IMPLEMENTATION AND EVALUATION OF THE DEPENDABILITY PLANE FOR THE DYNAMIC DISTRIBUTED DEPENDABLE REAL TIME INDUSTRIAL PROTOCOL (D^3RIP)

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖMER BERAT SEZER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2013

Approval of the thesis:

IMPLEMENTATION AND EVALUATION OF THE DEPENDABILITY PLANE FOR THE DYNAMIC DISTRIBUTED DEPENDABLE REAL TIME INDUSTRIAL PROTOCOL (D^3RIP)

submitted by **ÖMER BERAT SEZER** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by**,**

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Assoc.Prof. Dr. Şenan Ece Schmidt _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

Assoc. Prof. Dr. Klaus Werner Schmidt _____
Co-Supervisor, **Dept., of Mechatronics Eng., Çankaya U.**

Examining Committee Members:

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Şenan Ece Schmidt _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt Bazlamaççı _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün _____
Computer Engineering Dept., METU

Yusuf Bora Kartal _____
M.Sc. ASELSAN A.Ş

Date:
_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÖMER BERAT SEZER
Signature:

ABSTRACT

IMPLEMENTATION AND EVALUATION OF THE DEPENDABILITY PLANE
FOR THE
DYNAMIC DISTRIBUTED DEPENDABLE REAL TIME INDUSTRIAL PROTOCOL
(D^3RIP)

Sezer, Ömer Berat

M.Sc., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Şenan Ece Schmidt
Co-Supervisor : Assoc. Prof. Dr. Klaus Werner Schmidt

September 2013, 93 pages

Dynamic Distributed Dependable Real Time Ethernet Industrial Protocol (D^3RIP) is a real time industrial communication protocol that runs over shared-medium Ethernet with COTS hardware. The protocol consists of an interface layer that enables time slotted communication and a coordination layer that guarantees collision avoidance and timely delivery of real time messages generated by the control application. At the current development stage, these two layers of the protocol are fully implemented and tested. The scope of this thesis is the implementation of a new plane for D^3RIP to achieve dependability. To this end, mechanisms of fault detection and roll back recovery are applied. The interface of the dependability plane to the existing interface layer and coordination layer is defined. Finally the dependability plane is implemented and integrated to the existing protocol stack. A number of tests under different fault scenarios are conducted to demonstrate the plane functionality.

Keywords: Ethernet, industrial communication network, real time industrial communication

ÖZ

DİNAMİK DAĞITILMIŞ GÜVENİLİR GERÇEK ZAMANLI ENDÜSTRİYEL PROTOKOLÜ (D^2G^2EP) İÇİN GÜVENİLEBİLİRLİK DÜZLEMİ GERÇEKLENMESİ VE DEĞERLENDİRİLMESİ

Sezer, Ömer Berat

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Şenan Ece Schmidt
Ortak Tez Yöneticisi : Doç. Dr. Klaus Werner Schmidt

Eylül 2013, 93 sayfa

Dinamik Dağıtılmış Güvenilir Gerçek Zamanlı Endüstriyel Protokolü (D^2G^2EP), (COTs) orjinal donanımıyla ortam paylaşımlı Ethernet üzerinde çalışan gerçek zamanlı endüstriyel haberleşme protokolüdür. Protokol zaman oluklu iletişimi sağlayan arayüz katmanından (AK) ve kontrol uygulaması tarafından üretilen gerçek zamanlı mesajların iletimini ve çakışmayı önlemeyi garantileyen koordinasyon katmanından (KK) oluşur. Mevcut geliştirme aşamasında, bu iki protokol katmanı eksiksiz gerçeklenmiş ve test edilmiştir. Bu tezin kapsamı D^2G^2EP'in güvenirliliğini sağlayan yeni bir düzlem uygulamasıdır. Bu amaçla, hata belirleme ve hata öncesi duruma geri döndürme mekanizmaları oluşturulmuştur. Varolan arayüz katmanı ve koordinasyon katmanı için güvenirlilik düzlemi arayüzü tanımlanmıştır. Son olarak güvenirlilik düzlemi uygulanmış ve varolan yapıya entegre edilmiştir. Farklı hata senaryolarına göre bir çok test gerçekleştirilmiş ve düzlemin işlevselliği gösterilmiştir.

Anahtar Kelimeler: Ethernet, endüstriyel iletişim ağları, gerçek zamanlı endüstriyel haberleşme

To My Family

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Associate Prof. Dr. Şenan Ece Schmidt and my co-supervisor Associate Prof. Dr. Klaus Werner Schmidt for their valuable supervision and support. In addition to this, I like to thank them for giving me an opportunity to study industrial communication protocols. I also thank my colleagues Adem Kaya and Yusuf Bora Kartal for their contribution on integration of my thesis work to the system and system tests. My thesis was a part of a research project that was funded by The Scientific and Technological Research Council of Turkey (TUBITAK). I would like to thank TUBITAK for their project support.

I would like to thank my family, my wife and my colleagues in TUBITAK-UZAY for their support.

TABLE OF CONTENTS

ABSTRACT v
ÖZ vi
ACKNOWLEDGEMENTS viii
TABLE OF CONTENTS ix
LIST OF TABLES xi
LIST OF FIGURES xii
INTRODUCTION 1
BACKGROUND 3
2.1 Real -Time Ethernet for Industrial Communication Protocols 3
2.1.1 Requirements 5
2.1.2 Real Time Ethernet Protocols 6
2.2 Dependability 10
PREVIOUS WORK 15
3.1 Dynamic Distributed Dependable Real Time Industrial Protocol (D^3RIP) Protocol Overview 15
3.2 D^3RIP Formal Protocol Model: 17
3.2.1 Generic Interface Layer: 17
3.2.2 Generic Coordination Layer: 20
3.2.3 Generic Shared Medium Model: 23
3.2.4 Generic Dependability Plane Model: 24
3.3 D^2RIP Implementation 27
3.3.1 Interface Layer (IL) : 28
3.3.2 Coordination Layer (CL): 34
3.3.3 D^2RIP Implementation Summary and Its Operation 37
DEPENDABILITY PLANE IMPLEMENTATION 41
4.1 Overview 41
4.1.1 DP Implementation 42
4.2 Data Structures 52
4.3 Actions and Operation 54
4.3.1 Actions 54

4.3.2 Operations 55
EVALUATION OF THE DEPENDABILITY PLANE 59
5.1 Example Description 59
5.2 Performance Parameters 66
5.3 Experiments and Results 69
CONCLUSION & FUTURE WORK 73
6.1 Conclusion 73
6.2 Future Work 74
REFERENCES 75
APPENDIX 79
XML FILES 79

LIST OF TABLES

TABLES

Table 1:Shared Medium Industrial Ethernet Protocol.....10
Table 2: Frame Header Structure..... 30
Table 3: vCL_Q_TYPE 53
Table 4: vIL_Q_TYPE..... 53
Table 5: QUEUE_TYPE..... 53
Table 6: Synchronization Accuracy and Sequential Actions in D^3RIP Operation Without Any Fault 67
Table 7: Sequential Actions in D^3RIP Operation With Fault..... 67
Table 8: Synchronization Accuracy and Sequential Actions in D^2RIP Operation Before Adding Dependability Plane [44] 68

LIST OF FIGURES

FIGURES

Figure 1: Industrial Communication Levels [14] 4
Figure 2: Additional Protocol on Ethernet Layers [29] 8
Figure 3: TC-Net Structure [31] 9
Figure 4: Dependability Threats [20] 11
Figure 5: Domino Effect [38] 12
Figure 6: Creating Control Point and Rollback [40] 13
Figure 7: D^3RIP Layer Architecture 15
Figure 8: Time Slot Structure 16
Figure 9: IL Model as TIOA 19
Figure 10: Internal Functions in IL Layer 20
Figure 11: CL Model as TIOA 22
Figure 12: Update Functions for CL 23
Figure 13: SM Model as TIOA 24
Figure 14: Functions in Dependability Plane 25
Figure 16: Data Encapsulation of RT and Long nRT messages [44] 28
Figure 17: Message Transmission in IL layer 29
Figure 18: The Algorithm of the Transmit Function 32
Figure 19: The Algorithm of the Receive Function 33
Figure 21: Message Structure 35
Figure 22: Message Format in CL [29] 35
Figure 23: CL Algorithm [29] [41] 36
Figure 24: The Message Transmissions of the Layers and Functions of D^2RIP 37
Figure 25: The Timing of the Sending RT request with RT packet 38
Figure 26: The Timing of the Sending RT request without RT packet 38
Figure 27: The Timing of the Sending nRT packet 39
Figure 28: Message exchange of DP with other layers. 42
Figure 29: Structure of DP 43
Figure 30: CL Implementation with DP 44
Figure 31: Transmission Part of IL with DP Implementation 45
Figure 33: DoListenCLModule Thread in AP 47
Figure 34: listenCoordinationLayer Thread in DP Implementation 49
Figure 35: listenInterfaceLayer in DP Implementation 50
Figure 36: f_{AT} Implementation in DP 51
Figure 37: Dependability Plane UML Class Diagram 52
Figure 38: The Timing of the D^3RIP without Any Fault 56
Figure 39: The Timing of the D^3RIP with Fault 57
Figure 40: A Manufacturing System [44] 59
Figure 41: State Machines of Workcell [12] 60
Figure 42: The Timing Diagram for the PLC Communication of the Example Workcell [44] 61

Figure 43: The Connection of the Controllers and Plant 61
Figure 44: D^{3}RIP XML Configuration File for Controller C 64
Figure 45: SimpleNet XML Configuration File for Controller C 64
Figure 46: Simulator XML Configuration File for Controller C 66
Figure 47: The Experiment Result of the First Experiment 69
Figure 48: The Experiment Result of the Second Experiment 70
Figure 49: The Experiment Result of the Third Experiment 70

CHAPTER 1

INTRODUCTION

Industrial control applications are nowadays realized using distributed controller devices that are connected by a real-time communication network. The amount of the transmitted data has been increased with the new control systems and the demand of these systems will be increased more and more in the near future. Traditional bus and control network solutions such as CAN [1], ProfiBus [2] and LonWorks [3] do not support the demanded requirements, because of their low speed, high cost and incompatibility with other devices and equipment.

Therefore, a different protocol is needed to support the stated requirements. Ethernet (IEEE 802.3) is a common proposition for the support of industrial control applications since it is cheap, commonly used, high speed andcompatible with other protocols. However, there is a problem to use Ethernet in real time (RT) communication. The reason is the CSMA/CD (Carrier Sense Multiple Access / Collision Detection) access protocol. In CSMA/CD access protocol, if a collision occurs on the network, the node which sends the packet to the other node, waits a random time to resend the packet. Also, the random amount of time is double increased if a collision occurs again. This causes non-determinism and impairs the RT timing requirements. To overcome this problem, there are various solutions in the literature. Common solutions are:

- Modification of the Medium Access Control
- Adding Transmission Control Over Ethernet
- Using Switched Ethernet

In modification of MAC solution, specialized chips (ASICs) are used to modify the Ethernet hardware. Sercos [4], Ethercat [5], Profinet IO [6] are examples of modification of MAC solutions. They are used as RT Ethernet protocol, but their high cost and incompatible with other equipments are the problem of the modification of MAC. In adding transmission control over Ethernet solution, there are several different ways of doing this. Master /slave, Token Passing and TDMA methods are used to solve the problem by adding transmission control over Ethernet. Virtual Token Passing Ethernet [7], Ethernet Powerlink [8], Modbus/TCP [9], Ethernet for Plant Automation (EPA) [10], FTT Ethernet [11] are examples of adding transmission control over Ethernet. In using switched Ethernet solution, there are multiple transmission paths and switches are used instead of hubs that is, each network ınterface card (NIC) only receives traffic which is addressed to it. However, this solution is not enough to make Ethernet real-time due to the non-deterministic queuing delays in switches.

The new RT Ethernet protocol, Dynamic Distributed Dependable Real Time Ethernet Industrial Protocol (D^3RIP) is proposed in article [12]. This protocol is fully distributed, uses COTS Ethernet hardware and time-slotted transmission control based on the IEEE 1588 time synchronization protocol [13]. No hardware modification is required. It supports both RT and nRT traffic. D^3RIP is an extension of the two-layer protocol D^2RIP by dependability functionality in the form of a dependability plane. The interface layer (IL) and coordination layer (CL) of D^2RIP were implemented in [41]. In this thesis, the additional dependability plane of D^3RIP is studied, implemented and evaluated based on an application example, In this example, four distributed controller devices communicate with each other over D^3RIP. Several test scenarios show the functionality of the dependability plane.The remainder of the thesis is organized as follows. RT Ethernet for industrial communication, requirements of them and dependability are discussed and available RT Ethernet protocols are reviewed in Chapter 2. Formal protocol models and the implementation of a generic shared medium, a generic interface layer, a generic coordination layer and a generic dependability plane are explained in Chapter 3. The implementation of the dependability plane is described in detail in Chapter 4. The test scenario with 4 controllers and configuration of simulator, performance parameters, experiments and results are studied in Chapter 5. The conclusion and future works are presented in Chapter 6.

CHAPTER 2

BACKGROUND

2.1 Real -Time Ethernet for Industrial Communication Protocols

In industrial applications, industrial communication network and protocol are used for communication among control nodes and equipments. RT access, deterministic behavior and RT are the reasons why industrial communication protocols are used so often in control applications. In control applications, different components are used to implement the control system: controllers, remote controllers, supervisory stations, actuators and sensors are some of the components that are used. Sensors collect feedback data, controllers control the system according to receiving data from sensors using actuators. Actuators transform input signals into motion. Supervisory stations are the intelligent part of the control system. It is used as a monitoring part and computer in the system. All different parts are connected with each other using industrial communication networks.

Nowadays, industrial communication networks are widely used by industrial control applications and these industrial control applications become more complex and large-scale. Also computer aided industrial control devices with the network access are manufactured in recentyears. These developments make industrial control systems become an important industrial and academic research topic. Different industrial communication networks have been developed for the last twenty years for these systems.
In different industrial communication networks, messages for the different purposes are transmitted to each device in the system. These industrial communication networks are divided as follows: [14] (Figure1)

- T1) Device level data transmission between sensors, controllers and actuators: The receiving sampled data is periodic and it must be sent with time constraints.
- T2) Control level data transmission between supervisory controllers and the system components: It is needed that controllers and the system components at different hierarchical levels communicate each other for their coordination in the system. Mostly, components and controllers send the data which is event-based and requires deterministic response times, to each other. Because of the changing of the system behavior in discrete time, the next state of the system and the message which is sent in that case, have been already known using system dynamic model. For example, the controller which controls the two machines sends a message to the second machine to start, when the first machine completes its operation.
- T3) Information level data transmission: Mostly, it is used for the nRT and event-based communication.

When these traffic types are analyzed, there are four requirements that should be fulfilled by

the network to make it usable for industrial control: there should be RT traffic transfer, synchronized communication, dependable operation and support for nRT traffic. In RT traffic transfer requirement, when a node in the control system wants to send a message to other nodes, this message transfer time should be less than a deadline time of the message. In synchronized communication requirement, before the RT communication starts, all nodes in the system are synchronized to get the RT message successfully. In dependability requirement, if there is a failure in the system, the system should be able to fix the problem and resume its correct operation. In support for nRT traffic requirement, nRT messages should be sent without corrupting the RT traffic.

