ISSN 1612-6793
Nr. ZFI-BM-2016-19

Master's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Monitoring Mechanism Design for a Distributed Reconfigurable Onboard Computer

Tanyalak Vatinvises

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

15. August 2016

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Ramin Yahyapour, Georg-August-Universität Göttingen
Second Supervisor: Dr. rer.nat. Andreas Gerndt, German Aerospace Center (DLR)

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 15. August 2016

Abstract

The Onboard Computer - Next Generation (OBC-NG) project is initiated by the German Aerospace Center (DLR) to develop a distributed and reconfigurable onboard computer of Commercial off-the-shelf (COTS) components. Its goal is to utilize the higher computational power of COTS components and to maintain high system reliability, which is required for spacecraft. However, COTS have a lower robustness than the traditional space-qualified components and therefore a higher probability of failures. Thus, a monitoring system is indispensable. This thesis focuses on the design of a monitoring system, to evaluate the health status of the distributed components. It shall detect failures before any corrective action, e.g. migrating the tasks to the other functioning components, can be triggered.

Different monitoring techniques have different trade-offs, in terms of monitoring efficiency and monitoring overhead. Initially, various monitoring concepts are investigated, in order to qualitatively analyze their trade-offs and designs. Three monitoring mechanisms, PULL, PUSH and PUSH-PULL, are selected and modeled. Afterwards, the models are simulated on the discrete event simulator OMNeT++ and tested under different environment as well as monitoring mechanism settings. The simulation results, which represent a quantitative investigation of the designed monitoring mechanisms, are used to evaluate and compare them. The derived verdicts are used to identify the most suitable mechanism and settings for the OBC-NG system. The results show that PUSH mechanism is more suitable for OBC-NG system than the currently implemented PULL mechanism.

Contents

1 Introduction **1**
1.1 Motivation and Problem Statement . . . 1
1.2 Purpose and Goals . . . 2
1.3 Task . . . 3
1.3.1 Research Phase . . . 3
1.3.2 Design Phase . . . 3
1.3.3 Simulation and Implementation Phase . . . 3
1.3.4 Evaluation Phase . . . 3
1.4 Chapter Overview . . . 3

2 Background **5**
2.1 Onboard Computer Next Generation: OBC-NG . . . 5
2.1.1 System Architecture . . . 5
2.1.2 Middleware . . . 7
2.1.3 SpaceWire . . . 10
2.2 Distributed System Design . . . 11
2.2.1 Reliability . . . 11
2.2.2 Availability . . . 12
2.2.3 Redundancy . . . 13
2.2.4 Threats . . . 13
2.2.5 Design Validation: Fault Injection . . . 15
2.3 Monitoring . . . 15
2.3.1 Monitoring Mechanism Classifications . . . 15
2.3.2 Fundamental Problems of Distributed System Monitoring . . . 17
2.4 Related Work . . . 18
2.4.1 Traditional Heartbeat Monitoring . . . 18
2.4.2 Hybrid Heartbeat Monitoring . . . 19
2.4.3 Dynamic Heartbeat . . . 20

3 Design of OBC-NG Monitoring **23**
3.1 OBC-NG Monitoring Requirements 23
3.1.1 Reliability and Redundancy in Monitoring System 23
3.1.2 Availability and MTTR 24
3.1.3 Avoiding Incorrect Judgement 24
3.1.4 Reducing Monitoring Cost 25
3.2 Monitoring Mechanisms Concepts 25
3.2.1 Current Monitoring Mechanism: PULL Mechanism 25
3.2.2 Alternative 1: PUSH Mechanism 26
3.2.3 Alternative 2: PUSH-PULL Hybrid Mechanism 27
3.3 Monitoring Mechanism Settings 27
3.3.1 Number of Observers 27
3.3.2 Heartbeat Interval 27
3.3.3 Resend Mechanism 28

4 Simulation Specification and Implementation **31**
4.1 Simulation Tool: OMNeT++ 31
4.1.1 OMNeT++ Components Specifications 32
4.1.2 OMNeT++ User Interfaces 32
4.1.3 Output Formats and Types 32
4.2 Simulation Objectives 33
4.3 Performance Indicators 33
4.3.1 Monitoring Overhead 33
4.3.2 Fault Response Time 33
4.3.3 Monitoring System Stability 33
4.4 OBC-NG System Simulation 34
4.4.1 Manager Submodule 35
4.4.2 Application Submodule 36
4.4.3 Routing Submodule 37
4.4.4 Queue Submodule 37
4.4.5 Packets 38
4.5 Monitoring Mechanisms Simulation 38
4.5.1 PULL Model Simulation 38
4.5.2 PUSH Model Simulation 40
4.5.3 PUSH-PULL Hybrid Model Simulation 41
4.6 Monitoring Mechanism Settings Simulation 42
4.6.1 Number of Observers 42
4.6.2 Mechanism Setting: Static Heartbeat Interval 42
4.6.3 Mechanism Setting: Dynamic Heartbeat Interval 43

4.7 Environment Settings Simulation 43
4.7.1 Fault Injection: Node Failure 43
4.7.2 Network Traffic Simulation 44
4.8 Simulation Execution 44
4.8.1 Input Specifications 45
4.8.2 Automated Tests 45
4.9 Test Suites 45

5 Simulation Results and Evaluation 51
5.1 Test Results 51
5.2 Result Evaluation 67
5.2.1 Monitoring Overhead 67
5.2.2 Fault Response Time 67
5.2.3 Monitoring System Stability 68

6 Conclusion and Future Work 71
6.1 Final Results 71
6.2 Future Work 72

Bibliography 77

A Appendix Table 79
A.1 Test Suite 5 Results 79

List of Figures

2.1 An example of OBC-NG system [2] 6
2.2 Monitoring service of different types of nodes 6
2.3 Basic software architecture of OBC-NG [2] 7
2.4 Structure of the OBC-NG middleware [2] 8
2.5 An example of a decision graph [2] 9
2.6 Structure of OBC-NG network layer [2] 9
2.7 SpaceWire packet format [3] 11
2.8 Relationship between MTTF, MTTR, and MTBF [4] 12
2.9 Failure, Error and Fault [4] 14
2.10 The Bathtub failure rate curve [5] 14
2.11 The processes of basic monitoring [6] 15
2.12 Heartbeat flow in Microsoft operation manager [7] 19
2.13 Heartbeat self-test and mutual detection [8] 21

3.1 MTTR components in OBC-NG 24
3.2 Heartbeat flows in PULL mechanism 26
3.3 Heartbeat flows in PUSH mechanism 26
3.4 Resend mechanism with resend timeout and resend threshold 29

4.1 OBC-NG system simulation on OMNeT++ 34
4.2 Structure of OBC-NG node on OMNeT++ 35
4.3 Channel settings in NED file 35
4.4 Routing and connections 37
4.5 The structure of an OBC-NG packet and a simulated packet 38
4.6 Monitoring design with PULL mechanism 39
4.7 Monitoring design with PUSH mechanism 41
4.8 Monitoring design with PUSH-PULL hybrid mechanism 42
4.9 Application packet flows 44

5.1 Number of monitoring messages sent per node in different monitoring mechanisms and settings . . . 52
5.2 Fault response time of PULL and PUSH mechanisms when different nodes fail . . 53
5.3 An example of random fault injection time of 2000 simulation runs . . . 55
5.4 Fault response time and number of occurrences of PULL mechanism . . . 56
5.5 Fault response time and number of occurrences of PUSH mechanism . . . 56
5.6 Fault response time and number of occurrences of PUSH-PULL mechanism . . . 57
5.7 Application packet size and end simulation time of PULL mechanism with 100 ms HB interval . . . 59
5.8 Application packet size and end simulation time of PULL mechanism with packet factor of 80 bytes . . . 60
5.9 Application packet size and end simulation time of PUSH mechanism with packet factor of 80 bytes . . . 60
5.10 Application packet size and end simulation time of PUSH-PULL hybrid mechanism with packet factor of 80 bytes . . . 61
5.11 Application packet size and end simulation time of PULL mechanism . . . 62
5.12 Application packet size and end simulation time of PUSH mechanism . . . 62
5.13 End simulation time of PUSH mechanism with the application packet size over 1.05 MB . . . 63
5.14 Application packet size and end simulation time of PUSH-PULL hybrid mechanism 64

List of Tables

1.1 Comparison of the commercial, space and avionics domains [1] 2

2.1 Transmission message types and sizes 10

4.1 Test suite 1 General configurations 46
4.2 Test suite 1 Test case settings 46
4.3 Test suite 2 General configurations 47
4.4 Test suite 3 General configurations 48
4.5 Test suite 3 Resend mechanism configurations 48
4.6 Test suite 4 General configurations 49
4.7 Test suite 4 Test case settings 4.1 49
4.8 Test suite 4 Test case settings 4.2 49
4.9 Test suite 5 General configurations 50

5.1 Network overhead of PULL and PUSH mechanism with different Observer role settings 52
5.2 Fault response time of PULL mechanism when M fails at 1000 ms 54
5.3 Fault response time of PUSH mechanism when M fails at 1000 ms 54
5.4 Fault response time of a random failure in different mechanisms and HB intervals (ms) 57
5.5 Approximate simulation time when the packet of size reaches 1.05 MB 63
5.6 Average end simulation time of different mechanisms with different HB intervals . 64
5.7 Average end simulation time of PULL and PUSH mechanisms with different HB intervals 65
5.8 End simulation time of PUSH mechanism with packet factor of 80 66
5.9 Percentage of the tests with different packet factors that run over the threshold . . 66

List of Abbreviations

OBC-NG	Onboard Computer - Next Generation . . . 2
DLR	German Aerospace Center . . . 2
COTS	Commercial off-the-shelf . . . 2
OBC	Onboard Computer . . . 3
OBSW	Onboard Software . . . 3
PN	Processing Node . . . 5
IN	Interface Node . . . 5
FPGA	Field-programmable gate array . . . 5
M	Master . . . 6
W	Worker . . . 6
O	Observer . . . 6
API	Application Program Interface . . . 7
OS	Operating System . . . 7
RODOS	Realtime Onboard Dependable Operating System . . . 7
HB	Heartbeat . . . 8
ACK	Acknowledgement . . . 8
MetOp	Meteorological Operational Satellite . . . 10
ESA	The European Space Agency . . . 10
SWIFT	Swift Gamma Ray Burst Explorer . . . 10
JWST	James Webb Space Telescope . . . 10
NASA	The National Aeronautics and Space Administration . . . 10
NeXT	New X-ray Telescope . . . 10
JAXA	The Japan Aerospace Exploration Agency . . . 10
EOP	End-of-Packet . . . 11
MTTR	Mean Time To Repair . . . 12
MTTF	Mean Time To Failure . . . 12
MTBF	Mean Time Between Failures . . . 12
IDE	Integrated Development Environment . . . 45
ScOSA	Scalable on-board Computing for Space Avionics . . . 72

Chapter 1

Introduction

The goal of this chapter is to provide an introduction to the topic, and to discuss the motivations as well as problem statement of this work. The purpose and goals are explained, along with the task details. The chapter is finalized with the chapter overview.

1.1 Motivation and Problem Statement

As human explore deeper into the space, the complexity of the spacecraft design increases dramatically. The size and the amount of collected data grow because better sensors with higher resolution are used and mission periods are getting longer. However, deep space missions have the limitation of low telemetry bandwidths and long propagation delays between spacecraft system and ground control [9]. The bandwidth limits the amount of data to be transmitted and therefore the preprocessing of data before transmitting is required. Long command delay, which is caused by the communication latency, is another issue to be concerned of because todays' space system is becoming more autonomous and requires faster reaction time [1]. *Rosetta* spacecraft and its robotic comet lander, *Philae*, is a good example for this requirement, since it requires onboard processing of optical data for navigation as well as for reducing and filtering images before transferring to ground [10].

Table 1.1: Comparison of the commercial, space and avionics domains [1]

Operational Environment	**Commercial**	**Space**	**Avionics**
Mission duration	Years	Years	Hours
Maintenance intervention	Manual	Remote	After mission
Outage response time	Hours	Days (Cruise phase)	Milliseconds
Resources			
- Power	Unlimited	Minimal	Medium
- Spare parts	Unlimited	None	After mission

As shown in table 1.1, the operation environment of space domain makes the system maintenance harder, compared to other domain. Spacecraft are usually designed with high focus on reliability to provide the operation under harsh environment for long mission period as months or years with minimum maintenance [1]. In order to provide reliability, fault avoidance and fault tolerance techniques are deployed. Radiation-hardened components are one of the solutions to improve the level of fault avoidance. However, those components are expensive and have limited processing power. More importantly, faults still occur. In addition, they require one-to-one mapping for warm and cold redundancy. In contrast, Commercial off-the-shelf (COTS) components, which have lower costs, are used to provide higher processing power.

The German Aerospace Center (DLR) has initiated the Onboard Computer - Next Generation (OBC-NG) project to design a distributed and reconfigurable system by utilizing COTS along with space-qualified components [2]. Unfortunately, current COTS standards lack some of the dependability guarantee because it is not as robust as the space-qualified components [1]. Their lower level of robustness comes with a higher probability of component failures. Therefore, OBC-NG system requires an efficient monitoring system to monitor the state of the overall system. In case a component failure is detected, a reconfiguration is triggered.

1.2 Purpose and Goals

The main purpose of this work is to investigate the monitoring concepts, design and optimize monitoring mechanisms for the OBC-NG systems. The efficient monitoring mechanisms can help decreasing the duration from failure occurrence to failure detection time. The faster the failure is detected, the faster the system can react. The failure detection time is highly dependent on the frequency of monitoring. However, the higher monitoring frequency, the more overhead is created. For this reason, the combination of the mechanisms and their settings should be chosen wisely to reduce the monitoring overhead, and maintain the system reliability and availability.

1.3 Task

The task is divided into four phases: Research, Design, Simulation and Implementation, as well as Evaluation Phase.

1.3.1 Research Phase

In research phase, Onboard Computer (OBC), Onboard Software (OBSW) and spacecraft system fundamentals are investigated. The requirements of the OBC-NG system are analyzed. Various monitoring concepts and mechanisms, which are used in different applications of distributed systems, are collected to analyze their advantages and disadvantages as a qualitative trade-off analysis.

1.3.2 Design Phase

Base on the gained knowledge from the previous phase. The monitoring concepts are chosen and modeled along with their settings. In addition, the environment settings are defined in order to test the model in the next phase.

1.3.3 Simulation and Implementation Phase

In this phase, the OBC-NG system is simulated on a discrete event simulator: *OMNeT++*. The monitoring models are implemented and simulated to evaluate their efficiency and measure their communication overhead. The models are also tested under different scenarios to compare their monitoring efficiency.

1.3.4 Evaluation Phase

The test results from the previous phase are evaluated according to the specified criteria, such as monitoring overhead and fault response time. At the end of this phase, the suitable monitoring mechanisms and their settings, e.g. monitoring frequency and number of observers needed, are determined in the simulation results analysis.

1.4 Chapter Overview

The structure of this report is as follows: After a brief introduction in this chapter, chapter 2 provides more detailed information of distributed systems and spacecraft design and gives an

overview of monitoring. In addition, various monitoring mechanisms are investigated in the related work section. Chapter 3 presents the monitoring requirements and the design concepts. The details about simulation and implementation are explained in chapter 4. Chapter 5 presents the simulation results and their evaluation. Finally, chapter 6 concludes the work and provides a future outlook.

Chapter 2

Background

This chapter contains the basic information for the following chapters. It introduces the concepts, structure and functionality of the OBC-NG system in section 2.1. In section 2.2, the fundamentals of the distributed systems design are explained. Afterwards, the basic concepts of monitoring are described in section 2.3. Finally, the monitoring approaches are investigated at the end of the chapter in the related work, section 2.4.

2.1 Onboard Computer Next Generation: OBC-NG

This section begins with the overview of the OBC-NG system architecture and software architecture with the focus on the Middleware layer and its functionality and finally emphasizes its monitoring and monitoring-related functionality.

2.1.1 System Architecture

OBC-NG Nodes

Figure 2.1 shows an example of OBC-NG system architecture with different types of nodes connected with point-to-point links. The first type, Processing Node (PN), provides the computing resource for processing data and managing the system. The hardware of a PN consists of Main Processing Unit, router and optional coprocessor e.g. Field-programmable gate array (FPGA). The second type is Interface Node (IN). Its hardware components consist of microcontroller, router, interfaces to periphery and mass storage. It is used to connect the PNs with the peripheries. If it connects to mass-memory, it has the role of *Storage* and if it connects to sensors or actuators, it has the role of *Interface* [11].

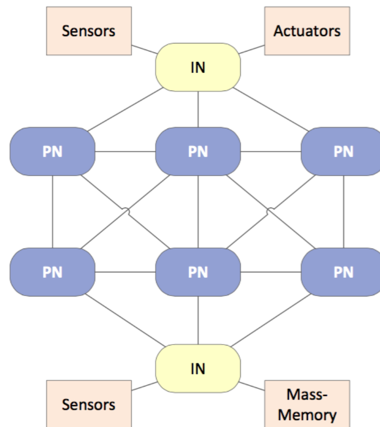


Figure 2.1: An example of OBC-NG system [2]

Peng et al. explained the roles of the PNs as following. The roles of the Master (M) are controlling and monitoring the other nodes and distributing the tasks. Observers (Os) have the role of monitoring the Master and higher priority Observer(s) as shown in figure 2.2. The PNs, which have no management functions of monitoring and managing the tasks, are Workers (Ws). They perform the data processing tasks assigned by the Master [11]. The roles can be assigned and re-assigned to the chosen nodes during system runtime because of the reconfigurable characteristic of OBC-NG nodes. Moreover, more than one role can be assigned to a single node. For example, the Master can also be assigned to perform other tasks along with the management roles. However, two management roles, i.e. Master and Observer, or two Observers, cannot be assigned on the same node because if the node fails, the management roles will not be able to detect each other's failure.

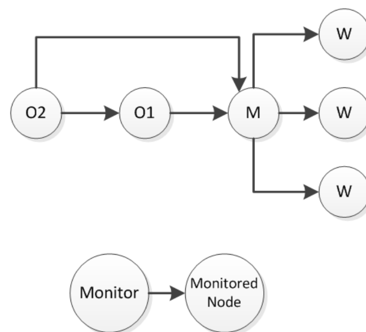


Figure 2.2: Monitoring service of different types of nodes

Software Architecture

Figure 2.3 shows the OBC-NG software architecture, which is the three layers on top of hardware layer. Our focus is on the *Middleware layer*, which is between the *Application layer* and the *Operating System layer.* It offers an Application Program Interface (API) for developing applications and management, monitoring and reconfiguration of application tasks [2]. For the Operating System (OS), DLR has chosen two OS for OBC-NG project for different purposes; Linux for complex applications and Realtime Onboard Dependable Operating System (RODOS) for time-critical applications [2].

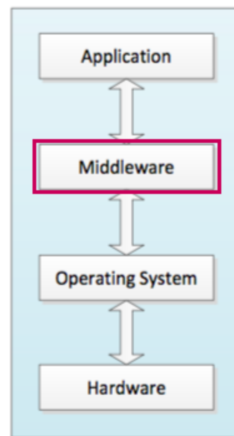


Figure 2.3: Basic software architecture of OBC-NG [2]

2.1.2 Middleware

OBC-NG middleware consists of API, Tasking Framework, Management and Network Protocol layer as can be seen in the figure 2.4 below [2]. The *API layer* provides the communication service and passes messages to be handled by the *Tasking Framework*. The OBC-NG applications run as tasks within the Tasking Framework. The services, which are related to the monitoring system design, are monitoring and reconfiguration services in the *Management layer* and the communication services in *Network protocol layer*. All middleware services are provided using message-triggered and event-triggered mechanisms [2].

