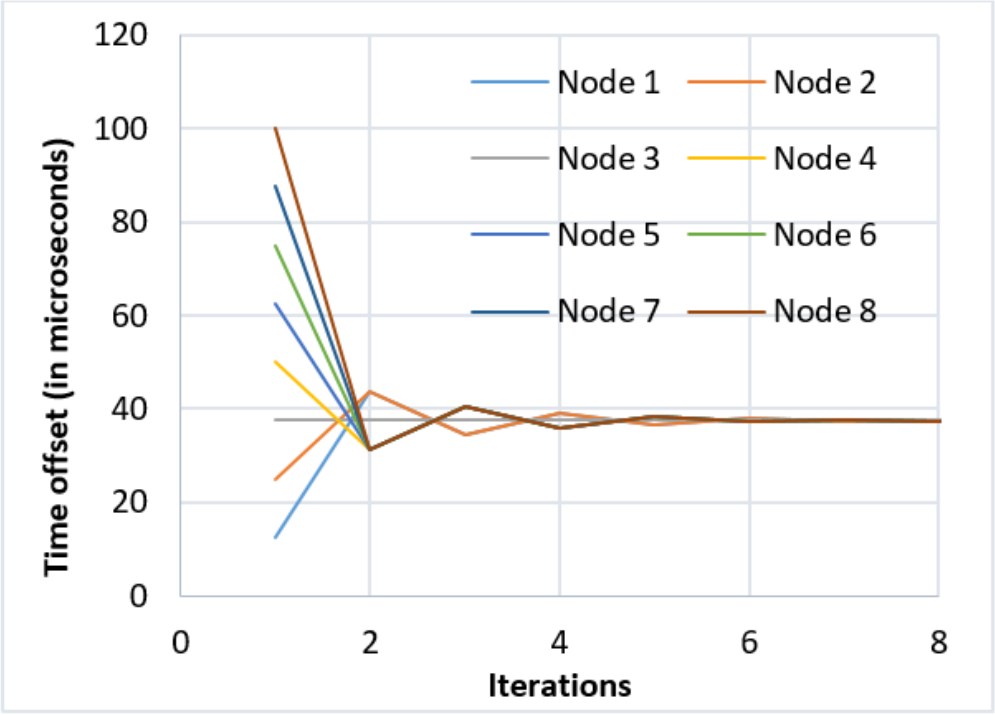


Fig. 4.1 Simulation for N = 8, S=4

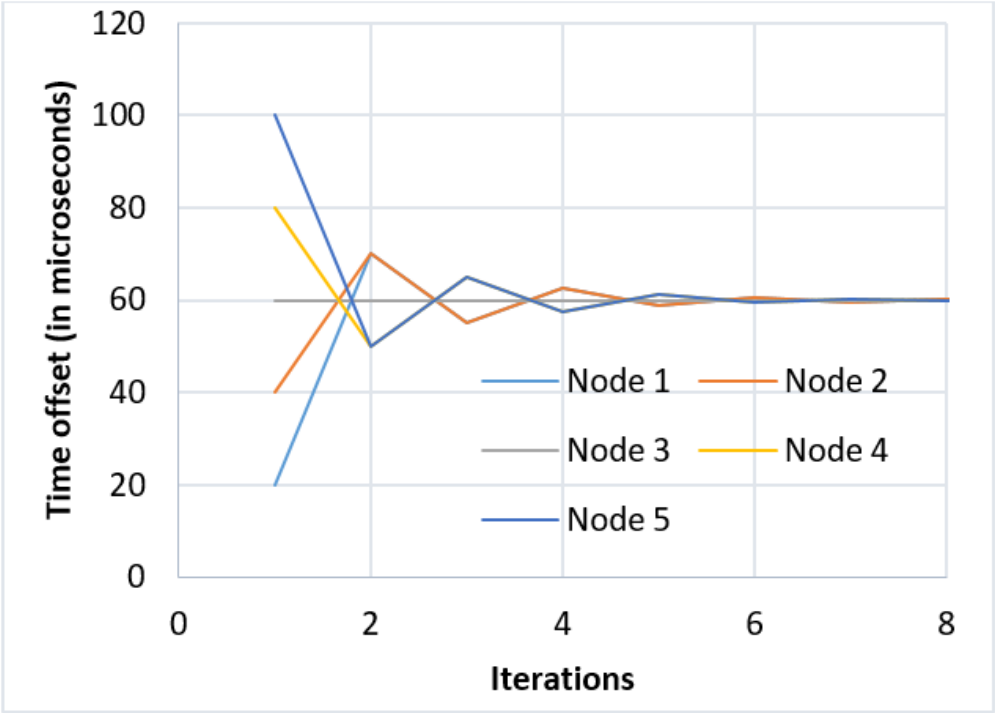


Fig. 4.2 Simulation for N = 5, S = 4

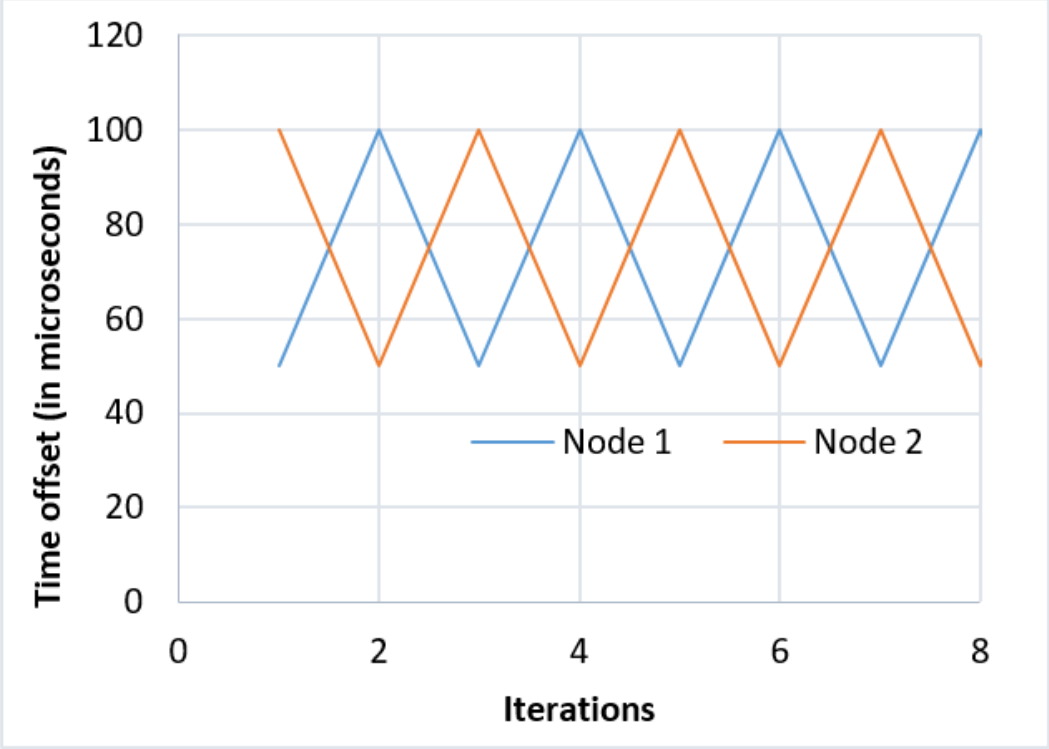


Fig. 4.3 Simulation for N = 2, S = 4

i. Effect of weighing factor

Calculated correction value is divided by a weighting factor (WF). Choosing a no weighing factor (equivalent to choosing weighing factor= 1) does not allow convergence of a N=2 node system as depicted in Figure 4.3. It was observed that using a weighing factor < 1 caused the divergence, refer Figure 4.4 (WF=0.5). Whereas using a weighing factor much greater than 1 caused the weighing factor to converge sluggishly leading to more no. of iterations required for convergence, refer Figure 4.5 (WF=4). Figure 4.6 depicts the convergence for WF=1.5. Considering various observations, the optimized value of weighing factor of 2 has been chosen.

ii. Results with modification to existing algorithm

Figure 4.7 to Figure 4.9 are graphs generated using weighting factor as 2. Using this modification, convergence is smooth and possible with a set of two nodes. For the FTA algorithm to operate, $N \geq 4$ is required to tolerate single Byzantine fault [31]. However, with this modification, $N \geq 3$ is required to tolerate single Byzantine fault.

iii. Verification of fault tolerance

Fault was injected in Node-1 by forcing the time offset to Constant high (out of bound value), Constant low (out of bound value), monotonically decreasing and High-Low transitions. Graphs in Figure 4.10 to Figure 4.13 show that a fault introduced in a single node (Node-1) do not affect convergence and it also verify the single failure criteria of the algorithm. In this case bus guardian forces the faulty node to be fail-silent.

iv. Effect of stack size

Stack size (S) effects the final value at which all nodes will converge. Stack size was varied and results were generated. It was found that final converged value is close to nodes of initial slots when stack size is less and vice versa. Simulation has been done for $S < N$ and results are shown in Figure 4.14 to Figure 4.17.