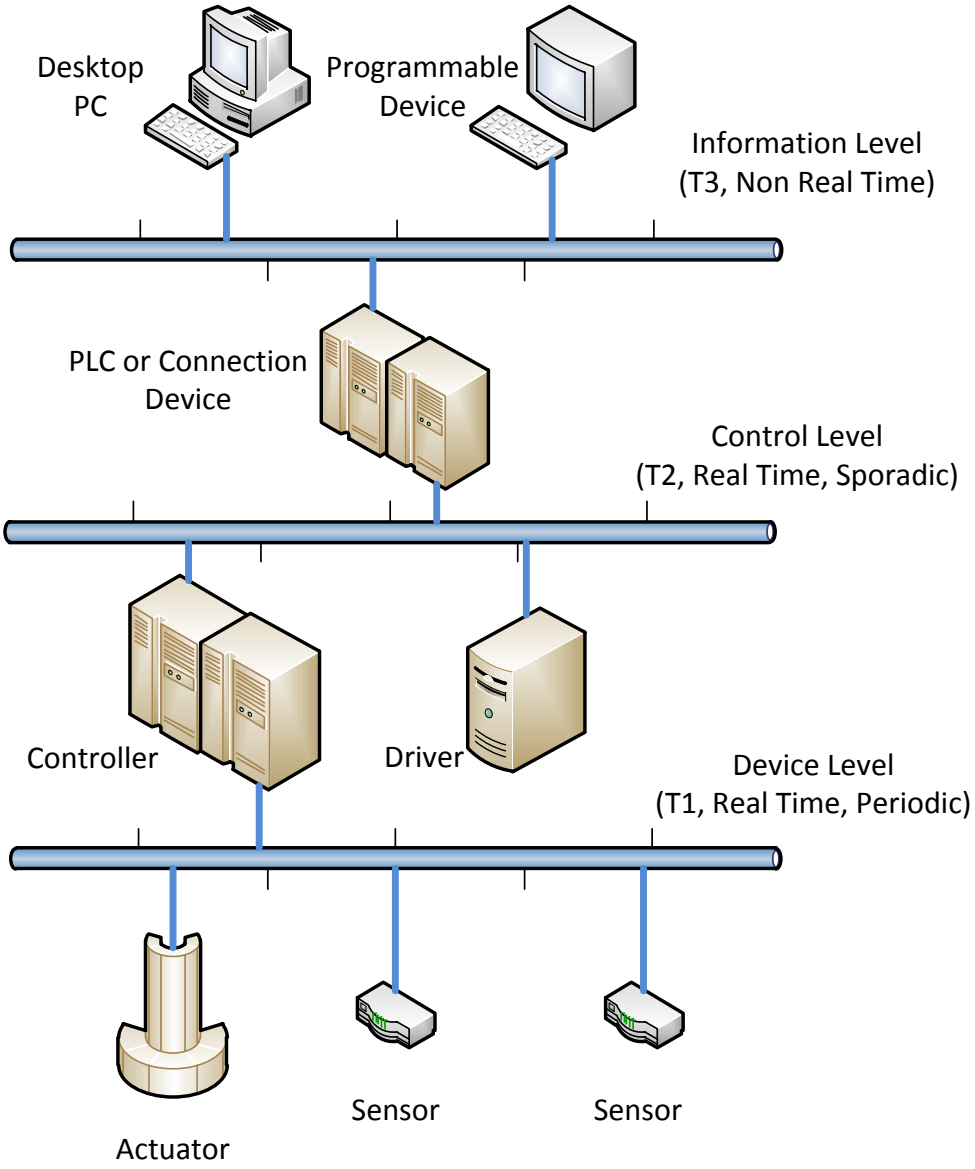

Figure 1: Industrial Communication Levels [14]

In 1980's CAN, Lonworks, Profibus started to be used as industrial communication network. [15]. But, their implementation cost is high, expanding the system is difficult and they are not compatible with other communication protocols. So, these problems are the reason for developing and using different protocols. Ethernet can be used for industrial communication protocol. However, Ethernet is not directly usable as industrial communication protocolwithout any modification on hardware or software. Because, it does not support the RT traffic when collisions occur in the system. When a collision happens, back-off algorithm runs and the node which wants to send a message to the system, waits a random time. It creates non-determinism on the system. So, generic Ethernet without any modifications cannot be used for industrial communication. However, the application of Ethernet is simple, widely used and low cost are the reason why there is a considerable research effort on modifications and additions to Ethernet in order to make ıt usable as a RT communication protocol.

2.1.1 Requirements

The requirements for the development of the real-time Ethernet protocol are listed below: [16]

Real Time Data Transmission: Message transmission time is measured between the applications which are sent and received. The requirements of the message transmission time for the different level communications are different. While the applications including human operators require 100 ms transmission time, applications working with programmable logıc controllers (PLCs) require 10 ms transmission time and applications which coordinate many devices, require 1 ms transmission time.

Synchronization Support: In industrial communication network, RT response time and common reference time between nodes are provided by synchronization protocol. The sensitivity of the synchronization is defined the maximum deviation between the time of two nodes [16]. To protect this sensitivity of the synchronization, guard periods are used and this causes the increasing of the time delay. The most common and used synchronization protocol for Ethernet is IEEE 1588 time synchronization protocol [13] [17].

IEEE 1588 time synchronization protocol works according to Precision-time protocol (PTP). In this protocol, time difference and delay time between the selected master node and other nodes are calculated using message exchanges between master node and slave nodes. Thus, nodes are synchronized. Except IEEE 1588, special time synchronization mechanisms are used in EtherCAT [4] and Sercos (IEC 61491) [5] protocols.

Non-Real Time Traffic Support: It is provided that while nRT traffic is supported, the RT traffic is not affected nRT traffic.

Compatibility: The most important reason that makes Ethernet an attractive technology is inexpensive hardware and software interface. It is required that when industrial Ethernet works, it is compatible with standard Ethernet to make implementation with COTS (Commercial Off-The-Shelf) components and to take advantage of inexpensive hardware andsoftware interface. In addition to this, it is also supposed thatcommonly used application protocols such as HTTP and FTP and synchronization protocols such as IEEE 1588 are supported.There are backward-compatibility requirements. For this reason, it is expected that

once a protocol has been established, it works for years. As a result, an industrial Ethernet protocol should be conducive to adding new devices.

Dynamic Resource Separation for the Real-Time Traffic: The communication requirements of the industrial system which communicates with a network, change dynamically in time [18]. For instance, in the self-triggered control concept, at the device level, calculation times are reserved before. In addition to this, high-level controllers which coordinate the distributed systems, communicate only when they are needed. It is supposed that according to instantaneous needs, RT bandwidth should be separated to devices in the industrial Ethernet protocols.

Dependability: Dependability is an important requirement for the applications which have critical security constraints and work in the industrial control systems [19]. Availability, safety, integrity and maintainability are the elements of dependability [20]. To talk about the dependability of a distributed industrial control system which communicates with the network, it is provided that the dependability of the network and controller is necessary. When designing a dependable industrial communication network, dependable synchronization and the consistence of values which are sent with messages, are important. The problem of dependability stands outmore RT Ethernet-based solutions due to non-deterministic feature of Ethernet [21]. Dependable communications provide that accurate information should be sent to the right place, at the right timeand right order. Dependability support is often done by the separation of the static additional capacity according to default worst case [22]. For example, for the TDMA-based protocol additional time slots might be allocated to the transmitting nodes inorder to send each message which is lost, in repetition time and only half of the capacity can be used.

2.1.2 Real Time Ethernet Protocols

In the literature, there are four major approaches to add Ethernet real-timeliness:

- Changing the non-deterministic sending messages mechanism with the hardware modification on Ethernet network interface card,
- Minimizing response time and the probability of the collision,
- Removal of the probability ofcollision on shared medium using point-to-point connections and switches,
- Constructing layers on top of shared medium to avoid collision.

Specialized Hardware: EtherCat [4], SERCOS III [5] and ProfiNet [6] use specially designed node and switch hardware. Ethercat and ProfiNet use IEEE 1588 for time synchronization. On the other hand, SERCOS III uses special messages to synchronize the nodes in the system. These three protocols are supported by special designed dependable protocols. Special designed, Twinsafe Protocol operates as separate layer under EtherCAT protocol. Devices get addresses and data safety is provided with CRC. In SERCOS III Safety, there are sequence numberanda timestamp in the message. The receiver node sends an acknowledgment message to the sender node. Devices get addresses and data safety is provided with HDLC coding.PROFIsafe is developed for ProfiNet [6]. Sequence number and a timestamp are added in the message. Devices get addresses and data safety is provided with CRC.

Non-Guaranteed Approaches: MODBUS/TCP [9] and similar protocols work on TCP/IP to be compatible with standard Ethernet [23] [24]. With traffic shaping, it low delays can be achieved in these systems. In these approaches, there is no guarantee that messages will be transmitted in time.

Switched Ethernet: Since collisions are possible on standard Ethernet, the solution of the non-deterministic network access problem is full-duplex, switched and point-to-point Ethernet (IEEE802.3x). With this structure, even if shared medium and the collision problem are eliminated, the problem of network access is carried to queueuing delays in the network [22][25][26]. To provide the RT communication, Ethernet switches that make scheduling and prioritization are needed. Giving priority to the messages, according to these priorities, providing different service like 802.1p and 802.1Q Ethernet protocols and protocol extensions are proposed. Unlike the standard Ethernet protocols, these protocols require specialized switches. Under the assumption of an infinite buffer for real-time traffic, even if scheduling analysis can be made, the actual conditions require the use of a limited buffer [27]. On switched Ethernet, the implementation of the sensitive time synchronization which is important for RT communication, can be difficult.

Ethernet/IP (EIP) [28] works on the TCP/IP with full-duplex Ethernet switches which have special prioritization mechanism. Ethernet /IP protocol does notensure the real-time communication. Time synchronization is made with special messages which are compatible with IEEE 1588 protocol. Also, the coordination between sender and receiver, is provided with the ping messages. There is also timestamp in the messages. Devices get addresses and data safety is provided with CRC.

Constructing Layer on Shared Medium: A variety of academic and industrial protocols are proposed to prevent collisions on shared medium by adding RT properties. These protocols aim at adding a layer on IEEE 802.3 that prevents collision and non-deterministic sending messages after collision. NRT and RT traffic pass over this layer. On this layer, there may be a specific protocol which is responsible for transmission of RT traffic. TCP-UDP/IP layers may be responsible for transmission of RT traffic. Figure 2 shows the additional protocol on Ethernet layers.

There are 3 different approaches for adding layer on medium access layer:

- Time Division Multiple Access (TDMA)
- Master-Slave
- Token Passing

Time Division Multiple Access (TDMA) : In this approach, time is divided into equivalent slots. The owners of one or more time slots are determined statically for each node. Time synchronization between all nodes in the system, is important for communication between nodes. This approach provides reliable network access for all nodes. Working with low efficiency is the disadvantage of TDMA. If a node does not send a message in the time slot which is belongs to that node, another node in the system cannot send a message in that unused time slot. In addition to this, the delay in the software and switches is also considered while choosing the time slot. If the messages in the network arelostdue to network errors, additional time periods must be allocated to send messages again.

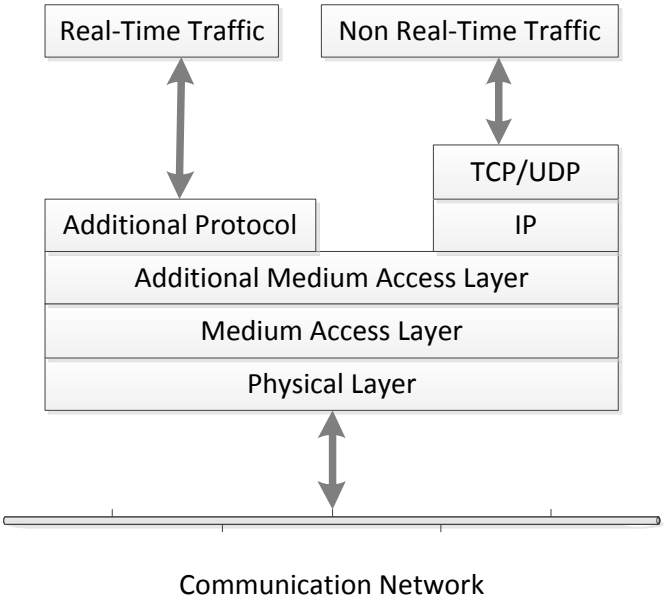

Figure 2: Additional Protocol on Ethernet Layers [29]

Master-Slave: A chosen master node sends messages to the other nodes (slaves) to ask whether it needs to send a message or not (polling). Slave nodes only send message when master nodes poll them. This approach is used in the network which has small number of nodes. The efficiency of master slave is affected negatively while polling the system. Especially in cases wherethe trafficis very variable and nodes do not have any message to send, the efficiency decreases. Also, the delay time which is passed when the master node waits slave node's answer, decreases the efficiency. When the number of nodes is large, the polling cycle time for all nodes, is more than the delay time of sending message. In that case, the delay time is much more than acceptable limit. The speed of software in the slave nodes also is the one of thedetermining factors of the polling process time. If the software is too slow, the importance of the network speed is ignored and the efficiency of the network is decreased. In addition to the problem of the efficiency, master-slave communication is not a suitable structure for distribution. Because of master node, it is single-centered and there might be a problem at a single point.

Token Passing: In this approach, one node can send a message if and only if it has a token to send a message. When it sends its message, it transmits the token to another node with a special message. In token-based approaches, the possibility of losing the token, token circulationtime which causes decreased communication speed and the difficulty of adding a new component are the disadvantages of token passing system.

There are lots of solutions which are created in industry and academia. These solutions and standards follow the approaches which are explained the section above and they carry on the negative aspects of them.

Time Critical Control Network (TC-Net) [30] is implemented with adding a layer on standard Ethernet which provides the token passing. NRT traffic has low priority. Time

synchronization is provided with the special message. The dependability of the protocol is also provided using an extra TC-Net card. Figure 3 shows the TC-Net structure. [26][31].

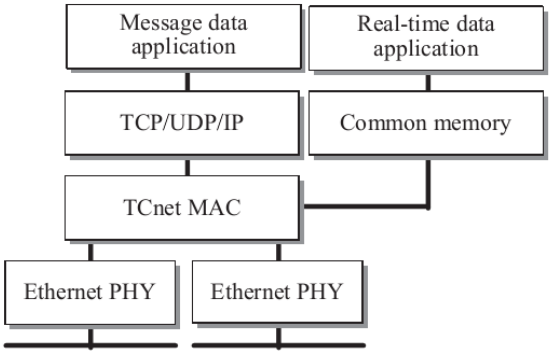

Figure 3: TC-Net Structure [31]

Powerlink (EPL) [8] is implemented with adding a layer on standard Ethernet which provides the master-slave. With the inefficiency of the master-slave structure, EPL efficiency is calculated as 25% [22]. Time synchronization is provided with the special message similar to the IEEE 1588 protocol. RT and nRT data are sent in different time slots. Sequence number and a timestamp are added in the message. Devices get addresses and data safety is provided with CRC.

Ethernet for Plant Automation (EPA) [10] works with static TDMA. Time slots for nRT and RT are determined before the communication. It supports both RT and nRT traffic. IEEE 1588 time synchronization protocol is used to synchronize the nodes in the network. The disadvantage of this protocol is static slot scheduling and TDMA. Slot scheduling is done by periodic message broadcast. Also, to the guard periods and error recovery precautions cause low efficiency in TDMA solutions like EPA. [16] [26]

In FTT Ethernet protocol, master/multi-slave model is used to implement the protocol. It uses COTS Ethernet hardware.The communication is TDMA based and time slot durations are fixed. Nodes can be connected to share or switched medium. It supports both RT and nRT traffic, also in addition to them, there is online admission control to guarantee timeliness to the RT traffic. Also there is no specific synchronization protocol. But, elementary cycle begins with master node trigger. When master node broadcast to trigger message, elementary cycle is started with that trigger message. The disadvantage of the FTT Ethernet protocol is that master-slave method. Master-slave models have single point of failure, undistributed structure and low efficiency. [11][32]

In Virtual Token Passing Ethernet (VTPE), if a node wants to hold a network, it should takethe virtual token to send a message to other node. In this method a virtual token is circulating between nodes and it works with closing the binary exponential back off (BEB) algorithm. When there is collision in the system, it provides that the nodes send the RT message again immediately. In this protocol, Ethernet hardware is not modified. It uses

COTS (commercial off-the-shelf) Ethernet. Software Ethernet driver modification is required for RT stations. Figure 4 shows the algorithm of VTPE.The disadvantage of the VTPE is losing token which is the dependability problem of token passing method. [7]

The focus of this study is working with compatible components without changing the working principle of Ethernet and providing guaranteed real-time performance for shared media protocols. Table 1 shows the comparison of the defined requirements and performance criteria for these protocols. In table, A/I: Academic/ Industrial Purpose, RT Cap: RT Data Transmission Capacity, nRT Cap: NRT Data Transmission Capacity, Time Sync: Time Synchronization Protocol are used as abbreviations.

Table 1: Shared Medium Industrial Ethernet Protocol

A/I	Protocol	Medium Access	Delay	Node Number	RT Cap.(bps)	nRT Cap.	Time Sync
I	EPA	TDMA	5ms, 100µs	32, 64	12.28M	0,85	IEEE 1588,10µs, 1µs
I	EPL	Master-Slave	400µs, 5.5ms	4,15	15.2M, 32M	19.6%, 4.4%	IEEE 1588,1s
I	Time Critical Control Network (TCNet)	Token-Passing	2ms/ 20ms/ 200m	24,13	58.4M/ 51.2M/ 7.2M, 45.6M/ 40.8M/ 4.8M	0%, 20%	-
A	FTT-E	Master-Slave	1ms	Unspecified	36M, 36%	0,11	Periodic Time Sychronization Message from Master Node
A	VTPE	Token-Passing	5.8ms	256	Under 40% Ethernet Cap.	Unspecified	-

2.2 Dependability

Dependability is defined as the ability to deliver service that can be justifiably trusted. Also, it includes the attributes below: [20][33].