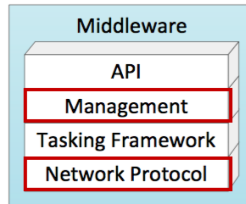


Figure 2.4: Structure of the OBC-NG middleware [2]

Management Layer

Monitoring Service The monitoring function is assigned to PNs to monitor the health of the other nodes and trigger reconfiguration if a node failure is detected. In the current prototype, the Master monitors the other PNs as well as INs by sending a small message called a Heartbeat (HB) to them periodically at a specified interval. The nodes those are alive will send an Acknowledgement (ACK) back to the Master. An Observer is assigned to monitor the Master, using the same procedure. In order to increase the reliability of the system, the Observer role is assigned to two PNs as can be seen in figure 2.2. As explained in subsection 2.1.1 Observer 1 observes Master, and Observer 2 observers Observer 1 and the Master. If Master detects failure, it triggers reconfiguration. If Observer 1 or Observer 2 detects a failure, it check if there is a healthy higher priority management role and inform it to reconfigure. If there is no higher priority management node, it triggers the reconfiguration.

Reconfiguration Service Reconfiguration is categorized into two types: *planned reconfiguration* and *reconfiguration due to a failure*. When the monitoring system detects failure, it triggers the latter type of reconfiguration. The reconfiguration is triggered by the Master or the highest priority Observer, in case the Master failed, by broadcasting a reconfiguration message with the highest priority. The purpose of reconfiguration is either to replace the failed component or to isolate it from the rest of the system [1]. OBC-NG system redistributes its tasks to the remaining nodes. During the reconfiguration, nodes stop the message transmission to reduce payload on the network and reduce the reconfiguration time.

The reconfigurator (the Master or the highest priority Observer) searches the decision graph, as an example in figure 2.5, for the next configuration to reconfigure itself and the other nodes. The decision graph contains the specific configurations for each specific failure when it occurs. It shows the configuration ID and the next configuration of each node failure ID (N-x). The initial configuration ID is 0 (C0). The configuration is based on the failed node ID, e.g. if N1 (node ID = 1) fails, the next configuration ID is 1 (C1). Each configuration has two types of arrays to specify how the roles or tasks are distributed to the nodes. The management configuration arrays specify

the roles of the nodes and application configuration arrays specify the tasks to run on the nodes. Failed node can be isolated until the last node is left and there is no more productive configuration. In other words, the configuration is at a leave of the decision graph. In that case, the system enters *safe-mode* and is solely handled by the Master.

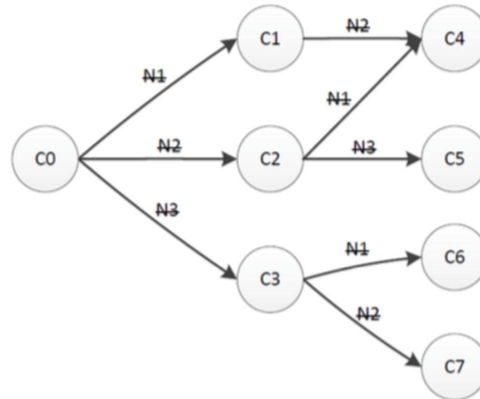


Figure 2.5: An example of a decision graph [2]

Network Protocol Layer

The *Network Protocol* functions with the support of the other components in the network layer as shown in figure 2.6 below. The Network Protocol transmits and receives messages through the *Underlying Protocol* via an abstract layer of the *Network Connector*. The *Event Handler* handles the received data and the *Timer Service* offers timer functionality from hardware.

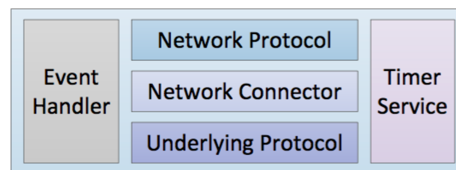


Figure 2.6: Structure of OBC-NG network layer [2]

Functionality of Network Layer The network layer provides the functionality of reliable, unreliable and large-sized messages transmission as well as subscription and broadcasting mechanisms for communication service among the distributed nodes. Apart from the communication service, it supports other services in the higher layer, i.e. OBC-NG middleware layer. The acknowledgement, resend and error notification mechanisms of the reliable message transmission can be utilized in the monitoring service, such as for sending HB ACK or resending HB request. For the reconfiguration,

broadcasting service is used to send the reconfiguration messages to all the nodes and the network layer incorporates the Network Protocol and Timer Service to make the system wait for a specific time to be sure that all the nodes has finished the reconfiguration.

Message Types and Sizes Messages are categorized into different types and have specific sizes as listed in the table 2.1. For a reliable message, the receiver sends an ACK to the sender when it receives the message within a specified time period. If the ACK is missing, the message will be resent. On the contrary, an unreliable message has no acknowledgement and resend mechanisms.

The maximum size of messages can be specified. The current maximum size is set at 1.05 MB with the maximum transmission segment of 55400 bytes.

Table 2.1: Transmission message types and sizes

Message type	Message size (bytes)
_DATA_RELIABLE	23
_DATA_UNRELIABLE	23
_ACKNOWLEDGE	27
_PULLREQ	23
_PULLRESPONSE	23
_ERROR_NOTIFY	21
_RECONF	22
_HEARTBEAT	18

2.1.3 SpaceWire

SpaceWire is planned to be integrated in OBC-NG system because it is widely used in many missions in space domain by several space agencies, such as Meteorological Operational Satellite (MetOp), Rosetta, Mars-Express by The European Space Agency (ESA), Swift Gamma Ray Burst Explorer (SWIFT), James Webb Space Telescope (JWST), Hubble Robotic Repair Mission by The National Aeronautics and Space Administration (NASA) and Bepi Colombo, New X-ray Telescope (NeXT) by The Japan Aerospace Exploration Agency (JAXA) [12].

A SpaceWire network is constructed from point to point links. Each link is a full-duplex, bi-directional, serial data link, which can operate at the data rates between 2 -200 Mbits/s [3]. Similar to the OBC-NG requirements, there is no restriction on the topology of a SpaceWire network. In term of reliability, a crossbar implementation in SpaceWire allows a fully meshed network, with each router of each node interconnected with each other [12].

SpaceWire is a packet switching network and wormhole routing switches are used to reduce to the

latency of transmission. In addition, wormhole switching minimizes the amount of buffer memory needed in the routing switches [12]. Fault tolerance can be achieved when more than one link connects a pair of routing switches using group adaptive routing to with rapid recovery from a link failure [12]. There is no limit on the size of SpaceWire packets. The packet format in figure 2.7 shows that, apart from destination address and payload, there is the End-of-Packet (EOP) field to indicate the end of the packet [12]. However, the maximum packet size can be specified to prevent a blocking for an indefinite time. Packets that exceed a certain maximum payload size are split up into multiple packets.

Figure 2.7: SpaceWire packet format [3]

For time synchronization, SpaceWire supports the distribution of time information to all nodes in the network with very low latency of a few microseconds. This feature is important for time synchronizing of distributed nodes for monitoring and real-time capabilities [13].

2.2 Distributed System Design

In this section, the distributed system design is investigated, focusing on the useful aspects for designing the monitoring system. Two important dependability attributes, which are in our concern in this report are, *reliability* and *availability*. According to Hamed and Jaber, there are three principles in high availability distributed system. The first one is to eliminate the single points of failure by adding *redundancy* to the system so that a failure of a component does not result in the failure of the whole system. The second principle is to have a *reliable crossover*, which is the ability to switch between components when a component fails. The third one, which is the goal of monitoring, is *failure detection* to trigger the crossover or in OBC-NG, a reconfiguration [14].

2.2.1 Reliability

Reliability is the probability that the system functions correctly as expected for a given period of time under the specified operating conditions [15]. Reliability is defined over an interval of time rather than a time instant, which is the case for availability [16]. The following equation 2.1 is used to reliability or R(t). However, reliability is a statistical probability and there are no absolutes or guarantees.

$$R(t) = e^{\lambda t} \tag{2.1}$$

t is the mission time, or the time the system must execute without an outage
λ is the constant failure rate over time (failures/hour)

2.2.2 Availability

Availability represents the probability that the system is available to operate or delivery its service at a specific time instant [4][16]. Availability of a system can be affected by a failure, a maintenance and an upgrade of the system, due to the installation of new hardware or software [8]. In space system, the cause of the second case can be the mission phase changes.

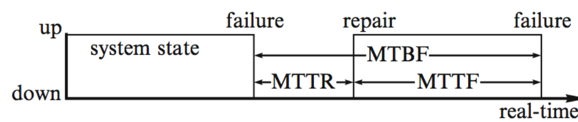


Figure 2.8: Relationship between MTTF, MTTR, and MTBF [4]

In order to assess the availability of the system, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) have to be specified. Their relationship is shown in figure 2.8. MTTF is the average time of normal operation before the failure. MTTR is the average time which is used to repair system and restore to its status. MTBF is the average time between failures in repairable system, when the system is resumed to normal working state and repeat the cycle [8].

In a system with constant failure and repair rates, system availability can be calculated using the following formula 2.2. MTBF is calculated from the sum of MTTF and MTTR. MTBF applies to the ground based or repairable systems. Otherwise, it can also be defined as the average time to the first failure [8].

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.2}$$

The formula 2.2 shows that high availability can be achieved either by a long MTTF or by a short MTTR [8]. The focus of this project is to reduce the duration between failure occurrence and failure detection time within MTTR before the repair or recovery actions are triggered.

For system level prediction, MTTR is calculated by summing the product of the MTTR and failure rates of each replaceable item. The result is then divided by the sum of all replaceable items' failure rates as the formula 2.3 below.

$$MTTR_{system} = \frac{1}{\lambda} \sum_{i=1}^{n} \lambda_i \; MTTR_i \tag{2.3}$$

λ is the constant failure rate over time (failures/hour)
λ_i is the constant failure rate over time of the ith item to be repaired (failures/hour)

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{2.4}$$

2.2.3 Redundancy

Redundancy is duplication of components or repetition of operations to provide alternative functional channels in case of failure [17]. It can be specified into *structural redundancy*, which is referred as a hardware method [16] and *functional redundancy*, which is achieved by a software method. Functional redundancy is a system design and operations characteristic that allows the system to respond to component failures in a way that it is sufficient to meet the mission requirements [15].

Azambuja et al. categorized redundancy into time and space redundancy. *Time redundancy* uses the outputs generated by the same component(s) by comparing the values at two different moments in time, separated by a fixed delay [18]. For *Space redundancy*, the outputs generated by different components at the same time are compared. If there is a mismatch, the failure is detected [19].

For the monitoring design, the redundancy concept will be used to determine the number of Observers. Hamed and Jaber explained about redundancy simulation in their work with *N-x criteria*. N is the total number of components in the system. x represents the number of components used to stress the system. The (N-1) criteria means the model is stressed by evaluating performance with all possible combinations where one machine fails. The (N-2) criteria means the model is stressed by evaluating performance with all possible combinations where two machines fail simultaneously [14].

2.2.4 Threats

The threat of reliability and availability are failures, errors and faults. As depicted in figure 2.9, *fault* is the cause of error and failure. *Error* is the deviation of component behavior from expected behavior [16]. *Failure* is the result of the deviation of the delivered service from the correct service [4].

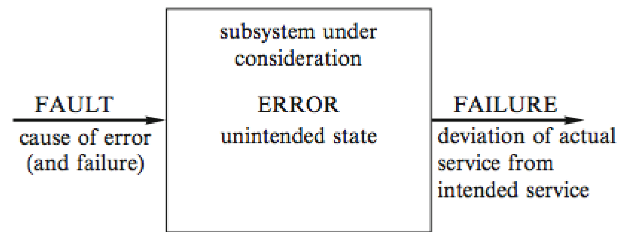


Figure 2.9: Failure, Error and Fault [4]

Failure Rate

Klutke et al. mentioned that a failure rate curve is in the shape of a bathtub curve. At the beginning, in *Infant Mortality Period*, the failure rate is high and gradually decreases. In the second period, *Random Failure Period*, the failures occur randomly by chance [5]. It is the useful life period and has constant failure rate, which is required for calculating MTTR because the calculation assumes a constant failure rate [20]. During *Wearout Period*, the failure rate raises because of the wearout failures, which result from the components aging [5].

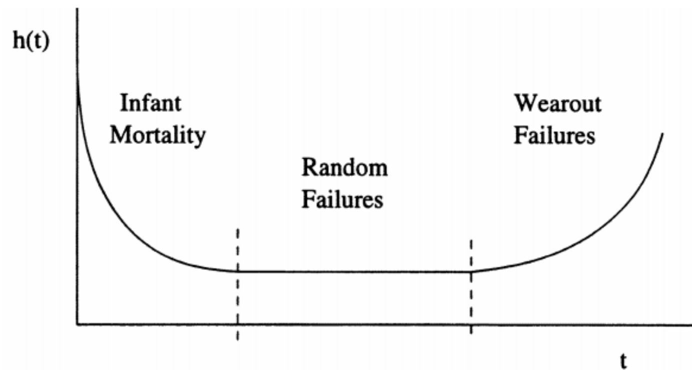


Figure 2.10: The Bathtub failure rate curve [5]

False Avoidance and False Tolerance

The goal of false avoidance is to ensure a component, subsystem, or system does not fail. Fault free system is only theoretically possible and still vulnerable to random failures. In false tolerance system, a component might fail but the system has the ability to maintain the functionality after random failures occurred. System continues to operate but might produces lower level of service rather than failing completely. The lower performance is called *graceful degradation* and might

be because some failed components are switched off in reconfiguration process [1]. The most important method supporting fault tolerance is redundancy.

2.2.5 Design Validation: Fault Injection

Fault injection technique is used to evaluate the efficiency of error detection techniques and assess the fault tolerant systems. Its goal is validate the design with respect to reliability [16]. This report focuses on the simulation-based fault injection. Fault injection at random time on random node will be simulated to represent the random failure in the useful period of the failure rate curve.

2.3 Monitoring

According to Joyce et al., the monitoring of distributed systems involves the collection, interpretation, and presentation of the analyzed information about hardware components or software processes [21]. The monitoring processes are depicted in figure 2.11. Monitoring is essential for the management of distributed systems, such as debugging and error correction, and can be performed on a single object or a group of related objects in the same monitoring domain [6].

Monitoring process

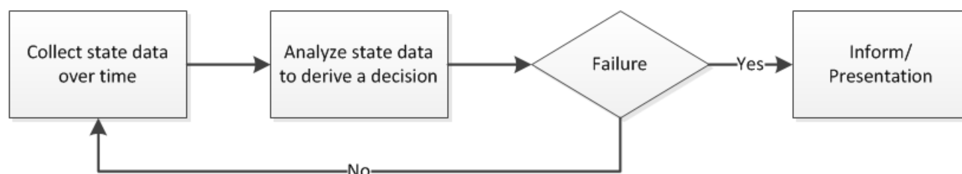


Figure 2.11: The processes of basic monitoring [6]

Monitors need be constantly aware of the existence and health of the components in system with high error detection accuracy and high responsiveness [22]. Zou et al. mentioned some other monitoring requirements in his work, which are to provide redundancy, avoid incorrect judgment and reduce monitoring cost [23].

2.3.1 Monitoring Mechanism Classifications

Monitoring mechanisms can be classified base on different aspects.

Base on how a failure is detected, monitoring mechanisms can be categorized into two categories. In *active monitoring*, sensors continuously send a small-sized message to show that it is alive to its control center. In this case, it is an *implicit detection* because the control centers sense the communication from the sensors and a missing alive-message after a predetermined timeout implies failure. The control center has an overview of the health its sensors. In *passive monitoring*, control center expects no message from sensors unless there is something wrong. This is an *explicit detection*, in which the monitored sensors are able to detect failures and send an alarm to control center [22].

Base on which kind of monitoring information is obtained, monitoring can be classified into *time-driven monitoring* and *event-driven monitoring*. Time-driven monitoring is based on acquiring periodic health status information to provide an instantaneous view of the behavior of an object or a group of objects. The status of an object has a duration in time. For example, a node is alive for a specific duration. Event-driven monitoring is based on obtaining information about the occurrence of specific events, which provide a dynamic view of system activity, as only information about the changes in the system are collected. An event is an atomic entity that reflects a change in the status of an object and occurs instantaneously. For example, message sent, counter reach threshold [6].

Alternatively, monitoring mechanisms can be categorized **base on error detection domain**. In the Design Principles for Distributed Embedded Applications by Hermann Kopetz, he explained that a real-time operating system must support error detection both *temporal domain* and *value domain* [4]. One of the methods to achieve these requirements is to use the *watchdogs*. Temporal domain is used to detect the failure of *fail-silent* nodes. These nodes have self-checking mechanism and either function correctly or stop functioning and produce no results after they detect an internal failure. [24]. A standard technique is the provision of a watchdog signal (heartbeat) that must be periodically produced by the operating system of the node. If the node has access to the global time, the watchdog signal should be produced periodically at known absolute points in time. An outside observer can detect the failure of the node as soon as the watchdog signal disappears. In contrast to temporal domain, for the value domain error detection, challenge-response protocol is executed by the error detector node. It provides an input pattern to the node and expects a defined response pattern within a specified time interval. The functional units required for computing the response are checked and if the response pattern deviates from the expected result, an error is detected.

A failure in the system can be detected at different **levels of granularity**. The current OBC-NG implementation it is on node level. When a heartbeat of a node is missing, all the tasks are migrated to other nodes [2]. Theoretically, the detection could be done in a finer grained level, i.e., task level or subnode level such as FPGA, embedded GPU. When there is a failure, all the tasks will not be migrated to new node but only the specific task will be moved. Consequently, fewer tasks will be interrupted and less bandwidth will be needed.

2.3.2 Fundamental Problems of Distributed System Monitoring

There are a number of fundamental problems associate with the monitoring of distributed systems. The efficiency of the system can be affected by monitoring system and vice versa.

The effects of overall system on monitoring system

As stated by Mansouri and Sloman, in distributed system, it is difficult to obtain a global view of all the system components because of the transmission delay. The transferred monitoring messages from source may already become out of date when it arrives its destination. Moreover, in the monitoring system that the sequence of information is important, time synchronization to provide the mean of determining the ordering is necessary [6].

Another issue they mentioned is, the amount of monitoring information generated in a large system can overburden the monitor and therefore, filtering and processing of monitoring information is necessary. Moreover, as the monitoring system shares the resource with the observed system, if they compete for the resources, the monitoring behavior may alter and affects monitoring result [6].

The effects of monitoring system on overall system

The monitoring system could affect the overall system because of the resource sharing as well. Monitoring creates overhead and also requires processing power, communication bandwidth and memory. It increases the application's executing, workload on the Master and Observer nodes. Thus, excessive monitoring leads affects system performance [6].

In Wan et al.'s study about heartbeat cycle effect on the high availability performance dual-controller RAID system, the system contains main and passive controller. An unsuitable heartbeat cycle leads to wrong fault interpretation of controller status and wrong takeover. They mentioned that, if the heartbeat cycle is relative short if read and write request is still in the wait queue of the master controller, the system interpret that main controller has failed. The master control will start the takeover procedures and transfer request to passive controller. This transfer process takes time and greatly increases the system response time to the request. The delayed heartbeat leads to wrong fault takeover when the system works normally. In addition, the frequency of heartbeat cycle adjustment should also be in consideration when design because it is proved in their work that the frequent changes of heartbeat cycle results could affect the performance of monitored system [8].