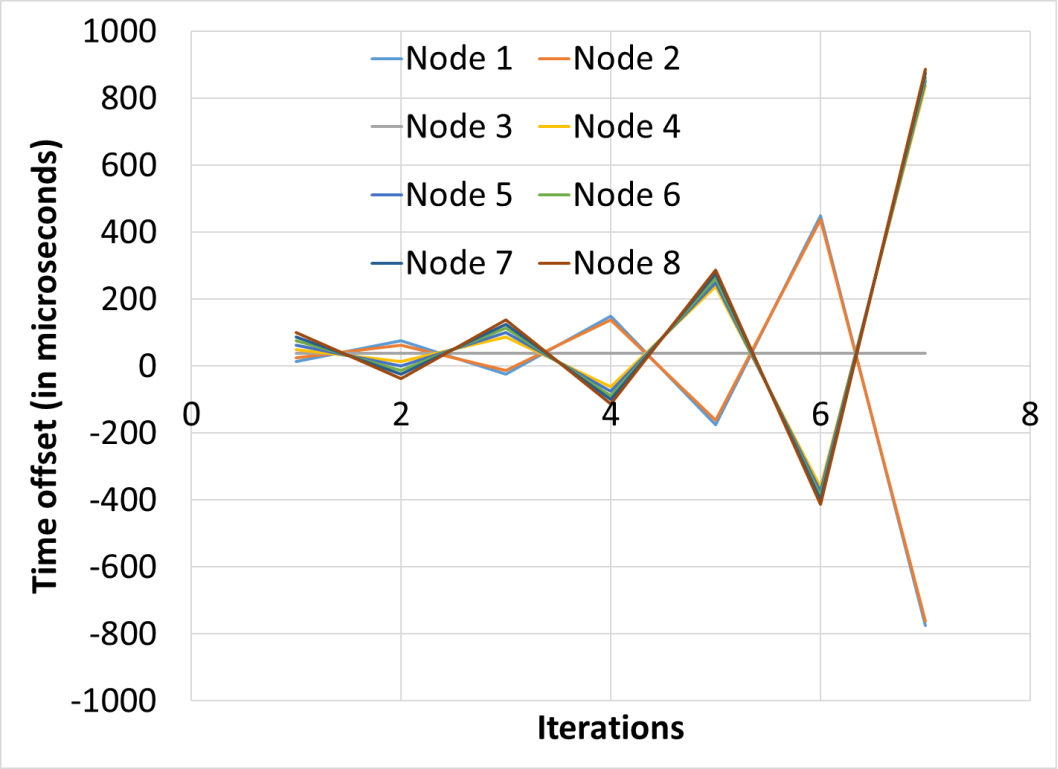


Fig. 4.4 Simulation with WF =0.5, N = 8, S = 4

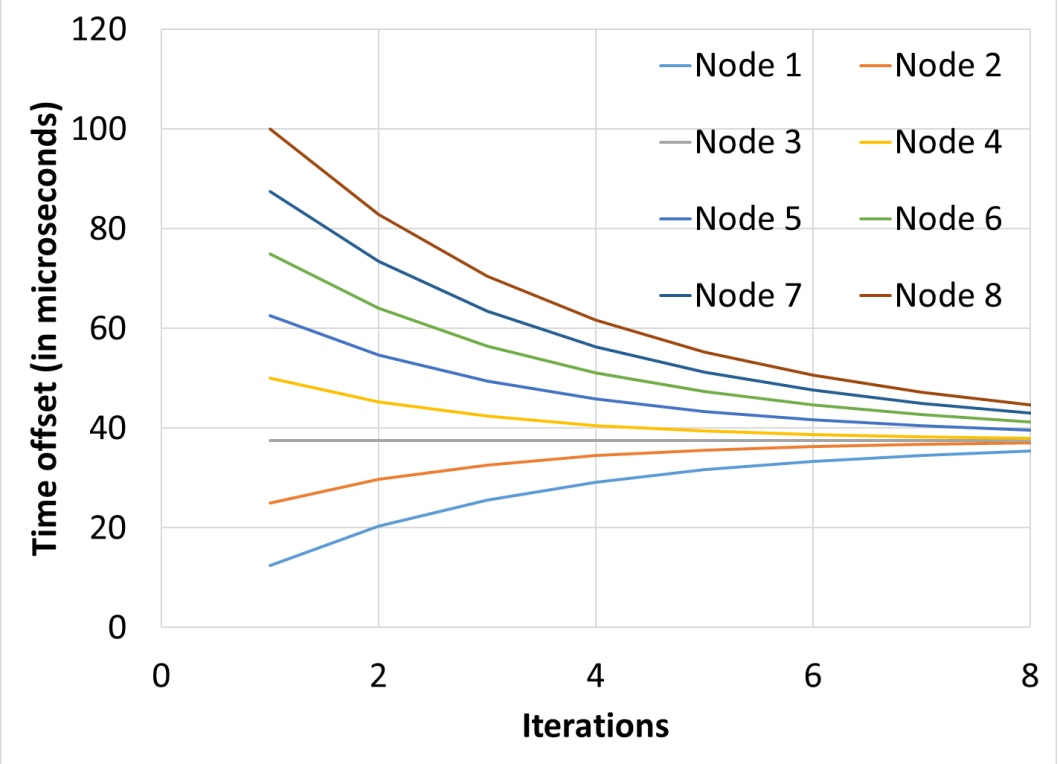


Fig. 4.5 Simulation with WF = 4, N = 8, S = 4

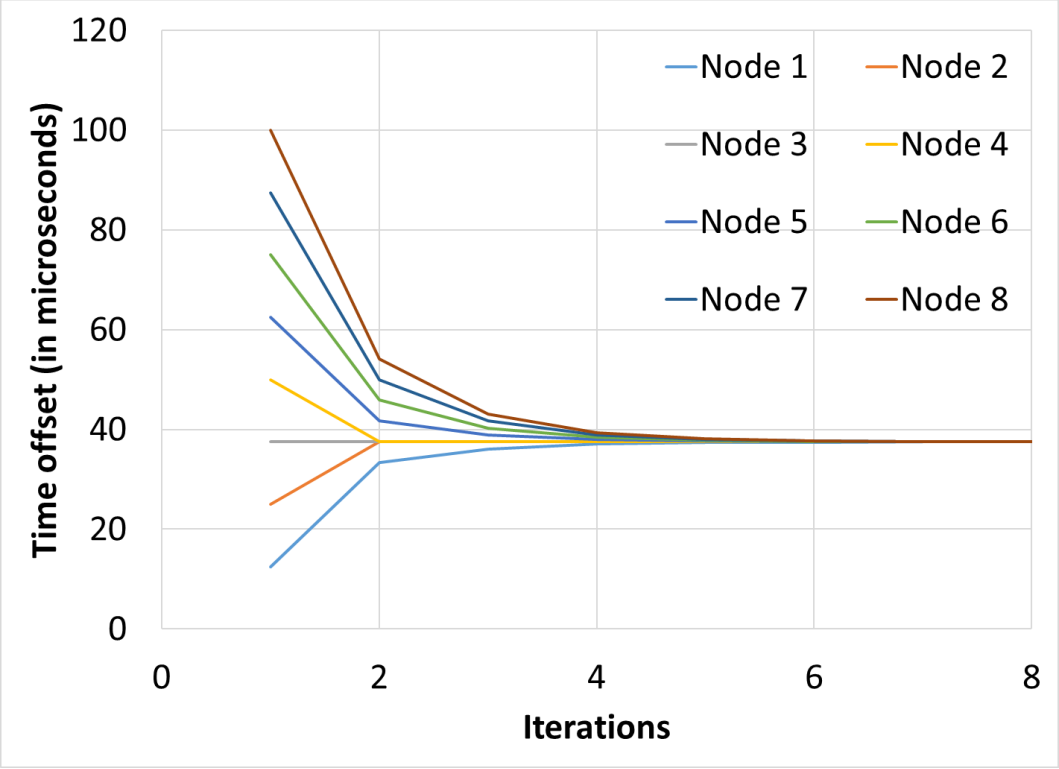


Fig. 4.6 Simulation with WF = 1.5, N = 8, S = 4

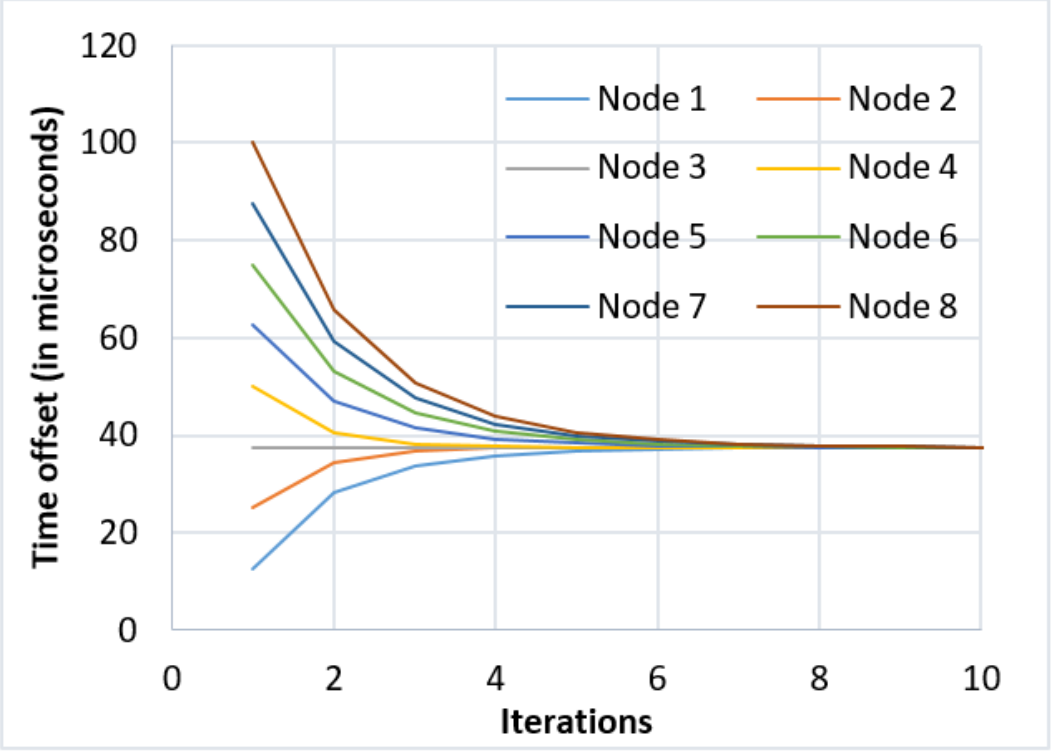


Fig. 4.7 Simulation with WF =2, N = 8, S = 4

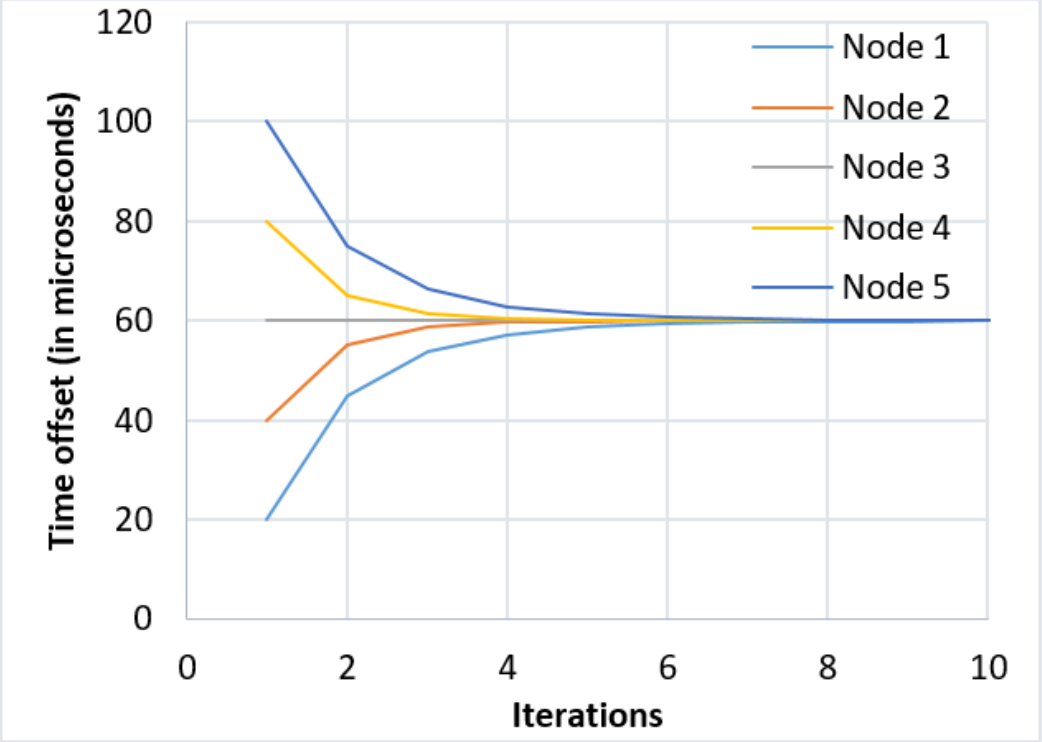


Fig. 4.8 Simulation with WF =2, N = 5, S = 4

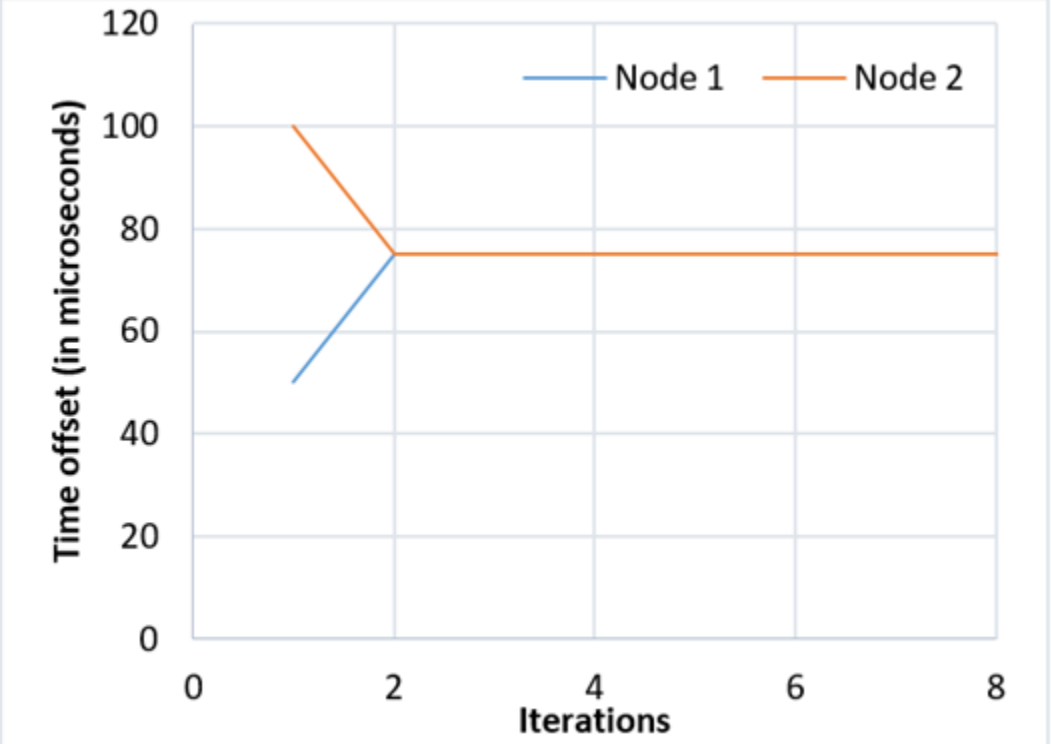


Fig. 4.9 Simulation with WF =2, N = 2, S = 4

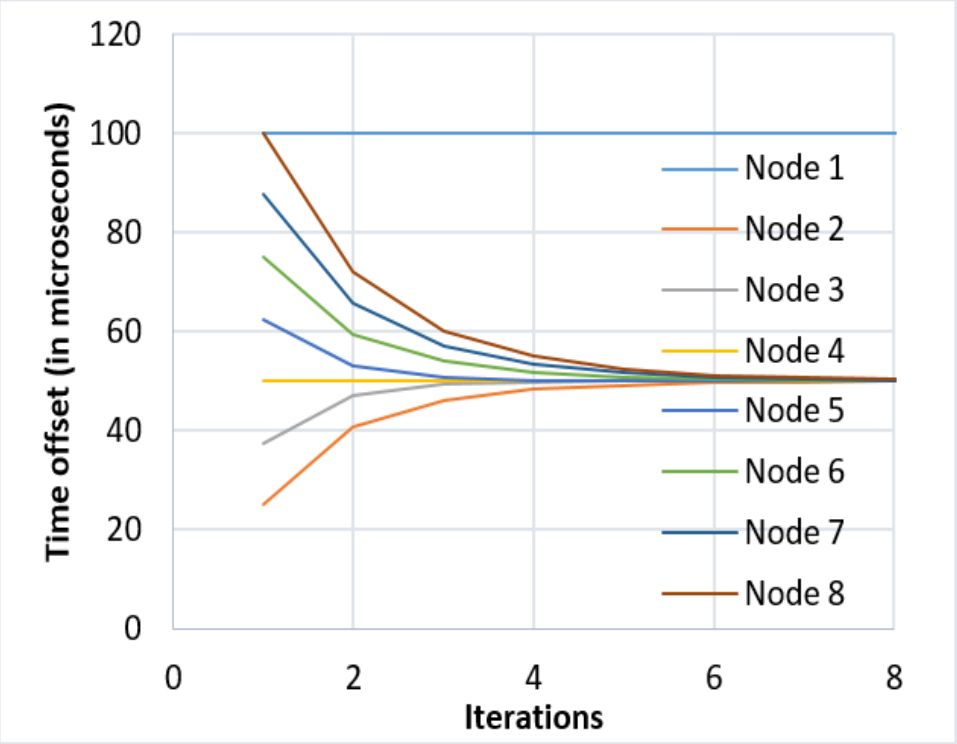


Fig. 4.10 Simulation with fault in Node-1 (Clock offset stuck at constant high value), WF=2, N = 8, S = 4