- Availability: A system is ready to provide the right service.
- Reliability: A system continues to right service in a time.
- Safety: A system does not lead to irreversible errors at the user level.

- Maintainability: A system can be conducive to repair and can be available to maintain when needed
- Integrity: System changes are suitable for designed sequence and there are not any unexpected system changes in the system design sequence.

System is dependable when it fulfills (some of) the above attributes. Also, system must have precautions against threatened dependability of the system elements at the design and operation stages. Threatened dependability of the system elements are divided into the three main categories. These categories are: [20] [33] [34]

- Component-Level Errors (Faults)
- System-Level Errors (Errors)
- User-Level Errors (Failures)

Figure 4 shows the faults cause-effect relationship.

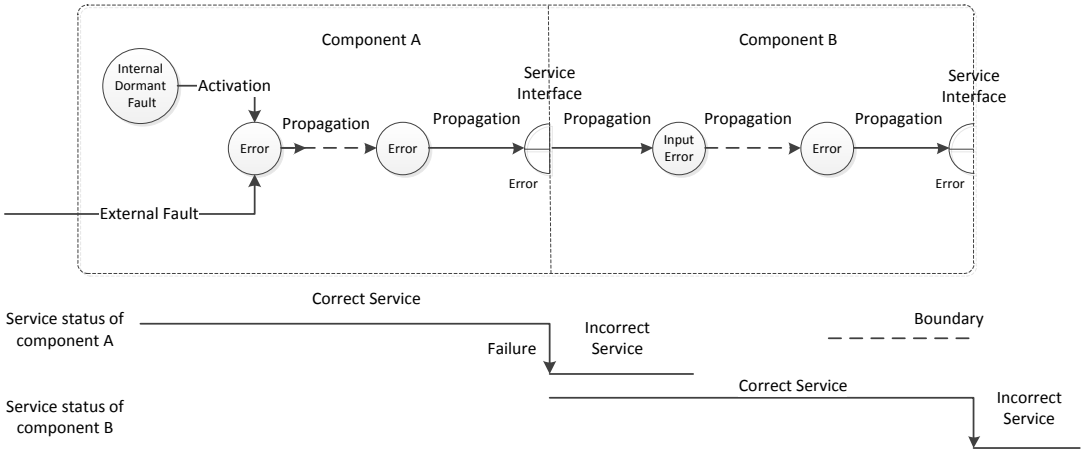

Figure 4: Dependability Threats [20]

As can be seen in Figure 5, faults which occurred and are not solved at the component level proceed to the user level. After that the system cannot work properly. Bringing back the correct function of the system, error conditions should be eliminated.

Means (dependability activities) are activities for elimination of errors and allocation of the dependability of the system at various levels. They are divided into 4 main groups: [20] [33] [35]

- Component-Level Fault Prevention Activities: Fault prevention activities are the activities which prevent the faults at the design stage. Keeping records of faults at the designed system and modify it during the design process is the most common example. [36] [37].
- Component-Level Fault Removal Activities: Error detection, classification and validation phases of the system design phase of these activities aims to eliminate the errors. System verification is a method which provides confirmation of fault before debugging and supports system requirements after debugging.

- Component-Level Fault Forecasting Activities: Fault forecasting activities are the activities which determine the state changes that cause user-level faults after completed system design.
- Component-Level Fault Tolerance Activities: Activities of detection and elimination of errors that can occur during operation of the system. Elimination of effects of the system faults is called system recovery [20]. The most common method without having to initialization (reset) while system is operating is making checkpoint and rolling back.

In this method, functions in a distributed system record their state in error conditions that may occur. In the event of any error, functions return to their pre-recorded states within the scope of the error recovery scenario operated by the system. Although at first glance it seems to be an easy method of application, in some conditions rollback mechanism causes consecutive rollbacks which might return the system to its initial state. In other words, it causes reset of the system. Figure 5 shows that situation which is called domino effect.

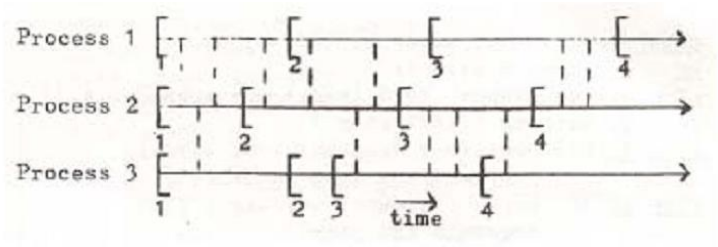

Figure 5: Domino Effect [38]

In Figure 6, lines with dashed vertical show the communication of functions. 3 processes in Figure determine the rollback point periodically. For example in process 3, an error after 4th recovery block is identified. This situation causes that process 3 returns to the 4th recovery block. When process 3 returns to the 4th recovery block, the other two processes have to return to their previous recovery point to be compatible with process 3. The reason why the other two processes have to return to their previous recovery point is that process 3 communicates with the other two processes between error and 4th recovery block. With the same logic, rollback mechanism causes that the system returns to its initial state, like toppled dominoes one after the other.

In order to stop the domino effect, a communication mechanism between processes is recommended [39] [40]. In these articles, proposed communication mechanisms cause additional load on the system message traffic. However, it seems that [40]'s proposed idea causes less additional load on the system message traffic than other one. In [40], the use of a common reference time between functions is proposed to reduce the additional load on the system message traffic. Figure 6 shows the proposed control point description and rollback mechanism.

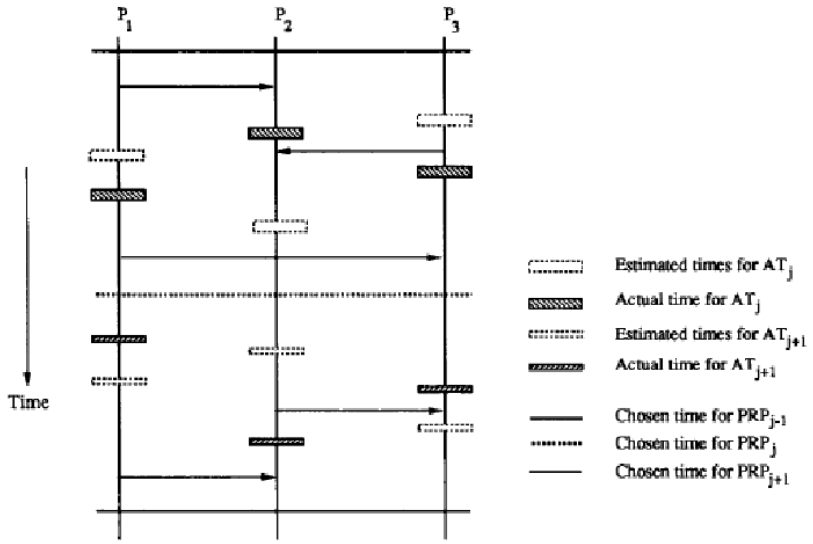

Figure 6: Creating Control Point and Rollback [40]

Figure 7 shows that distributed nodes run synchronous with each other. Common period of time is determined for synchronous nodes to make an acceptance test. Node which does not complete acceptance test within the specified time, sends its delay time to other nodes to determine the synchronous recovery points. This reduces the additional load on the system message traffic. However, currently only available in recovery point messages are used for identification purposes in the network, and this adversely affects the efficiency of the network. In our framework, distributed nodes are synchronous. In addition to this, communication between the nodes is on the shared medium. Dependability plane in our framework uses the advantages of these two features and it is aimed to eliminate the additional message load on the system.

CHAPTER 3

PREVIOUS WORK

3.1 Dynamic Distributed Dependable Real Time Industrial Protocol (D^3RIP) Protocol Overview

Dynamic Distributed Dependable Real Time Ethernet Industrial Protocol can be used for the communication of controllers in distributed control systems. Dynamic Distributed Real Time Ethernet Industrial Protocol works over Ethernet protocol with non-real and RT traffic. There is no need to change the physical MAC layer, it uses COTS Ethernet hardware. D^3RIP protocol works on shared medium without using any switch. It realizes TDMA on top of Ethernet, whereby synchronization is achieved by the IEEE 1588 protocol. It requires small software Ethernet driver modification and modifications of the software stack between MAC and Application layer. Figure 7 shows the layered architecture of D^3RIP. The Dependability Plane works over D^2RIP structure that is implemented in [41].

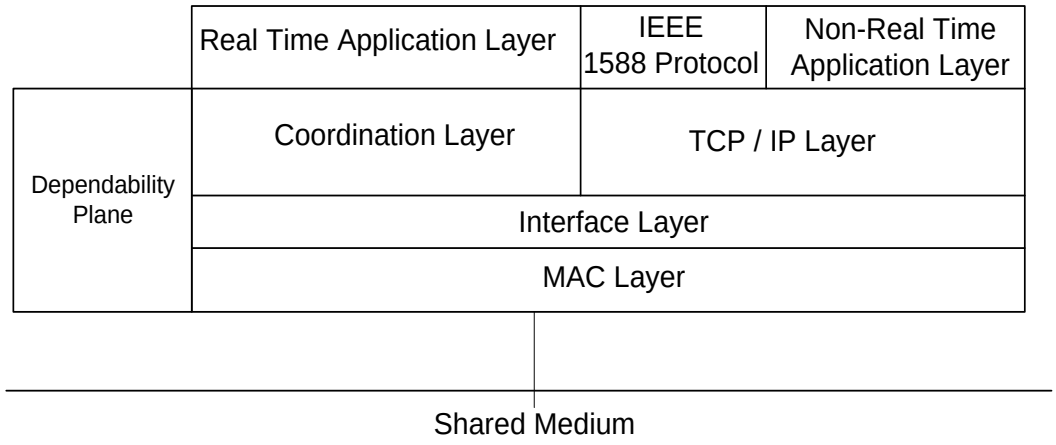

Figure 7: D^3RIP Layer Architecture

In D^3RIP Layer Architecture, there are 3 different layers added to the original Ethernet layer architecture namely interface layer (IL), coordination layer (CL) and dependability plane (DP). Interface layer is responsible for the time-slotted TDMA structure that is implemented and synchronized with other nodes using IEEE 1588 time synchronizing protocol. At the beginning of each time slot, CL sends information to the IL about the usage of the time slot and IL sends Ethernet frame within time slot when it gets information from CL. Coordination layer is responsible for determining the allocation of the type of slot whether

RT or nRT and the allocation of the owner of the slot. CL is implemented in the user space of Linux and IL is implemented in the kernel space of Linux in previous works [29][42]. In these works, CL is implemented as 2 types: DART (Dynamic Allocation Real-Time Protocol) and URT (Urgency-Based Real-Time Protocol). In DART, variables in the protocol are hold in the form of allocated RT slots. In URT, control application variables are stored in the form of the communication requests [29]. Also, IL is implemented as 2 types: RAIL (Real-Time Access Interface Layer Protocol) and TSIL (Time-Slotted Interface Layer). In RAIL, slot allocations for nRT and RT traffic are made statistically. In TSIL, slot allocations for nRT and RT traffic are made dynamically by CL [42].

Dependability Plane is responsible for dependability of the framework. DP makes an acceptance test whether the protocol works without problem or not. If there is a fault, it sends stored CL parameters using rollback message to CL and when CL gets the rollback message from DP, it warns application layer to resend the fault messages. Thus DP protects the framework. Timing of messaging between layers is important to determine the slot timing duration. Figure 8 illustrates the timings of 1 slot. CL_{cmp} which is a calculation time for CL, IL_{cmp} which is a calculation time for IL and Message Tx, which represents transmission of an application message on Ethernet.

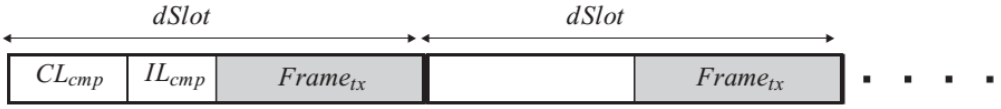

Figure 8: Time Slot Structure

D^{3}RIP Layer Architecture works on the RT operating system. RT features of the operating system kernel are gained with RT patches over the Linux kernel. The latest stable kernel is 3.6.2. The configurations below are needed for RT operating system after RT patches implemented over Linux kernel:

- Activate Tickless System (Dynamics Ticks).
- Activate High Resolution Timer Support.
- Set“Preemption Model” parameter to “Fully Preemptible Kernel(RT)”.
- Set “Timer frequency” parameter to “1000Hz”.
- Deactivate “Suspend to RAM and standby”.
- Activate “Timestamping in PHY devices”.
- Activate “PTP Hardware Clock (PHC)”.
- Activate “PTP clock support”.
- Deactivate “Show timing information on printks”.
- Set “I/O scheduler” parameter to “Deadline”.

3.2 D^{3}RIP Formal Protocol Model:

3.2.1 Generic Interface Layer:

Interface Layer Generic Model is defined using TIOA Model in [12][43]. Figure 9 shows the IL Model using TIOA. There are 6 parameters in the IL Model: *dSlot, t0,t1,t2, t3, M , Q , AIL,HIL*.

- *dSlot:* Slot duration,
- *t0,t1,t2, t3*: Time of events,
- *M:* The type of transmitted messages,
- *Q*: The type of a FIFO queue messages,
- *AIL*: Abstract variable of IL.
- *HIL*:Abstract variable of IL

IL Model as TIOA has variables to define the model and operations: now_i^a , $next_i^d$, $TxRT_i^d$, $TxnRT_i^d$, $RxRT_i^d$, $RxnRT_i^d$, $RTIL_i^d$, $myIL_i^d$, vIL_i^d, $reqIL_i^d$.

- now_i^a: Analog variable which evolves with the time derivative of 1 , the updated time information is provided by this variable.
- $next_i^d$: The end of the current time slot is stored by this variable.
- $TxRT_i^d$: The buffer that stores the RT messages to be transmitted.
- $TxnRT_i^d$: The buffer that stores the nRT messages to be transmitted.
- $RxRT_i^d$: The buffer that stores the RT messages that are received.
- $RxnRT_i^d$: The buffer that stores the nRT messages that are received.
- $RTIL_i^d$: The variable that stores the type of next slot whether RT or nRT.
- $myIL_i^d$: The variable that stores whether the device owns the next time slot or not.
- vIL_i^d: The variable that holds the additional information of the protocol operations.
- $reqIL_i^d$: The variable that stores the request to the CL to determine RTIL and myIL.
- $sendVIL_i^d$: The variable that stores the whether vIL is sent or not.
- at_i^d: The variable that shows the acceptance test result.
- $checkPT_i^d$: The variable that stores the rollback status of that node.