2.4 Related Work

Several monitoring techniques from the previous subsection implement the heartbeat monitoring and watchdog mechanism. Heartbeat is categorized as an active and implicit monitoring because the monitoring messages are sent and sensed continuously and the missing of the messages implies failure. The advantages of heartbeat mechanism are, it only needs to maintain few states, low management is required and overhead on the system is low [22]. Zou et al. have categorized heartbeat monitoring into traditional and hybrid heartbeat monitoring.

2.4.1 Traditional Heartbeat Monitoring

In traditional heartbeat monitoring, the heartbeat monitor adopts a model which could be PULL or PUSH models depending on different status realization patterns and which component initiates the monitoring message [23][25].

PULL model

Rachuri et al. explained that PULL has the concept of querying. The monitor node queries for the required information on need basis. This is the advantage of PULL because it can be used to request a specific data at anytime. However, if the query rate is depending on the rate of event occurrences because the mechanism is not efficient when query rate is low and event occurrence is high [26].

In term of failure detection, detection nodes in PULL model send request message to detected node. After detected nodes receive the message, they passively send response message back. Therefore, there are two way of transmission and with the same monitoring rate, the communication cost is higher monitoring model with one way communication [23].

In summary, PULL model can be implemented in monitoring system on need basis or periodically. The current monitoring system design of OBC-NG uses PULL model for periodically monitoring node's health status.

PUSH model

PUSH model has the concept of continuous collection and is useful in the system where continuous sensing is required. In monitoring system, monitored node periodically sending its status to the monitor [23]. It needs only half of the amount of messages for equivalent failure detection efficiency because it is pushing one way [25]. However, the period and data to be sent have to be predetermined. Therefore, it cannot be used to request specific information.

2.4.2 Hybrid Heartbeat Monitoring

PUSH-PULL Hybrid model

In hybrid heartbeat monitoring, the advantages of the two models can be combined. PUSH model has high consistency but lower efficiency, while PULL model has lower consistency but higher efficiency (if PULL model is used on need basis). The switch between PUSH and PULL styles can be according to user requirements and resource status [27].

Zuo et al. combines these two by setting the period of PULL higher than PUSH. If the monitor receives PUSH heartbeat messages within the specific period, the timer gets into next period. Otherwise, monitor activates the timer and adopts PULL model to detect the monitored node if it is invalid or not. If the monitor does not receive PULL response in time, the module is judged as invalid [23].

The advantage of this model is, it is more flexible and can be used for specific purposes because it can sense the occurrence of the events and query when the event occurs [26]. This model is also implemented on Microsoft Operation Manager, which is a good example of the usage of PUSH and PULL for different purposes. Figure 2.12 below shows how each mechanism is implemented. PUSH is used to check the health of the nodes and PULL is used to recheck if the missing heartbeat is because the node fails or the connection fails [7]. Therefore, two types of failure can be differentiated, if there is no response to ping, the missing heartbeat is because of the connection, not a node failure.

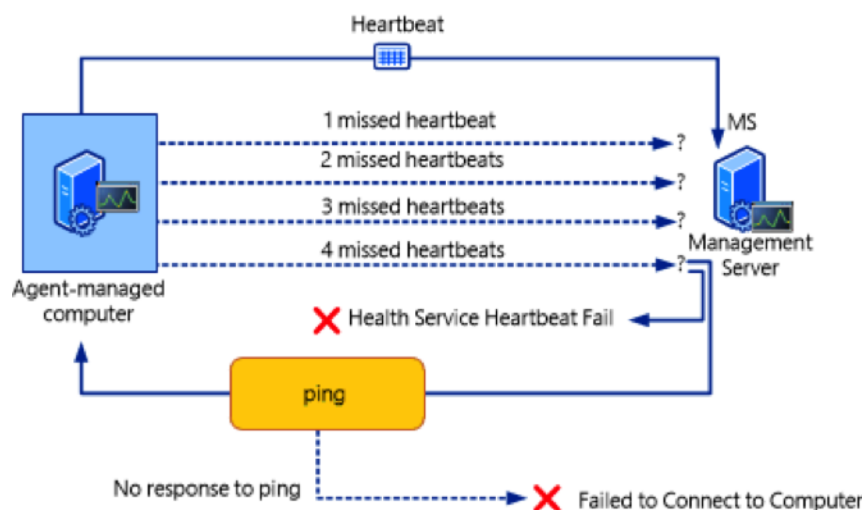


Figure 2.12: Heartbeat flow in Microsoft operation manager [7]

In summary, PUSH-PULL hybrid mechanism can be used for different purposes. First PUSH is used to periodically sending heartbeat to the monitor and detecting failure. When a failure is detected, PULL can be used after a missing push heartbeat is detected for retrieving diagnostic information, such as request the state from specific node, or check the network connection. PULL can also be used in the resend mechanism for requesting heartbeat to confirm the invalidity or failure of the node.

PULL-PUSH Hybrid model

On the other hand, Zhao also mentioned PULL-PUSH hybrid model, where both consumers and suppliers are passive. There is an event channel, which actively requests from the suppliers and passes the response to the consumers. This model gives the channel more control on how to coordinate the components because PULL and PUSH can be executed with various frequencies on each component [28].

Summary of Monitoring Models

In summary, PULL mechanisms have two ways of communications, which increase network overhead if the querying frequency is high. Therefore, PULL is more suitable for querying data on need basis as it can be used to request a specific data at anytime.

In contrast, PUSH is useful when continuous sensing is required but the data sent is predetermined and fixed. It is suitable for checking the health status of the nodes since the message sent can be predetermined as heartbeat without any system details.

PUSH-PULL hybrid model can be used to differentiate two types of failures, such as node failure and network failure. In addition, it can be used to confirm the invalidity or failure of the node. PULL-PUSH hybrid model has passive consumers and producers but the model requires an event channel.

2.4.3 Dynamic Heartbeat

Heartbeat monitoring can be improved by dynamically adjusting the heartbeat period to adapt to different network conditions or system status [23]. It can be implemented in other monitoring models, such as PULL or PUSH. An implementation of dynamic heartbeat is introduced in the application of the adaptive heartbeat design of high availability RAID dual-controller. As shown in 2.13, Heartbeat self-test mechanism for detecting internal error of in the controller and mutual detection between the two controllers are implemented [8]. Heartbeat interaction is made up of ping to inquire information and acknowledgement (ACK) as a response.

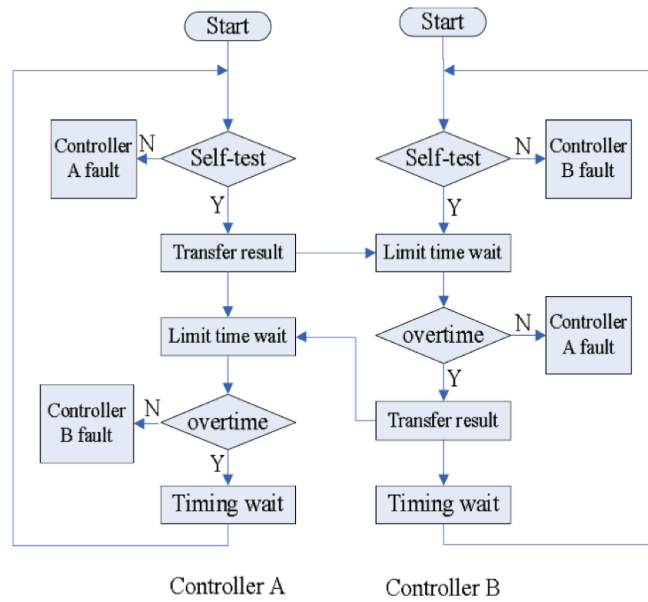


Figure 2.13: Heartbeat self-test and mutual detection [8]

For dynamic heartbeat, the mechanism called *Grade-Heartbeat* is implemented to allow the controller to adapt its heartbeat interval using real-time monitoring module to monitor the read and write requests frequency. Heartbeat interval is set dynamically according to the previous 20 requests interval time. Their experiment also shows that frequent changes in heartbeat interval result in the decline of overall system performance [8].

Chapter 3

Design of OBC-NG Monitoring

This chapter is divided into three sections. Section 3.1 presents the OBC-NG monitoring requirements. The monitoring concepts from the related work are analyzed in section 3.2. Finally, the mechanism settings are listed and explained in section 3.3.

3.1 OBC-NG Monitoring Requirements

General monitoring requirements and fundamental problems of distributed system monitoring are considered along with the OBC-NG's specific requirements and combined into the OBC-NG monitoring system requirements. These requirements are used to specify the performance indicators in the next chapter.

3.1.1 Reliability and Redundancy in Monitoring System

For the reliability, OBC-NG system requires at least one node to always be responsive to ground command at all times. Therefore, we need to make sure that there is always a node assigned as a Master in the system. In the system with only one Observer, if the Master fails and the Observer fails later before the detection, the Master's failure will never be discovered and reconfiguration will never be triggered. Consequently, no node has the Master role and the remaining Worker nodes in the system are not monitored. In contrast, if there are two Observers and one of the Observers fails at the same time as Master, the failure will be detected by the other Observer and reconfiguration will be triggered. Therefore, redundancy is required to provide the reliability.

3.1.2 Availability and MTTR

In a high availability system, MTTR should be low. MTTR of an OBC-NG system is the time from the failure occurrence to when the system repaired, i.e. reconfigured. MTTR consists of fault response time and reconfiguration time. As shown in figure 3.1, monitoring frequency and resend mechanism of the monitoring system influence the first part, which is the fault response time.

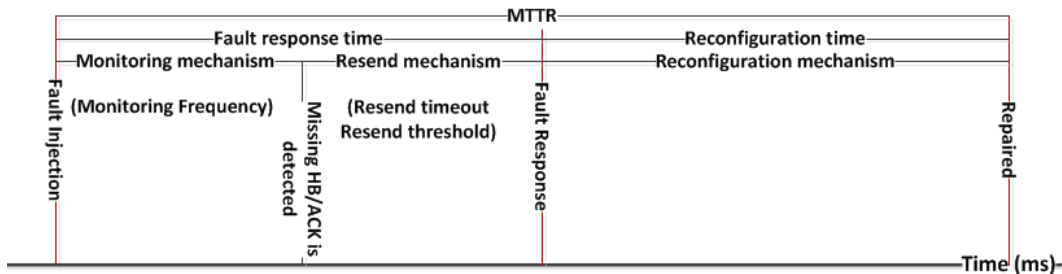


Figure 3.1: MTTR components in OBC-NG

3.1.3 Avoiding Incorrect Judgement

Incorrect judgment should be avoided because the false alarms usually have greater effects on the system efficiency than having an undetected failed node. The reason is, when a failure is detected, every node in the system is reconfigured. From the investigation of distributed system monitoring problems in subsection 2.3.2, transmission delay is a significant cause of incorrect monitoring interpretation and also affects the accuracy of failure detection. Some delayed HBs maybe misinterpret by the monitor as missing HBs. Network traffic is considered the main cause of the delay. In addition, clock synchronization is necessary for determining the timeliness of the monitoring messages [6]. The network layer of OBC-NG system provides the timer service but the transmission delay has to be estimated and added to the watchdog timer to monitor the arrival time of monitoring messages.

An obvious trade-off exists between the probability of false alarm and the fault response time. In order to decrease the response delay, the timeout value needs to be decreased but that leads to a higher probability of false alarm.

Furthermore, to avoiding incorrect judgment, the monitoring system should be fault tolerant and remains the level of accuracy after a reconfiguration occurred.

3.1.4 Reducing Monitoring Cost

The monitoring system should not create too much overhead that it affects the efficiency of other applications in the system. The number and size of monitoring messages and bandwidth usage should be measured and compared with the overall bandwidth. The smaller the monitoring messages, the less data has to be processed. Consequently, the CPU time and the memory consumption are reduced.

3.2 Monitoring Mechanisms Concepts

This section introduces the currently implemented monitoring mechanism and the other alternatives collected from section 2.4. The PULL, PUSH and PUSH-PULL hybrid models are analyzed with the same monitoring roles of Master and Observers as the currently implemented model. PULL is chosen because it is used in the current monitoring system of OBC-NG. PUSH is chosen because it is suitable for continuous sensing, which is required for nodes' health status monitoring. PUSH-PULL is chosen to utilize PUSH for monitoring and PULL for requesting PULL HB when a PUSH HB is missing. On the other hand, for PULL-PUSH mechanism is not chosen because it requires an event channel and therefore, it is not suitable for point-to-point topology of OBC-NG. In addition, if PULL response is missing, PUSH cannot be used for requesting a HB in resending mechanism.

The OBC-NG system has a Master serving as the monitor of the other nodes and two Observers as the monitors of the Master, as mentioned in the monitoring service subsection. In this section, the Master or an Observer is referred to as a monitor.

3.2.1 Current Monitoring Mechanism: PULL Mechanism

After a the first reconfiguration is finished, the monitor waits for the specified HB interval and initiates the HB mechanism by sending a HB request to its monitored node(s) and starting the timer. When a monitored node receives the HB request, it sends the HB response (ACK) back to its monitor. The monitor listens to response, as shown figure 3.2 below. If ACK does not arrive within the specified duration, the resend mechanism begins. The details of the resend mechanism are explained in subsection 3.3.3. PULL is a request-response mechanism and therefore, resend mechanism can be implemented. Moreover, it can also be used as resend mechanism in other hybrid mechanisms.

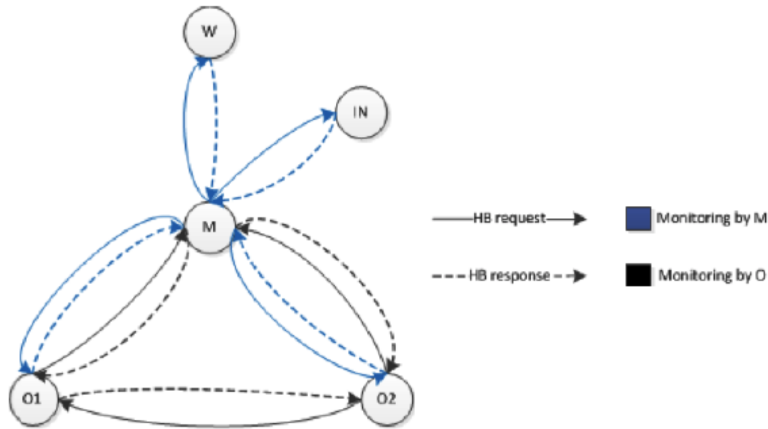


Figure 3.2: Heartbeat flows in PULL mechanism

3.2.2 Alternative 1: PUSH Mechanism

Similar to PULL mechanism, PUSH begins after the reconfiguration is finished but it is initiated by the monitored node. Each monitored node is set up in a way that it sends a HB message to its monitor at every specified interval. Therefore, there is just one way of monitoring message flow for each monitoring pair as can be seen in figure 3.3. Each monitor node sets up a watchdog timer for its monitored node(s). If the HB arrives within the duration, the watchdog timer is reset. Otherwise, the resend mechanism begins.

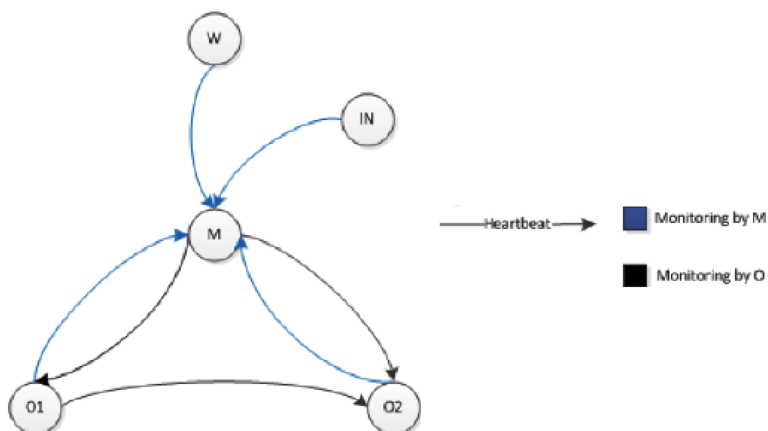


Figure 3.3: Heartbeat flows in PUSH mechanism

3.2.3 Alternative 2: PUSH-PULL Hybrid Mechanism

PUSH and PULL mechanisms are combined to allow the implementation of the resend mechanism into the PUSH model. Before the monitor senses a missing HB, the HB mechanism is the same as PUSH in the previous subsection. After the first missing HB is sensed, the monitor waits for a specific time duration, then switch to PULL mechanism, in which monitors start sending HB requests and wait for the responses. Afterwards, if there is no HB response, PULL resend mechanism begins.

3.3 Monitoring Mechanism Settings

3.3.1 Number of Observers

Number of Observers is determined by the assumption of the simultaneously failure occurrences. In a system with the mechanism setting of with one Observer, maximum number of monitors that can fail simultaneously is one (The M or an O). In the design, it is assumed that the maximum number of failures that could happen at the same time is two. Therefore, the redundancy concept of *N-x criteria* from section 2.2.3 is implemented on the monitoring system to specify the number of monitors in the system and measure the monitoring performance. N is the total number of management roles (The M and Os) and x is the number of monitor nodes that can fail simultaneously and can be detected. If there are two Observers, N = 3 (two Os and one M) and x = 2, and two of them can fail at the same time.

3.3.2 Heartbeat Interval

HB interval effects the fault response time and the monitoring overhead. When a failure occurs, the monitoring system detects it at the next missing HB, then waits until the timeout and starts resend mechanism. HB interval is the duration between two HBs. The larger HB interval causes less monitoring overhead but increases detection time. The smaller HB interval causes higher overhead but the missing HB is detected faster because the HB is checked more often and the difference between failure occurrence and the next HB is lower. However, the higher number of HBs is expected, the higher chance of a missing HB. One possible solution is to increase the reconfiguration timeout, which is the duration between when a missing HB is detected and when a reconfiguration starts. If the duration is over without receiving a HB, then the reconfiguration can be triggered.

Static Heartbeat Interval

In static HB interval setting, every node periodically sends HBs at the same interval. The advantage of this setting is, it has less synchronization problem but the disadvantage is, it is not adjustable to the amount of network traffic. The baseline of HB interval is set at 200 ms and the intervals of 100, 500, 1000 ms are chosen to be tested and compared with the baseline.

Dynamic Heartbeat Interval

There are two options of adjusting the HB interval for dynamic HB mechanism. One option is to determine if there is a congestion or high traffic in the network by measuring the network, and readjust the HB interval accordingly. Another option is, to calculate the new HB timeout at the monitor according to the previous PULL HB response or PUSH HB arrival time.

3.3.3 Resend Mechanism

Resend mechanism affects the fault response time and detection accuracy. It contains two values, resend timeout and resend threshold, as seen in figure 3.4. *Resend timeout* is the duration from when a missing HB is expected to when a HB request is resent. *Resend threshold* is the maximum counts of resending HBs (n). If the threshold is reached and there is no response from the monitored node, the reconfiguration will be initiated. Resend threshold is only implemented in the resend mechanism of PULL and PUSH-PULL model, when the mechanism switches to PULL because PULL HB is a reliable message. As mentioned in subsection 2.1.2, if there is no ACK for the reliable message from the receiver, the message or in this case, a PULL HB, will be resent.

In PUSH mechanism, HB is an unreliable message and is initiated from the monitored node, thus the monitor cannot resend HB request. However, resend timeout can be used in PUSH mechanism by setting resend threshold as 0, consequently, the resend timeout will be used as a reconfiguration timeout. *Reconfiguration timeout* is the duration from the start of the first resend timeout until the end of the last one. In other words, it is the duration from the detection of a missing monitoring message to the reconfiguration.