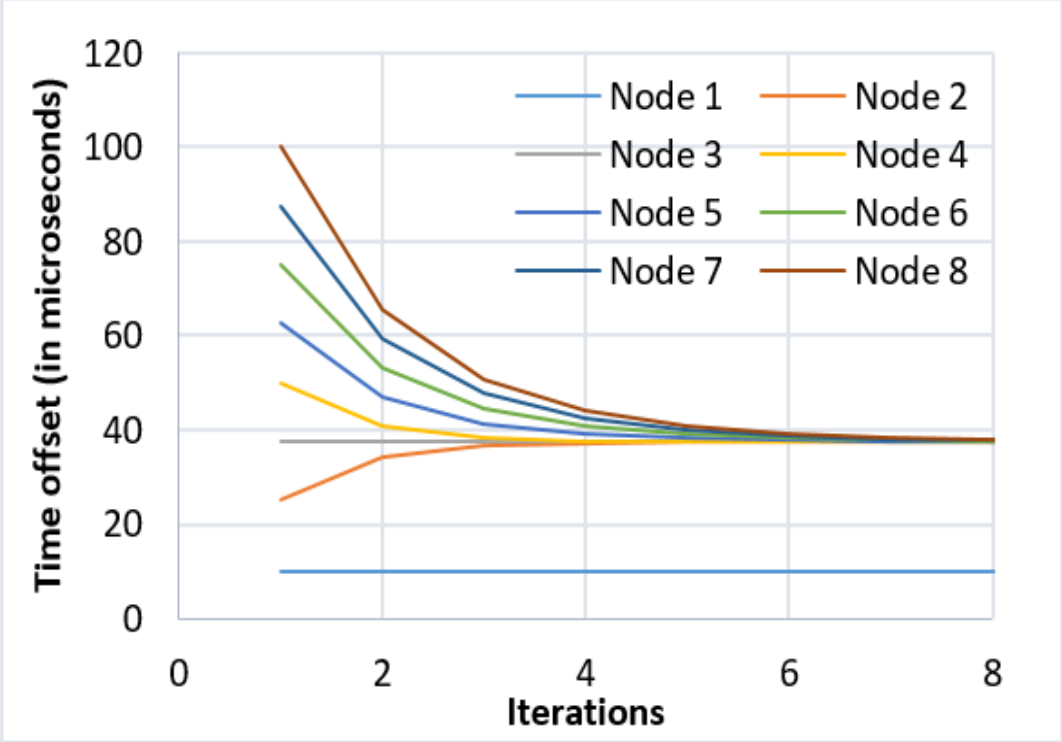


Fig. 4.11 Simulation with fault in Node-1 (Clock offset stuck at constant low value), WF=2, N = 8, S = 4

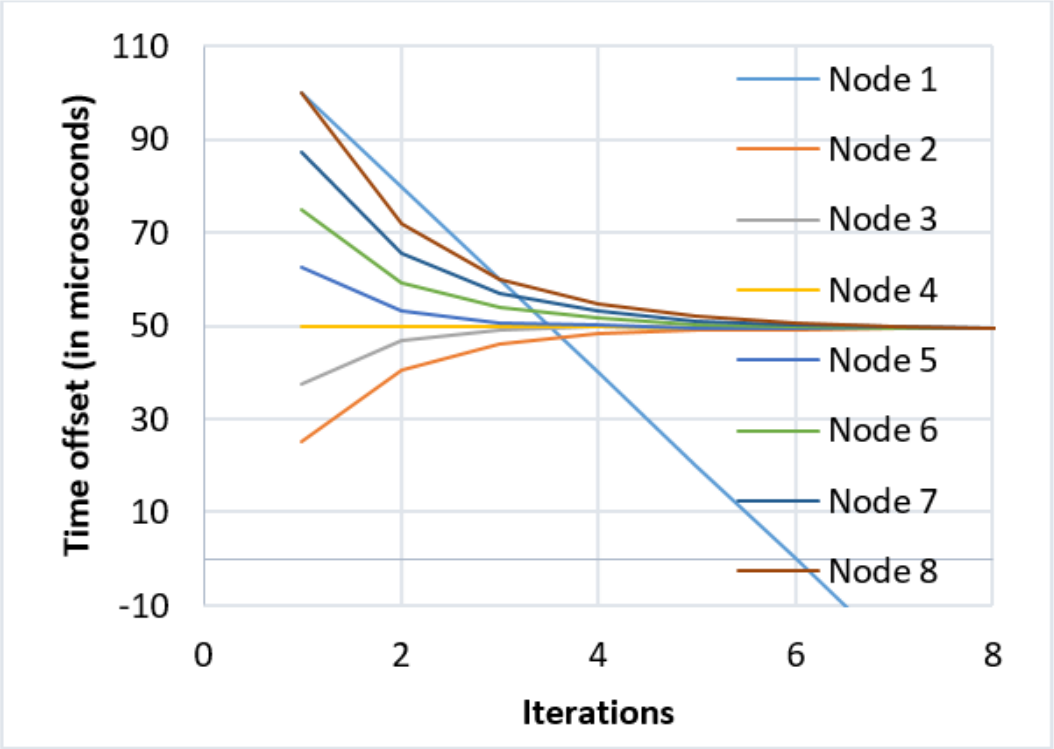


Fig. 4.12 Simulation with fault in Node-1 (Clock offset stuck at monotonically decreasing value), WF=2, N = 8, S = 4

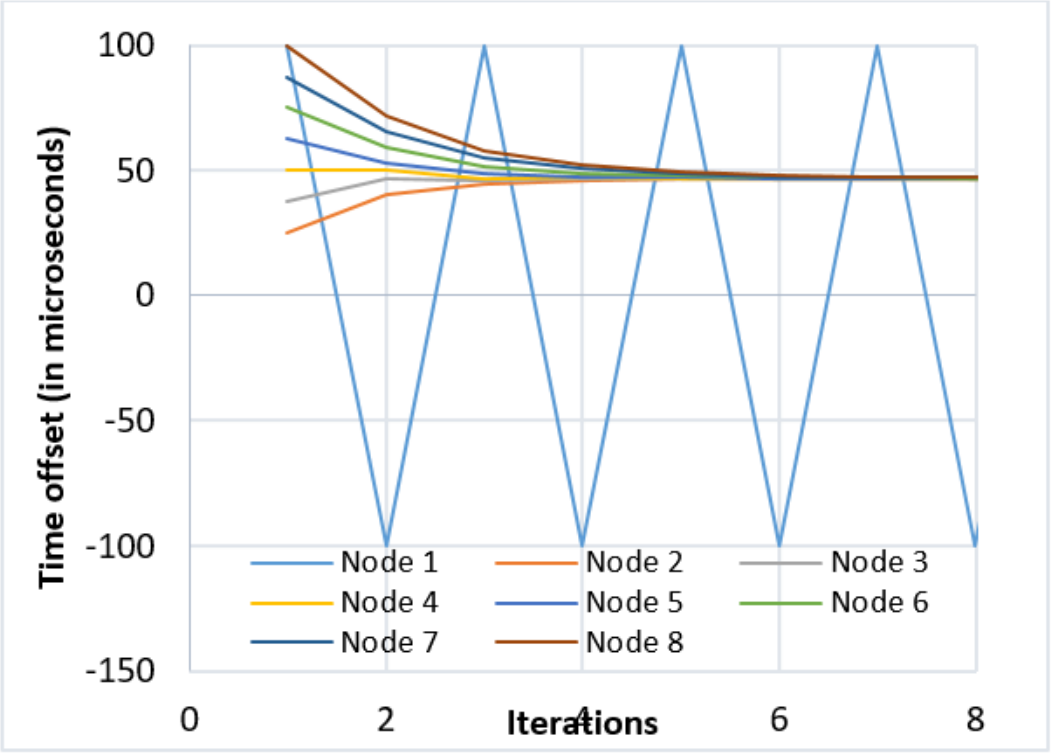


Fig. 4.13 Simulation with fault in Node-1 (Clock offset with varying transitions), WF=2, N = 8, S = 4