Actions in IL:

- *output IL2SM (m:M)*$_i$
- *output UPDVIL(vIL:* $A_{IL,}$ *TxnRT: Q, RxnRT:Q)*$_i$
- *input SENDRES(atRes:bool, rbSt: int)*
- *input RBACK(ILHT:* $H_{IL,}$ *cLHT:* H_{IL}*, rbSt: int)*
- *input SM2ILDP (m:M)*$_i$
- *input CL2ILRT(*b_{my}*:bool,* b_{RT}*:bool, m:M)*$_i$
- *input AP2ILNRT(m:M)* $_i$
- *input IL2APNRT(q:Q)* $_i$
- *output IL2CLRT (m:M)* $_i$
- *internal UPDATE()*$_i$
- *output REQRT()*$_i$

TIOA $IL_i(dSlot: int, t_0: int, t_1: int, t_2: int, t_3: int, M: Type, Q: Type, A_{IL}: Type, H_{IL}: Type)$

states

$now_i^a: R := dSlot$
$\texttt{TxRT}_i^d: M :=$ empty
$\texttt{TxnRT}_i^d: Q :=$ empty
$\texttt{RxRT}_i^d: M :=$ empty
$\texttt{RxnRT}_i^d: Q :=$ empty
$\texttt{RTIL}_i^d: bool :=$ false
$\texttt{myIL}_i^d: bool :=$ false
$\texttt{reqIL}_i^d: bool :=$ false
$\texttt{sendVIL}_i^d: bool :=$ false
$\texttt{at}_i^d: bool :=$ true
$\texttt{checkPT}_i^d: int :=$ -1
$\texttt{vIL}_i^d: A_{IL} :=$ InitV

transitions

internal UPDATE()$_i$

pre:

$now_i^a = dSlot$

eff:

$\texttt{vIL}_i^d = f_{upd}(\texttt{vIL}_i^d, \texttt{RTIL}_i^d)$
$now_i^a := 0$
$\texttt{reqIL}_i^d :=$ true
$\texttt{sendVIL}_i^d :=$ true

output UPDVIL$(vIL, TXnRT, RXnRT)_i$

pre:

$\texttt{sendVIL}_i^d =$ true
$now_i^a = t_0$

eff:

$\texttt{sendVIL}_i^d :=$ false

input SENDRES$(atRes, rbSt)_i$

eff:

$\texttt{at}_i^d :=$ atRes
$\texttt{checkPT}_i^d :=$ rbSt

input RBACK$(ilHT, clHT, rbSt)_i$

eff:

$\texttt{vIL}_i^d := \texttt{ilHT.vIL}_i^d$
$\texttt{TXnRT}_i^d := \texttt{ilHT.TXnRT}_i^d$
$\texttt{RXnRT}_i^d := \texttt{ilHT.RXnRT}_i^d$
$now_i^a := 0$
$\texttt{reqIL}_i^d :=$ true
$\texttt{sendVIL}_i^d :=$ true
$\texttt{at}_i^d :=$ true

output REQRT()$_i$

pre:

$\texttt{reqIL}_i^d =$ true
$now_i^a = t_1$

eff:

$\texttt{reqIL}_i^d =$ false

signature

output IL2SM$(m: M)_i$
output UPDVIL$(vIL: A_{IL}, TXnRT: Q, RXnRT: Q)_i$
input SENDRES$(atRes: bool, rbSt: int)$
input RBACK$(ilHT: H_{IL}, clHT: H_{CL}, rbSt: int)$
input SM2ILDP$(m: M)$
input CL2ILRT$(b_{my}: bool, b_{RT}: bool, m: M)_i$
input AP2ILNRT$(m: M)_i$
input IL2APNRT$(q: Q)_i$
output IL2CLRT$(m: M)_i$
internal UPDATE()$_i$
output REQRT()$_i$

input IL2APNRT$(\texttt{RxnRT}_i^d)_i$

eff:

set $\texttt{RxnRT}_i$ empty

input AP2ILNRT$(m)_i$

eff:

$\texttt{TxnRT}_i^d$.Push(m)

output IL2SM$(m)_i$

pre:

$(now_i^a = t_2) \wedge \texttt{myIL}_i^d \wedge$
$(\neg(\texttt{TxRT}_i^d \text{ empty}) \wedge \texttt{RTIL}_i^d) \vee (\neg\texttt{RTIL}_i^d \wedge$
$\neg(\texttt{TxnRT}_i^d.\text{Top empty}))$

eff:

if $\texttt{RTIL}_i^d$
  set $m = \texttt{TxRT}_i^d$
  set $\texttt{TxRT}_i^d$ empty
else
  set $m = \texttt{TxnRT}_i^d$.Top
  $\texttt{TxnRT}_i^d$.Pop
set $m.rbSt = \texttt{checkPT}_i^d$
set $m.atRes = \texttt{at}_i^d$
$\texttt{myIL}_i^d =$ **false**

input SM2ILDP(m)

eff:

if $\texttt{RTIL}_i^d$
  $\texttt{RxRT}_i^d = m$
else
  $\texttt{RxnRT}_i^d$.Push(m)

output IL2CLRT$(m)_i$

pre:

$now_i^a = t_3$
$\neg(\texttt{RxRT}_i^d \text{ empty})$

eff:

set $m = \texttt{RxRT}_i^d$
set $\texttt{RxRT}_i^d$ empty

input $\text{CL2ILRT}(b_1, b_2, m)_i$

eff:

$\text{RTIL}_i^{\text{d}} = b_1$

$\text{myIL}_i^{\text{d}} = f_{\text{my}}(\text{vIL}_i^{\text{d}}, \text{RTIL}_i^{\text{d}}, b_2, i)$

$\text{TxRT}_i^{\text{d}} = m$

Trajectories T

stop when	**evolve**
$\text{now}_i^{\text{a}} = dSlot$	$d(\text{now}_i^{\text{a}}) = 1$
$(\text{sendVIL}_i^{\text{d}} = \text{true}) \wedge (\text{now}_i^{\text{a}} = t_0)$	
$(\text{now}_i^{\text{a}} = t_1) \wedge (\text{reqIL}_i^{\text{d}} = \text{true})$	
$((\text{now}_i^{\text{a}} = t_2) \wedge (\text{myIL}_i^{\text{d}} = \text{true}) \wedge (\neg(\text{TxRT}_i^{\text{d}} \text{ empty}) \wedge \text{RTIL}_i^{\text{d}}) \vee (\neg \text{RTIL}_i^{\text{d}} \wedge \neg(\text{TxnRT}_i^{\text{d}}.\text{Top empty})))$	
$\text{now}_i^{\text{a}} = t_3 \wedge \neg(\text{RxRT}_i^{\text{d}} \text{ empty})$	

Figure 9: IL Model as TIOA

$myIL_i^d$ and vIL_i^d variables are updated by internal operations f_{my} and f_{upd}. After the data transmission is finished, `UPDATE()` is called for the update of variables of the next slot. *vIL* is updated first. When IL does not need the information if the next time slot is RT or nRT, reqIL gets false value ($reqIL_i^d$=false) and the owner of the next time slot is determined locally. If IL needs to information if the next time slot is RT or nRT, reqIL gets true value ($reqIL_i^d$=true). IL triggers the action `REQRT` to the CL. `REQRT()` requests the type of the next time and the owner of the next time slot. After the calculation time of CL, the action `CL2ILRT(`b_1`,`b_2`,m)` indicates the type of the next time slot (b_1), the owner of the next time slot (b_2) and RT message (m) in CL if it has. *RTIL* and *myIL* variables update their new values according to b_1 and b_2.

In `IL2SM(m)`, IL sends messages to all nodes, when current time now equals to next starting time *next* and *myIL* is true. The value in $RTIL_i^d$ determines if the transmitted message is RT or nRT message. Also, in `SM2ILDP(m)`, $RTIL_i^d$ is important too, to send message to upper layer RT buffer ($RxRT_i^d$) or nRT buffer ($RxnRT_i^d$). In `IL2CLRT(m)`, RT messages are send to the upper layer immediately. In `IL2APNRT(`$RxnRT_i^d$`)` and `AP2ILNRT(m)`, upper layer can reach $TxnRT_i^d$ and $RxnRT_i^d$ buffers in the IL at any time.

If there is no collision on the shared medium, IL sends message to SM in a one slot and after every update in IL, parameters which are related protocol are same. But, *RTIL* variable gets its b_1 parameter from upper layer CL and this b_1 parameter shows the next time slot is RT or NRT. Also, in f_{my} `(`vIL_i^d`,`$RTIL_i^d$`,`b_2`,i)` function gets its b_2 parameter from CL and b_2 parameter indicates that the next time slot belongs to the device *i* or not.

$$f_{my}(vIL_i^d, false, -, i) = \text{true}$$
$$(\forall j \in I - \{i\}) \quad f_{my}(vIL_i^d, false, -, i) = \text{false}$$
$$f_{my}(vIL_i^d, true, b_2, i) = b_2$$

In `UPDVIL`, if *sendVIL* is true, IL sends vIL parameters of that time slot to DP in each time slot. In `SENDRES(atRes,rbSt)`, IL gets information from the DP about acceptance test

result (*atRes*) and rollback status (*rbSt*). After getting *atRes* and *rbSt*, these variables are sent to other nodes with `IL2SM(m)`.In `RBACK`, IL gets stored *vIL* parameters, *Rx* and *Tx* message from DP and updates its *vIL* parameters with the received *vIL* parameters.

Also, IL obeys some requirements below:

- Transmission window covers all messages.
  $m.length < dSlot - rem \quad \forall m \in M$
- The time between $REQRT_i$and $CL2ILRT_i$is rem-cmp. After each $REQRT_i$ function, $CL2ILRT_i$occurs.
- If $REQRT(t)_i$ and $REQRT(t)_j$i, j ∈ I, i≠ j occur at the same time t, then, it holds for the

next occurrence of$CL2ILRT(b_1, b_2, m)$ and $CL2ILRT(\hat{b_1}, \hat{b_2}, \hat{m})_j$ that and $b_2 = b_1 = \hat{b_1}$ true ➔ $\hat{b_2}$= false [12]

In framework, IL asks to the CL layer whether the next time slot is RT or nRT. There are 3 variables in vIL such as*vIL.cnt, vIL.cyc, vIL.slots*. These vIL parameters show which node owns the nRT slot at that time slot. Internal functions f_{upd} and f_{my} are defined in Figure 10. In f_{upd} function, *vIL.cnt* is incremented with modulo *vIL.cyc.* After sending action `REQRT` to the CL, CL does action `CL2ILRT` (*b1*, *b2*, *m*). In f_{my} internal function, if the next time slot is RT, it determines with action `CL2ILRT`, f_{my} function returns b_2 variable which returns with action `CL2ILRT`. If the next time slot is nRT, it determines with $RTIL_i^d$ and *vIL.nRTSet*, f_{my} internal function returns true and the owner of the next time slot is determined by IL.

$$f_{\text{upd}}(\text{vIL}_i^{\text{d}}, \text{RTIL}_i^{\text{d}}).\text{cnt} = \{ \ (\text{vIL}_i^{\text{d}}.\text{cnt} + 1) \mod \text{vIL}_i^{\text{d}}.\text{cyc}$$

$$f_{\text{my}}(\text{vIL}_i^{\text{d}}, \text{RTIL}_i^{\text{d}}, b_2, i) = \begin{cases} b_2 & \text{if } \text{RTIL}_i^{\text{d}} = \text{true} \\ \text{true} & \text{if } \neg\text{RTIL}_i^{\text{d}} \wedge \text{vIL}_i^{\text{d}}.\text{cnt} \in \text{vIL}_i^{\text{d}}.\text{nRTSet} \\ \text{false} & \text{otherwise.} \end{cases}$$

Figure 10: Internal Functions in IL Layer

3.2.2 Generic Coordination Layer:

Coordination Layer Generic Model is defined using TIOA Model in the article [12] [43]. Figure 11 shows the CL Model using TIOA. There are 8 parameters in the CL Model : del_i, *t0, M, Q, V,* A_{CL}, H_{CL}, *InitCL* .

- del_i:A processing delay value,
- *t0*: Time of event,
- *M*: The type of transmitted messages,
- *Q*: The type of a FIFO queue messages,
- *V*: A vector of messages with type M,
- A_{CL}: Abstract variable of CL.

- H_{CL}: Abstract variable of CL.

CL Model as TIOA has variables to define the model and operations: $send_i^a$, Tx_i^d,Rx_i^d, $RTCL_i^d$,$myCL_i^d$, ch_i^d, $reqCL_i^d$, vCL_i^d .

- $send_i^a$: Analog variable which evolves with the time derivative of 1defines the passage of time after a request is sent by IL, it is bounded by del_i.
- Tx_i^d: The buffer that represents the transmission of messages.
- Rx_i^d: The buffer that represents the reception of messages.
- $RTCL_i^d$: The variable that stores the type of next slot whether RT or nRT.
- $myCL_i^d$: The variable that stores whether the device owns the next time slot or not.
- ch_i^d : The channel variable.
- $reqCL_i^d$: The variable that stores the request from IL to determine RTIL and myIL.
- $sendvCL_i^d$: The variable that stores the whether vCL is sent or not.
- vCL_i^d: The variable that holds the additional information of the protocol operations.

Actions in CL:

- *input AP2CL (m:M, ch: int)*$_i$
- *input IL2CLRT(m:M)*$_i$
- *input REQRT(m:M)*
- *output UPDVCL(vCL:* $A_{CL,}$ *Tx: V, Rx:Q,* $RTCL_i^d$*: bool)*$_i$
- *output RBACK(ILHT:* $H_{IL,}$ *cLHT:* H_{IL}*, rbSt: int)*$_i$
- *output CL2ILRT(*$RTCL_i^d$*:bool,*$myCL_i^d$*: bool, m:M)*$_i$
- *input CL2AP(q:Q)*$_i$

The transmission of messages for different channels are supported in CL. Channels have 2 parameters namely *b* which shows the device number and *c* which shows the channel number of that device. Current message for one channel is stored in Tx message buffer. Tx and Rx buffers are used to stores messages. $RTCL_i^d$, *myCl* and channel variable *ch* indicate if the next time slot is RT or nRT, the owner of the next time slot and its channel. These variables are updated when *vCL* is updated with internal functions. Send variable shows the passage of the time after a request is sent by IL and it is also bounded by del_i.

When CL gets message $\texttt{REQRT}(t)_i$ from IL layer, $RTCL_i^d$, *myCL* and *ch* variables are updated for the next time slot using $g_{RT}(vCL_i^d, RTCL_i^d, t)$ and $g_{my}(vCL_i^d, RTCL_i^d, t, i)$ functions. After getting $\texttt{REQRT}(t)_i$ message from IL, send analog variable gets 0 and *reqCL* boolean variable gets true. In the computation, unique sender for RT slot is determined to avoid collision. *myCL* variable shows the owner of the next time slot. While computing in CL layer, $RTCL_i^d$, $myCL_i^d$, vCL_i^d, $reqCL_i^d$, ch_i^d, *t*(timing information) and *m.par* (RT message parameters) are used by CL. If send (the computation time) does not exceed the del_i and $reqCL_i^d$ variable is true, CL sends $\texttt{CL2ILRT}(RTCL_i^d, myCL_i^d, m)$ to the IL layer. Also del_i variable should be less than rem-cmp time. In $\texttt{CL2ILRT}(RTCL_i^d, myCL_i^d, m)$,if $myCL_i^d$ variable is true, Tx_i^d buffer gives message *m* to the IL layer.

In action UPDVCL, if *sendVCL* is true, CL sends *vCL* parameters of that time slot to DP each time slot. In action RBACK, IL gets *vCL* parameters, Rx and Tx message from DP and update its *vCL* parameters with the receved *vCL* parameters.

Figure 11: CL Model as TIOA

CL shares the action $AP2CL(dat,p,ch)_i$ with the upper layer. and the action $IL2CLRT(m,t)_i$ with the lower layer . After $AP2CL(dat,p,ch)_i$, data *dat* and the protocol parameters *p* are stored in $Tx_i^d[ch].data$ and $Tx_i^d[ch].par$ on the CL layer. After $IL2CLRT(m,t)$, the message *m* from IL is stored in Rx_i^d buffer and vCL_i^d variable is updated with the action of $g_{upd}(vCL_i^d, m.par, t)$. Upper Layer control application shares the action $CL2AP(Rx_i^d)_i$ with CL. After $CL2AP(Rx_i^d)$, message in Rx_i^d buffer is sent to the control application. The decision variables vCL_i^d, the slot type $RTCL_i^d$, variable which indicates the owner of the next slot time $myCL_i^d$ and related channel variable ch_i^d are updated with $g_{upd}(vCL_i^d, m.par, t)$, $g_{RT}(vCL_i^d, RTCL_i^d, t)$ and $g_{my}(vCL_i^d, RTCL_i^d, t, i)$ functions.