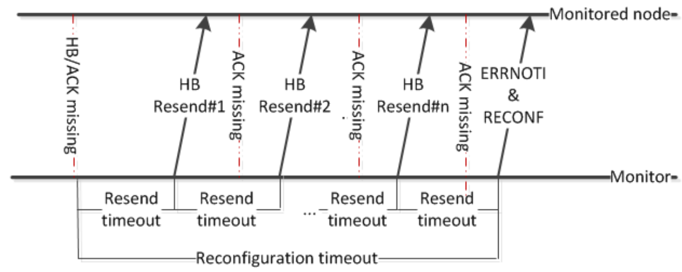


Figure 3.4: Resend mechanism with resend timeout and resend threshold

Chapter 4

Simulation Specification and Implementation

This chapter shows the simulation specifications of monitoring models, which are designed in the previous chapter. It begins in section 4.1 with the introduction to OMNeT++, which is the simulation tool used for implementing and testing the models. Afterwards, the simulation objectives and the performance indicators are explained in section 4.2 and 4.3. The details of how OBC-NG system is simulated are shown in section 4.4. Section 4.5 presents the implementation of monitoring mechanisms on OBC-NG system. The mechanism and environment settings simulation are explained in section 4.6 and 4.7. Section 4.8 describes how to the simulations are executed. Finally, the five test suites, which combine each monitoring mechanism, its settings and environment settings, are explained in the last section.

4.1 Simulation Tool: OMNeT++

OMNeT++ is an object-oriented discrete event network simulation framework. It is not a simulator but it provides infrastructure, libraries and frameworks for the simulation. Its modular architecture divides the network models into modular and reusable components. Each active component in the simulation model is called a *simple module*, which is a basic unit of other *compound modules* and *models*. The modules are connected via a channel [29]. In addition, it also uses message-triggered and event-triggered mechanisms as OBC-NG middleware as mention in subsection 2.1.2.

The main functions of the class cSimpleModule are *initialize* and *handleMessage*. The first one is invoked by the simulation kernel once at the beginning and the second one at each message arrival. Messages, packets and events are all represented by cMessage objects. Most models need to schedule future events in order to implement timers, timeouts, and delays. It can be done in simple module by sending a *self-message* to itself. After the self-message is sent or scheduled, it will

be held by the simulation kernel until the schedule time is reached then the message is delivered to the modules via *handleMessage* [29]. Since the simulation kernel is event based, the event can be scheduled at different points in real time to the same simulation time. When the simulation time is reached, it is seen as the events are executed in parallel.

4.1.1 OMNeT++ Components Specifications

The behaviors and designs of the components are specified in different types of files [30].

- **.ned file**: NED or Network Description is a high-level language that describes the structure of the modules at every level, from simple module to a model. It also describes the parameters, gates, and network topology of the model.
- **.cc file**: The file contains the behaviors of each module, which are specified in C++ language.
- **.msg file**: The communication between module is via messages. Message Definitions file is for defining various types of message or packet and specifying data fields. OMNeT++ will translate message definitions into C++ classes.
- **.ini file** or a configuration file contains the configurations of the model and different parameters of each simulation run.

4.1.2 OMNeT++ User Interfaces

The simulations can be run using Tkenv or Cmdenv user interfaces. *Tkenv* is a GUI toolkit and graphical simulation environment, which links into simulation executables for graphical interactive simulation execution. It is very useful for model development and verification. *Cmdenv* is a command-line user interface for batch execution. It can be used to execute many runs by one command [29].

4.1.3 Output Formats and Types

The output of simulation can be recorded as vector or scalar. Vector output is recorded during the simulation into *output vector file* (.vec). The scalar values are collected during the simulation in a variable and recorded at the end of the simulation in *output scalar file* (.sca). Moreover, if the simulation is run with record event log option, the sequence chart is produced and stored in *event log file* (.elog). It contains the graphical view of modules, events and messages sequence and transmission duration. The first three types of output files can be exported as *Scalable Vector Graphics file* (.svg) [30].

4.2 Simulation Objectives

Two main objectives of the simulation are to evaluate the monitoring efficiency and to measure the overhead of different monitoring mechanisms and their settings. They are tested under different environment settings, i.e. high network load and node failures. Fault injection is simulated in order to see how fast the monitoring system reacts when different type of nodes fail at different fault injection time. The outcomes of simulation are used to evaluate different monitoring design options for OBC-NG system.

4.3 Performance Indicators

Performance indicators are used to measure the efficiency of the monitoring models. They are chosen according to the OBC-NG monitoring requirements from section 3.1 to prevent the fundamental problems of distributed system monitoring, which are previously presented in subsection 2.3.2.

4.3.1 Monitoring Overhead

An efficient monitoring system should create as low overhead as possible. The bandwidth consumption is measured to specify the communication overhead and the total number of monitoring messages sent from each node to measure the workload on different roles. Both HB and ACK are measured over the simulation time. CPU overhead is not simulated because in the current implementation, every node in the network is COTS component and therefore it is assumed that the computing power is sufficient.

4.3.2 Fault Response Time

Fault response time in this work is the duration between fault injection time and when the remaining healthy nodes receive reconfiguration messages. It is right before the reconfiguration mechanism starts and the nodes handle the reconfiguration messages. At the end of each simulation, the difference between the simulation times of those two events is recorded.

4.3.3 Monitoring System Stability

Monitoring system stability is the ability to avoid incorrect judgment and running into a wrong configuration, when there is high network load in the system. The system starts up in the initial configuration with configuration ID = 0. In the test without node failure, the reconfiguration

should not be triggered. If the monitor searches the decision graph for the next configuration and tries to initiate reconfiguration by sending reconfiguration message with configuration ID $\neq 0$, the timestamp is recorded and *endSimulation* function is called.

4.4 OBC-NG System Simulation

Figure 4.1 shows an OBC-NG system simulation on the OMNeT++ framework. The simulated network consists of six COTS nodes and each node represents a PN. Each PN is simulated as a compound module.

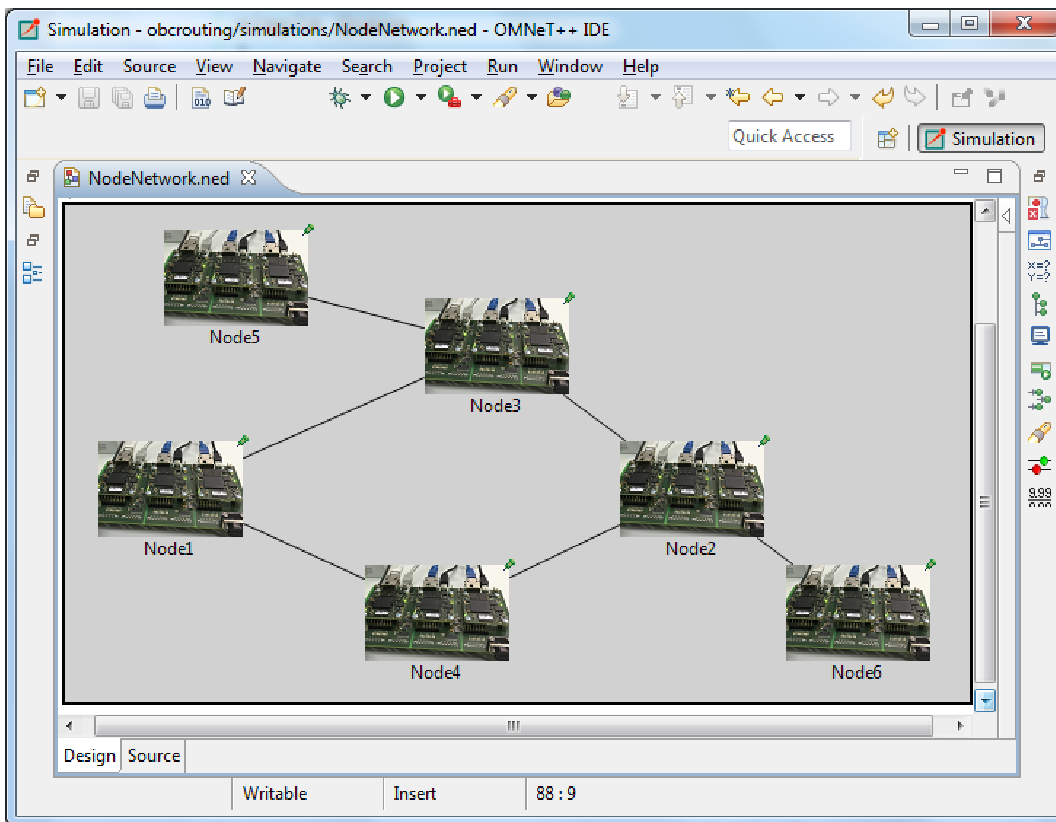


Figure 4.1: OBC-NG system simulation on OMNeT++

Each compound module consists of four different simple modules, *Manager*, *userApp*, *Routing*, and *Queue*, to perform different functions as shown in figure 4.2.

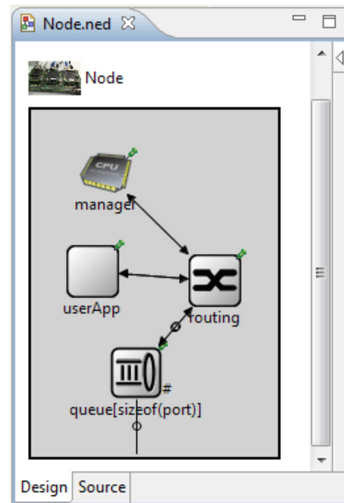


Figure 4.2: Structure of OBC-NG node on OMNeT++

The nodes are connected through ports and links (channels). The simulated channel between the nodes has a channel delay of 3 ms and SpaceWire's maximum data rate of 200 Mbit/s as shown in figure 4.3.

```
channel C extends DatarateChannel
{
    parameters:
        delay = 3 ms
        datarate = 200 Mbps
}
```

Figure 4.3: Channel settings in NED file

4.4.1 Manager Submodule

The Manager submodule provides the functionality of initializing the node according to its roles as well as handling messages and HBs. It also provides the monitoring and reconfiguration services. Manager submodule enables the Master and Observers to perform management and monitoring tasks and the Workers to perform application tasks and sending HBs.

Node Initialization

When a node is initialized, node ID is used to check its role with the management configuration array. If it is the Master, it broadcasts the reconfiguration messages to configure other nodes when the system starts. Manager submodule uses node ID to identify the role and status of the node. For

example, a node with node ID 1 is the Master and it is alive. If the node has no management role, it is initialized and starts the assigned task as a Worker. The task distribution is specified in the application configuration array. The fault injection event is also scheduled according to a specific or a random point in the simulation time in this submodule at the beginning of the run.

Messages Handling

Manager handles its self-message, which is for scheduling events in its module as explained in section 4.1. If it is an incoming message from other module, the message is handled through the network protocol. Manager submodule also handles reconfiguration and error notification messages, which are created in the network protocol.

Heartbeat Sending and Monitoring

To conduct the monitoring, Manager submodule sends PULL HB from the monitor or PUSH HB from the monitored node. For PULL mechanism, HB ACK is created and monitored by network protocol as a mechanism of a reliable message. However, in PUSH model, PUSH HB has to be handled in Manager. The watchdog timeout value is set as an attribute of the node. If the watchdog timer is over, the reconfiguration is triggered. For PUSH-PULL, if the watchdog timeout is reached, Manager starts PULL mechanism, which is handled in the network protocol in the same way as the traditional PULL mechanism.

Selecting New Configuration and Reconfiguration

If the node is the Master or the highest priority Observer (in case the Master fails) and the reconfiguration needs to be initiated, it searches the decision graph for the next configuration ID, initiates reconfiguration and broadcasts reconfiguration command via the network protocol.

4.4.2 Application Submodule

Application submodule, *userApp* in figure 4.1, is used for performing application tasks and in this work, it is used for traffic generation. The application generates a packet by specifying the source, destination and size of the packet. The packet size can be fixed or varies in each transmission. The generated packets are sent out via gate *out* to the routing module, as seen in figure 4.4. The application address is set differently from node ID so that Routing submodule, which is explained in the next subsection, can differentiate them and send the different types of message to the right submodule. The IDs < 50 are reserved for the management service and application ID starts from 51. The application module schedules *applicationEvent* to start the application of sending packets at

the specified time. After a packet is sent, the next *applicationEvent* is scheduled to send the next packet in the next interval.

4.4.3 Routing Submodule

Routing submodule connects the Manager and userApp submodules to the Queue submodule as shown in figure 4.4. It provides the functionality of static shortest-path routing between the parent modules, i.e. PNs and sets the routing table when the simulation begins [29]. Each node queries the topology of the network independently. However, the routing table is not updated during the simulation. Therefore, when a node fails, its routing function still works and can forward application and other management packets.

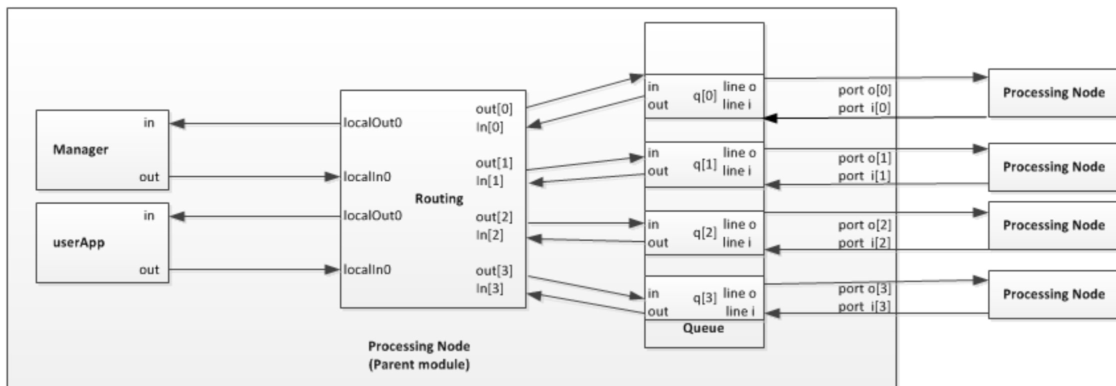


Figure 4.4: Routing and connections

4.4.4 Queue Submodule

Queue submodule is for queuing up packets to be sent. It is a point-to-point interface module. As can be seen in figure 4.4, packets from Routing module that arrive at *in* gate of Queue submodule are sent out through *line* gate, which is connected to the internode connection. If a packet arrives and there is a packet being transmitted, the arrived packet will be queued up. The size of the queue is limited by the frame capacity, which can be specified in the submodule's .ned file. If the packet queue is over the queue size, the packet will be discarded and drop counts will be recorded [29]. In the simulation, no limit is set because if the monitoring message got dropped because of the frame capacity, the monitoring result will change and we want to limit the causes and only focus on the delayed HB, which is caused by the amount of traffic.

4.4.5 Packets

In figure 4.5, the *Transmission* struct is the packet structure of OBC-NG and the *packet* struct below is how a packet is simulated on OMNeT++ in .msg file. The parameters of each packet are set similarly to OBC-NG transmission protocol. However, each simulated packet has no data and there is no need to specify the maximum data size or data type in the parameter. In the simulation, data size is set using *setByteLength* function and it will be used to calculate the transmitting time from data rate specified in the channel property in .ned file. The packet type is differentiated by a packet name. The name and size of a packet are set when it is created. The type of a packet created by the underlying network and transmission protocols is also used to set the packet name using *setName* function of OMNeT++.

```
struct Transmission {
    uint16_t _sender;
    uint16_t _receiver;
    int64_t _timestamp;
    uint8_t _type;
    uint8_t _data [MAX_Transmission_Data_SIZE];
};

packet Packet
{
    int srcAddr;
    int destAddr;
    int hopCount;
    int64_t timestamp;
    char data[];
}
```

Figure 4.5: The structure of an OBC-NG packet and a simulated packet

4.5 Monitoring Mechanisms Simulation

Different monitoring mechanisms utilize different types of messages and resend mechanisms. Consequently, the resend timeout and resend threshold have to be set differently. This section shows how each monitoring mechanism is simulated.

4.5.1 PULL Model Simulation

Message Type

In PULL mechanism, the HB request of type _HEARTBEAT is a reliable message of size 18 bytes. The HB response is the ACK of type _ACKNOWLEDGE, which is 27 bytes.

Heartbeat Creation and Monitoring

After a node gets a reconfiguration message, the first *heartbeatEvent* is scheduled by the monitor in *handleReconfig* function of Manager submodule at the current simulation time + HB interval. The *heartbeatEvent* is handled by *heartbeatService* function of PULL mechanism. The function first checks the management role of the node in management configuration array to specify the HB destination, i.e. its monitored node(s). After the HB is sent, the function reschedules the new HB of the next interval. ACK is the response to the HB and is created and monitored by the network protocol.

Resend Mechanism

Figure 4.6 shows PULL mechanism and its resend mechanism. If the network protocol senses a missing ACK, it waits for a specific duration (resend timeout) then resends HB request again until the resend threshold (n) is reached. The ACK is monitored in the network protocol as well as the resend mechanism.

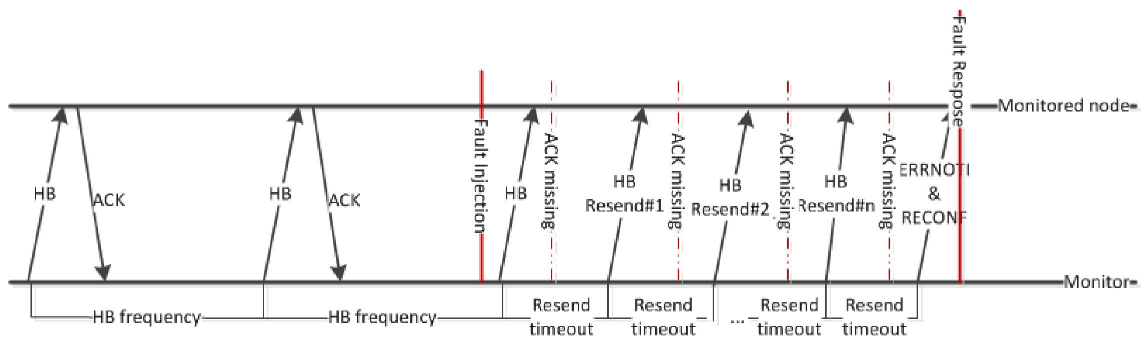


Figure 4.6: Monitoring design with PULL mechanism

The reconfiguration timeout can be calculated using equation 4.1 below.

$$Reconfiguration\ timeout\,(ms) = Resend\ timeout \times (Resend\ threshold + 1) \tag{4.1}$$

4.5.2 PUSH Model Simulation

Message Type

Due to the fact that PUSH HB does not require a request-response mechanism, it should be set as a new message type so that the network protocol does not send an ACK back when it arrives at the monitor. PUSH HB messages are simulated as a new message type (_PUSH_HEARTBEAT) with the same size as _HEARTBEAT (18 bytes) but as an unreliable message instead of a reliable message.

Heartbeat Creation and Monitoring

Similar to PULL mechanism, after a node gets a reconfiguration message, the first *pushHeartbeat-Event* is scheduled in *handleReconfig* function to the value of current simulation time + HB interval. However, PUSH HB is created by monitored node and handled in *pushHeartbeatService* function, which specifies the node's monitor address, sends the PUSH HB and reschedules the PUSH HB to the next interval.

At the monitor, a watchdog timer for each of its monitored node is set and scheduled as an event. If the PUSH HB arrives, the watchdog timer is rescheduled to next interval and if not, the function *handleDelayedPushHeartbeat* is called. In the function, if the node is highest priority monitor, it searches the decision graph for the new configuration and initiates the reconfiguration. If it is not, it checks if there is any higher priority monitor, which is alive, and informs it.