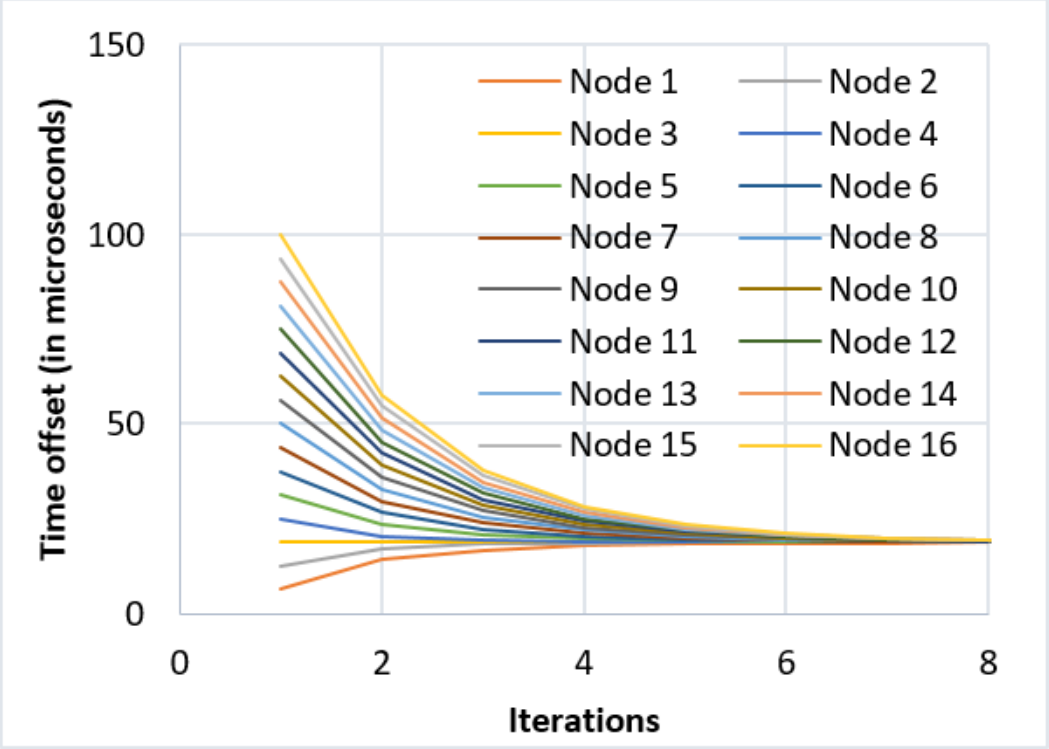


Fig. 4.14 Simulation with WF= 2, N = 16, S= 4 to analyze effect of stack size

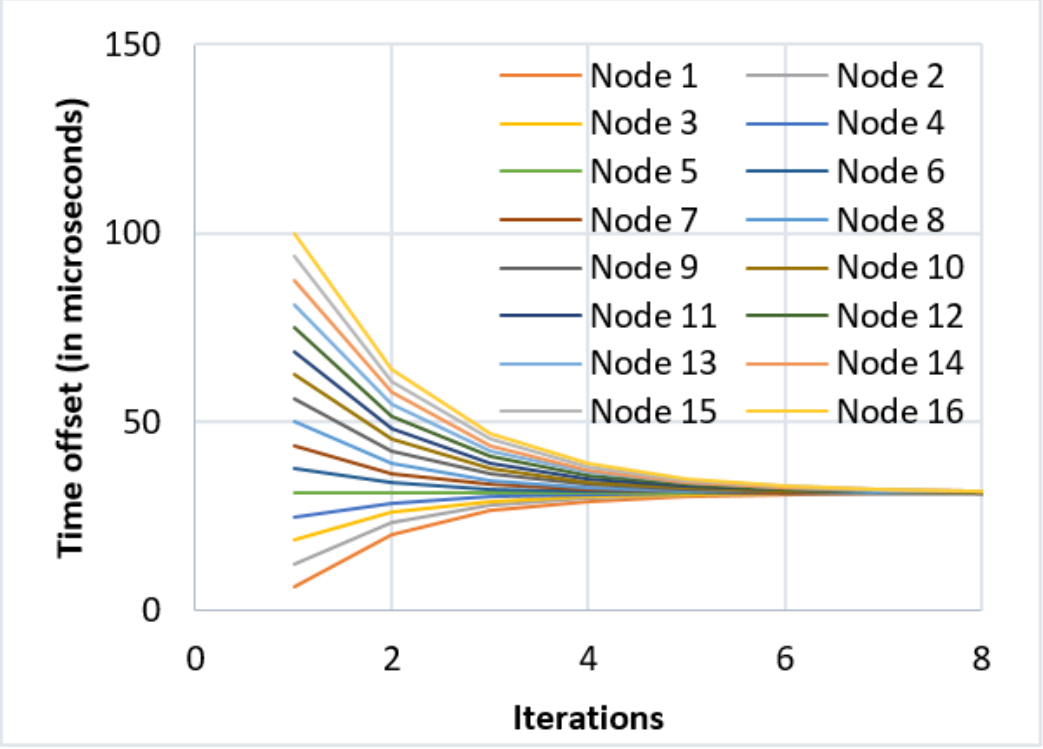


Fig. 4.15 Simulation with WF= 2, N = 16, S= 8 to analyze effect of stack size

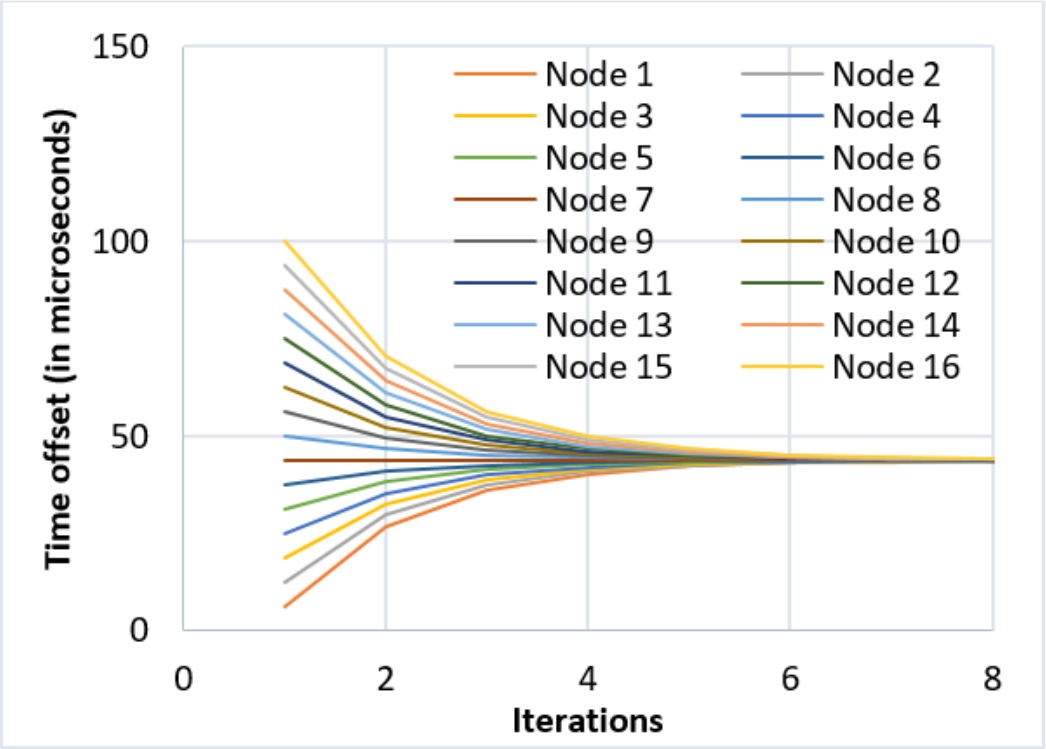


Fig. 4.16 Simulation with WF= 2, N = 16, S= 12 to analyze effect of stack size

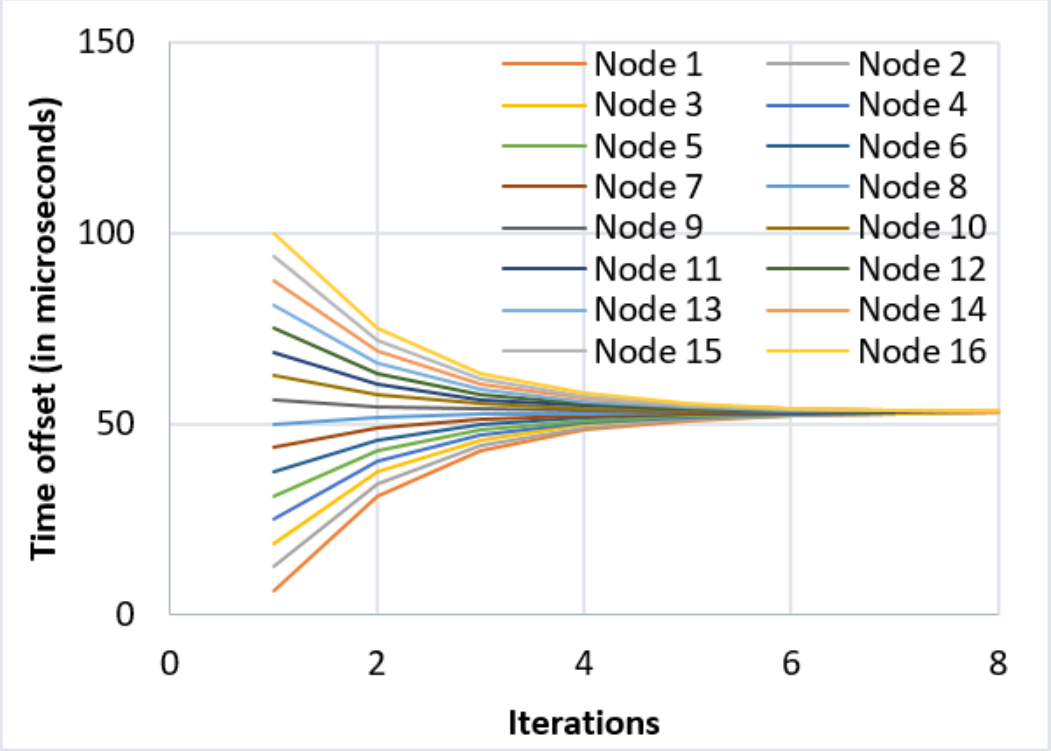


Fig. 4.17 Simulation with WF= 2, N = 16, S= 15 to analyze effect of stack size

4.5 Majority voting Group membership algorithm

A majority voting based group membership algorithm for sensor network will provide implicit acknowledgement and a consistent view of status of all nodes in the network. Each node in the network maintains a list of valid frame receipt status called as Local Membership Vector (LMV). In each slot the node updates its LMV based on the condition whether valid frame was received or not. If a valid frame is received from the expected node, then an entry is marked as valid (‘1’) else marked as invalid (‘0’).

In sending slot, the node will piggyback and transmit the LMV as a header along with the data frame to all the nodes. Thus each node piggybacks the acknowledgments of the previous transactions from all other nodes with its own broadcast transmission. This kind of acknowledgment does not require any separate frame.

Every node receives the LMV of all other nodes and a matrix called as “Acknowledgement State Matrix” is formed as shown in Figure 4.18. Thus, enough data is collected to perform a membership algorithm. To get a consistent view of status of nodes throughout the network, a strict majority voting is carried out and the final result (Global Membership) is calculated. It is possible that there can be an equal amount of ones and zeroes if the number of nodes in the system is even. The system will then vote zero since it is safer to assume that an application is down. A node which finds itself voted out goes into idle state (or shut down), so as to not disturb the timely transmission of other nodes. Acknowledgment is done at communication controller level and does not require any processing from host. It should be noted that all nodes together decide about the health of a node.

To summarize, the controller autonomously checks the acknowledgment; provides the Global

membership using the algorithm and the host computer may or may not use it. Also, this algorithm will send a faulty node to idle-state if it is itself faulty or sending faulty frames, thus adding a feature of fail silent.

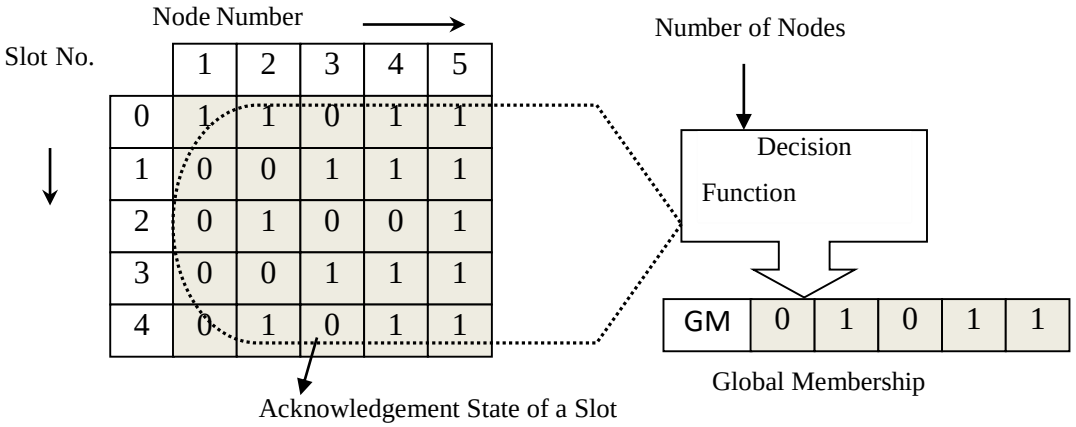


Fig. 4.18 Bitwise strict majority voting membership algorithm

4.6 Simulation results of Group membership algorithm

Simulation has been performed on ‘spyder’. Spyder is a scientific Python Development Environment. Python is an interactive, object-oriented, high-level and general-purpose interpreted programming language. Its Language Constructs and Object-Oriented Approach aim to help programmers write clear, logical code for small and large-scale projects. The ‘tkinter’ module has been used to develop the Graphical User Interface (GUI) for simulating the group membership algorithm. ‘tkinter’ is the standard GUI library for Python. Python when combined with tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

For the group membership simulation, in the beginning the number of nodes in the system has to be given input by user. GUI will display the Local Membership Vector (LMV), Acknowledgement State Matrix (ASM) and Global Membership Vector (GMV) corresponding

to every node. There is provision of injecting faults (Transmit fault, Receive fault or both) using button clicks.

i. Normal operating condition with no faults

When there are 5 nodes operating in normal working state, it was observed that for a system of 5 nodes, 3 have successfully been inferred as 'active' by the algorithm (through GMV of all nodes). If this observation is generalized it can be stated that for a system with N nodes, N/2 (for even N) or (N+1)/2 (for odd N) nodes will become 'active' in 1st TDMA cycle of initialization.

It can be observed that for a system of 5 nodes, all 5 nodes have successfully been inferred as 'active' by the GMV of all nodes by the end of 2 TDMA cycles, refer Figure 4.19. If this observation is generalized it can be stated that for a system with N nodes, all N nodes will become 'active' after 2nd TDMA cycle (initialization).

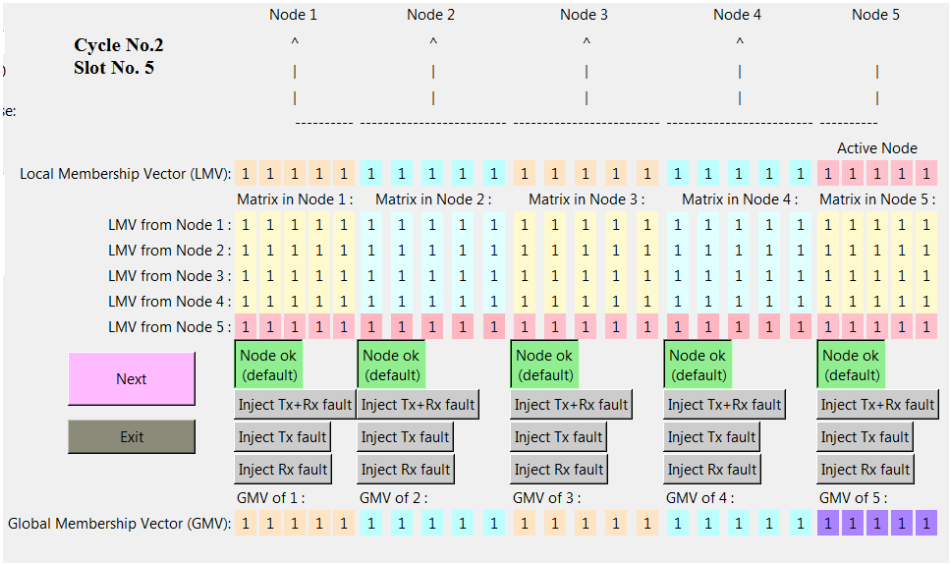


Fig. 4.19 Initialization result with all nodes non-faulty

ii. One Node failure (Both Transmit / Receive fault)

When a node is powered off or is completely disconnected from the system, the faulty node does not receive valid data and it doesn't transmit valid data as well. The same has been incorporated accordingly in the LMVs and consequently in ASMs through the algorithm. It has been observed (refer Figure 4.20) that the algorithm detects the fault because all the other nodes (non-faulty ones) have the corresponding faulty node's bit transformed from '1' to '0' in their respective GMV.

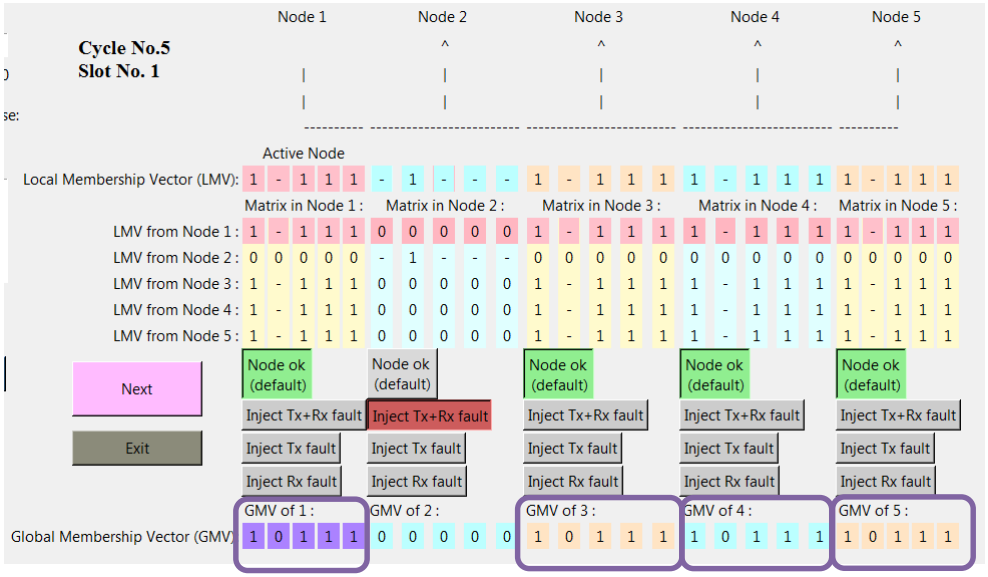


Fig. 4.20 One node failure (Both transmit / receive failure)

iii. One Node Transmit failure

Similar to the Transmit / Receive fault detection, transmit fault in a network is detected and that node is removed from the active set of nodes, refer Figure 4.21.

Observe that a send fault can only occur to a processor when it is in the broadcast slot, and a receive fault can only occur to a processor different from the broadcaster. Also, notice that messages cannot be corrupted, and that a send fault is consistent: no processor receives a

message from a send-faulty broadcaster.

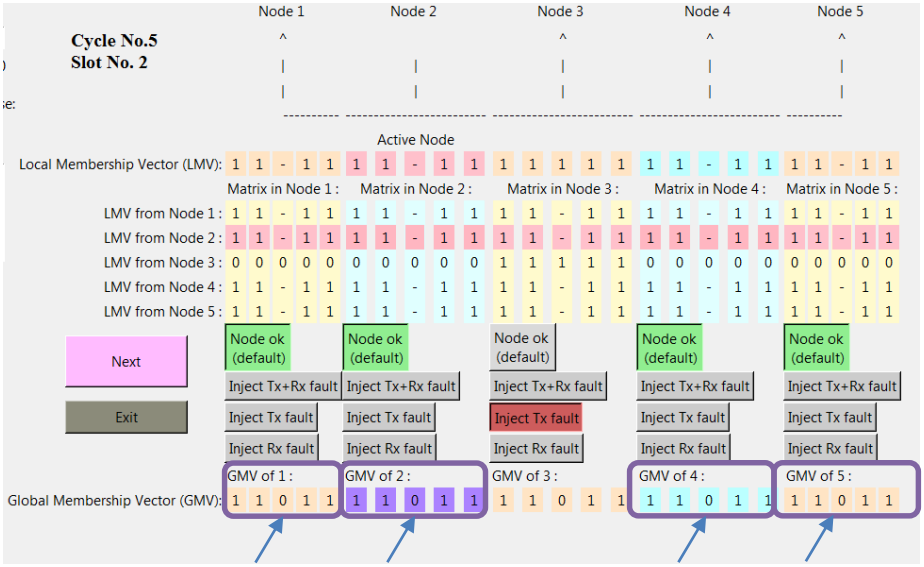


Fig. 4.21 One node transmit failure

iv. One Node Receive failure

If a node suffers a symmetric send fault, then after having completed the Implicit Acknowledgement algorithm, it will reconsider its clique assignment. However, when a single node suffers a symmetric receive fault it will form a clique with only a single member (refer Figure 4.22). Note that the GMV of the node with receive fault is different than the all the other non-faulty nodes.