$g_{my}(vCL_i^d, false, t, i)$ = (false, 0),

$g_{my}(vCL_i^d, true, t, i)$ = (true, ch),
➔$g_{my}(vCL_i^d, true, t, j)$ = (false, 0) for all j ∈ I – {i}

In CL, variables are stored in the form of communication requests. Priority queue is a queue which stores the communication requests in the form of (*b,c,eT,dT*). *b* indicates a device, *c* shows the channel of that device, *eT* holds the eligibility time of the message and *dT* is the deadline time of the message which is started when the request is issued. In other words, the message in the priority queue is sent by device *b* with channel *c* at the eligible time *eT* and it should be sent before the deadline time *dT*. m.par.req parameter is a set of request from control application. After getting messages from control application, messages are pushed into *vCL.PQ* and they are ordered according to their eligibility time *eT* and deadline time *dT*. The request which is the most urgent eligibility time *eT* is sent to the lower layer.

$$g_{\mathrm{RT}}(\mathrm{vCL}_i, \mathrm{RT}_i, t) = \begin{cases} \text{true} & \text{if } \mathrm{vCL}_i.\mathrm{PQ.Top}.eT \le t \\ \text{false} & \text{otherwise} \end{cases}$$

$$g_{\mathrm{my}}(\mathrm{vCL}_i, \mathrm{RT}_i, t, i) = \begin{cases} (\text{true}, a) & \text{if } \mathrm{PQ}_i.\mathrm{Top}.b = i \wedge \mathrm{RT}_i \\ & = \text{true} \wedge \mathrm{PQ}_i.\mathrm{Top}.c = a \\ (\text{false}, 0) & \text{otherwise} \end{cases}$$

Figure 12: Update Functions for CL

Figure 12 shows the update functions for CL. In $g_{upd}(vCL_i^d, m.par, t)$ function, if RTCL equals true,in other words , the next time slot is RT slot, the first request in the priority queue is popped and the request is sent to the lower layer. However, if RTCL is false (the next time slot is nRT slot), the first request reenters the priority queue. If the eligibility time of the request at the top of the priority queue (*vCL*$_i^d$*.PQ.Top.eT*) is smaller than current time t, $g_{RT}(vCL_i^d, RTCL_i^d, t)$ function returns true. Thus, the type of next slot time is determined whether it is RT or nRT. If the request at the top of the priority queue's owner is (*PQ*$_i$*.Top.c*) that device and the request at the top of the priority queue's channel is that device's channel $g_{my}(vCL_i^d, RTCL_i^d, t, i)$ function returns true and that device's channel *a*.

3.2.3 Generic Shared Medium Model:

Shared Medium Layer Generic Model is defined using TIOA Model in the article [12] [43]. Figure 13 shows the SM Model using TIOA. There are 6 variables in the SM Model : *mess*^{d}, *coll*^{d} , *next*^{d} , *now*^{a}, *M, N.*

- *mess*^{d}: The parameter that indicates currently transmitted message
- *coll*^{d}: Theparameter that indicates whether the collision is happened or not.
- *next*^{d}: The parameter that indicates the next reception time
- *now*^{a}: Analog parameter which evolves with the time derivative of 1 , the updated time information is provided by this variable.

- *M:* The type of transmitted messages.
- *N:* The parameter that indicates the number of messages.

Actions in SM:

- *input IL2SM(m:M)*$_i$
- *output SM2ILDP(m:M)*

TIOA $SM(N : int, M : type)$

states

$\texttt{mess}^{\texttt{d}} : M :=$ empty
$\texttt{coll}^{\texttt{d}} :$ Bool := false
$\texttt{next}^{\texttt{d}} :$ int := 0
$\texttt{now}^{\texttt{a}} :$ Real := 0

transitions

input IL2SM$(m)_i$
eff:
if $(\texttt{coll}^{\texttt{d}} = \text{false}) \quad \wedge(\texttt{mess}^{\texttt{d}}$ is empty)
$\texttt{mess}^{\texttt{d}} := m$
$\texttt{next}^{\texttt{d}} := m.\text{length}$
else
$\texttt{coll}^{\texttt{d}} :=$ true
$\texttt{next}^{\texttt{d}} := 0$
set $\texttt{mess}^{\texttt{d}}$ empty
$\texttt{now}^{\texttt{a}} := 0$

trajectories

stop when
$(\texttt{now}^{\texttt{a}} = \texttt{next}^{\texttt{d}}) \wedge (\texttt{mess}^{\texttt{d}}$ not empty)
evolve
$d(\texttt{now}^{\texttt{a}}) = 1$

signature

input IL2SM$(m : M)_i$
output SM2ILDP$(m : M)$

output SM2ILDP(m)
pre:
$(\texttt{now}^{\texttt{a}} = \texttt{next}^{\texttt{d}}) \wedge (\texttt{mess}^{\texttt{d}}$ not empty
eff:
set $\texttt{mess}^{\texttt{d}}$ empty
$\texttt{next}^{\texttt{d}} := 0$

Figure 13: SM Model as TIOA

There are 2 actions shared with the IL layer namely `IL2SM(m)` and `SM2IL(m)`. In `IL2SM(m)`, if *mess* variable is empty in SM layer, message *m* is sent to the SM layer and next variable is updated. If *mess* variable is not empty in SM layer, collision is occurred and *coll* variable equals true. In this case *next* variable is set to 0 and mess variable is set to empty. If next equals to now in SM layer, SM does action `SM2IL` to the IL layer. In `SM2IL` transition, *m* variable is set to *mess* variable. Thus the message *m* can be sent by SM. After that, *mess* variable is set to be empty.

3.2.4 Generic Dependability Plane Model:

Dependability Plane (DP) Generic Model is defined using TIOA Model [43]. Figure 14 shows the DP Model using TIOA. There are 7 parameters in the DL Model : *cyc*, t_0,t_1,A_{IL}, A_{CL}, A_{DP}, *InitDP* .

- *cyc:* The variable shows the number of the cycle.

- t_0,t_1: Time of events.
- A_{IL}: Abstract variable of IL.
- A_{CL}: Abstract variable of CL.
- A_{DP}: Abstract variable of DP.
- *InitDP*: Abstract variable of DP

Figure 14: Functions in Dependability Plane

DP Model as TIOA has variables to define the model and operations: $atRes_i^a$, $rbReq_i^d$, $rtSlot_i^d$, $stNo_i^d$, $nodeID_i^d$, $rbSt_i^d$, cnt_i^d, $vILHist_i^d[cyc]$, $vCLHist_i^d[cyc]$, vDP_i^d, $rVCL_i^d$.

- $atRes_i^a$: Boolean variable keeping the acceptance test results
- $rbReq_i^d$: Boolean variable keeping the rollback requirement.
- $rtSlot_i^d$: Boolean variable keeping the type of the time slot.
- $stNo_i^d$:Integer variable used for state transition
- $nodeID_i^d$:Integer variable keeping the message transmitting node ID
- $rbSt_i^d$:Integer variable keeping the rollback state number
- cnt_i^d: Integer counter variable for periodic operation
- $vILHist_i^d[cyc]$: Data structure keeping the vIL history
- $vCLHist_i^d[cyc]$: Data structure keeping the vCL history
- vDP_i^d: Data structure to keep the information required for dependability checks. Now it just holds the non real-time slot ownership information as an integer array
- $rVCL_i^d$. Data structure keeping the transmitting node's vCL variable for dependability checks.

Also, some new parameters are added to the header of the messages.

- $nodeID_i^d$: transmitting node's ID.
- vCL_i^d: transmitting node's vCL.
- $atRes_i^a$: transmitting node's acceptance test result.
- $rbSt_i^d$: the rollback state in case of a failure.

Actions in DP:

- *input* $UPDVIL(vIL: A_{IL,}\ TxnRT: Q,\ RXnRT: Q)_i$
- *input* $UPDVCL(vCL: A_{CL,}\ Tx: V,\ Rx:Q,\ RTCL_i^d: bool)_i$
- *input SM2ILDP(m:M)*
- *internal* $ATEST()_i$
- *output* $RBACK(ILHT: H_{IL,}\ cLHT: H_{IL},\ rbSt: int)_i$
- *output* $SENDRES(atRes:bool,\ rbSt:int\ )_i$

Dependability Plane has interfaces with IL and CL. There are input actions shared with IL and CL namely `UPDVIL(vIL)` and `UPDVCL(vCL,RTCL)`. In `UPDVIL(vIL)`, vIL history $vILHist_i^d[cyc]$ is updated with vIL decision variable taken from IL. In `UPDVCL(vCL,RTCL)`, the type of the time slot, $rtSlot_i^d$ is updated with *RTCL* which is taken from CL layer that shows the next time slot is whether RT or nRT. Also, vCL history $vCLHist_i^d[cyc]$ is updated with vCL decision variable taken from CL. Also, an action `SM2ILDP(m)` is occurred in DP. In `SM2ILDP(m)`, transmitting node's ID (*m.nodeID*) is assigned to $nodeID_i^d$, transmitting node's vCL is assigned to $rVCL_i^d$ which keeps the transmitting node's vCL variable for dependability checks. The rollback state number $rbSt_i^d$ of message m is assigned to $rbSt_i^d$ in DP. If transmitting node's acceptance test result $atRes_i^a$ equals to false, the state transition number $stNo_i^d$ equals 3, if $atRes_i^a$ equals to true, the state transition number $stNo_i^d$ equals 1.

There are 2 output actions from dependability plane: `SENDRES(`$atRes_i^a$`,`$rbSt_i^d$`)` and `RBACK(`vIL_i^d`,`vCL_i^d`)`. Before `SENDRES` occurs, the state transition number $stNo_i^d$ should be 2, and after `SENDRES` $stNo_i^d$ is -1. Before `RBACK` occurs $stNo_i^d$ should be, and after that, counter variable for periodic operation equals to rollback state number plus 1 ($cnt_i^d = rtSlot_i^d + 1$). If counter for periodic operation cnt_i^d equals to number of the *cycle*, cnt_i^d gets 0.

$vILHist_i^d[cyc]$ and $vCLHist_i^d[cyc]$ variables are assigned to vIL_i^d and vCL_i^d variables in IL and CL . The state transition number $stNo_i^d$ equals -1 and the rollback requirement $rbReq_i^d$ gets false.

In the dependability plane, there is also an internal action `ATEST()`. Before `ATEST()` occurs, the state transition number $stNo_i^d$ should be 1, and after `ATEST()`, $stNo_i^d$ is -1.The counter variable cnt_i^d is incremented by 1 firstly in `ATEST()`. Then if rollback requirement $rbReq_i^d$ equals false, $f_{AT}(vCLHist, vDP, cnt, nodeID, rtSlot)$ function result is assigned to the transmitting node's acceptance test result $atRes_i^a$ and if acceptance test result $atRes_i^a$ equals true, the rollback state number $rbSt_i^d$ equals the counter cnt_i^d minus 1 ($rbSt_i^d = cnt_i^d - 1$). If acceptance test result $atRes_i^a$ equals false, rollback requirement is needed and rollback state number $rbSt_i^d$ equals the counter cnt_i^d minus 2 ($rbSt_i^d = cnt_i^d - 2$). If rollback state number $rbSt_i^d$ equals -1, rollback state number $rbSt_i^d$ equals cyc minus 1 ($rbSt_i^d = cyc - 1$). If the counter cnt_i^d equals to cyc, counter cnt_i^d is assigned to 0 and , the state transition number $stNo_i^d$ is assigned to 2.

Figure 15 shows the example of f_{AT}. In function of acceptance test f_{AT} , there are 4 parameters using input of function, $vCLHist_i^d$, vDP_i^d, cnt_i^d, $nodeID_i^d$, $rtSlot_i^d$. If vIL history $vILHist_i^d[cnt-1]$ equals to the transmitting node's vCL variable for dependability checks $rVCL_i^d$ and if the type of the time slot $rtSlot_i^d$ equals true (slot is RT) and the device of the on the top of the priority queue of vIL history $vILHist_i^d[cnt-1].PQ.Top.b$ equals the *nodeID*, f_{AT} function returns true in other words acceptance test result is passed . If slot $rtSlot_i^d$ equals false (slot is nRT) and $vDP_i^d[cnt-1]$ equals *nodeID*, f_{AT} function returns true, otherwise f_{AT} function returns false and acceptance test result is failed.

```
fAT(vCLHT_i^d, vDL_i^d, rVCL_i^d, nodeID_i^d, rtSlot_i^d)
 if (rtSlot_i^d == true) ∧ (vCLHT.PQ_i.Top.b == nodeID)
  if vCLHT_i^d == rVCL_i^d
   return(true)
 else if (rtSlot_i^d == false) ∧ (vDL[cnt − 1] == nodeID)
  return(true)
 else
  return(false)
```

Figure 15: Example for f_{AT}

3.3 D²RIP Implementation

D²RIP is the predecessor of D³RIP which includes all layers as described before except for the dependability plane. D²RIP is implemented over Intel Gigabit Ethernet driver on PCs. While developing the framework, Lubuntu (Linux Kernel 3.6.2 with the RT patch) is used as

the operating system. Several changes are made on kernel configuration of Linux to provide precise timing and needed task scheduling mechanism. These changes are: [44]

- The amount of memory is increased for TCP/UDP socket I/O queues.
- BIOS Settings are changed to disable non-maskable interrupts. Because operating system cannot disable these interrupts.
- For low latency, network interface card NAPI is disabled.
- To create an interrupt for all incoming messages, InterruptThrottleRate is set to 0.0
- For instant message transmission starting, TxIntDelay is set to 0.0
- To prevent power saving state, EEE is set to 0.0
- To disable re transmission in Ethernet, E1000_COLLISION_THRESHOLD is set to 0.

In implementation, synchronization of nodes is needed to run system. IEEE 1588 protocol with hardware time-stamping is used for precise clock synchronization. For IEEE 1588 protocol, Intel Gigabit CT Desktop Adapter is used. In this adapter, e1000 driver module with version 2.2.14 is used. So, while software development, codes regarding implementation are added on the e1000 driver module.
In D^2RIP, standard Ethernet frames are used. Different types of frames are used in standard Ethernet frames. In RT frame type, RT messages have 14 Bytes CL message header and CL messages have 14 Bytes Ethernet header. In fragmented nRT frame type, fragmented payload messages have 8 Bytes fragment header and nRT Fragment messages have 14 Bytes Ethernet header. Figure 16 below shows the data encapsulation of RT and long nRT messages.

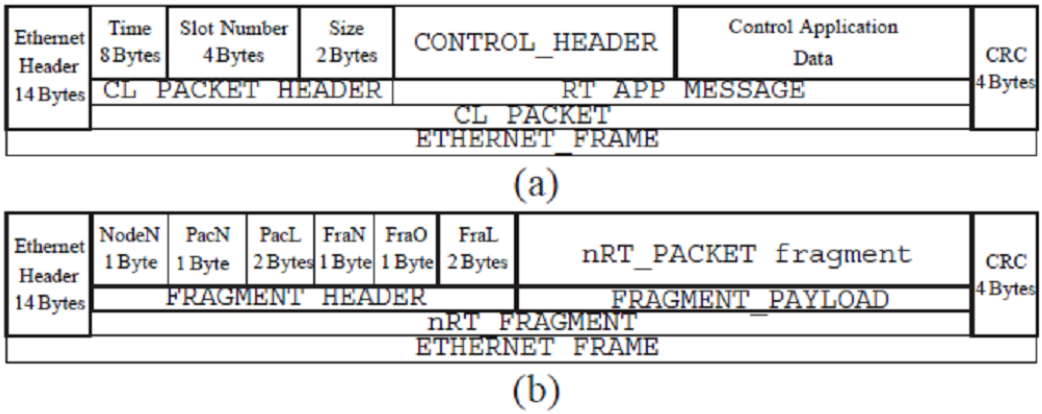

Figure 16: Data Encapsulation of RT and Long nRT messages [44]

3.3.1 Interface Layer (IL) :

Interface layer (IL) is the layer which lies between shared medium and coordination layer (CL). It gets RT and nRT messages from shared medium and stores RT messages in the RxRT message buffer and stores nRT messages in the RxnRt message buffer. It sends RT messages to the CL using `IL2CLRT` message and sends nRT messages to the application layer using `IL2APNRT`. IL layer gets also RT message and nRT messages from CL and

application layer. It stores these messages in the TxRT and TxnRT message buffers and they are sent to the shared medium using `IL2SM` message.
Interface layer was implemented in RT Linux kernel-space part. Figure 17 illustrates the message transmission in IL.