Resend Mechanism

As can be seen in figure 4.7, the monitored nodes of PUSH mechanism initiate the monitoring messages. Consequently, resend mechanism cannot be implemented and the resend threshold is 0. If the monitor senses a missing HB, it uses the value of the resend timeout as the reconfiguration timeout as shown in equation 4.2. After the timeout, the reconfiguration is triggered. However, if the reconfiguration timeout is long enough, monitored nodes have the chance to send the next HB of the next interval.

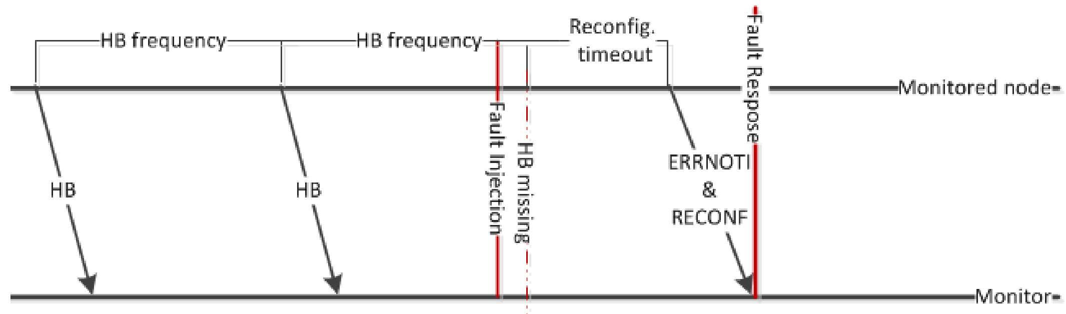


Figure 4.7: Monitoring design with PUSH mechanism

$$Reconfiguration\ timeout\,(ms)\ =\ Resend\ timeout \tag{4.2}$$

4.5.3 PUSH-PULL Hybrid Model Simulation

Message Type

Three types of messages are used in this model. Before a node failure is detected, each monitored node sends _PUSH_HEARTBEAT messages to its monitor. When a PUSH HB is missing, the monitor sends _HEARTBEAT messages to recheck the invalidity of the node. If the monitored node is healthy, it sends a message of type _ACKNOWLEDGE back.

Heartbeat Creation and Monitoring

PUSH HB is created and monitored the same way as in the traditional PUSH model. However, when a PUSH HB is missing, the function *handleDelayPushHeartbeat* at the monitor handles the situation differently by starting sending a PULL HB.

Resend Mechanism

The resend mechanism is the same as in PULL model. However, the resend threshold should be reduced by one to get the desired number of HB resends because after the mechanism changes from PUSH to PULL, the first HB is a normal HB of PULL mechanism. Afterwards, the resend threshold of PULL starts as seen in figure 4.8.

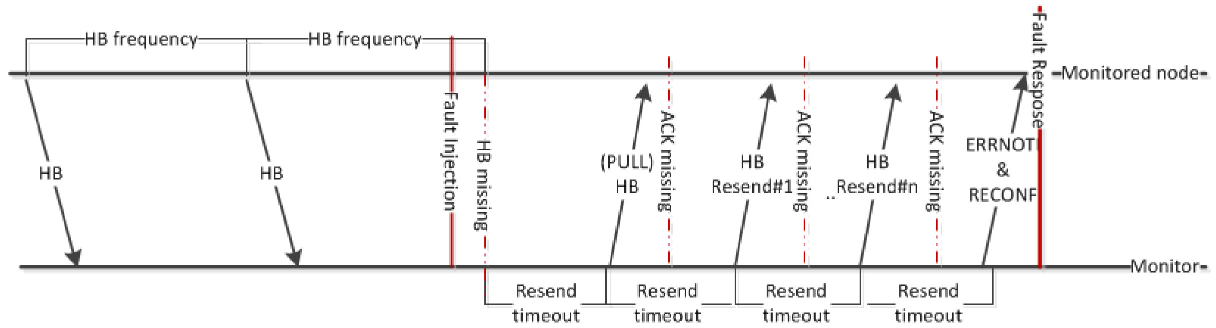


Figure 4.8: Monitoring design with PUSH-PULL hybrid mechanism

Reconfiguration timeout of PUSH-PULL can be calculated using the formula 4.3 below.

$$Reconfiguration\ timeout\,(ms)\ =\ Resend\ timeout\ \times\ (Resend\ threshold\ +\ 2) \tag{4.3}$$

4.6 Monitoring Mechanism Settings Simulation

In this section, the settings of the monitoring mechanism are listed. In addition, the details of how they are simulated in different mechanisms are explained.

4.6.1 Number of Observers

The desired number of Observers is specified in the configuration file (.ini). In Manager submodule, the monitoring roles and the destinations of monitoring messages are set accordingly. For example, in the setting with one Observer, Observer 2 in the management configuration array becomes a Worker and does not have monitoring functionality.

4.6.2 Mechanism Setting: Static Heartbeat Interval

The HB interval parameter is also specified .ini file. After a node get a reconfiguration message, the first *heartbeatEvent* or *pushHeartbeatEvent* is scheduled in *handleReconfig* function of Manager submodule at that specified HB interval. When the event is handled, a HB is sent and the event is rescheduled at the same interval.

4.6.3 Mechanism Setting: Dynamic Heartbeat Interval

There are two options of simulating dynamic HB. The first approach is to measure network traffic and adjust HB interval accordingly. In PULL mechanism, the monitor changes the interval of the HB request, and in PUSH, the monitor sends a message to the monitored node to change the interval of the next HB to prevent the synchronization issue. The monitor node itself changes the duration of the watchdog timer to prepare itself for the new interval. The disadvantage of this approach is, the change occurs at the next HB interval and it might be too late to adapt to the network traffic.

Another approach is to adjust the watchdog timer according to the previous HB arrival. The new watchdog timer is set when the HB arrives. The simulation times of the previous and the current HB arrivals are recorded. The difference between these two values is added with a slack time to calculate a new watchdog timer. The slack time is calculated using the equation 4.4 below. The slack factor makes the next interval slightly larger than the actual difference of the HB arrivals to prepare for higher network traffic. In this case, the monitor can sense the traffic and adjust the watchdog timer accordingly.

$$Slack\, time\, (ms) \;=\; (simulation\, time\, of\, current\, HB\, arrival - previous\, HB\, arrival) \times slack\;\; factor \tag{4.4}$$

4.7 Environment Settings Simulation

4.7.1 Fault Injection: Node Failure

Node failure is simulated by scheduling a *faultInjectionEvent*, which is handled by the chosen node at the scheduled time. At the chosen node, its *isAlive* status is updated to false and it will stop sending HBs. On Tkenv GUI, the node is labeled as dead. The purpose of fault injection is measuring fault response time of each mechanism.

In the design, a random failure is injected in each simulation randomly because random failures with constant failure rate are used for calculating MTTR. The duration between fault injection and fault response time of each mechanism is recorded. However, a fault is supposed to be detected within the simulation time (5000 ms) so that the fault response time can be recorded. Therefore, the failures are random in the range of 0 - 4000 ms to make sure that they will be detected in every HB interval setting.

The random function *rand* is used with the seed of wall-clock time, which is specified in *srand* function as *srand(time(NULL))*. It is used to generate random number for fault injection time and

also the failed node ID.

4.7.2 Network Traffic Simulation

Network traffic is generated by userApp submodule in figure 4.2. The traffic is scheduled at the module initialization at a specific interval. Every node sends a packet to the node with the next higher application ID and the highest one sends to the lowest, as shown in figure 4.9. As mentioned in subsection 4.4.2, the application ID is set to a different values from node ID so that the Routing submodule can distinguish them and send the packet to the correct destination submodule. The address of the source and the destination is set with *setSrcAddr* and *setDestAddr* function.

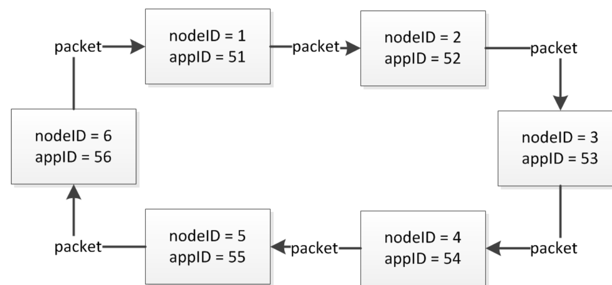


Figure 4.9: Application packet flows

Message size can be static with the same message size or dynamically increasing. Both types of message size can be set with *setByteLength* function. In the case of dynamic size, the size of the message is increased in each interval and is calculated using the equation 4.5 below. Packet factor indicates how fast the message size increases. The higher value of packet factor, the steeper the size increases.

$$Application\ packet\,size\,(bytes)\ =\ current\ \ simulation\ \ time \times\ packetfactor \tag{4.5}$$

4.8 Simulation Execution

The input specifications for setting the simulation parameters are explained in subsection 4.8.1. The combinations of the simulated monitoring mechanisms, mechanism settings and environment settings are selected and combined into different test suites in section 4.9. The test runs can be automated as described in subsection 4.8.2 below.

4.8.1 Input Specifications

Most test settings are specified as parameter values or seeds in the configuration file (.ini). However, the resend threshold and resend timeout are specified as macros since the resend mechanism is provided by the underlying network protocol, not an OMNeT++ module.

4.8.2 Automated Tests

Instead of executing each test case manually, automated tests can be performed to execute every test case in each test suite. The parameter values of each test case settings can be specified in the configuration file. Then change the run configurations in the Integrated Development Environment (IDE) to command line and change run number to * to run all the test cases with every parameter combinations.

4.9 Test Suites

The monitoring system testing is divided into five test suites according to the test objectives. The first test suite compares the monitoring overhead and the fault response time of the PULL and PUSH mechanisms without the resend mechanism. The second test suite aims to compare the settings with different number of Observers. The third and fourth test suites test the monitoring models under different environment settings, i.e. fault injection and network traffic. Finally, the fifth test suite verifies the dynamic HB mechanism.

Test suite 1: Monitoring Mechanisms and Observer Roles

General Purpose

This test aims to compare PULL and PUSH mechanisms without resend mechanism by using the same mechanism settings. Different monitoring roles of the Observers are also tested. The evaluation focuses on the monitoring overhead, workload of each role, and fault response time.

General Configurations

Table 4.1: Test suite 1 General configurations

Mechanism	PULL, PUSH
Number of Observers (nodes)	2
Heartbeat interval (ms)	Static: every 200
Reconfiguration timeout (ms)	30
Resend threshold (times)	0
Node failures	See the test case settings below
Application traffic	None
Simulation time limit (ms)	5000

Test case settings

Table 4.2: Test suite 1 Test case settings

TestID	Node failure	Test details
1.1.1	0 node failure	Observer role setting 1: O2 Observes M and O1
1.1.2		Observer role setting 2: O2 Observes only O1
1.2	1 node failure	Each role fails at 1000 ms

Test Suite 2: Number of Observers

General Purpose

In the previous test, the default number of Observers (two nodes) is tested with different Observer role settings. The chosen setting is used in this test and the following tests. The goal of this test is to compare fault response time of the settings with different number of Observers when the Master fails.

General Configurations

Table 4.3: Test suite 2 General configurations

Mechanism	PULL, PUSH
Number of Observers (nodes)	1, 2
Heartbeat interval (ms)	Static: every 100, 200, 500, 1000
Reconfiguration timeout (ms)	30
Resend threshold (times)	0
Node failures	The Master fails at 1000 ms
Application traffic	None
Simulation time limit (ms)	5000

Test Suite 3: Fault Injection

General Purpose

The goal of this test is to measure the fault response time of each monitoring mechanism and HB interval setting with the same reconfiguration timeout of 120 ms. The timeout is calculated from the baseline setting of the currently implemented mechanism, PULL. As shown in table 4.5, the resend timeouts and resend thresholds are adapted to different resend mechanisms of PUSH and PUSH-PULL models. The number of repetitive runs of each test setting is 2000. In each run, a random node fails at a random time between 0 - 4000 ms to allow the settings with HB interval of 1000 ms to detect the failure before the simulation time limit of 5000 ms.

General Configurations

Table 4.4: Test suite 3 General configurations

Mechanism	PULL, PUSH, PUSH-PULL Hybrid
Number of Observers (nodes)	2
Heartbeat interval (ms)	Static: every 100, 200, 500, 1000
Resend timeout (ms)	See the resend mechanism configurations below
Resend threshold (times)	
Node failures	A random node fails at random time between 0 - 4000 ms
Application traffic	None
Number of repetitive runs (runs)	2000
Simulation time limit (ms)	5000

Resend Mechanism Configurations

Table 4.5: Test suite 3 Resend mechanism configurations

Mechanism	Resend timeout (ms) x Resend threshold (times)	Reconfiguration timeout (ms)
PULL	30×3	120
PUSH	120×0	120
PUSH-PULL	30×2	120

Test Suite 4: Network Traffic

General Purpose

The network traffic is simulated to test the monitoring system because it is the main cause of transmission delay and incorrect monitoring judgment. The cause of the missing HB needs to be determined, whether it is because of a node failure or the HB is sent out from a healthy node but delayed by the network traffic. This test aims to measure the tendency of monitoring mechanisms to run into a wrong configuration, when there is high application traffic in the network.

In test 4.1, different mechanisms with the same reconfiguration timeout of 120 ms are investigated. In test 4.2, PULL and PUSH mechanisms without resend mechanism are tested to compare the

stability of the mechanism. Each run has no simulation time limit and runs until a node get reconfigured and the simulation ends. The system should not be reconfigured, since there is no failure injection. End simulation time and the traffic from each node at the point, when the system run into a reconfiguration, are recorded and compared.

General Settings

Table 4.6: Test suite 4 General configurations

Mechanism	PULL, PUSH, PUSH-PULL Hybrid
Number of Observers (nodes)	2
Heartbeat interval (ms)	Static: every 100, 200, 500, 1000
Resend timeout (ms)	See the resend mechanism configurations below
Resend threshold (times)	
Node failures	None
Application traffic	Each node send a packet every 50 ms Packet size (bytes) = current simulation time × packet factor Packet factor = 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180
Simulation time limit (ms)	-

Test Case Settings

Table 4.7: Test suite 4 Test case settings 4.1

TestID	Mechanism	Resend timeout (ms) x Resend threshold (times)	Reconfiguration timeout (ms)
4.1.1	PULL	30×3	120
4.1.2	PUSH	120×0	120
4.1.3	PUSH-PULL	30×2	120

Table 4.8: Test suite 4 Test case settings 4.2

TestID	Mechanism	Resend timeout (ms) x Resend threshold (times)	Reconfiguration timeout (ms)
4.2.1	PULL	30×0	30
4.2.2	PUSH	30×0	30

Test Suite 5: Dynamic Heartbeat Interval

General Purpose

The purpose of this test is to measure the efficiency of the dynamic HB monitoring mechanism under high traffic in the network. Apart from the dynamic HB setting, the other test settings are the same as test suite 4.

In this mechanism, the monitors adjust the watchdog timers according to the duration between the simulation time of the previous and the current HB arrivals multiply by a slack factor as explained in subsection 4.6.3. At the beginning, where there is no previous HB arrival time, the monitor uses the static HB interval values and the default reconfiguration timeout. In contrast to the previous test suite, this test suite has a simulation time limit. The limit of 500000 ms has to be set because if dynamic HB is adjusted appropriately to the network traffic, the simulation will not end and the result will not be recorded.

General Settings

Table 4.9: Test suite 5 General configurations

Mechanism	PUSH
Number of Observers (nodes)	2
Heartbeat interval (ms)	Dynamic: starting at 100, 200, 500, 1000 Slack factor = 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
Reconfiguration timeout (ms)	120
Resend threshold	0
Node failures	None
Application traffic	Each node sends a packet every 50 ms Packet size (bytes) = current simulation time × packet factor Packet factor = 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180
Simulation time limit (ms)	500000

Chapter 5

Simulation Results and Evaluation

The first part of this chapter presents the test results from the five test suites in section 4.9 and the explanation of each test. The second part is the test analysis and the evaluation base on the OBC-NG monitoring requirements in section 3.1 and the performance indicators in section 4.3.

5.1 Test Results

Test Suite 1 Results: Monitoring Mechanisms and Observer Roles

Test 1.1 Results

Figure 5.1 combines the results of test 1.1.1 and 1.1.2 so that they can be compared to each other. The graph shows number of monitoring messages sent from each node in two different Observer role settings. In this configuration, node ID 1 is M, 2 is O1 and 3 is O2. The rest are Ws. In PULL mechanism, the monitoring messages are HB and ACK and in PUSH is PUSH HB.

In Observer role setting 1, in both PULL and PUSH, the highest monitoring workload is on M. This is also the case for PULL in Observer role setting 2, M has the highest workload comparing to the other nodes but lower than in setting 1. In contrast, for PUSH, O2 has highest monitoring workload instead of M.

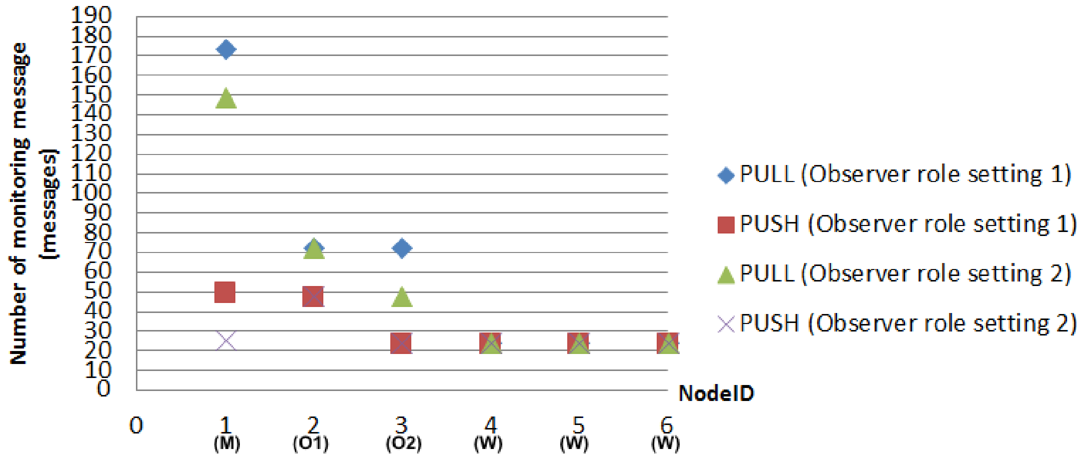


Figure 5.1: Number of monitoring messages sent per node in different monitoring mechanisms and settings

The network overhead of total monitoring messages of all nodes in each Observer role settings is calculated from the sum of the number monitoring messages multiply by its size as shown in message size table 2.1 in chapter 2. The HB and PUSH HB are 18 bytes and ACK is 27 bytes. Table 5.1 shows that the network overhead of PULL in Observer role setting 1 is 2.5 times higher than PUSH. Similarly, in Observer setting 2, the network overhead of PULL is 2.52 times higher than PUSH.

Table 5.1: Network overhead of PULL and PUSH mechanism with different Observer role settings

Observer role setting	Network overhead (bytes)	
	Mechanism	
	PULL	PUSH
1	8730	3492
2	7650	3041

In summary, in Observer role setting 2, where O2 observes only O1 and does not observers M, creates less network overhead and less workload on the M. Therefore, this setting is chosen for the following tests.

Test 1.2 Results

Figure 5.2 shows the bar graph of fault response time when each node fails. The x-axis of the generated graph is the node ID of M after the reconfiguration and y-axis is the fault response time. As can be seen in the graph, PUSH mechanism always has a higher response time than PULL. In this test, the average response time of PULL is 56 and PUSH is 66.