So, it is considered that this fault is an asymmetric send fault where a malicious sender node sent correct data to all nodes except a single one. In this way, receive fault in a network is detected and that node is removed from the active set of nodes. Note that since the node is out of working set of nodes its GMV has no bound to follow the property of 'Agreement' anymore. And since no data is received on the node, all the bits of its GMV are calculated to come out as zero.

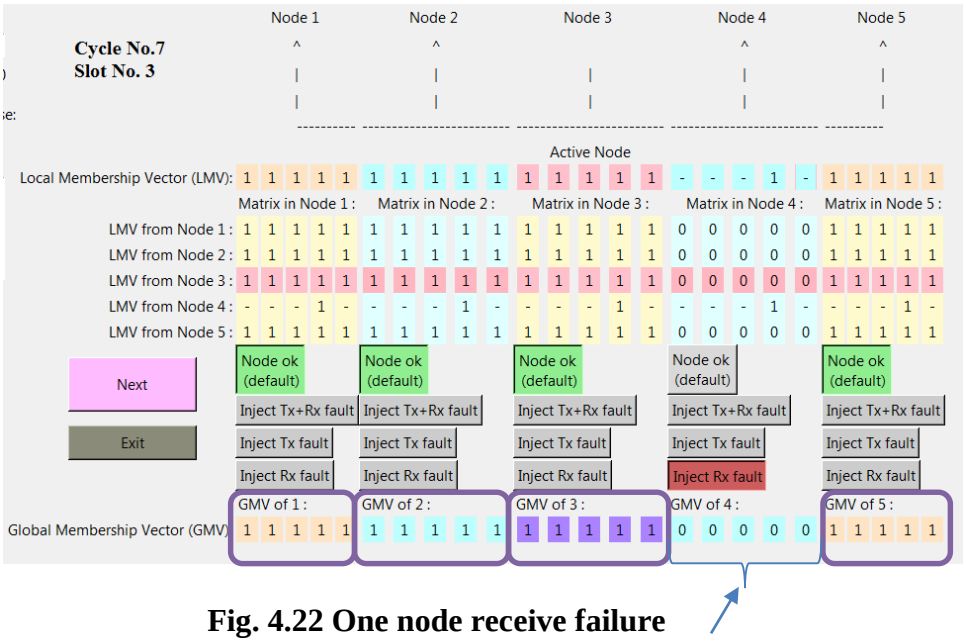


Fig. 4.22 One node receive failure

CHAPTER 5: Experiments and Observations

5.1 Custom protocol Hardware Architecture

A custom protocol and architecture named as Deterministic Fault Tolerant Communication Protocol (DFTCP) is formulated. DFTCP is based on time triggered communication with conflict-free static TDMA bus access and dual redundant communication channels. It is master-less and messages are transmitted as broadcast. All nodes are time synchronized using distributed clock synchronization algorithm. Each node consists of a host computer and a communication controller based on time triggered architecture as shown in Fig. 5.1.

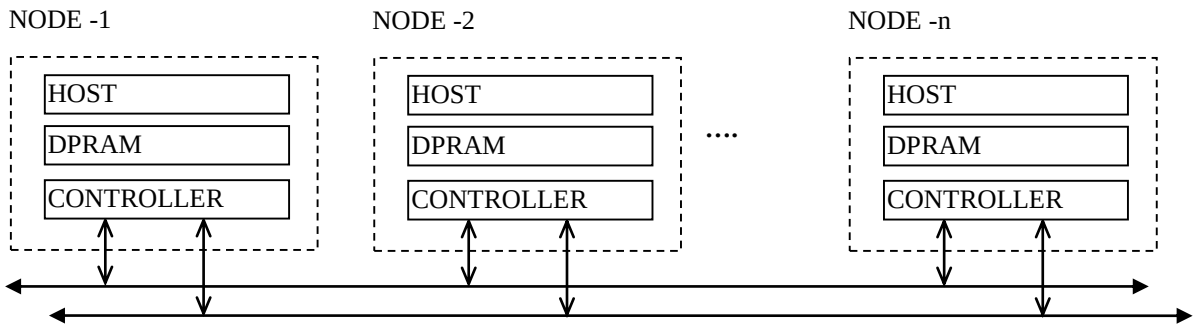


Fig. 5.1 Architecture of a DFTCP network

Host is responsible for execution of user application, whereas the communication controller executes the communication protocol. The incoming and outgoing messages are stored in DPRAM and it acts as a temporal firewall between the host and controller. A temporal firewall provides control-free interface between two subsystems. An understandable abstraction of the subsystem behind the firewall, confines the impact of most changes to the encapsulated subsystem, and limits the potential of error propagation.

The protocol is designed to isolate and/or tolerate single fault. Media access is through an independent bus guardian at for each communication link of the node. It is implemented using

a separate onboard clock. Bus guardian is a kind of hardware watchdog to ensure fail-silent behavior and it guards against “babbling idiots”. It guarantees that a faulty host can’t kill protocol operation.

This protocol will be specifically designed for safety-critical fault-tolerant applications. The system will have fault tolerance implemented in both hardware and software. Whereas the hardware relies on duplicated communication channels, the software uses algorithms that control such basic services as membership agreement, clique avoidance, and clock synchronization. To tolerate the failure of a node, nodes can also be replicated, & may grouped into Fault Tolerant Units (FTUs).

5.2 Considerations in hardware implementation

To select transmission medium best suited for an application; this protocol does not specify the medium or signaling method. In fact, there are no restrictions on the signaling method because DFTCP is not an arbitration-based protocol. An encoding technique such as modified frequency modulation, which has fewer than one transition per bit, can be used to increase the channel capacity on twisted pairs. DFTCP also scales well to high transmission speeds for fiber- optic systems since it requires no bit-wise arbitration.

The interface between a host computer and the DFTCP controller can be realized with a dual-ported RAM that contains the control registers for the DFTCP controller, the descriptor fields of the modes, and the memory for the incoming and outgoing data objects.

The present global time and the recent history of membership fields are available in special registers. Eventually, DFTCP must be implemented in a hardware communication controller. DFTCP’s conflict-free media-access protocol simplifies the interface at the signaling level and

makes the protocol scalable to very high transmission speeds.

i. Start-up and Integration

The change of an unsynchronized network to a synchronized one is performed by the startup and Integration Logic.

When a node is started after power-on or reset, it is unsynchronized and does no transmission. If this unsynchronized node finds a running network, it adopts the time-base, and this procedure is called as integration. If the unsynchronized node does not find any valid frames on the network, it will send special-frame named as Network-Startup Frame, to start communication and to allow other nodes to acquire to its time-base. This procedure is termed as Startup. The controller goes through the following sequence of events:

1. Initialization of the DFTCP controller.
2. Listen on both the broadcast-channels for frames for the duration of the listen timeout.
    a. If a frame is received, the controller adopts the slot number obtained from the received frame and integrates. The listen timeout of each controller is same and is equal to twice the TDMA cycle.
    b. If a silent network is detected and if startup from this node is allowed (each node may or may not have startup rights as configured by the Communication Schedule) the node sends ‘network startup frame’ after a unique Startup-TimeOut. This node, now expects that other nodes will integrate on this frame. If it receives a data frame from any other node, it goes to active state & starts its normal operation.
3. When more than one node transmits startup frames simultaneously, a collision is sensed, by detecting CRC error, and then it restarts its unique startup timer and retries startup after

time out. The collision can occur only at the startup. As all nodes with startup rights have unique timeout no more collisions are expected.

4. If startup is not allowed from this node, it will continue to be in listen state till it receives any valid data frame or Network-Startup frame. There is no Listen-Timeout for these nodes.

ii. Protocol states

For the operation, seven protocol states are defined:

i. *Idle:* The execution of the protocol is halted until the controller is given a reset.

ii. *Init:* initialization.

iii. *Listen:* Listen for any valid frame to integrate until the listen timer expires. If valid frame received, then it will integrate & go to Active state or Passive state, for monitoring node. If no valid frame received before listen timer expires then controller will go to startup wait state if this node has startup rights.

iv. *Passive:* Monitoring node will enter this state. The controller is synchronized but frame transmission is prohibited.

v. *Startup wait:* If no frames received for the unique startup-timeout, then it enters the Startup state. If valid frame received, then it will integrate & will go to Active state.

vi. *Startup:* It facilitates the integration of other controllers by periodically sending Network startup frames until it receives a response from another controller, or if it runs out of startup retries; in this case, it goes back to the listen state. On receiving the response from other controllers, it will go into the active state.

vii. *Active:* The controller is synchronized with the group. It transmits frames according to its schedule-table.

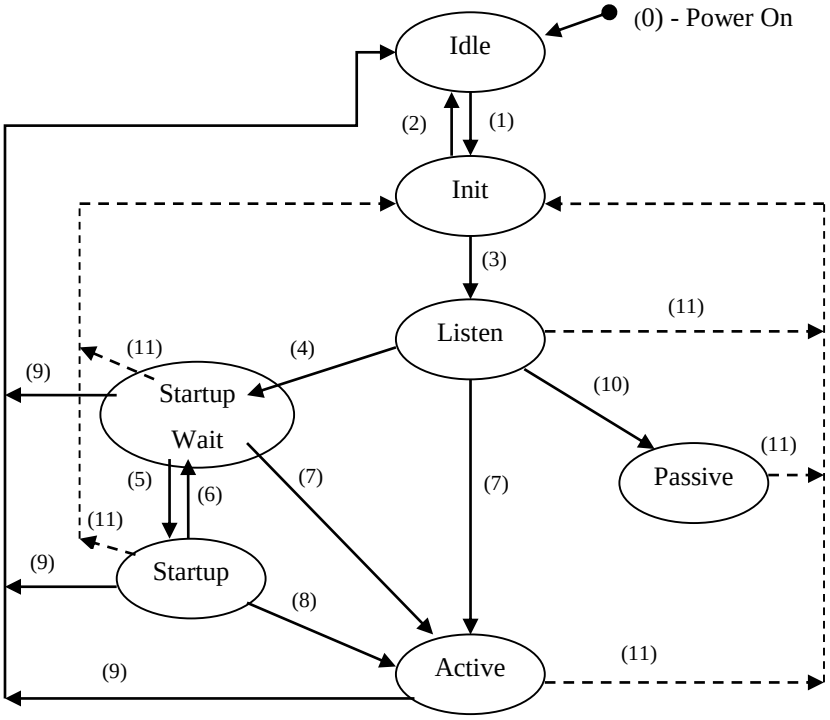


Fig. 5.2 State diagram for protocol

Table 5.1 : Events causing state transition (Refer Figure 5.2)

(0)	Power Switched ON.
(1)	Reset from Host, start on the operation of the controller.
(2)	Schedule-Table CRC check failed or controller initialization error.
(3)	Initialization Completed successfully
(4)	Listen timeout expired, startup is allowed
(5)	Startup-Timer expires.
(6)	Collision detected.
(7)	Startup Frame or Data Frame from any other node was received. The node has integrated on it.

8)	Response from other nodes were received i.e. other nodes have integrated by the Startup Frame sent.
(9)	Synchronization error, Membership error, Periodic Schedule-Table CRC check failed.
(10)	If the node is a monitoring node and any valid frame received.
(11)	Host has given a RESET command.

5.3 Experimental Set-up

The objective is to validate clock synchronization algorithm and group membership algorithm. Since the jitter to be achieved is in the order of few micro-seconds and parallel processing of many functions is required, an existing FPGA board with RS485 interface has been used. Refer Figure 5.3 for block diagram.

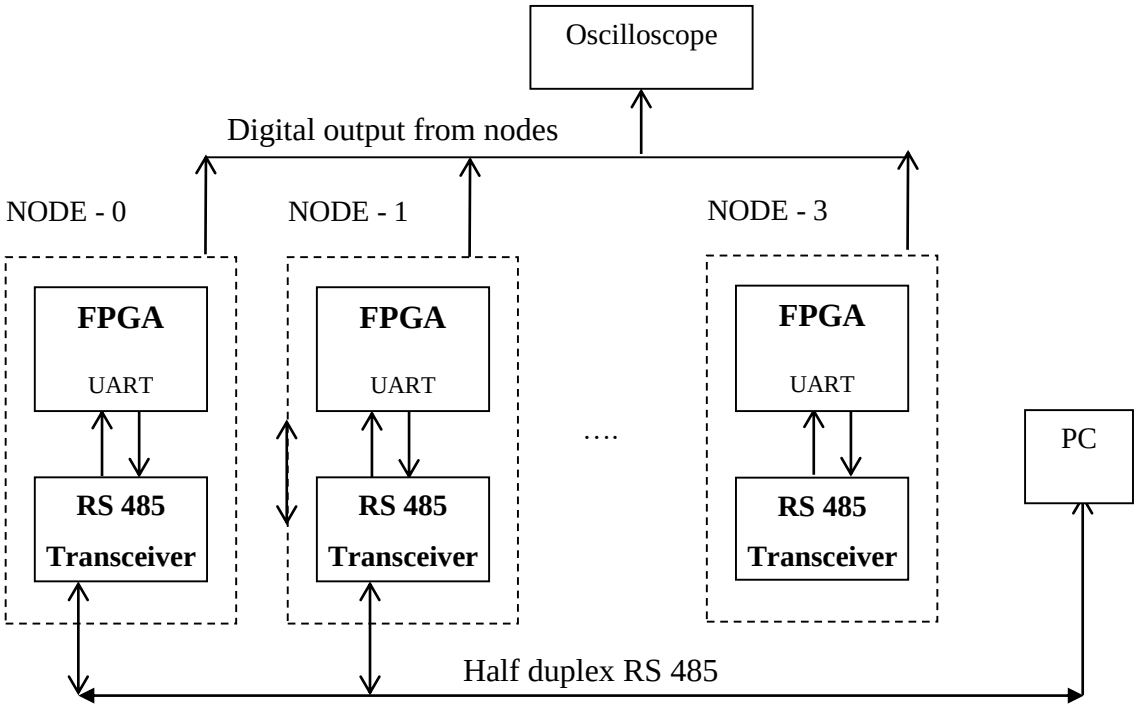


Fig. 5.3 Architecture of the custom TT network

The experimental set-up was used to observe the behavior of a network with 3 active nodes as shown in Figure 5.4.