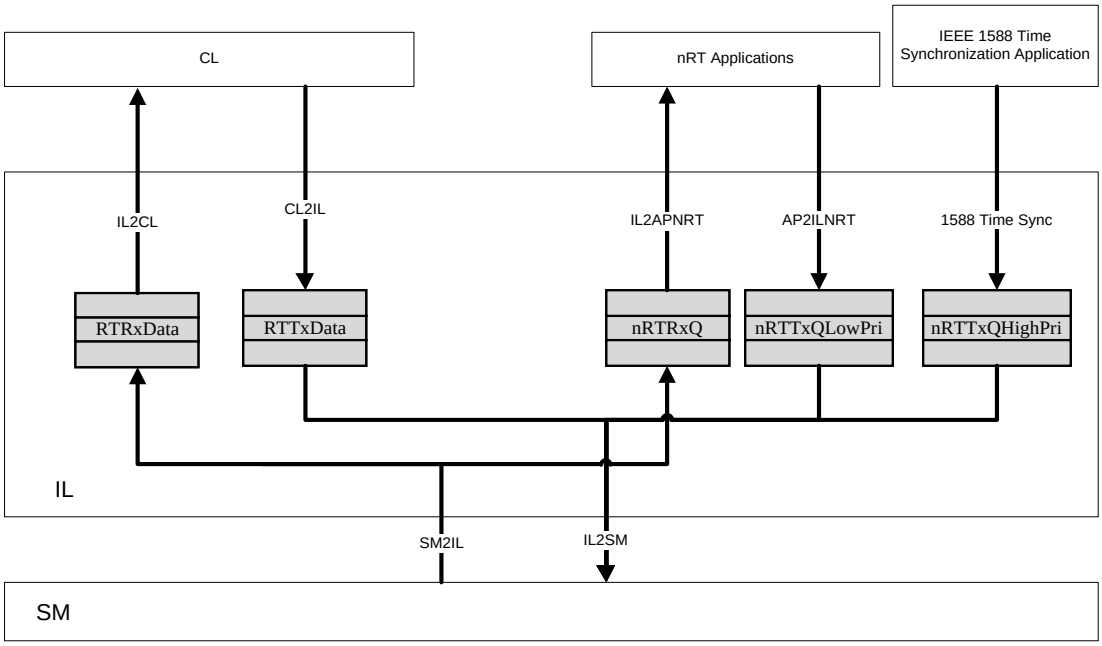

Figure 17: Message Transmission in IL layer

The communication method in D^{2}RIP is time division multiple access (TDMA). In TDMA structure, all nodes have different slots to transmit the message to the other nodes. IL provides this TDMA structure to avoid collision in the shared medium. This TDMA structure is synchronized using 1588 precise time synchronizing protocol and all nodes start TDMA at the same time to be synchronized using SYNC message. At the startup point, one of the nodes in the network behaves as a master node and sends a special SYNC message to all nodes to determine the start of the operation. After that, TDMA structure is started and all nodes in the network know that TDMA is started. It is also maintained using IEEE 1588 Precise Time Protocol. When TDMA structure is implemented on this framework, the issue which is related TDMA structure with fixed time slots is raised. Message packets especially nRT packets can be bigger than time-slot size. To overcome this problem, nRT packets which are bigger than determined message size should be fragmented before transmission and reassembled after transmission. So that, there are fragmenter and defragmenter threads in the IL thread. Table 2 shows the frame header structure. In the frame header structure, nodeID shows the id of the source node, packetID indicates the id of the packet which is transmitted from source node (nodeID), packetLength stores the total length of packet which is unfragmented, frameNum shows the total frame number, frameSeq indicates the sequence number of the frame, frameLength stores the data length in the frame. Fragmenter thread fragments the nRT packets which is bigger than determined message size and transmits the

fragmented packets to shared medium. After destination node receives the fragmented packets, reassembly thread assemblies the fragmented packets.

Table 2: Frame Header Structure

nodeID: unsigned char packetID: unsigned char packetLength: unsigned short int fragNum: unsigned char frameSeq: unsigned char frameLength: unsigned short int

If the received message is nRT, IL triggers the `IL2APNRT` action which is shared with application layer to transmit the nRT message and application layer sends nRT message to the IL using `AP2ILNRT`. Also, the time synchronization messages are sent as nRT message to the application layer. The difference between nRT and the time synchronization messages is 1588 messages' priority is higher than nRT messages'. So that, there are 2 priority queue in the IL to separate the nRT messages and time synchronization messages namely high priority queue (nRTTxQHighPri) and low priority queue (nRTTxQLowPri). When nRT packet comes from SM, IL puts time synchronizing packets into the high priority queue and puts nRT packets into the low priority queue. When an nRT slot comes for that device, IL controls the high priority queue firstly and packets in the high priority queue is sent to application layer.

There are two shared actions between the shared medium layer and the interface layer: `SM2IL` and `IL2SM`. In `SM2IL`, shared medium sends RT and nRT message to the IL. There are 3 different cases. If ETHERTYPE is 0x2200 in the frame, it means that fragmented nRT packet is received and this message is forwarded to Reassembly thread. After reassembling, packet is sent to IL. If the packet is nRT and shorter than packet size, it also sends to the IL. In third type, ETHERTYPE is 0x1100 in the frame, it is also sent to the IL. Then IL transmits the received message to the RTRxData or nRTRxQ buffer depends on receiving packet's protocol which is written in `IL2SM` action before packet is sent by other node.If receiving packet's protocol is RT, the received message is stored in the RTRxData message queue. If receiving packet's protocol is nRT, the received message is stored in nRTRxQ message queue. In `IL2SM`, if the device owns the time slot (myIL=true, if and only if cnt belongs to nRTSet), the type of time slot is determined and the message queue which is related the type of message has at least a message, IL sends message to the SM. If the slot is determined for RT traffic (RTIL=true), the message m is transferred from RTTxData. If the slot is determined for nRT traffic (RTIL=false), the message m is transferred from nRTTxQHighPri or nRTTxQLowPri depends on the type of nRT message. (IEEE 1588 Time Synch or nRT from AP) If the message m is nRT message from the application and it's longer than standard packet size, IL wakes up Fragmentation Thread and then send fragmented message m to SM.

There are also two shared actions between the interface layer and the coordination layer namely `IL2CL` and `CL2IL`. In `IL2CL`, interface layer gets packet from SM, it copies that packet into the RTRxData message queue, then it sends RT message m to the coordination layer. In `CL2IL` communication message, message m is copied from CL using

`copy_from_user` function. Then the message length is different from 1 byte, receiving message is copied into the RTTxData message queue. If the message length is 1 byte and the receiving message's first byte is set to RTIL and myIL variable is set according to return of slotOwner(RTIL). If the receiving message is 0xFF, RTIL is set to false. Thus, it can be seen that RTIL and myIL is determined according to message coming from CL.

For nRT communication between application layer and interface layer, there are 2 actions: `IL2APNRT` and `AP2ILNRT`. In `IL2APNRT`, the nRT message m which is stored in nRTRxQ message buffer is sent to the application layer and nRTRxQ message queue is set empty. In `AP2ILNRT`, application layer sends nRT message and it is stored in nRTTxQLowPrimessage buffer in the IL.

3.3.1.1 SM- IL Interface Implementation

In the implementation of SM-IL interface, Ethernet driver functions are used and new functions are implemented on the Ethernet driver source code. For implementing the interface between SM and IL, modular structure of Linux is used. The driver source code did not change directly, instead the driver source code of the network internet card (NIC) is downloaded from its website and modification of driver functions and new functions are added to the driver source code. After that, the original Ethernet driver module is removed and the updated version module is added to the kernel.

In the implementation of the IL, receive and transmit functions are used and modified. Binary exponential back-off algorithm is disabled. Because in back-off algorithm, if a collision occurs on the network, a node which wants to communicate to other one wait a random time and start a communication again. This causes that TDMA operation is collapsed. So back-off algorithm is disabled. Figure19 shows the algorithm of the transmit function and modification in the transmit function. In Figure 18, IL gets RT and nRT messages from user space. RT packets go into IL module firstly, then they are transmitted to SM by `e1000_xmit_frame()` function. nRT packets go into e1000e module firstly. If the packets length bigger than standard packet size, they are transmitted to fragmentation module. If the packet length is smaller than standard packet size, they are transmitted to `e1000_xmit_frame()` function. If the packets are 1588 time synch packets, they are transmitted to IL module.

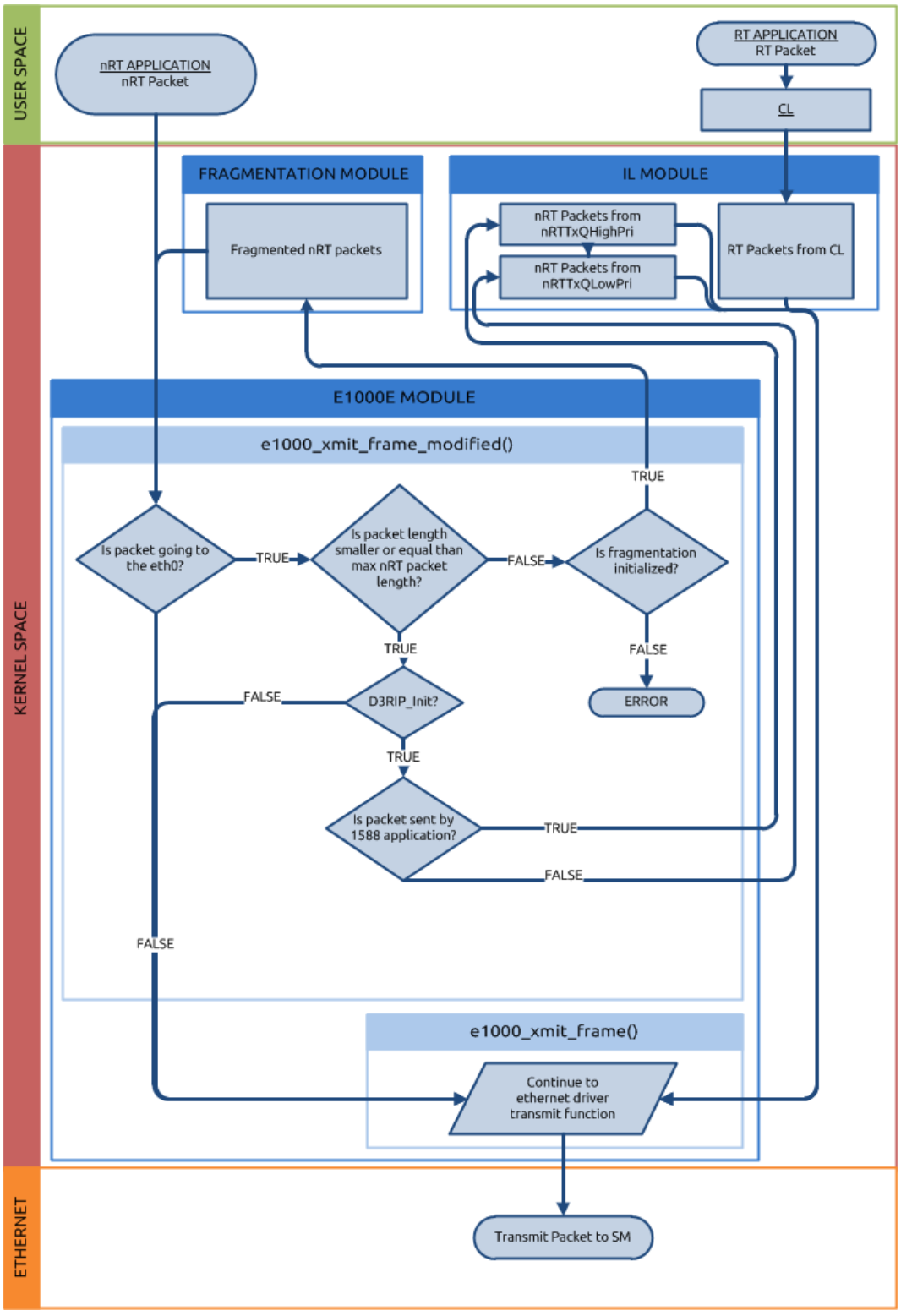

Figure 18: The Algorithm of the Transmit Function

Figure 19 shows the algorithm of the receive function and modification in the receive function. In Figure 19, packets come from SM and they go into the e1000e module. In this module, if packets are not fragmented, they continue to receive function and nRT

application. If packets are fragmented, they continue to defragmentation thread. If packets are RT, they continue to `IL2CL` function. Then they are sent to CL.

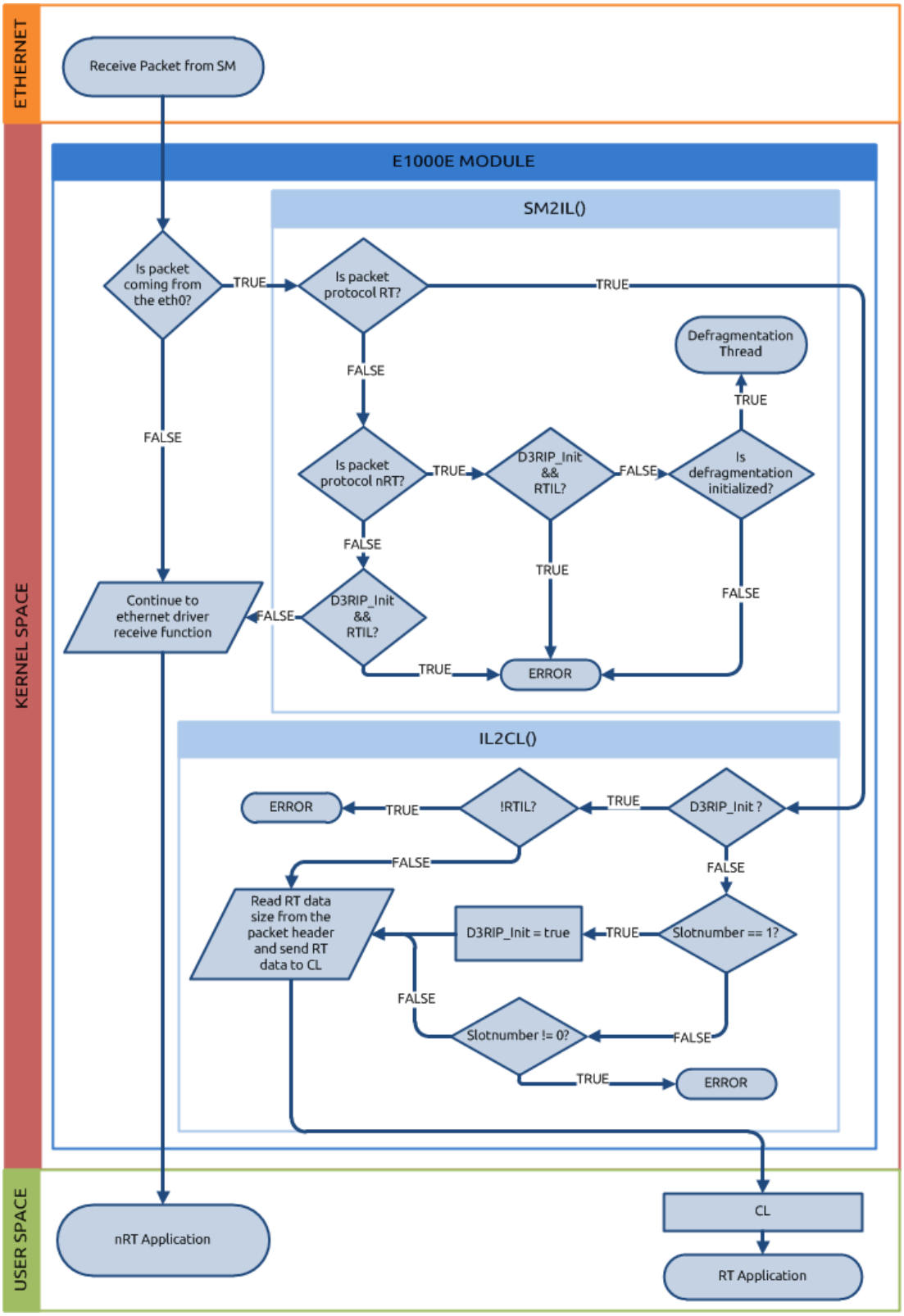

Figure 19: The Algorithm of the Receive Function

3.3.1.2 CL- IL Interface Implementation

In the implementation of IL-CL Interface, character device files are used to transfer the messages between IL and CL. charDev_IL2CL and charDev_CL2IL device files are used for the communication of each layer. The reason why using character device file is that IL is implemented on kernel space and CL is implemented on user space. `Copy_from_user()` and `copy_to_user()` functions in the IL are used to write messages to the file and to read messages from file.

3.3.2 Coordination Layer (CL):

Coordination layer (CL) is the layer which lies between the control application and interface layer (IL). It gets the RT messages from control application, sends the RT messages to the interface layer. While processing the RT traffic, it calculates the best performance of delivery of the RT messages to the network. Also, CL calculates the slot allocations according to RT messages coming from control application and coordinates the RT traffic.Figure 20 below shows the message transmission in CL.