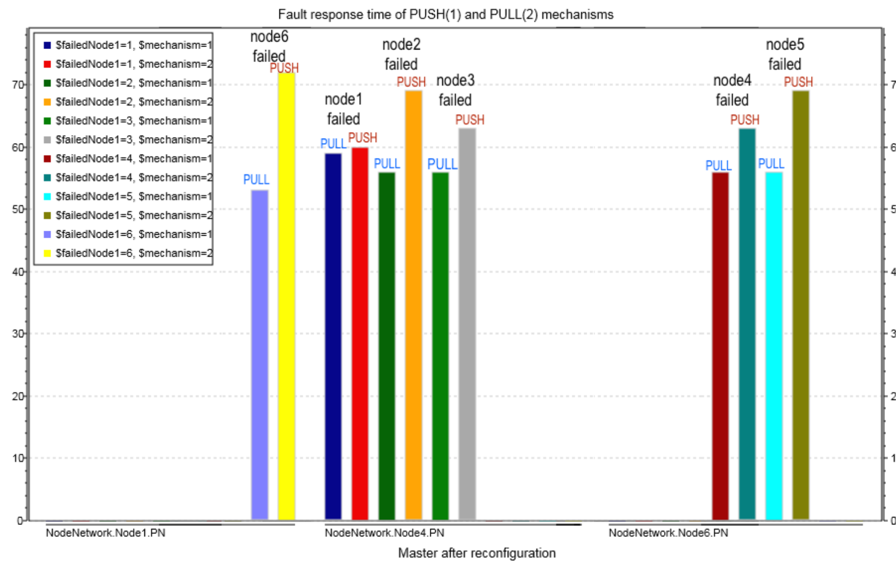


Figure 5.2: Fault response time of PULL and PUSH mechanisms when different nodes fail

Test Suite 2 Results: Number of Observers

The monitoring systems with one and two Os are simulated in this test. Table 5.2 and 5.3 below show that, in both monitoring mechanisms, the number of Observers does not affect the fault response time. In PULL, mechanism the fault response time of every HB interval are at 59 ms. In PUSH mechanism, the fault response time of 100, 200, 500 ms HB interval are at 60 ms and slightly lower in 1000 ms HB interval at 58 ms.

Table 5.2: Fault response time of PULL mechanism when M fails at 1000 ms

Heartbeat interval (ms)	Fault response time (ms)	
	Number of Observer (s)	
	1	2
100	59	59
200	59	59
500	59	59
1000	59	59

Table 5.3: Fault response time of PUSH mechanism when M fails at 1000 ms

Heartbeat interval (ms)	Fault response time (ms)	
	Number of Observer (s)	
	1	2
100	60	60
200	60	60
500	60	60
1000	58	58

Test Suite 3 Results: Fault Injection

Figure 5.3 shows an example of random fault injection times between 0 - 4000 ms of PULL mechanism with 100 ms HB interval. Its second-degree polynomial shows the trend of the graph, which is relatively flat, as the constant failure rate period of the bathtub curve mentioned in subsection 2.2.4. The random fault injection times of the other mechanisms and settings in this test also follow the same fashion since they are generated by the same random mechanism.

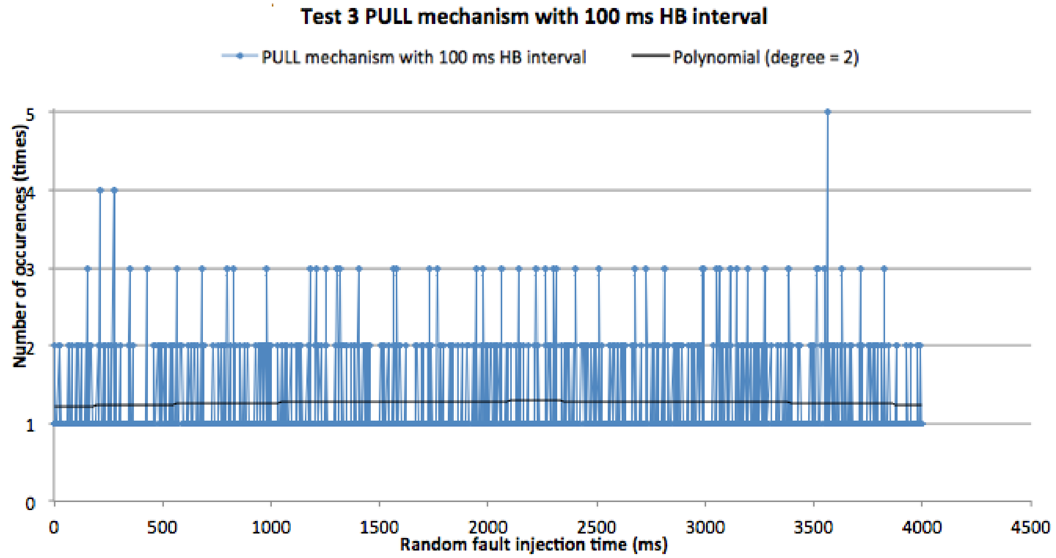
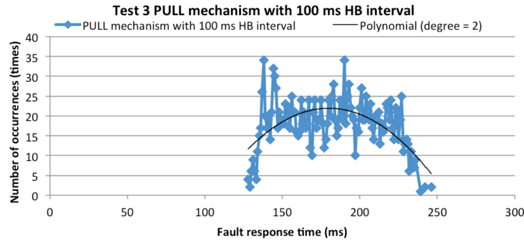


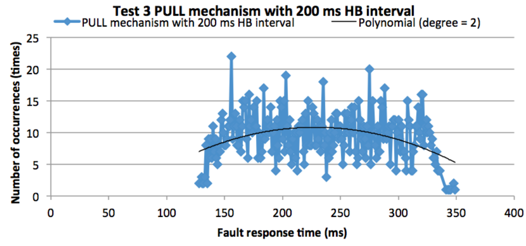
Figure 5.3: An example of random fault injection time of 2000 simulation runs

Figure 5.4 shows the fault response time and its number of occurrences of PULL mechanism with different HB intervals. The graphs are not flat as the fault injection time graph in figure 5.3 because the fault response time has lower range and higher number of occurrences than the fault injection. This is due to the different HB intervals. Comparing the subfigures of 5.4, 5.4(a), which has shortest HB interval, has the second-degree polynomial line of Gaussian bell curves with the highest height. The range of fault response time is the lowest and maximum number of occurrences is the highest. The height of the Gaussian bell decreases, the range of fault response times increases, maximum number of occurrences lowers in 5.4(b), 5.4(c), and 5.4(d), respectively.

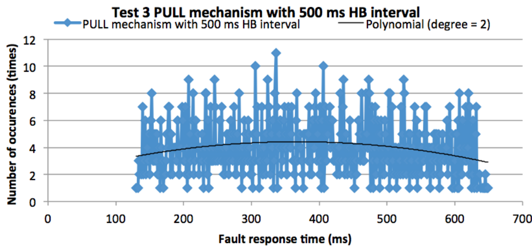
The fault response time graphs of PUSH mechanism in figure 5.5 and PUSH-PULL hybrid graphs 5.6 on the next page also follow the same trend as PULL mechanism, as mentioned previously.



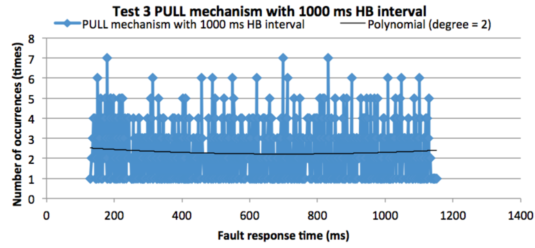
(a) 100 ms HB interval



(b) 200 ms HB interval

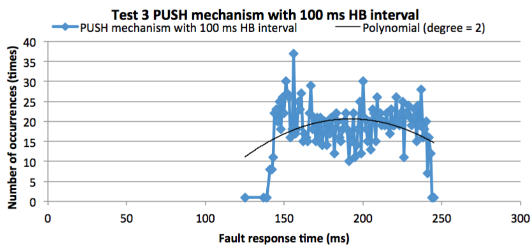


(c) 500 ms HB interval

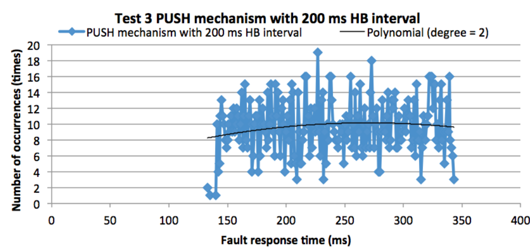


(d) 1000 ms HB interval

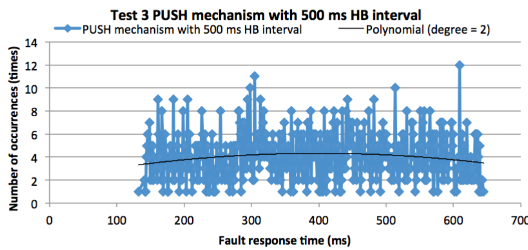
Figure 5.4: Fault response time and number of occurrences of PULL mechanism



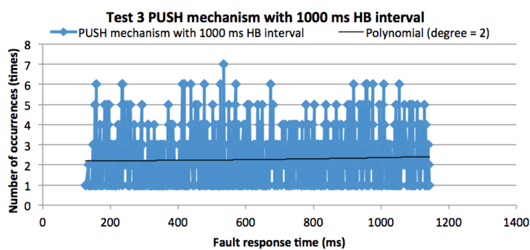
(a) 100 ms HB interval



(b) 200 ms HB interval

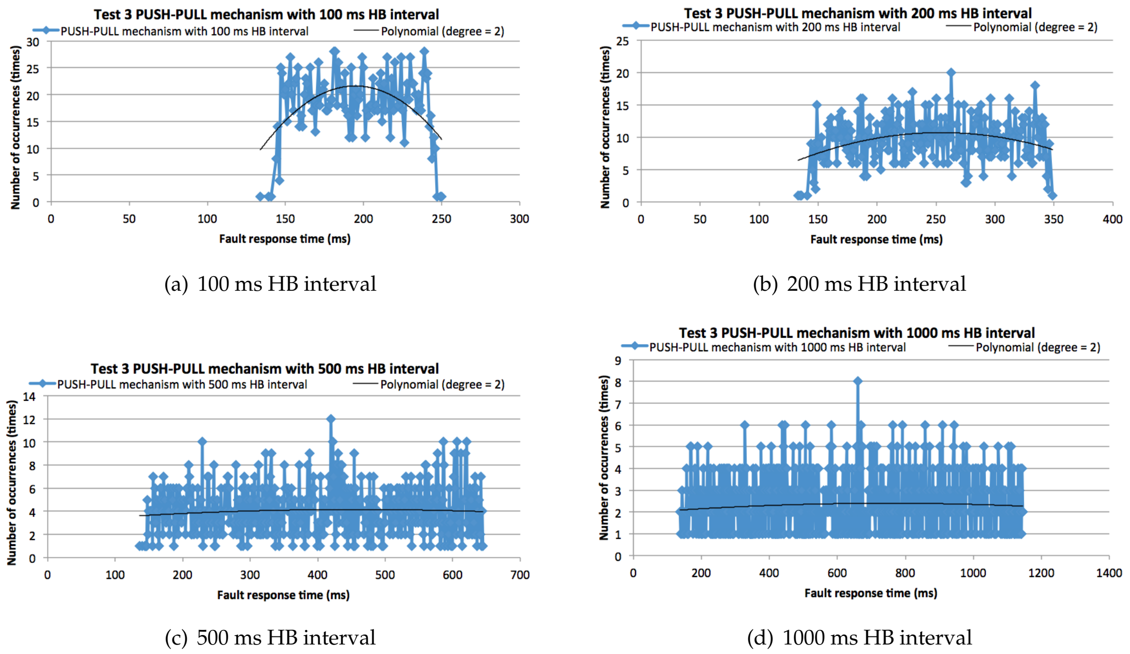


(c) 500 ms HB interval



(d) 1000 ms HB interval

Figure 5.5: Fault response time and number of occurrences of PUSH mechanism



(a) 100 ms HB interval

(b) 200 ms HB interval

(c) 500 ms HB interval

(d) 1000 ms HB interval

Figure 5.6: Fault response time and number of occurrences of PUSH-PULL mechanism

Table 5.4 shows detailed results of the test, which includes the minimum, maximum, mean, and range of the fault response times of different mechanisms and HB interval from 2000 simulation runs.

Table 5.4: Fault response time of a random failure in different mechanisms and HB intervals (ms)

HB interval (ms)	Mechanism											
	PULL				PUSH				PUSH-PULL			
	Min	Max	Mean	Range	Min	Max	Mean	Range	Min	Max	Mean	Range
100	128	246	181.80	118	125	245	191.62	120	134	250	194.92	116
200	128	349	233.19	221	133	343	242.66	210	133	349	245.86	216
500	130	649	384.20	519	132	645	392.36	513	136	645	395.70	509
1000	129	1151	633.95	1022	127	1143	653.49	1016	137	1145	650	1008
Mean (ms)	358.29				370.03				371.62			

Minimum Value

The minimum fault response time values of all mechanisms are between 128 -137 ms, which are slightly higher than the reconfiguration timeout (120 ms). This is because of there is the network latency of transferring HB, ACK, error notification and reconfiguration messages. From the test, PUSH-PULL has the highest minimum fault response time. The minimum fault response time is created when the fault is injected right before the ACK or PUSH HB from the monitored node is supposed to be sent out.

Maximum Value

From the observation of minimum fault response time, the assumption of the maximum fault response time is, it is created when the fault is injected right after the ACK or PUSH HB from the monitored node is sent. The expected fault response time is at the value of the full HB interval + reconfiguration timeout + some network latency as mentioned in minimum value analysis. However, the maximum values are 8 - 22 ms higher than the expected value. This is because, at the beginning of the simulation, nodes start the setting the timeout to send the first HB after the first reconfiguration is finished. Thus, if the failure is injected before the node finishes setting up, it takes longer to be detected since it has longer duration until the first ACK/PUSH HB is expected.

Range

The difference between the fault detection time ranges of each HB interval in the same mechanism is the approximate value of the HB interval.

Mean

The mean values of fault response time from the lowest to highest are of PULL, PUSH and PUSH-PULL mechanism. PULL and PUSH have bigger differences than PUSH and PUSH-PULL.

Test Suite 4 Results: Network Traffic

Test 4.1 Results

In figure 5.7, the message size and end simulation time of PULL mechanism with HB interval of 100 ms is shown to illustrate the application packets generation of the setting, which is the same as to the other HB interval settings. Each point on the line is, where an application packet is created, starting from the simulation time of 50 ms. The steepest line has the highest packet byte factor, as the message size gets larger faster and the line is shorter because it also runs into a

reconfiguration earlier. The other HB interval settings also follow this trend but the end of each line, which indicates the end simulation time, varies. In this setting, the end simulation time is in the range of 3133 - 6833 ms, which is earlier than the other two mechanisms.

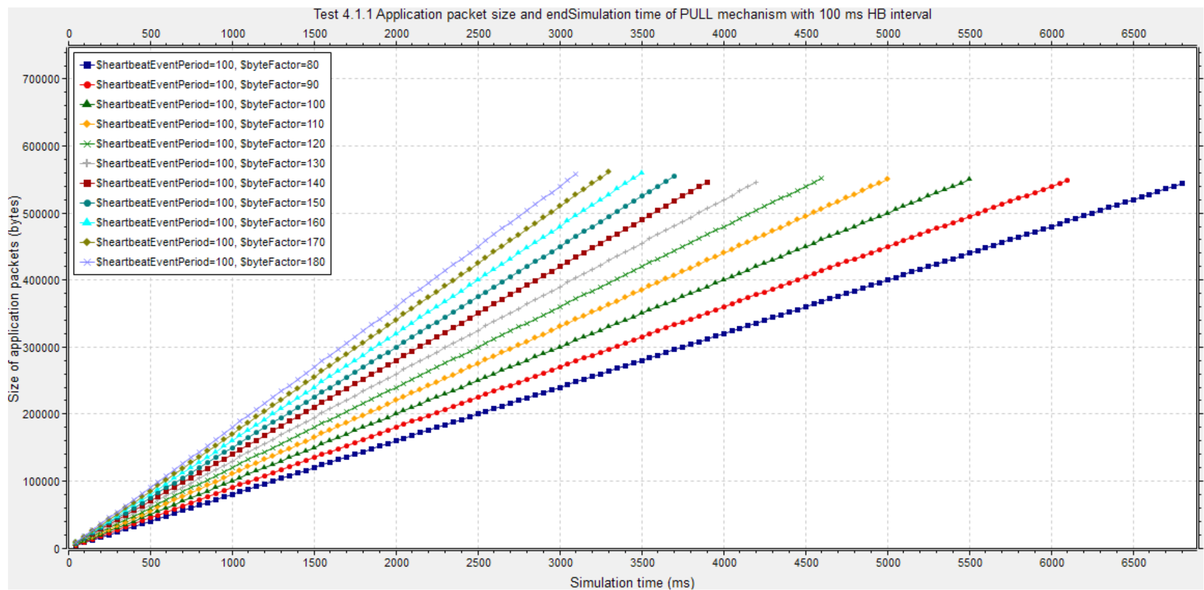


Figure 5.7: Application packet size and end simulation time of PULL mechanism with 100 ms HB interval

The three following figures are the results of the packet generation shown above. The end simulation time of the settings with the same packet byte factor of 80 bytes and different HB intervals of PULL, PUSH and PUSH-PULL hybrid, are presented.

The first figure, 5.8, shows that in PULL mechanism, the simulation of 100, 200, 500 and 1000 HB intervals ends at 6833, 6933, 7133 and 7133 respectively. Therefore, for PULL, the smaller HB interval, the lower the monitoring system stability.

In contrast to PULL model, figure 5.9 shows that in PUSH model, the settings with larger HB interval runs into wrong configuration earlier than the smaller HB interval. The simulation time of 100, 200, 500 and 1000 HB interval ends at 13880, 10588, 9258, and 9176 respectively. Unlike PULL in the previous test, for PUSH, the smaller the interval, the higher the stability of monitoring system. Even though both mechanisms have different trends, with the same mechanism settings, PUSH always have higher end simulation time.