Fig. 5.4 Experimental set-up

The protocol operation and the communication are handled by the controller implemented in a FPGA. The software has been divided into a number of modules that execute in parallel inside the FPGA. The software is developed using VHDL. The protocol includes limited services necessary to validate clock synchronization and membership algorithm.

Specifications assumed in VHDL code:

Software consists of a constant static schedule of a 4 node system with 2 TDMA rounds as shown in Figure 5.5. Data rate has been considered as 921.6 kbps with 16 bit CRC. Slot time of 5 msec and data length of 250 bytes was configured.

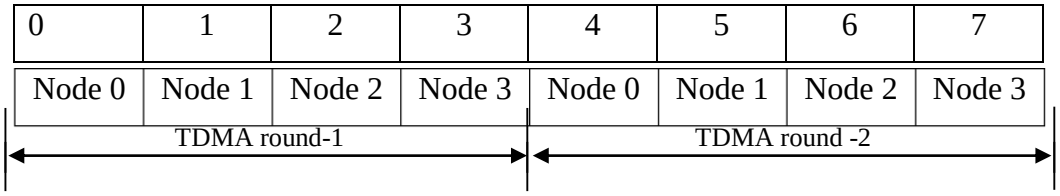


Fig. 5.5 Network cycle

The frame format used is as shown in Figure 5.6.

Control-Word	Membership Vector (MV)	Data	CRC-16
2-Bytes	*2-Byte*	*250 Bytes*	*2-Bytes*

Fig. 5.6 Frame format

The Control-Word contains slot number, size of the M, Channel-ID and reserved bits as shown in Figure 5.7.

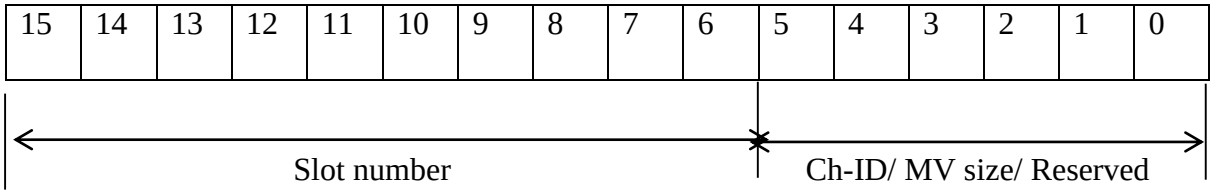


Fig. 5.7 Control word

Hence, for Slot No. 0, 2 TDMA rounds and 16-bit MV (Membership vector), control word is 0x20 0x00, for Slot No. 1 it is 0x 60 0x 00 and so on. MV indicated the healthy nodes (active nodes) in the network. In this configuration, TDMA round has 4 slots, 3 Active nodes (Node 0, Node1 and Node 3) has been considered. Node-2 has been considered as inactive/faulty and is not connected. So the membership vector with all 3 non-faulty nodes will be 0x0B 0x 0x00 (0000 1011 0000 0000).

5.4 Observations

Packets over RS485 network are captured in PC using “Free Device Monitoring Studio” software (screenshot in Figure 5.8) to view packets transmitted by each node on the network.

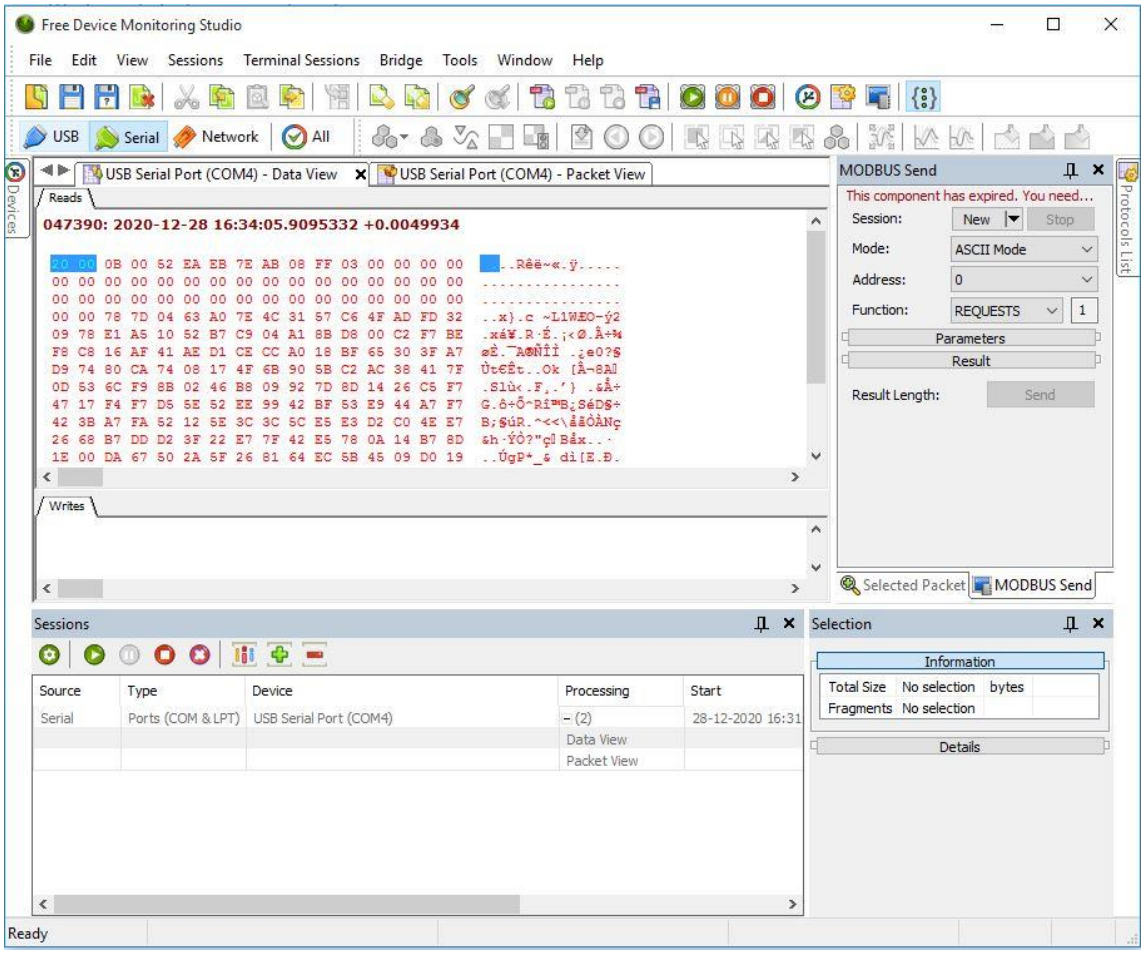


Fig. 5.8 Screenshot of software denoting data packets received from the nodes

Figure 5.9 shows the snapshot of the exported html file. It can be observed that all nodes (Node 0, Node 1 and Node 3) follow the schedule and has sent the data packets in its own scheduled slot accordingly.

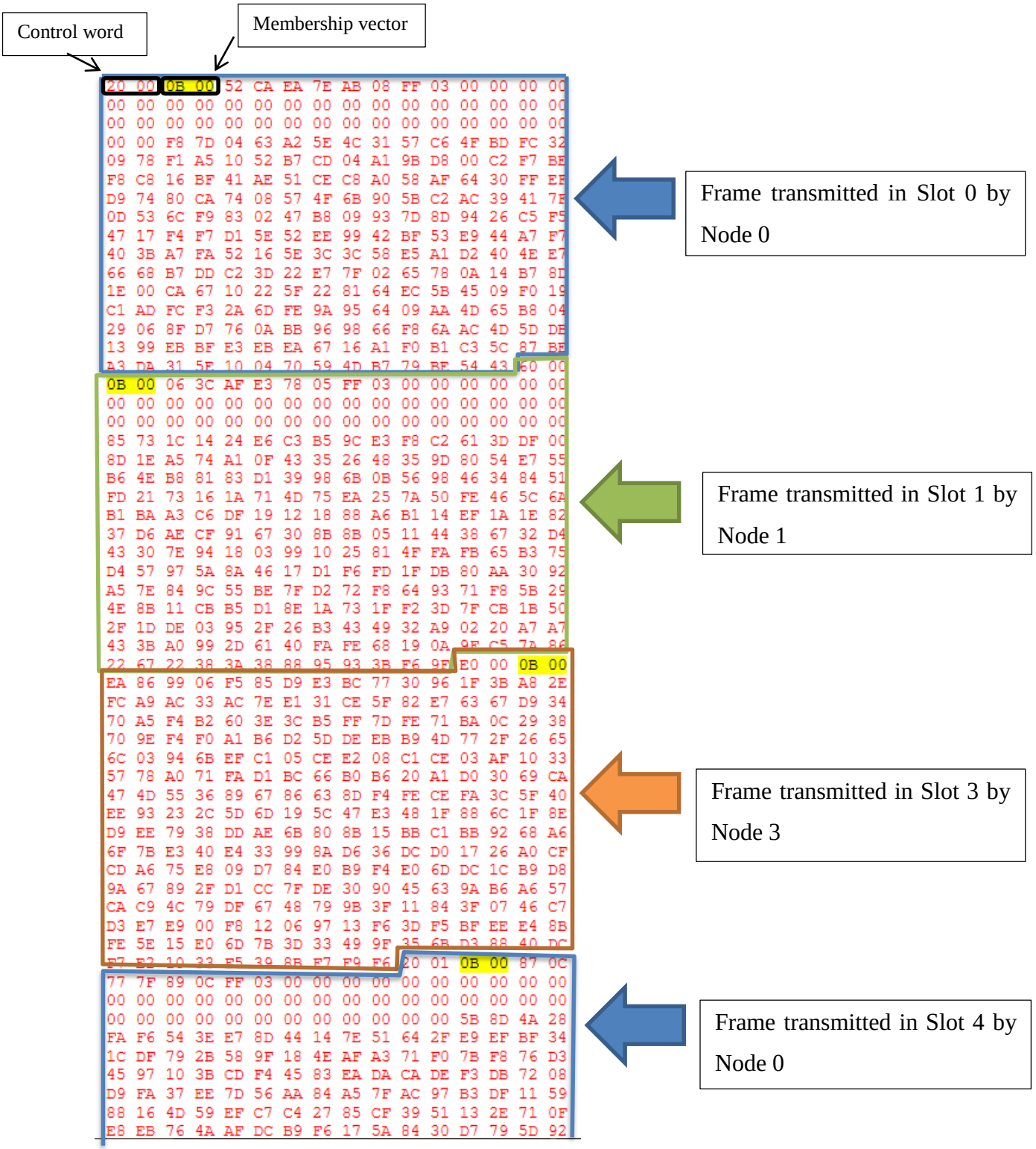


Fig. 5.9 Data packets received from the 3 node system over the broadcast network

i. Validation of clock synchronization algorithm

A digital output of FPGA has been programed to toggle on every network cycle change (every 5*4=20 ms) as per its global time, refer Figure 5.10. Objective is to see the global time is

synchronized in two nodes. The output from 2 nodes has been captured in oscilloscope and jitter was found to be in the order of µsecs:

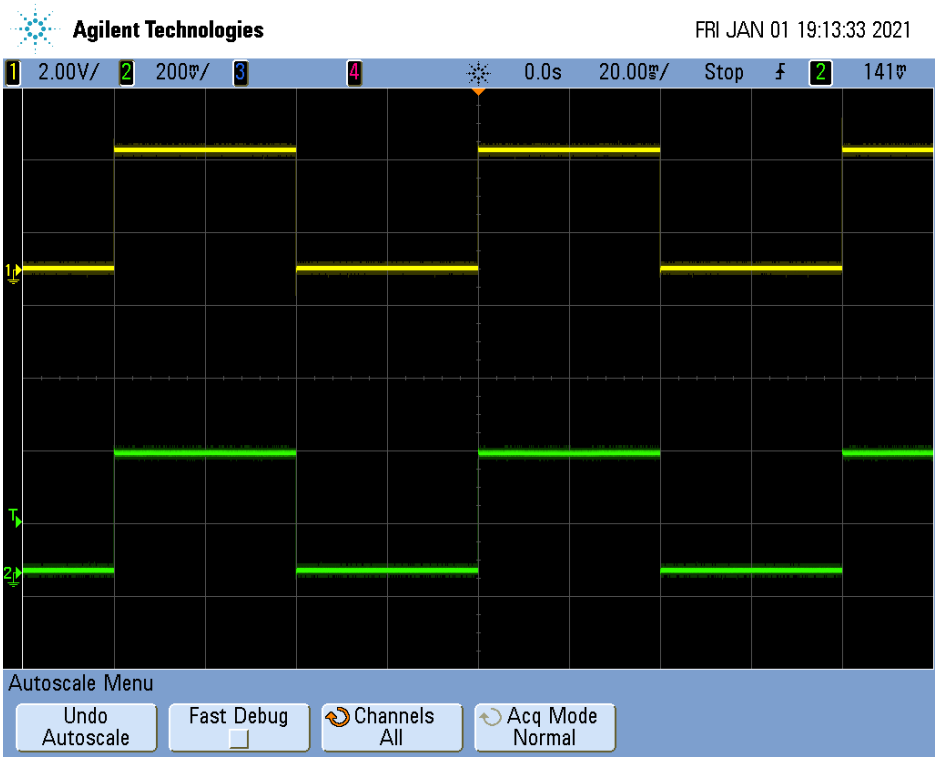


Fig. 5.10 Digital output from two 2 nodes, toggling after every network cycle

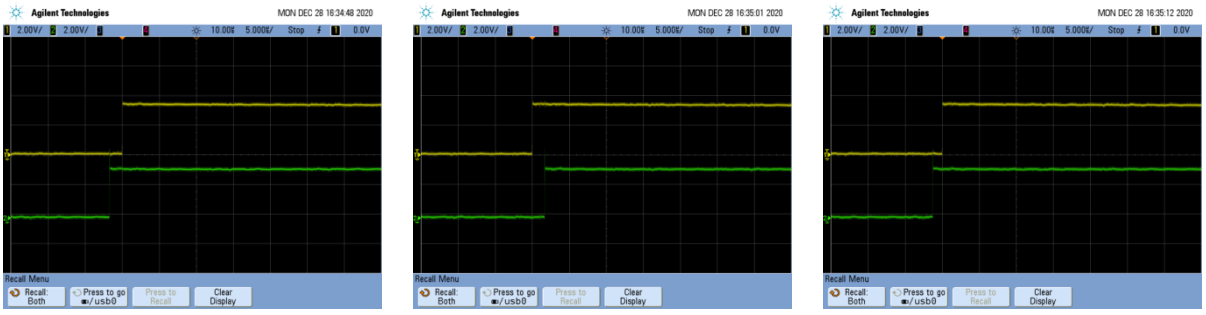


Fig. 5.11 Digital outputs from 2 different nodes, at 3 different time instants

It was observed that the global time was synchronized among the nodes. The clock synchronization algorithm managed to converge time of each node well within 10 µs which was the design tolerance for the algorithm. Refer Figure 5.11 which depicts synchronization at 3 different instants after resetting of nodes.

ii. Validation of membership algorithm

For validation of the membership algorithm, each node was powered off one by one and the broadcast packet transmission was observed. Following Table 5.2 denotes the observations:

Table 5.2 : Membership vectors over different fault conditions

Fault conditions			Expected MV (hex)	Observed MV (hex)
Node 1	**Node 2**	**Node 3**		
Ok	Ok	Ok	0B 00	0B 00
Faulty	Ok	Ok	0A 00	0A 00
Ok	Faulty	Ok	09 00	09 00
Ok	Ok	Faulty	03 00	03 00

It was observed that powering off of a node (which is the same as incorporating a transmit fault as well as receive fault), was successfully acknowledged by other nodes in the system. The membership vector accurately represented all and only the active nodes in the network with only implicit acknowledgement from other nodes.

CHAPTER 6: Conceptual design of sensor network

6.1 Sensor network system overview

Sensor network will consist of sensor nodes which will transmit data on DFTCP network. Each independent network of sensor nodes is named as Local Cluster (LC). Each LC is a network of sensors over redundant RS485 channels.

All sensor networks will send its measurement data one at a time over DFTCP. A smart sensor network is designed to consist maximum 10 local clusters. The LC will provide redundant RS485 interface to connect with Senor Network's Hub (SNH). Maximum RS485 segment length will be 24 meters. For distance more than 24 m, repeater will be used. Refer Figure 6.1.

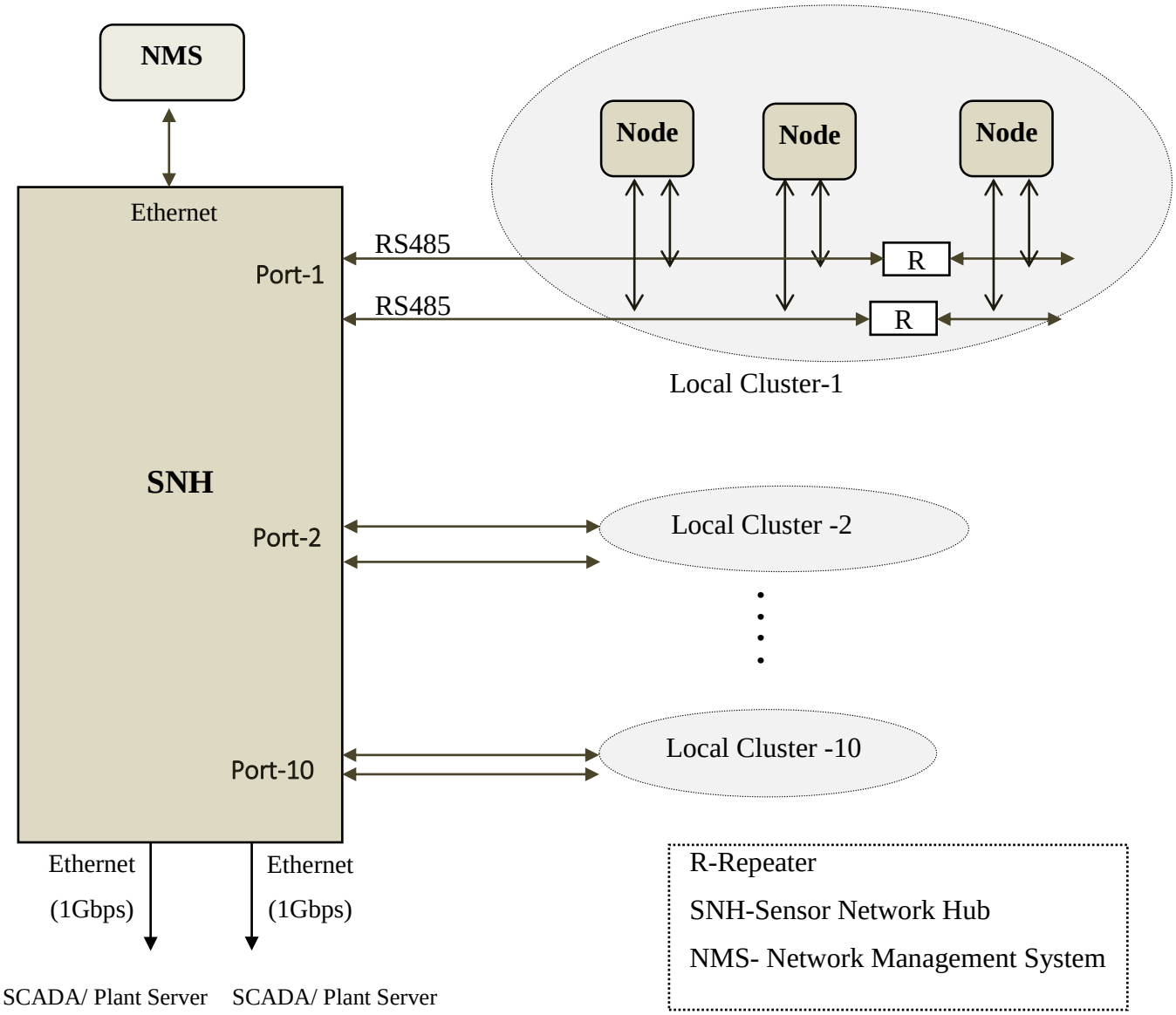


Fig. 6.1 Block Diagram of local cluster

i. Sensor Node (SN)

A sensor node will have sensor input, signal conditioning circuitry and redundant communication controller. Each sensor node perform data acquisition and transmits field value and diagnostics over the network.

Each controller will have independent bus guardian and RS485 channels. Signal from sensor input will be digitized and will be communicated on RS485 channel though DFTCP controller.

DFTCP parameters will be available in a schedule table. Schedule Table will be encrypted inside a non-volatile memory of each node. When powered on, it will check the integrity of schedule table. Refer Figure 6.2.

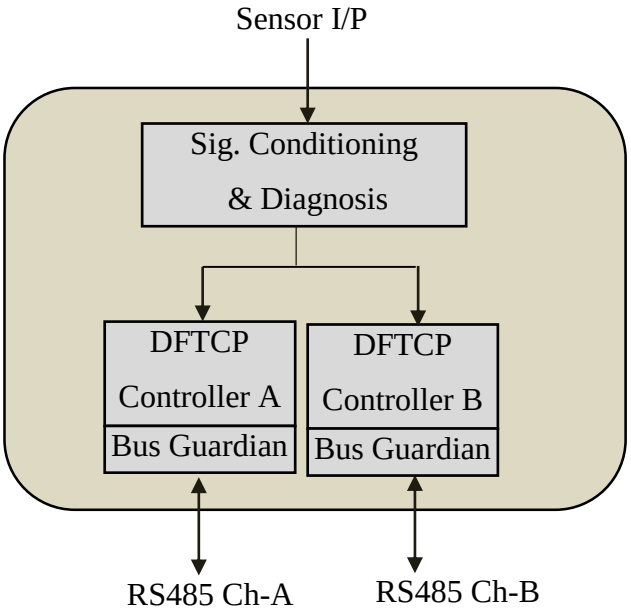


Fig. 6.2 Block Diagram of sensor node

ii. Sensor Network's Hub (SNH)

This device will allow integration of multiple local clusters. Maximum of 10 LC can be connected to SN Hub. It will have a separate RS485 port for each LC. Each LC will operate with an independent DFTCP schedule. SN Hub will act as a monitoring node for respective LC connected on that port. So, it will gather data in real time and will be synchronized with each LC separately. It will forward all acquired data on redundant 1Gbps Ethernet Network. SN hub will be rack mountable in a 19" width rack/panel which is widely used in nuclear plants and facilities. Refer Figure 6.3.

NMS which for Network Management Software, will be a PC based software. It will have functionalities such as: planning, analyzing data for decoding of time-stamped data packets,

maintain statistics and performance monitoring. NMS shall run on a Windows or Linux PC and provide operator console for user interface. This shall provide the facility to design / configure local cluster parameters.

From security perspective, following points shall be taken care of:

a. Schedule table as generated will be encrypted.

b. Downloading of encrypted file to each sensor node will be done after sensor node authentication.

c. Read back of firmware of configurable devices will not be possible.

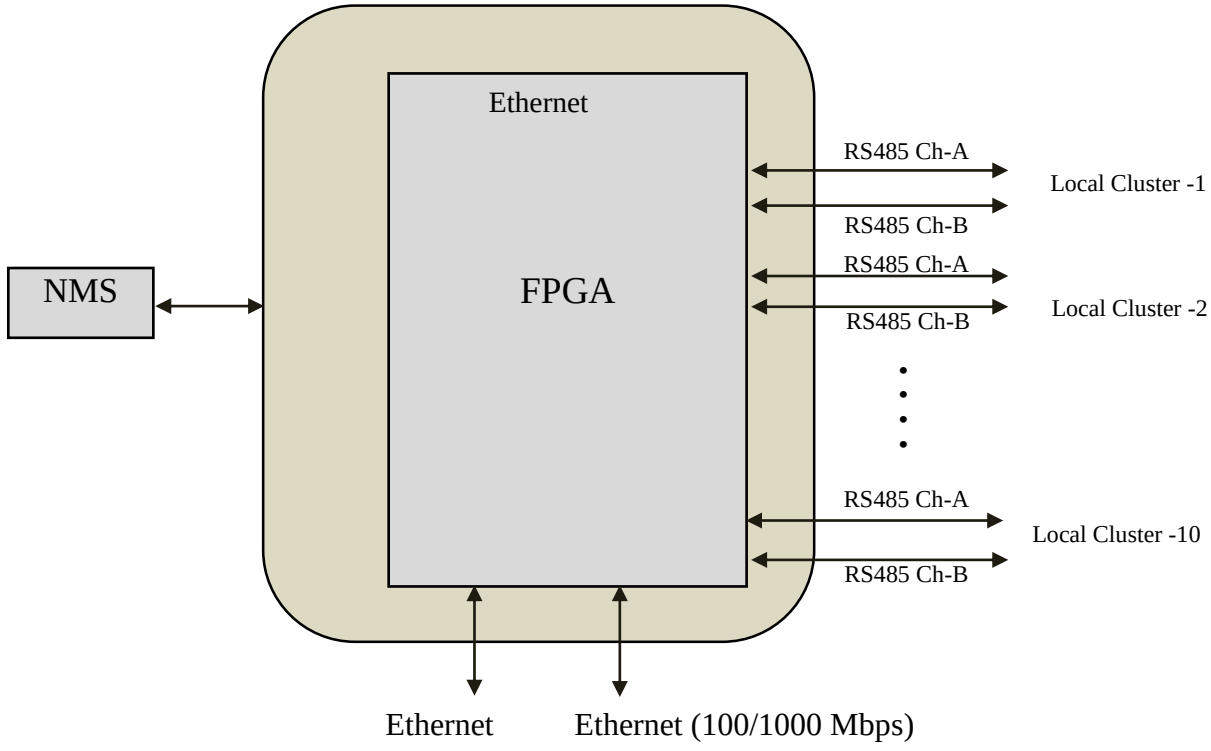


Fig. 6.3 Block Diagram of sensor network's hub

6.2 Performance

1) Each sensor node will communicate over DFTCP and will be time synchronized within 50 µsec (microseconds). Communication controller will be designed using FPGA/processor based on this requirement.
2) Sensor Node RS485 baud rate: 8 Mbps.