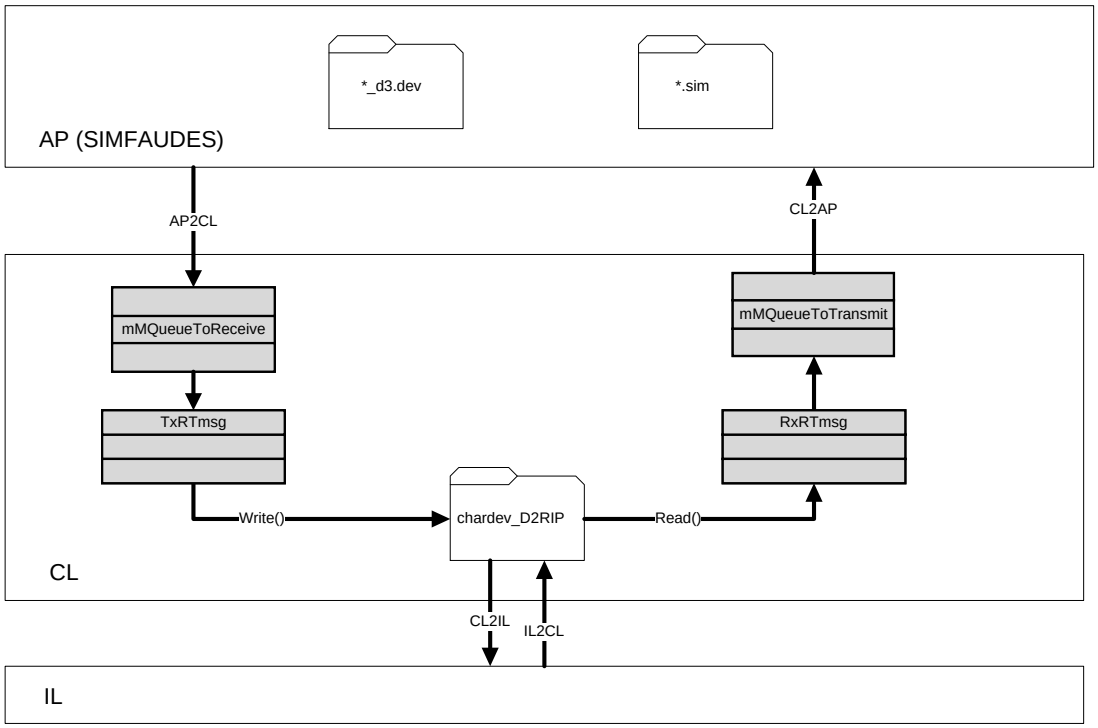

Figure 20: Message Transmission in CL layer

In coordination layer implementation, there is a thread which wakes up periodically at the start of the each time slot. At the beginning of the time slot, if mMQueueToReceive message queue has a message from AP, CL continues with `processAP2CL`. In `processAP2CL`, receiving message is copied to Tx. After that CL continues with the `processCLUPDATE` function. In this function, the type of the time slot (RT or nRT) is determined by looking at the PQ_dT priority queue. If there is no message in PQ_dT, RTCL is nRT. If there is any message in PQ_dT, RTCL is RT. The owner of the time slot is determined by looking at vCL

parameters in PQ_dT. If vCL parameter's node id equals that node's node id, mySlot returns true. If vCL parameter's node id does not equal that node's node id, mySlot returns false. After determining the type of the time slot and the owner of the time slot, if the time slot is RT and the owner of the time slot is mine (myCL=true), the message in the Tx is assigned to chardev_D2RIP character file. The reason why character file is used to communicate with IL is that CL is implemented on user-space. If the time slot is RT or nRT and the owner of the time slot is other (myCL=false), CL sends IL 1 byte to inform it.
If there is a message in chardev_D2RIP character file, CL calls `processCL2AP` function. In `processCL2AP`, receiving message is copied into Rx. After that gUpdate function is called to update the vCL parameters in the PQ_dT. Then the message in the Rx is assigned to mMQueueToTransmit message queue to send the message to Application Layer (AP). Figure 21 shows the structure of the message which is send to AP. It contains channel information (ch), CL protocol parameters and payload.

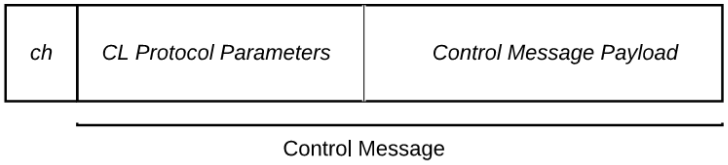

Figure 21: Message Structure

In CL, two priority queueare used to implement the protocol namely PQ_dT and PQ_eT.In PQ_dT, the communication requests are ordered with the deadline time of the events. In PQ_eT, the communication requests are ordered with the eligibility time of the events. PQ_dT and PQ_eT are used to store the communication requests which are needed at the time slot. Figure 22 shows the message format in the CL. It contains 3 parameters: number of requests (NoR), communication requests and control message payload. The number of requests (NoR) shows the count of request. Communication requests part contains requests with the node (b), channel (c), eligibility time (eT) and the deadline time (dT).

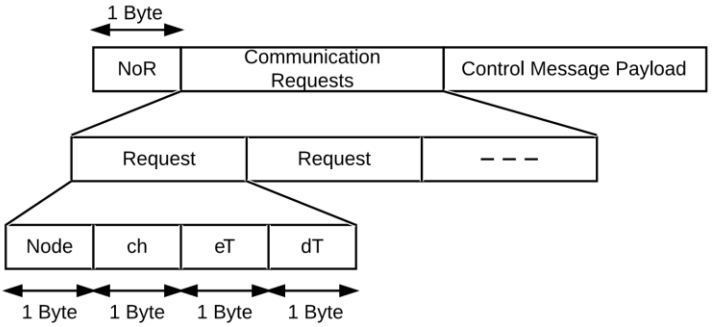

Figure 22: Message Format in CL [29]

Figure 23 shows the algorithm of the CL. CL waits until a RT message comes from IL in the runCL thread. Then it reads the start time from receiving RT message header. Meanwhile, it controls Tx for RT message. If there is no RT data available, it continues process CLUPDATE. In process CLUPDATE, RTCL and myCL are determined. If RTCL or myCL are false, CL sends 1 byte to IL and sleeps until the next slot. If RTCL and myCL true, CL gets packets from FIFO queue and send it to IL. It waits until a RT message comes from IL. Then it continues to process CL2AP. In this process, CL gets slot number, time and packet length info from the received RT message. If the packet length info is equal to the received RT message size and the packet length is not equal to the CL header length, the priority queue is updated and the received message is sent to AP. Finally, CL sleeps until next slot.

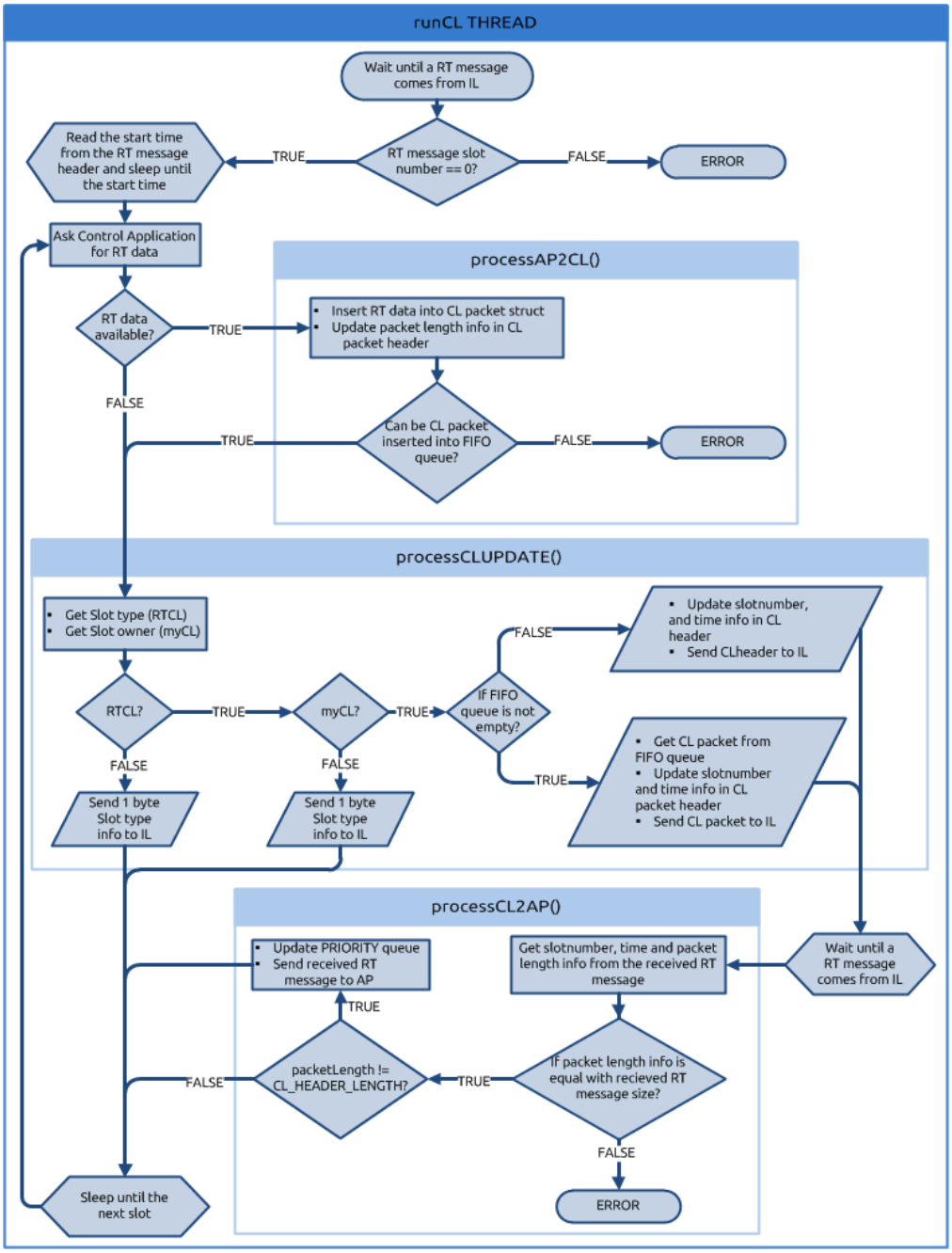

Figure 23: CL Algorithm [29] [41]

3.3.3 D²RIP Implementation Summary and Its Operation

As mentioned before, in D²RIP implementation, there are 2 layers over SM: CL and IL. Figure 24 shows the message transmissions of the layers and functions of D²RIP. When RT applications send message `AP2CL`, CL stores it in TxRT and it sends message `CL2IL`. After buffering in IL (inTxRT), IL sends SM and other nodes get the transmitted message using message SM2IL. Node which takes the RT message, stores its RxRT buffer. After that it sends to CL using message `IL2CL`. CL stores the message in RxRT and sends the message to the RT applications using `CL2AP`. When nRT applications send message to IL, IL buffers the nRT message in its TxnRT and sends the message to SM using `IL2SM`. Node which takes this nRT message, stores its RxnRT buffer in IL. After that it sends to nRT applications using UDP/TCP Socket.

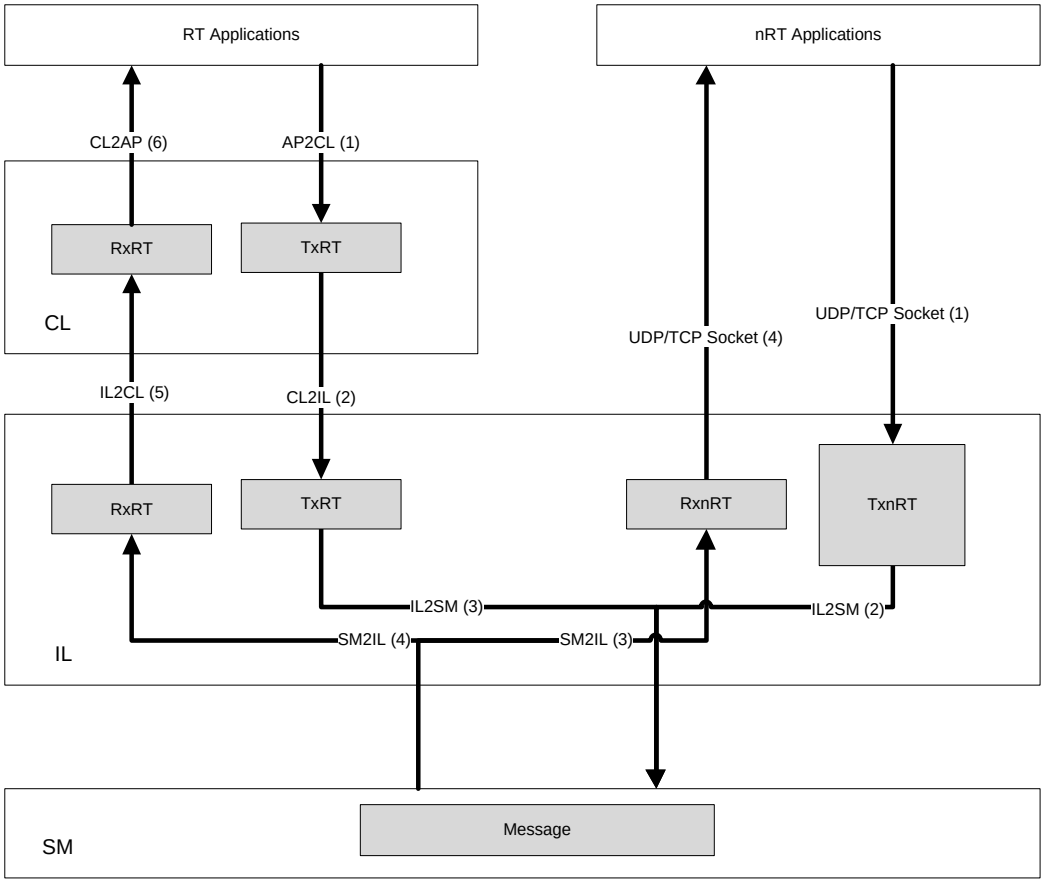

Figure 24: The Message Transmissions of the Layers and Functions of D²RIP

There are 3 different transmission events in the D²RIP framework: Sending RT request with RT packet, sending RT request without RT packet, sending nRT packet. Figure 25 shows the timing of the sending RT request with RT packet.

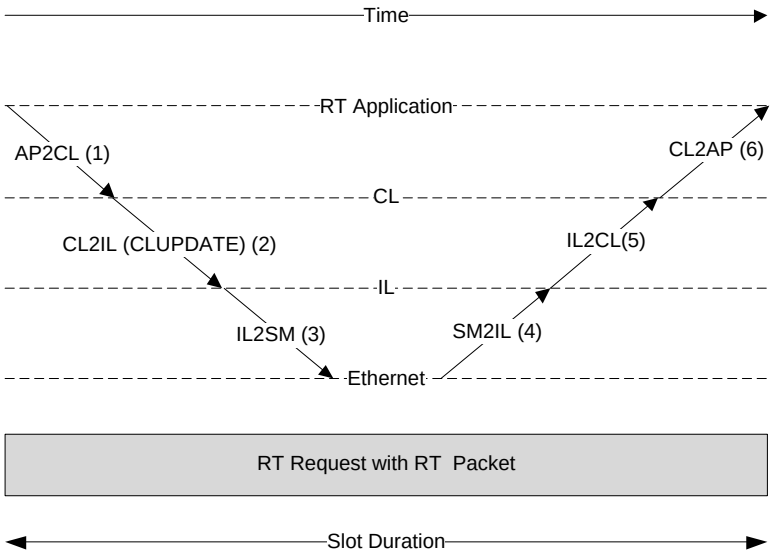

Figure 25: The Timing of the Sending RT request with RT packet

Figure 26 shows the timing of the sending RT request without RT packet.

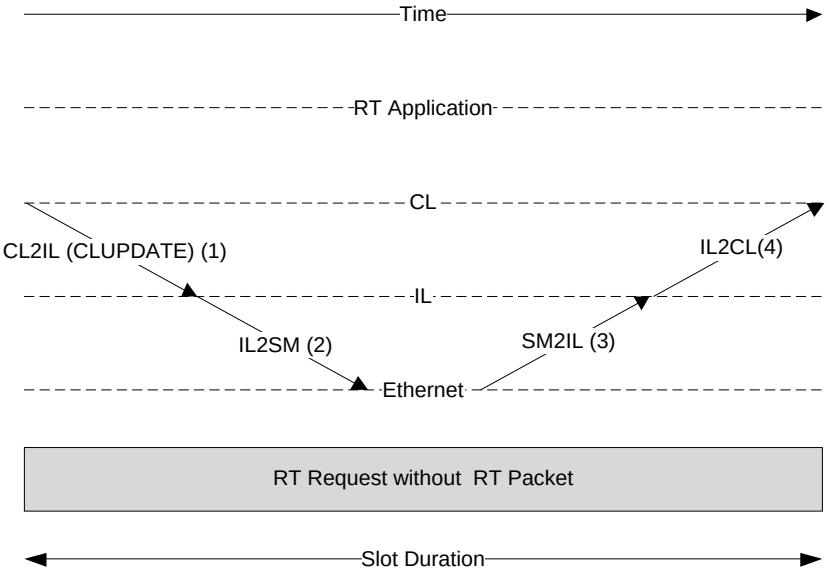

Figure 26: The Timing of the Sending RT request without RT packet

Figure 27 shows the timing of the sending nRT packet.