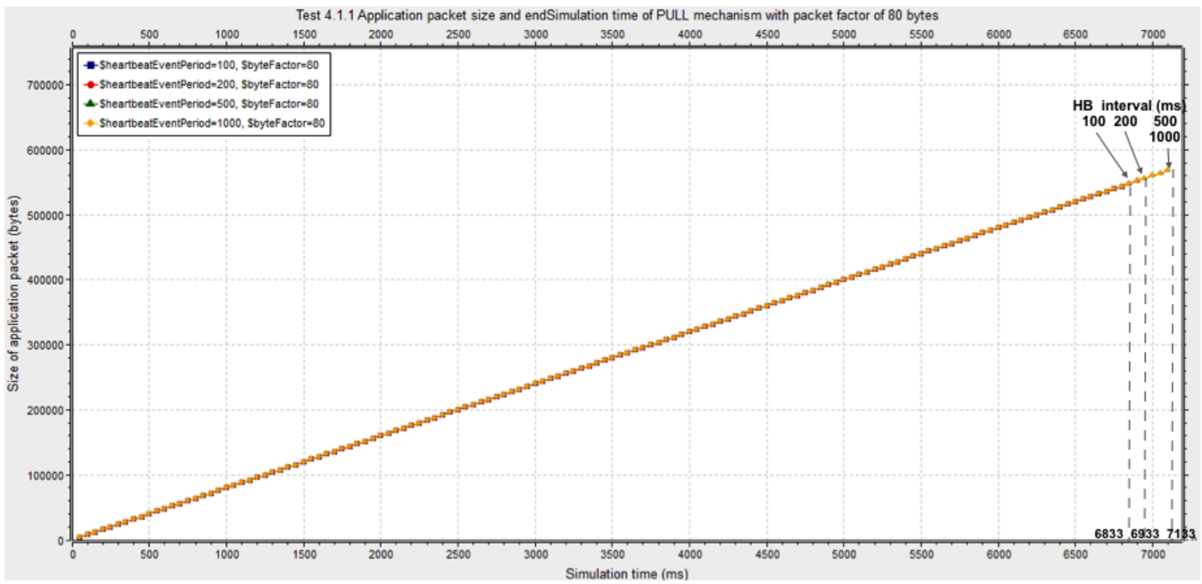


Figure 5.8: Application packet size and end simulation time of PULL mechanism with packet factor of 80 bytes

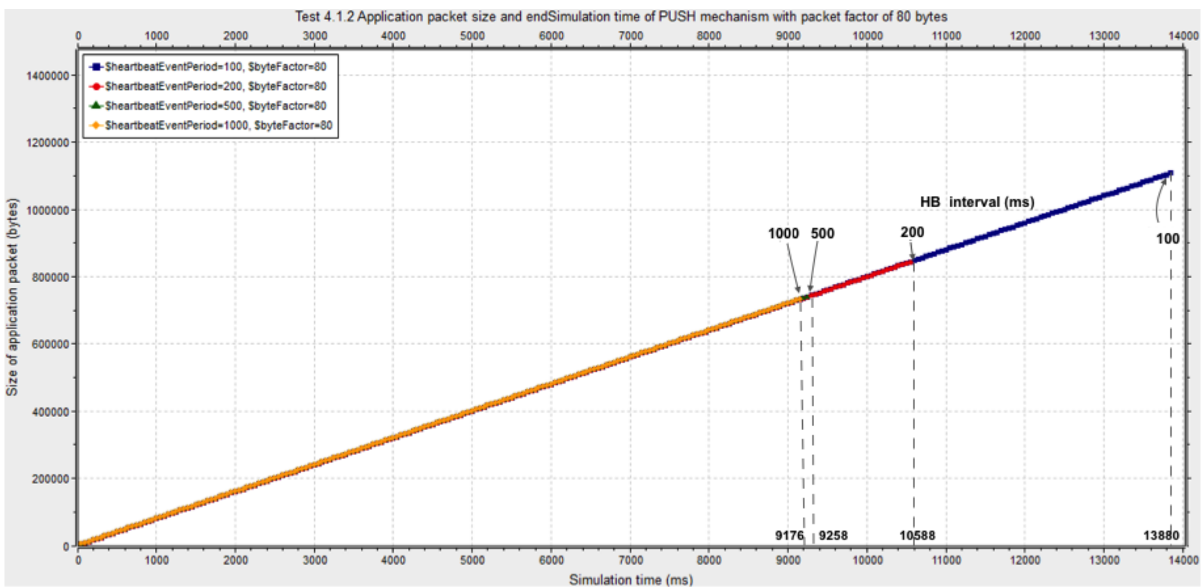


Figure 5.9: Application packet size and end simulation time of PUSH mechanism with packet factor of 80 bytes

Figure 5.10 shows that the end simulation time of PUSH-PULL hybrid is similar to PUSH in the previous figure. The settings with larger HB intervals run into wrong configuration earlier than the smaller ones. The simulation time of 100, 200, 500 and 1000 HB interval ends at 8592, 8587, 8680 and 7191, respectively and the values are between the values of PULL and PUSH mechanism.

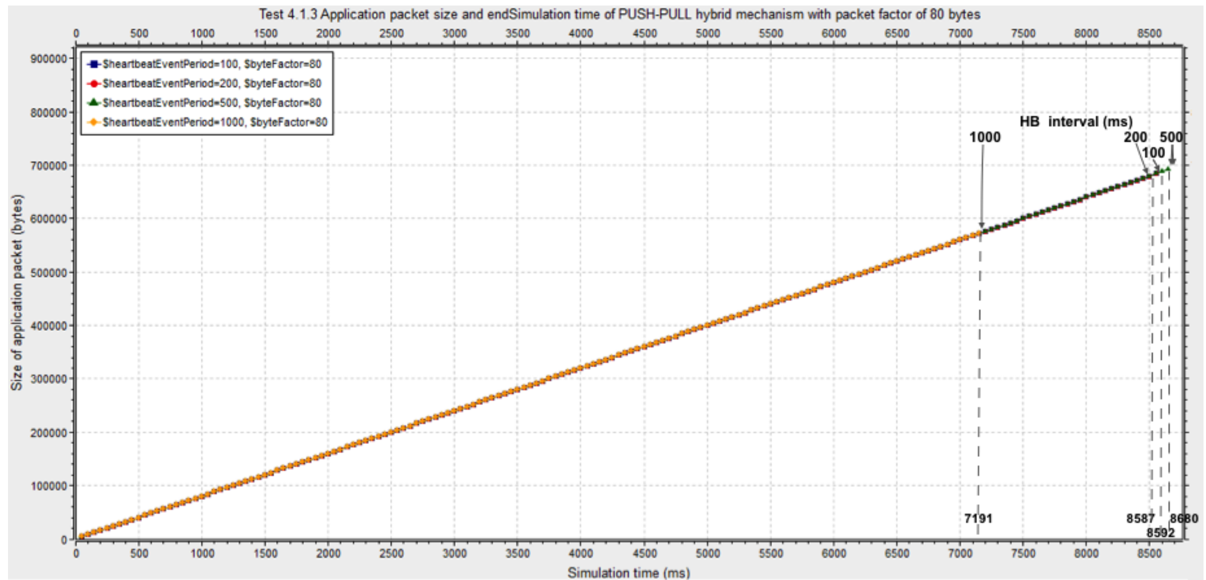


Figure 5.10: Application packet size and end simulation time of PUSH-PULL hybrid mechanism with packet factor of 80 bytes

The following graphs shows results of PULL, PUSH and PUSH-PULL models with every HB interval and packet byte factor to compare the packet size that each setting can handle without running into a wrong monitoring judgment.

Figure 5.11 shows the results of PULL model. The steepest lines, which have the factor of 180, the simulation ends earliest at 3133 ms. The simulation that run longest has HB interval of 500 and 1000 ms. In this test, no setting reaches 1.05 MB packet size, which is the current maximum data size of OBC-NG system.

In contrast to PULL, PUSH mechanism has higher end simulation time. Figure 5.12 shows that there are some settings of PUSH mechanism that reach 1.05 MB packet size before running into a wrong reconfiguration. Therefore, another graph in figure 5.13 is created with the minimum value of application packet of 1.05 MB to emphasize the settings that runs over the threshold.

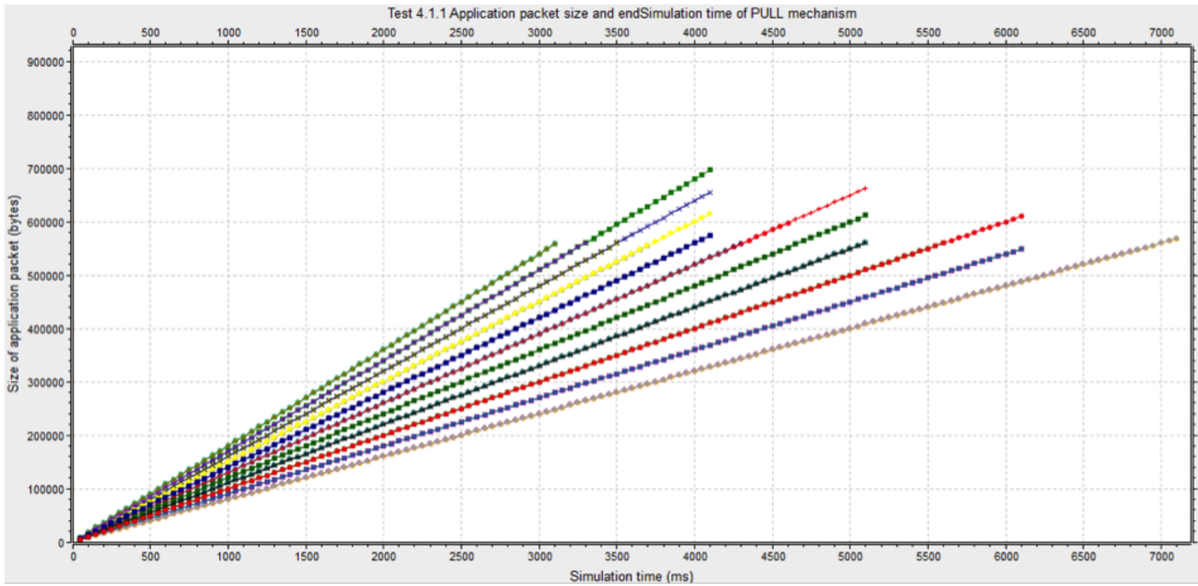


Figure 5.11: Application packet size and end simulation time of PULL mechanism

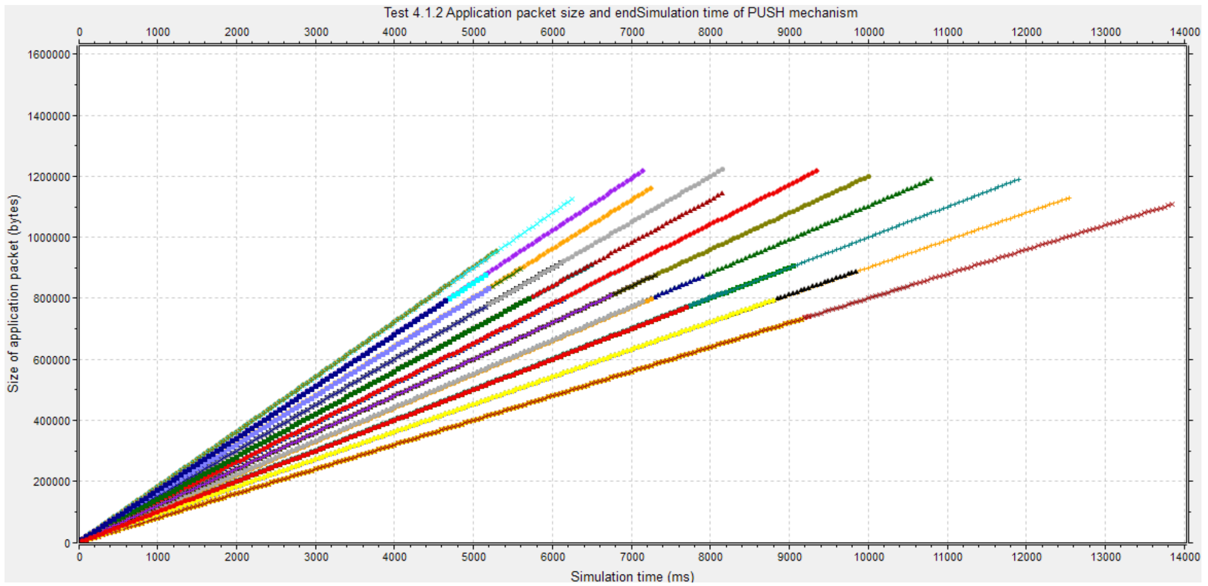


Figure 5.12: Application packet size and end simulation time of PUSH mechanism

The test results of PUSH mechanism that have their packet size generated over 1.05 MB before running into a wrong configuration are shown in figure 5.13 and they all have the 100 ms HB

interval. From this graph, the approximate simulation time, when packet is generated at the size of 1.05 MB, are recorded to use for evaluating the results of dynamic HB mechanism in the next test. For example, for packet factor 80 at the rightmost line, from 13200 ms onwards, the packet size generated is over 1.05 MB. Table 5.5 shows the record of the approximate values of simulation time when the packet reaches 1.05 MB.

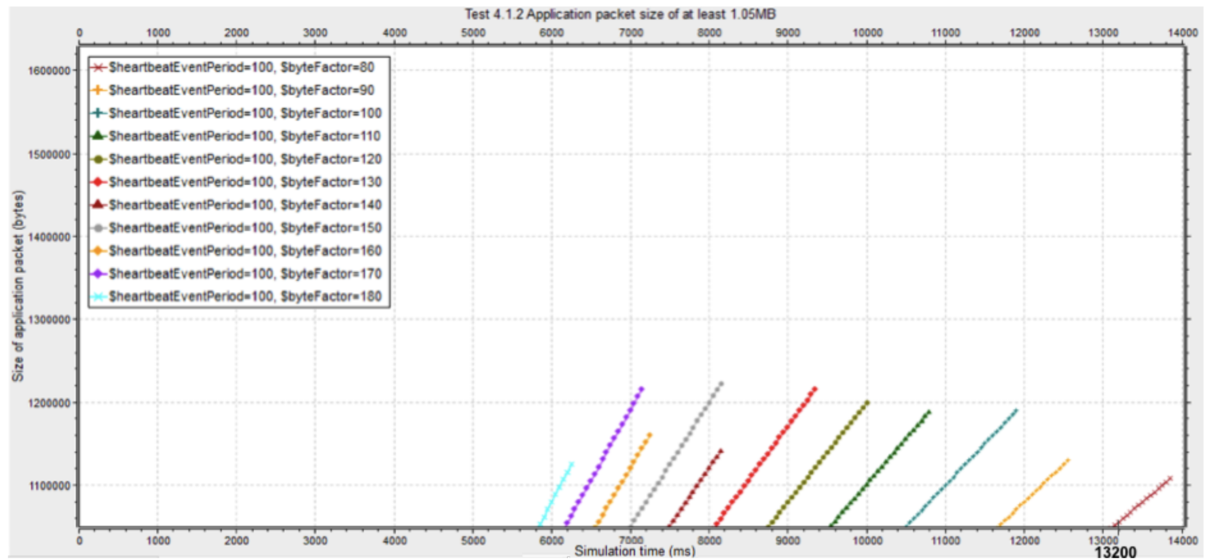


Figure 5.13: End simulation time of PUSH mechanism with the application packet size over 1.05 MB

Table 5.5: Approximate simulation time when the packet of size reaches 1.05 MB

Packet factor	Approximate simulation time (ms)
80	13200
90	11900
100	10600
110	9600
120	8800
130	8200
140	7600
150	7000
160	6600
170	6200
180	6000

The last figure of test 4.1, figure 5.14, shows the results of PUSH-PULL model. The end simulation time of each setting is between the value of PULL and PUSH model but does not reach 1.05 MB as well as PULL model.

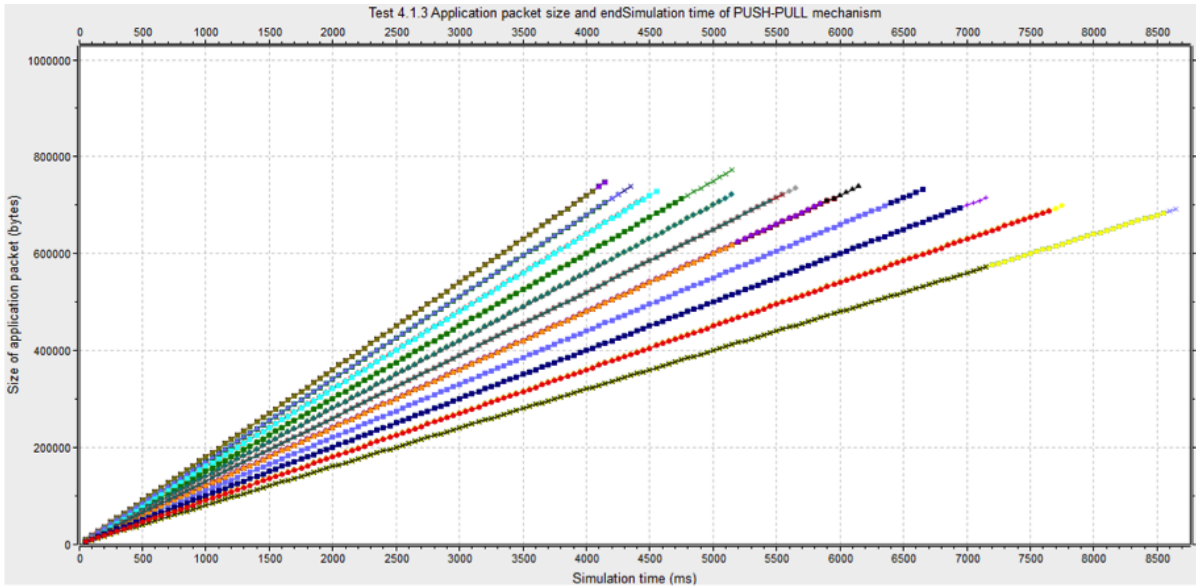


Figure 5.14: Application packet size and end simulation time of PUSH-PULL hybrid mechanism

Test 4.1 Summary The table 5.6 presents the average end simulation time of the three mechanisms with different HB intervals. PULL is more efficient with larger HB intervals. In contrast, PUSH and PUSH-PULL hybrid are mostly more efficient with smaller HB intervals except the case of PUSH-PULL with 100 ms HB interval, which has slightly earlier end simulation time than 200 ms HB interval.

Table 5.6: Average end simulation time of different mechanisms with different HB intervals

Mechanism	Average end simulation time (ms)			
	Heartbeat interval (ms)			
	100	200	500	1000
PULL	4451.18	4587.55	4723.73	4951.18
PUSH	9608	7310.64	6517.55	6646.91
PUSH-PULL	5800.64	5862.36	5680.18	5189.09

Test 4.2 Results

This test compares the end simulation time of PULL and PUSH model without resending mechanism. The result of PULL mechanism shows the same trend as test 4.1, the end simulation time is higher in higher HB interval. However, in PUSH it also follows the same trend as its result in test 4.1 with an exception that table HB interval of 100 ms has lower average end simulation time than 200 ms as seen in 5.7. This is similar to PUSH-PULL in test 4.1. The reason could be that HB interval of 100 ms is too frequent and the setting has no resend mechanism and reconfiguration timeout at only 30 ms and therefore the simulation ends early.

Table 5.7: Average end simulation time of PULL and PUSH mechanisms with different HB intervals

Mechanism	Average end simulation time (ms)			
	Heartbeat interval (ms)			
	100	200	500	1000
PULL	1221.82	1258.18	1403.64	1585.36
PUSH	5707.18	5768.36	5586.18	5095.09

Test Suite 5 Results: Dynamic Heartbeat Interval

From the previous test results, PUSH mechanism is more stable in high network load and has lower overhead. Therefore, it is chosen to implement the dynamic HB mechanism. The result table 5.8 is an example of the recorded output with packet factor of 80. The full table with every packet factor settings is shown in the appendix A.1. The result shows that dynamic HB mechanism increases the efficiency of the monitoring mechanism. Several tests, which are marked as $*$ in the table, run over 500000 ms simulation time.

In addition, there are also many tests that run over the point in simulation time that the application packet size reaches 1.05 MB. Those simulation time thresholds vary upon the packet byte factor and they are identified based on the result of the previous test in the table 5.5. The percentages of the settings that run over the threshold are recorded in the table 5.9. The table shows that the percentage of the run over the simulation time threshold in the settings of 200 and 500 are mostly higher than 100 and 1000 ms HB intervals.