Scalability:

Considering the synchronization within 50 µsec and 6 byte data and 2 byte of CRC-16:

Let N be the no. of nodes that are allowed in the network

Packet size: = 6 Bytes (Data) + 2 Bytes (CRC) + (N / 8) Bytes

= 8 + N / 8 Bytes

= 80 + 10N / 8 bits [considering 1 start bit, 1 stop bit and 8 data bits]

Transmission time = (80 + 10N / 8) bits * 1 / (8 µsec)

= 10 + 10N / 64

N * [Transmission Time + 50 µsec] < Cycle time

N * [10N / 64 + 50 + 10] µsec < CT

N * [10N / 64 + 60] µsec < CT

$10N^2 / 64 + 60N$ µsec < CT

Solving this quadratic equation for various values of 'Cycle Time', we get the values of No. of nodes N as shown in the following Table 6.1:

Table 86.1: No. of nodes as per cycle time

Cycle time (in millisec)	N	Total no. of nodes the network (10 Local Cluster)
1	16	160
2	31	310
3	70	700

So, if the cycle time of 1 millisecond is considered, Maximum number of nodes per LC will be: 160.

6.3 Timing Analysis for Data Communication

The data communication in DFTCP will be deterministic. This is because the worst case latency is very less, constant and computable. The worst case latency will be mathematically formulated in the timing analysis carried out further.

For any data communication, the time delay of a message, T_{delay}, is defined as the difference between the time when the source node wants to send a message, T_{src}, and the time when the destination node receives this message, T_{dest}, i.e.,

$\mathbf{T_{delay}} = T_{dest} - T_{src}$.

The total time delay can be broken into three parts, representing the time delays:

1. At the source node:

   The time delay at the source node includes the preprocessing time (T_{pre}), and the waiting time (T_{wait}). Preprocessing time (T_{pre}) is the sum of the computation time (T_{scomp}), the encoding time (T_{scode}). The waiting time (T_w) is the is the time a message must wait once a node is ready to send it. Depending on the amount of data the source node must send and the traffic on the network, the waiting time may be significant.

2. On the network channel:

   The network time delay includes the total transmission time of a message and the propagation delay of the network. This will depend on message size, data rate, and the length of the network cable.

3. At the destination node.

   The time delay at the destination node only includes the post-processing time (T_{post}), which is the sum of the decoding time (T_{dcode}), and the computation time (T_{dcomp}), at the destination node.

The time delay can be explicitly expressed by the following equations:

$\mathbf{T_{delay}} = T_{dest} - T_{src}$.

$= T_{pre} + T_{wait} + T_{tx} + T_{post}$

$= (T_{scomp} + T_{scode}) + T_{wait} + (T_{frame} + T_{prop}) + (T_{dcode} + T_{dcomp})$

For DFTCP, ($T_{scomp} + T_{scode}$) is included in the Pre-Send Interval (T_{PSI}) of the slot timing & (T_{dcode} + T_{dcomp}) is included in the Post Receive Interval(T_{PRI}) of the slot timing. The worst case delay can be approximated as:

$\mathbf{T_{delay}} = T_{PSI} + T_{wait} + T_{frame} + T_{prop} + T_{PRI}$

Waiting time:

The waiting time (T_{wait}), which is the time a message must wait once a node is ready to send it, depends on the network protocol and is a major factor in the determinism and the performance of a control network. In a real-time communication system, the data should be processed immediately for transmission or reception. It should not use FIFO for storing the data frames. Queuing will not be there in the DFTCP controller. Hence, there is no queuing time, only the blocking time. Irrespective of the slot time, each node will be allowed to transmit only at the start of its transmit interval. The host will provide the data for transmission at a fixed instant of time, before the slot for that controller begins, once in each TDMA round. This instant of time will be marked by the interrupt given by the controller. Hence the overall blocking time will be fixed and constant.

Let the time for which the data waits before transmission be represented by T_b. The receiver will have a slightly different perception of the global time bounded by the precision π. So the receiver may get the frame delayed by π more.

T_{wait} (min.) = $T_b - \pi$

T_{wait} (max) = $T_b + \pi$

Frame Time

The frame time, T_{frame}, will depend on the size of the data and the overhead. There will not be any padding or stuffing. Let N_{data} be the size of data in terms of bytes & N_{ovhd} be the number of bytes used as overhead. The frame time can then be expressed as:

$T_{frame} = [N_{data} + N_{ovhd}] * 8 * T_{bit}$

T_{frame} shall be used to estimate the slot time during the communication system design.

Propagation time:

Propagation time for the electrical signal for 100m is 0.5μs (Considering that the speed of propagation in a transmission line as $2/3^{rd}$ that of speed of light). The proposed, DFTCP prototype system will have the maximum length of the communication channel as 24m. For a maximum distance of 24m from source to node, the propagation time will not exceed 0.12μs.

Hence, the worst case delay was computed to be:

$\mathbf{T_{delay}} = T_{PSI} + T_{wait} + T_{frame} + T_{prop} + T_{PRI}$

Considering the above analysis, the delay is constant given by:

$\mathbf{T_{delay}} = T_b + [N_{data} + N_{ovhd}] *8* T_{bit} + T_{prop} + T_{PSI} + T_{PRI} \pm \pi$

The major factor determining the total delay will be the waiting time (which consists of the blocking time). The values of T_{PSI} & T_{PRI} cannot be calculated mathematically. It will depend on the controller processing speed & the amount of protocol services and the complexity of the algorithms to be executed.

CHAPTER 7: Conclusion and future work

7.1 Concluding remarks

A time triggered communication provides deterministic, fault tolerant and fail silent communication. This is best suitable for C&I system in which control data is communicated over shared medium. Clock synchronization and membership algorithm are necessary in time triggered communication for proper operation and to achieve determinism and fault tolerance.

A conceptual design of sensor network was formulated and the specifications of the deterministic fault tolerant time triggered protocol was designed. It will provide robust and fault tolerant communication for distributed sensors in a nuclear plant environment and also result in substantial savings in cabling and penetrations

The clock synchronization algorithm used in DFTCP was verified using simulation in Scilab and was validated by actual hardware. It was observed that the time values of distributed nodes communicating in a time triggered architecture converges to a single value. The correctness, convergence and fault tolerance property of the FTA algorithm was verified by the simulation results. Fault was injected and it was observed that time offset of all nodes converge and satisfy single failure criteria. It was also shown that the convergence was possible between two nodes by applying a fraction of the correction term instead of applying the complete correction term.

A majority voting membership algorithm has been formulated to get an implicit acknowledgment and a consistent view of status of all nodes of the network. Simulation of membership algorithm was carried out in Python. Faults were injected and it was observed that the algorithm detect transmit fault, receive fault and is capable in clique avoidance.

These algorithms were implemented in FPGA and a test setup is used to validate the correctness of these two algorithms. Time synchronization of the nodes has been observed with the help of an oscilloscope in which jitter within 10 microseconds was observed. Information regarding the faulty nodes is depicted in the membership vector broadcasted by each node. It was observed that faulty nodes were successfully acknowledged by other nodes in the system. Thus both: clock synchronization algorithm and the group membership algorithm were validated on hardware.

7.2 Future work

- Simulation and testing of integration and startup algorithm for time triggered communication and implementation.
- Development of prototype sensor node and network with time triggered communication. Integrated testing of the same.
- Development of time triggered communication over Ethernet.

References

[1] Amos Albert, “Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems”, *Embedded World Nurnberg, 17.*– pages 235–252, 2004.

[2] Real, Jorge, Sergio Sáez, and Alfons Crespo. "A hierarchical architecture for time-and event-triggered real-time systems" *Journal of Systems Architecture 101: 101652.*, 2019.

[3] Ataíde, Fernando H., Fabiano C. Carvalho, Carlos E. Pereira, and Marco A. Wehrmeister. "A comparative study of embedded protocols for safety-critical control applications." *IFAC Proceedings Volumes 39, no. 3, pages 87-94*, 2006.

[4] Kopetz, Hermann, and Günther Bauer. "The time-triggered architecture." *Proceedings of the IEEE 91.1 : 112-126*, January 2003.

[5] FlexRay “FlexRay Communications System Protocol Specification Version 3.0.1” *FlexRay Consortium*, October 2010.

[6] B. Abdul Rahim1, S.Krishnaveni , “Comparison of CAN, TTP and Flexray Communication Protocols”, *International Journal of Innovative Research in Computer and Communication Engineering Vol.2, Special Issue 4*, September 2014.

[7] Rushby, J., “A comparison of bus architectures for safety critical embedded systems”, *Technical report. SRI International. Menlo Park California*, 2001.

[8] Kopetz, H. and G. Grunsteidl, “TTP - A time-triggered protocol for fault-tolerant real-time systems”, In *IS Fault-Tolerant Computing (23th FTCS). IEEE Press. Toulouse, France* 1993.

[9] ISO 11898-1:2015 Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling

[10] Fuehrer, T., Mueller, B., Hartwich, F., and Hugel, R., "Time Triggered CAN (TTCAN)," *SAE Technical Paper* 2001-01-0073

[11] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, “TTEthernet: Time- Triggered Ethernet,” in *Time-Triggered Communication, R. Obermaisser, Ed. CRC Press*, Aug 2011.

[12] Veríssimo, P., Rodrigues, L. & Casimiro, A. CesiumSpray: “A Precise and Accurate Global Time Service for Large-scale Systems”, *Real-Time Systems 12, 243–294*, 1997.

[13] J. Rushby and F. von Henke. “Formal verification of the interactive convergence clock synchronization algorithm”. *Technical Report CSL-89-3R, Computer Science Laboratory, SRI International, CA, Menlo Park, USA*, February 1989.

[14] A. Schedl, “Design and Simulation of Clock Synchronization in Distributed Systems”.

Doctoral thesis, Institut fu¨r Technische Informatik, Technische Universita¨t Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, April 1996.

[15] S. Dolev. "Possible and impossible self-stabilizing digital clock synchronization in general graphs". *Real-Time Systems, 12(1):95–107*, January 1997.

[16] S. Dolev and J.L. Welch. "Self-stabilizing clock synchronization with Byzantine faults. In Proceedings of the 14th ACM Symposium on Principles of Distributed Computing", *page 256. ACM Press*, 1995.

[17] M. Papatriantafilou and P. Tsigas. "Self-stabilizing wait-free clock synchronization". In *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory, volume 824 of Lecture Notes in Computer Science, pages 267–277. Springer-Verlag Berlin Heidelberg, Germany*, July 1994.

[18] M. Lu, D. Zhang, and T. Murata, "Analysis of self-stabilizing clock synchronization by means of stochastic Petri nets", *IEEE Transactions on Computers, 39(5):597–604*, 1990.

[19] G. Ciardo and C. Lindemann. "Comments on analysis of self-stabilizing clock synchronization by means of stochastic Petri nets." *IEEE Transactions on Computers, 43(12):1453–1456*, 1994.

[20] IEEE Standard "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". *IEEE Press, New York, NY, USA, IEEE Standard No. 1588*, February 2009.

[21] E. Anceaume and I. Puaut, "Performance evaluation of clock synchronization algorithms" *Research Report 3526, Institut National de Recherche en Informatique et Syst`emes Al´eatoires (IRISA), Rennes, France*, October 1998.

[22] E. Anceaume and I. Puaut. "A taxonomy of clock synchronization algorithms" *Research Report 1103, Institut National de Recherche en Informatique et Syst`emes Al´eatoires (IRISA), Rennes, France*, July 1997.

[23] J. Welch and L. Lynch, "A new fault-tolerant algorithm for clock synchronization" *Information and Computation (formerly Information and Control), 77(1):1–36*, 1988.

[24] C. Fetzer and F. Cristian, "An optimal internal clock synchronization algorithm", In *Proceedings of the 10th Conference on Computer Assurance, pages 187– 196, Gaithersburg, MD, USA, IEEE*, June 1995.

[25] C. Fetzer and F. Cristian, "Integrating external and internal clock synchronization", *Real-Time Systems, 12:123–171*, March 1997.

[26] M. Pfluegl and D. Blough, "A new and improved algorithm for fault-tolerant clock synchronization", *Journal of Parallel and Distributed Computing, 27:1– 14*, 1995.

[27] K. Marzullo and S. Owicki, “Maintaining the time in a distributed system”, In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, pages 295–305*, 1983.

[28] F. Cristian, H. Aghili, and R. Strong, “Clock synchronization in the presence of omission and performance failures, and processor joins”, *In Proc. of 16th Int. Symp. on Fault-Tolerant Computing Systems*, July 1996.

[29] F. Cristian and C. Fetzer, “Fault-tolerant external clock synchronization”, In *Proceedings of the 15th International Conference on Distributed Computing Systems, pages 70–77, Los Alamitos, CA, USA, IEEE,* May 30–June 2 1995.

[30] J.M. Krause, M.J. Englehart, and D.A Shaner, “Achievable performance of fault tolerant avionics clocks”, In *AIAA Computing in Aerospace Conference, 8th, Technical Papers. Vol. 2 (A92-17576 05-61), pages p. 608–622, Baltimore, MD, American Institute of Aeronautics and Astronautics*, Oct. 21-24 1991.

[31] L. Lamport, R. Shostak, & M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems, Vol.4, No.3, pp. 382-401*, July 1982.

[32] W. R. Moore, N.A. Haynes, "A review of synchronisation and matching in fault-tolerant systems", *IEEE Proceedings, Vol. 131, Pt.E, No.4, pp. 119-124*, July 1984.

[33] Peter Puschner, Raimund Kirner, “Interfacing to Time-Triggered Communication Systems”, *IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC),* 2019.

[34] Li, Q., & Rus, D., “Global clock synchronization in sensor networks”, *IEEE Transactions on computers, 55(2), 214-226*, 2006.

[35] John Rushby, “An Overview of Formal Verification for the Time-Triggered Architecture”, *7th International Symposium, FTRTFT 2002 Co-sponsored by IFIP WG 2.2 Oldenburg, Germany*, September 9-12 2002.

[36] Shmuel Katz, Pat Lincoln, and John Rushby, “Low-overhead time-triggered group Membership”, In *Marios Mavronicolas and Philippas Tsigas, editors, 11th International Workshop on Distributed Algorithms (WDAG ’97), volume 1320 of Lecture Notes in Computer Science, pages 155–169, Saarbr¨ucken Germany, Springer-Verlag.*, September 1997.