Table 5.8: End simulation time of PUSH mechanism with packet factor of 80 ($*$ is the end simulation time over 500000 ms)

Packet factor	Slack factor	End simulation time (ms) Heartbeat interval (ms)			
		100	200	500	1000
80	0	316	616	1516	3016
	0.1	3906	4017	8611	9164
	0.2	3914	10570	22710	10549
	0.3	3922	20628	*	*
	0.4	21898	*	*	*
	0.5	21915	*	*	*
	0.6	21931	*	*	*
	0.7	29876	*	*	*
	0.8	*	*	*	*
	0.9	*	*	*	*

Table 5.9: Percentage of the tests with different packet factors that run over the threshold

Packet factor	Percentage of the simulation runs over the threshold (%) Heartbeat interval (ms)			
	100	200	500	1000
80	63.64	72.73	81.82	72.73
90	63.64	81.82	81.82	72.73
100	63.64	81.82	81.82	72.73
110	63.64	81.82	81.82	63.64
120	63.64	81.82	81.82	63.64
130	63.64	81.82	72.73	63.64
140	63.64	81.82	72.73	63.64
150	63.64	81.82	72.73	54.55
160	63.64	72.73	72.73	72.73
170	63.64	63.64	72.73	63.64
180	63.64	72.73	72.73	54.55

5.2 Result Evaluation

In this section, the results of different test suites in the previous section are summarized and compared. The analysis and evaluation is based on OBC-NG monitoring requirements and the performance indicators.

5.2.1 Monitoring Overhead

From the investigation of monitoring concepts in section2.4, PULL mechanism has two ways of monitoring messages and creates double the amount of overhead. However, the results of test 1.1.1 and 1.1.2 show that in OBC-NG system, PULL mechanism creates about 2.5 times higher overhead than PUSH mechanism. The reason is, a HB response in PULL is an ACK packet, which has the size of 27 bytes. Whereas PULL HB request and PUSH HB are 18 bytes. PUSH-PULL mechanism has the same overhead as PULL except the case that there is a failure and the resend mechanism is triggered.

Test 1.1.1 and 1.1.2, where O2 has different monitoring roles, are compared. The setting that O2 observes only O1 has less communication overhead. More importantly, monitoring workload of M in that setting is reduced.

5.2.2 Fault Response Time

The result of test 1.2 shows that the fault response time of PUSH mechanism is slightly higher than PULL mechanism.

In test 2, the fault response times of the settings with one and two Observers when M fails are the same. No matter which O detects the failure of M first, the reconfiguration is triggered by the higher priority O, if it is alive, to prevent race condition. Race condition occurs when two nodes select the configurations and trigger the reconfigurations at the same time. Even though the number of Observers does not decrease the fault response time, its redundancy is important for increasing the reliability of the monitoring system.

Test 3's result supports test 1.2 that the fault response time of PUSH mechanism is slightly higher than PULL mechanism. In addition, it gives more detail about the components of fault response time that it is depending on:

- The duration between fault injection and when the next ACK or PUSH HB is expected from the monitored node.
- The reconfiguration timeout

- Network latency to deliver the reconfiguration, error notification, HB and ACK messages. This is depending on the number of hops between the monitor and the monitored node.

Minimum value of fault response time is from the case that the fault injection occurs right at the time when a HB or an ACK is expected to be sent out. Therefore, the value is the reconfiguration timeout plus the network latency. **Maximum value**, however, includes the time from the failure to the first reconfiguration is finished if the failure occur before it is finished. The duration is added to the value of full HB interval + reconfiguration timeout + network latency, which is the case when fault injection occurs right after the time a HB or an ACK is sent. The **range** between maximum and minimum varies upon the HB interval.

Heartbeat Interval and Reconfiguration Timeout The settings with smaller HB interval, which have higher HB frequency, allow the missing HB to be detected earlier. The reconfiguration timeout needs to be set up to make sure the monitored node has failed to sent a HB. If the reconfiguration timeout is too short, the risk of misinterpreting a delayed HB as a node failure is higher. However, if the reconfiguration timeout is longer, the fault response time increases.

However, small HB intervals are not suitable for PULL mechanism since the monitoring messages are in two ways (request and response) so it has higher risk of losing one of the messages.

The previous assumption of PUSH mechanism was that the resend mechanism cannot be implemented and in order to implement it, the monitoring systems have to switch to PULL mechanism. However, the HB interval and reconfiguration timeout of PUSH mechanism can be set up in a way that it waits for the same amount of PUSH HB as the resend threshold. For example, with 100 ms HB interval, and 400 ms reconfiguration timeout. After a missing HB, the monitored node passes three HB intervals before the reconfiguration is triggered.

5.2.3 Monitoring System Stability

In test 4, network traffic is generated to test the monitoring system stability. For static HB interval, different mechanisms with the same reconfiguration timeout of 120 ms have different the end simulation time. The result of test 4.1 shows that PULL tends to run into a wrong configuration and end the simulation earliest because of its two ways of communication. PUSH mechanism has the higher end simulation time because it has one way of monitoring message and PUSH-PULL is in the middle because in the end, PULL is implemented as a resend mechanism. To analyze the mechanisms, PULL and PUSH mechanism without resend mechanisms are simulated in test 4.2 and the results confirm that PUSH mechanism is more stable than PULL, when there is high network load.

HB interval settings affect the stability differently. From test 4.1 and 4.2, PULL is more stable with the larger HB intervals and PUSH is more stable with the smaller intervals.

The results of dynamic HB interval mechanism in test 5 show that it increases the stability of every mechanism settings. However, 100 and 1000 ms HB interval shows lower efficiency than 200 and 500. The result of 100 ms HB interval supports the observation grade-heartbeat (dynamic HB) experiment in subsection 2.4.3 that, frequent changes in HB interval result in the decline of the monitoring system efficiency. In the case of 1000 ms HB interval, the adjustment is not frequent enough when packet factor is high and the application packet size increases steeply.

Chapter 6

Conclusion and Future Work

In this work, various monitoring mechanisms of distributed system are investigated to design the monitoring system of OBC-NG. Three mechanisms, PULL, PUSH and PUSH-PULL hybrid, are chosen to be tested and analyzed to find the one that has high efficiency, low overhead and high stability. The monitoring mechanism testing is divided into five test suites and the concluded results are shown in section 6.1. Finally, section 6.2 presents the future outlook how the designs and the results could be used for the further development of the OBC-NG system.

6.1 Final Results

Monitoring Mechanism

PULL, PUSH and PUSH-PULL mechanisms are tested with different mechanism settings and environment settings to compare the monitoring efficiency, overhead and stability. The monitoring models have been evaluated base on the OBC-NG monitoring requirements.

The monitoring mechanisms and mechanism settings have been analyzed as follows: The currently implemented model of **PULL** has the highest monitoring workload on M and the highest network overhead. Adjusting the role of Observer setting decreases the workload on the M, but the monitoring network load is still higher than in the other two mechanisms.

PUSH mechanism has the lowest monitoring overhead, 2.5 times lower than PULL. Its fault response time is slightly higher than PULL but it has a lot higher stability of the monitoring, when there is high network load.

Mechanism Settings

The higher **number of Observers** does not affect the fault response time but the reliability. The **HB interval** should be set differently according to the mechanism. Larger HB intervals are more efficient in PULL model and smaller intervals are better in PUSH and PUSH-PULL models. The results show the range of the fault response time of each HB interval. However, the exact value of a HB interval should be mission-specifically configured, using the range of the fault response time measured from the test and in the mission requirements. Lastly, the higher **reconfiguration timeout** decreases the probability of false monitoring judgment without the requirement of resend mechanism.

Dynamic Heartbeat Mechanism

Base on the test results mentioned above, PUSH has been chosen for implementing dynamic HB mechanism. The mechanism increases the stability of every HB interval setting and decreases the false monitoring judgment. However, the default value of HB interval at the beginning and the slack factor, to calculate the next interval, are important. If the HB interval is too short, the monitoring efficiency decreases because the frequency of the HB interval adjustments is high. In contrast to that, if the HB interval is too long, the HB interval adjustments are not frequent enough, in case of high network traffic in the system. Therefore, the HB frequency and slack factor should be chosen base on the maximum size of packets and the assumption of the frequency of application packets.

6.2 Future Work

In the future, PUSH mechanism, which is selected in the final results as the most efficient mechanism can be integrated in the Scalable on-board Computing for Space Avionics (ScOSA) project. This project is the successor of the OBC-NG and has similar design. It is also a distributed, scalable system with interconnected nodes, whereas nodes can be COTS as well as radiation-hardened components.

The traffic sensing mechanism of PUSH dynamic HB, which is used for adjusting the watchdog timer, implicitly informs the M about the current traffic load on the network. This enables the M to inform other W nodes that they shall prioritize their output on the network. To derive a correct verdict, the M needs to be positioned near or in the center of the network.

Heartbeat monitoring used in this work is a mechanism for implicit monitoring and the failure is detected by the monitor. In the future, explicit monitoring can be integrated in the form of an agent-based system, in which an application can use an agent service on the node to inform the M

and the monitoring system about a certain inconsistency. The M and the monitoring system then might have more time to evaluate the system health and to react in case required.

Moreover, the functionality of the monitor can be increased to evaluate the system health status, in case a missing HB or other negative symptoms are sensed. After a missing HB, during the reconfiguration timeout, the monitor has the possibility to prepare itself and the other nodes into a *pre-reconfiguration state*. Tasks, such as searching for the next configuration in the decision tree and the pre-distribution of checkpoint values for warm redundancy, can be conducted.

When a reconfiguration has to be triggered, the current mechanism reconfigures the entire system including the healthy nodes. In the future, partial reconfiguration can be designed to isolate the failed node and redistributing its tasks without interrupting the tasks of the healthy nodes.

Furthermore, monitoring can be conducted in a finer grained level, such as task level, instead of node level. When a failure occurs, a specific task is migrated to another node instead of migrating all the tasks. This requires a more detailed knowledge of the monitoring system, regarding the faults, which can be provided by a more sophisticated error messaging.

Bibliography

[1] D. P. Siewiorek and P. Narasimhan, “Fault-tolerant architectures for space and avionics applications,” electrical and Computer Engineering Department Carnegie Mellon University.

[2] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T.Peng, B. Weps, G. Fey, and A. Gerndt, “OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft,” in *2014 IEEE Aerospace Conference*, 2014, pp. 1–13.

[3] S. Parkes, *SpaceWire User's Guide*. STAR-Dundee Limited, 2012.

[4] H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*, 2nd ed. Springer US, 2011.

[5] G. A. Klutke, P. C. Kiessler, and M. A. Wortman, “A critical look at the bathtub curve,” *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 125–129, March 2003.

[6] M. Mansouri-Samani and M. Sloman, “Network and distributed systems management,” M. Sloman, Ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994, ch. Monitoring Distributed Systems, pp. 303–347. [Online]. Available: http://dl.acm.org/citation.cfm?id=184430.184469

[7] M. Technet. (2016) How heartbeats work in operations manager. [Online]. Available: https://technet.microsoft.com/en-us/library/hh212798.aspx

[8] Y. Wan, D. Feng, T. Yang, Z. Deng, and L. Liu, “The adaptive heartbeat design of high availability raid dual-controller,” in *Multimedia and Ubiquitous Engineering, 2008. MUE 2008. International Conference on*, April 2008, pp. 45–50.

[9] S. W. Heinen, S. Reid, “Automation through on-board control procedures: Operational concepts and tools,” american Institute of Aeronautics and Astronautics.

[10] M. Lauer, U. Herfort, D. Hocken, and S. Kielbassa, “Optical measurements for the flyby navigation of rosetta at asteroid steins,” in *Proceedings 21st International Symposium on Space Flight Dynamics–21st ISSFD. Toulouse, France*, 2009.

[11] T. Peng, K. Höflinger, B. Weps, O. Maibaum, K. Schwenk, D. Lüdtke, and A. Gerndt, "A Component-Based Middleware for a Reliable Distributed and Reconfigurable Spacecraft Onboard Computer," in *35th Symposium on Reliable Distributed Systems (SRDS).*

[12] S. M. Parkes and P. Armbruster, "Spacewire: a spacecraft onboard network for real-time communications," in *14th IEEE-NPSS Real Time Conference, 2005.*, June 2005, pp. 6–10.

[13] S. Parkes, *Supporting SpaceWire Applications*, STAR-Dundee Ltd, School of Computing, University of Dundee Dundee, DD1 4HN, Scotland, UK.

[14] A. Hamed and K. Jaber, "Analysis of quality of service in cloud storage systems," *IJFCST International Journal in Foundations of Computer Science Technology*, vol. 4, no. 6, p. 71–77, 2014.

[15] S. Kapurch, *NASA Systems Engineering Handbook*. DIANE Publishing Company, 2010. [Online]. Available: https://books.google.de/books?id=2CDrawe5AvEC

[16] M. Kooli and G. D. Natale, "A survey on simulation-based fault injection tools for complex systems," in *Design Technology of Integrated Systems In Nanoscale Era (DTIS), 2014 9th IEEE International Conference On*, May 2014, pp. 1–6.

[17] I. Eusgeld, F. Freiling, and R. Reussner, *Dependability Metrics: GI-Dagstuhl Research Seminar, Dagstuhl Castle, Germany, October 5 - November 1, 2005, Advanced Lectures*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008. [Online]. Available: https://books.google.de/books?id=HX5rCQAAQBAJ

[18] J. Azambuja, F. Kastensmidt, and J. Becker, *Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors*, ser. SpringerLink : Bücher. Springer International Publishing, 2014. [Online]. Available: https://books.google.de/books?id=b1clBAAAQBAJ

[19] D.-M. Kwai and B. Parhami, "Fault-tolerant processor arrays using space and time redundancy," in *Algorithms amp; Architectures for Parallel Processing, 1996. ICAPP 96. 1996 IEEE Second International Conference on*, Jun 1996, pp. 303–310.

[20] NASA. Mean time to repair predictions. [Online]. Available: http://llis.nasa.gov/lesson/840

[21] J. Joyce, G. Lomow, K. Slind, and B. Unger, "Monitoring distributed systems," *ACM Trans. Comput. Syst.*, vol. 5, no. 2, pp. 121–150, Mar. 1987. [Online]. Available: http://doi.acm.org/10.1145/13677.22723

[22] C.-f. Hsin and M. Liu, "A distributed monitoring mechanism for wireless sensor networks," in *Proceedings of the 1st ACM Workshop on Wireless Security*, ser. WiSE '02. New York, NY, USA: ACM, 2002, pp. 57–66. [Online]. Available: http://doi.acm.org/10.1145/570681.570688

[23] J. X. Zou, Z. Q. Zhang, and H. B. Xu, "Design of heartbeat invalidation detecting mechanism in triple module redundant multi-machine system," in *2009 IEEE Circuits and Systems International Conference on Testing and Diagnosis*, April 2009, pp. 1–4.

[24] F. V. Brasileiro, P. D. Ezhilchelvan, S. K. Shrivastava, N. A. Speirs, and S. Tao, "Implementing fail-silent nodes for distributed systems," *IEEE Transactions on Computers*, vol. 45, no. 11, pp. 1226–1238, Nov 1996.

[25] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A lazy monitoring approach for heartbeat-style failure detectors," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, March 2008, pp. 404–409.

[26] K. Rachuri, A. A. Franklin, and C. S. R. Murthy, *Coverage Based Expanding Ring Search for Dense Wireless Sensor Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 245–256. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89894-8_24

[27] H. Huang and L. Wang, "P amp;p: A combined push-pull model for resource monitoring in cloud computing environment," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 260–267.

[28] Y. Zhao, "A model of computation with push and pull processing," Technical Memorandum Electronics Research Laboratory University of California at Berkeley, dec 2003.

[29] A. Vargas, "Omnet++ user guide version 4.6," 2014.

[30] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60:1–60:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416290

Appendix A

Appendix Table

A.1 Test Suite 5 Results

Table A.1: End simulation time of PUSH mechanism with different packet factors and slack factors (∗ is the end simulation time over 500000 ms)

Packet factor	Slack factor	End simulation time (ms)			
		Heartbeat interval (ms)			
		100	200	500	1000
	0	316	616	1516	3016
	0.1	3906	4017	8611	9164
	0.2	3914	10570	22710	10549
	0.3	3922	20628	*	*
	0.4	21898	*	*	*
80	0.5	21915	*	*	*
	0.6	21931	*	*	*
	0.7	29876	*	*	*
	0.8	*	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
	0	316	616	1516	3016
	0.1	3506	3617	8182	8163
	0.2	3514	14694	*	9530
	0.3	3522	14721	*	*
	0.4	14563	54947	*	*
90	0.5	14576	*	*	*

	0.6	14590	*	*	*
	0.7	27004	*	*	*
	0.8	*	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
	0	316	616	1516	3016
	0.1	3205	3415	7117	8286
	0.2	3213	15936	*	8391
	0.3	3220	15967	*	*
	0.4	15939	*	*	*
100	0.5	15954	*	*	*
	0.6	15969	*	*	*
	0.7	26587	*	*	*
	0.8	*	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
	0	316	616	1516	3016
	0.1	2906	3017	6568	7178
	0.2	2914	20440	21748	7280
	0.3	2922	20503	*	8877
	0.4	15532	*	*	*
110	0.5	17238	*	*	*
	0.6	17255	*	*	*
	0.7	20124	*	*	*
	0.8	*	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
	0	316	616	1515	3016
	0.1	2705	2817	6069	7344
	0.2	2713	14398	14771	7453
	0.3	2720	14426	*	7562
	0.4	14229	14453	*	*
120	0.5	14241	14481	*	*
	0.6	14252	*	*	*
	0.7	14263	*	*	*
	0.8	14275	*	*	*
	0.9	14286	*	*	*
	1	14298	*	*	*

	0	316	616	1515	3016
	0.1	2506	2617	5614	6173
	0.2	2514	12534	6292	6275
	0.3	2522	32011	*	7874
	0.4	13172	*	*	*
130	0.5	13184	*	*	*
	0.6	13195	*	*	*
	0.7	13206	*	*	*
	0.8	13218	*	*	*
	0.9	13229	*	*	*
	1	13241	*	*	*
	0	316	616	1514	3016
	0.1	2306	2418	5110	6169
	0.2	2314	9353	5759	6368
	0.3	2322	9379	*	6472
	0.4	9357	12616	*	*
140	0.5	9370	12644	*	*
	0.6	9383	*	*	*
	0.7	18659	*	*	*
	0.8	26521	*	*	*
	0.9	26556	*	*	*
	1	26591	*	*	*
	0	316	616	1513	3016
	0.1	2205	2415	5077	5173
	0.2	2213	8968	5186	6545
	0.3	2220	8995	*	6657
	0.4	12204	*	*	6804
150	0.5	12221	*	*	*
	0.6	12238	*	*	*
	0.7	17763	*	*	*
	0.8	44916	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
	0	316	616	1516	3015
	0.1	2007	2217	4612	5177
	0.2	2015	4457	5287	5279
	0.3	4342	8289	*	6896
	0.4	8272	*	*	7019
160					

	0.5	8285	*	*	*
	0.6	8299	*	*	*
	0.7	14134	*	*	*
	0.8	213941	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
170	0	316	616	1516	3014
	0.1	1907	2020	4580	5142
	0.2	1915	4494	4688	5301
	0.3	4146	5155	*	5404
	0.4	9775	10482	*	7386
	0.5	9791	10510	*	*
	0.6	9806	*	*	*
	0.7	10292	*	*	*
	0.8	10303	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*
180	0	316	616	1516	3013
	0.1	1807	2017	4113	5187
	0.2	1815	4065	4778	5392
	0.3	3947	9358	*	5497
	0.4	9333	*	*	5603
	0.5	9349	*	*	*
	0.6	9365	*	*	*
	0.7	9687	*	*	*
	0.8	9699	*	*	*
	0.9	*	*	*	*
	1	*	*	*	*