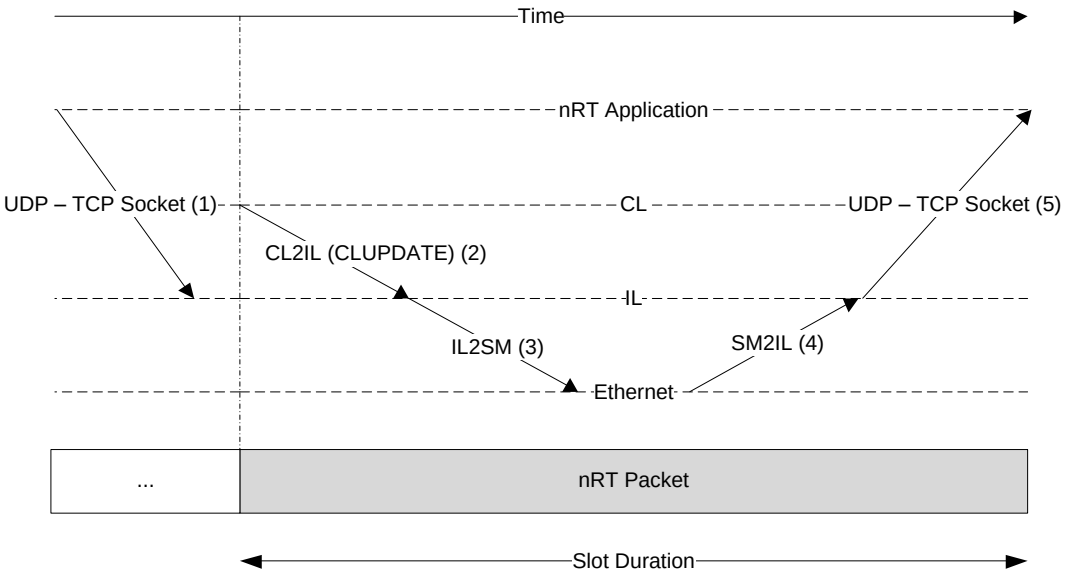

Figure 27: The Timing of the Sending nRT packet

CHAPTER 4

DEPENDABILITY PLANE IMPLEMENTATION

4.1 Overview

The dependability plane (DP) is the layer which has interfaces with the shared medium (SM), interface layer (IL) and coordination layer (CL). If there is a fault in the operation of D^3RIP, DP is supposed make sure that the fault is tolerated and the protocol maintains its correct operation. At the beginning of each time slot, the actions `UPDVIL` and `UPDVCL` are triggered by IL and CL to update the vCL and vIL history queues in DP. In the action `UPDVIL` , IL sends vIL parameters of the time slot (*cnt, cyc and slot*) to the DP. In the action `UPDVCL` , CL sends vCL parameters of the time slot priority queue of (nodeid, ch, eT and dT) tuples to the DP. DP also gets action `SM2ILDP` from IL when a message *m* is sent from any node. In the action `SM2ILDP`, there are 6 parameters (*atRes, rbSt, rVCl.nodeId, rVCl.ch, rVCl.eT, rVCl.dT*) which are sent for information about node which sends message *m*. In each time slot, the receiving vCL parameters and the vCL parameters of that time slot are compared with each other in the f_{AT} function in DP. If they are equal to each other, the function returns 1. Otherwise, the function returns 0. The returning parameter shows the acceptance test result. If a faultoccurred in the system (acceptance test result is failed), DP sends `RBACK` to the IL and CL to roll back the time slots. Messages *m* before the fault, are sent again and the system returns to its normal operation.

In this thesis, the dependability plane is implemented in the RT Linux user-space part. It communicates with the IL (in the kernel space) using a character device file and communicates with the CL using message queues. Figure 28 below shows message exchange of DP with other layers.

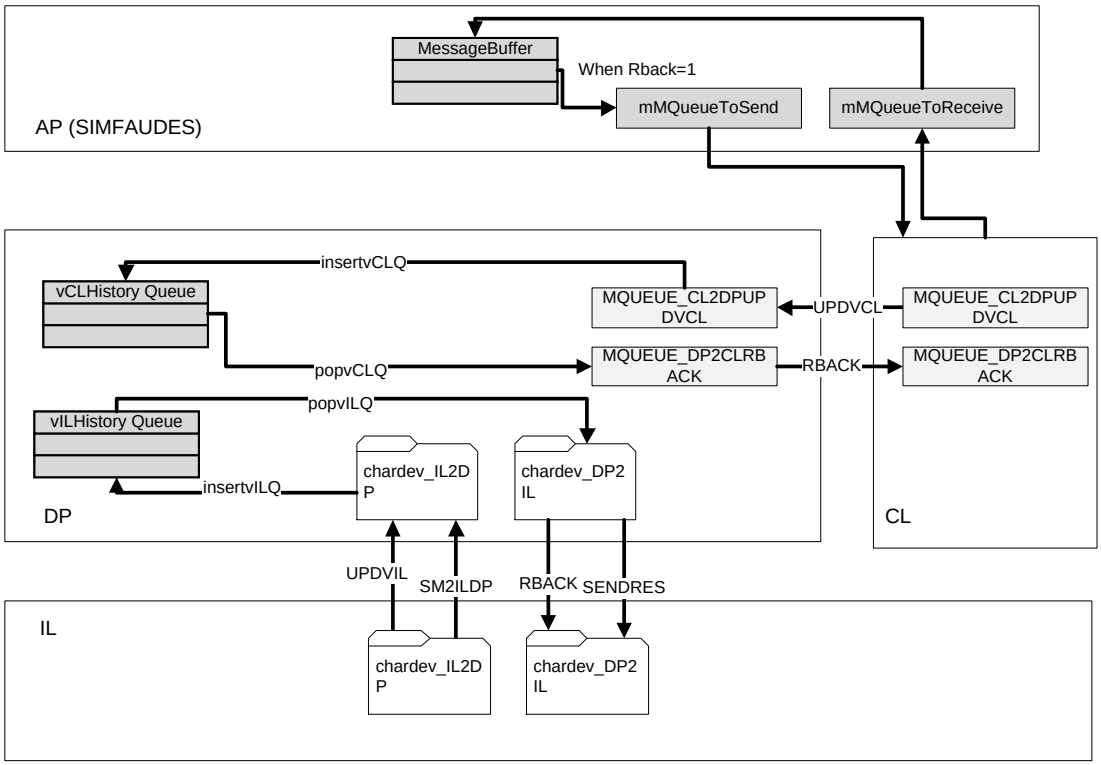

Figure 28: Message exchange of DP with other layers.

4.1.1 DP Implementation

In the dependability plane implementation, there are 2 threads which wake up periodically at the start of the each time slot. In the listenCoordinationLayer thread, DP is listening to the CL. If the MQUEUE_CL2DPUPDVCL message queue has a message from DP, DP continues with `processCL2DPUPDVCL`. In `processCL2DPUPDVCL`, DP calls the UPVCL function. In this function, the received vCL parameters are inserted into the vCLHistory FIFO queue. In the listenInterfaceLayer thread, DP is listening to the IL. If chardev_IL2DP character device file has a message from IL, DP looks at the first bit of the message. If that message bit (msg[0]) equals to 0, it continues with `processIL2DPUPDVIL`. In `processIL2DPUPDVIL`, DP calls the UPVIL function. In this function, the received vIL parameters are inserted into the vILHistory FIFO queue. If that message bit (*msg[0]*) equals to 1, DP continues with `processSM2IL`. In this process, the received message parameters (atRes, rbSt, rvCl.nodeId, rvCl.ch, rvCl.eT, rvCl.dT) are assigned to the variables. If atRes equals to 1, DP continues with the `ATEST()` function. In this function, DP calls `fAT()` and the variables (atResrbSt) are changed to coordinate the operations. In `fAT`, vCL parameters which are came with SM2ILDP (*rvCl.nodeId, rvCl.ch, rvCl.eT, rvCl.dT*) and vCL parameters which came with UPDVCL (*nodeid, ch, eT and dT*) are compared to each other. If they are equal, `fAT()` returns true. If they are not equal, `fAT()` returns false. After that the obtained results are sent with `SENDRES`. In `SENDRES`, atRes and rbSt are sent to IL. In IL, these variables are inserted to the message header when a message is sent to other nodes Thus, other nodes can learn the results of the acceptance test. If the time slot passes the acceptance test, the protocol operation continues with the next

time slot. If the time slot doesn't pass the acceptance test, DP triggers `RBACK` in IL and CL. In the action `RBACK`, the latest vCL and vIL parameters in the FIFO queues are popped up and these variables are sent to CL and IL. Figure 29 shows the structure of DP

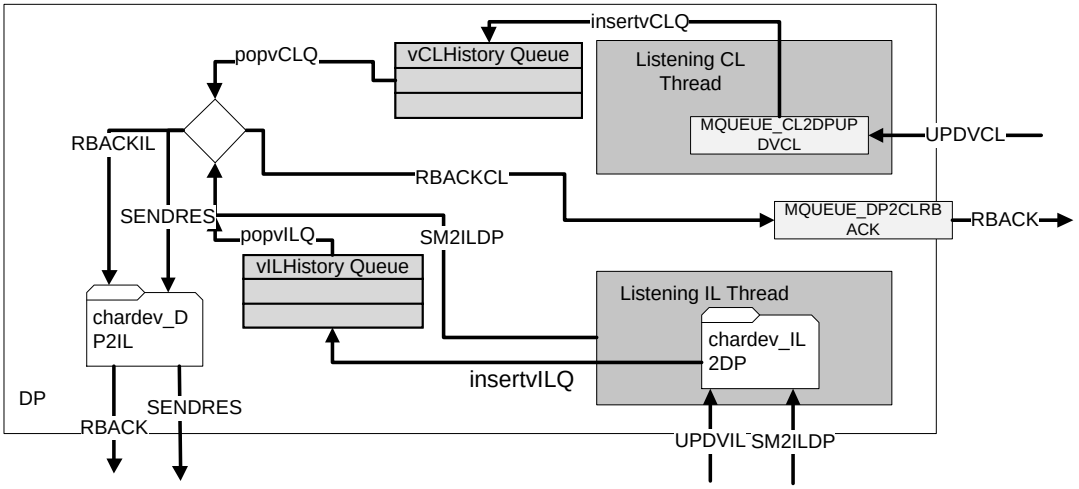

Figure 29: Structure of DP

In DP implementation, some changes are also made in IL, CL and AP, since the DP has interfaces with IL and CL. So there must be interfaces with DP in CL and IL. The implementation of these interfaces in CL and IL and the detailed information of the changes on the other layers with the flowchart of the IL, CL, AP are explained in the following sections. The first change on the other layers is adding a process and thread to CL. Figure 30 shows the CL implementation for dependability support. In CL, `processCL2DPUPDVCL` function is added to the runCL thread to send vCL parameters to the DP using the message queue. In `processCL2DPUPDVCL`, CL gets the vCL parameters of the time slot from PQ_dT. It assigns the vCL parameters to msgCL2DP. Then CL sends msgCL2DP to DP using message queue MQUEUE_CL2DPUPDVCL. Also, a new thread, Listen Dependability Plane Thread is created in CL to listen to DP. In this thread, CL listens to DP to get a RBACK message whenever DP issues `RBACK`. In `processDP2CLRBACK` function, CL reads vCL parameters from the message queue MQUEUE_DP2CLRBACK. It creates free space to insert vCL parameters into the PQ_dT. Then it assigns the msgDP2CL to vCL parameters and they are inserted into the priority queue.

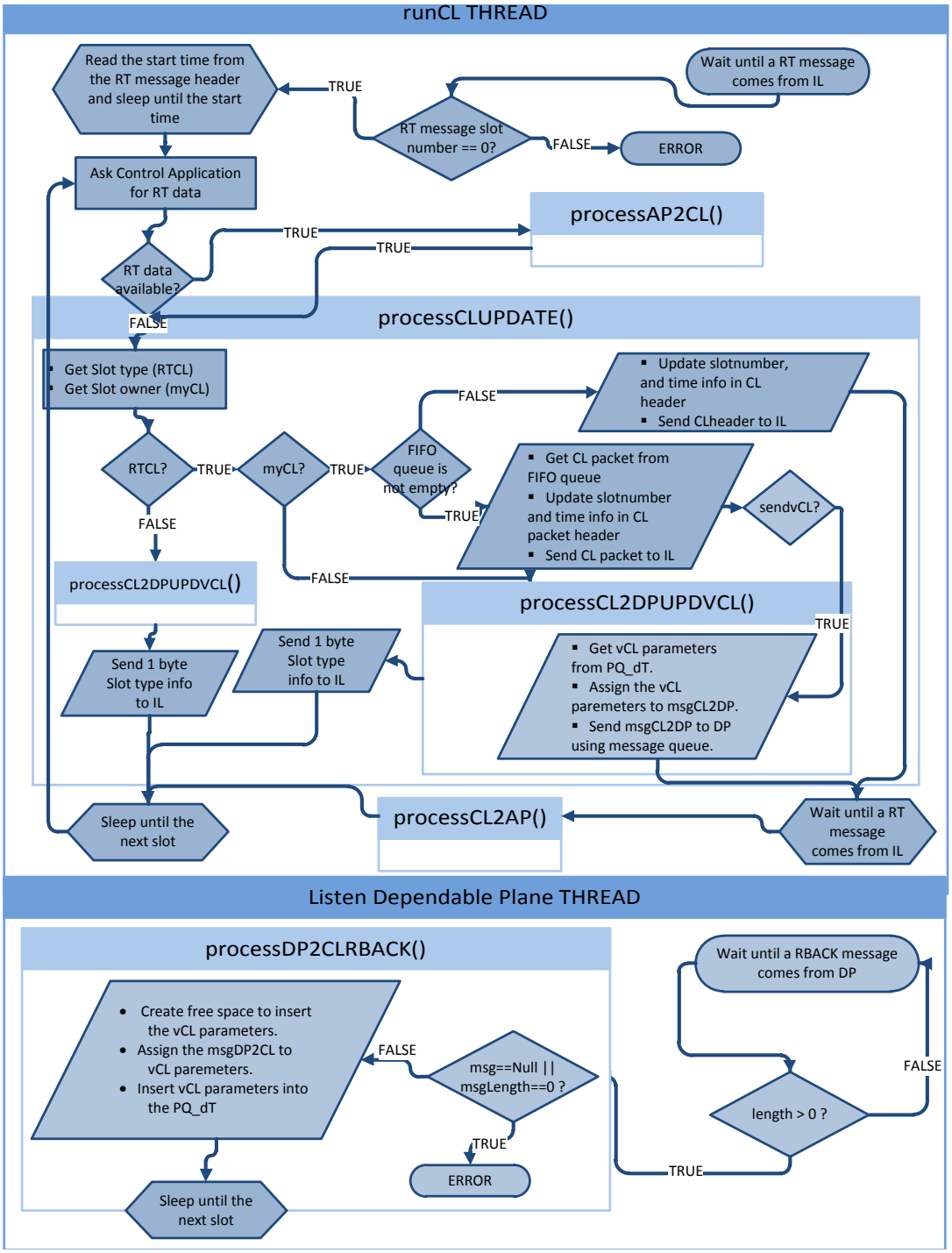

Figure 30: CL Implementation with DP

The second change on the other layers is adding a process and functions to IL. Figure 31 shows the transmission part of IL with dependability support. In IL, `UPDVIL` function is added to send vIL parameters to DP. In IL module, after getting slot information or RT data from CL, message is read and IL continues with `CL2IL`. After `CL2IL` function, `UPDVIL` function is called. After sending vIL parameters to DP, IL continues with `IL2SM` function. In the IL2SM function, IL sends atRes, rbSt and vCL parameters to the DP to loopback. Also, `SENDRES` and `RBACK` mechanisms are added to IL. If IL reads `SENDRES` or `RBACK`

from DP, it controls the first bit *(MyBuf[0])* of the receiving message. If it equals to 0, IL continues `RBACK` process. If it equals to 1, IL continues `SENDRES` process.

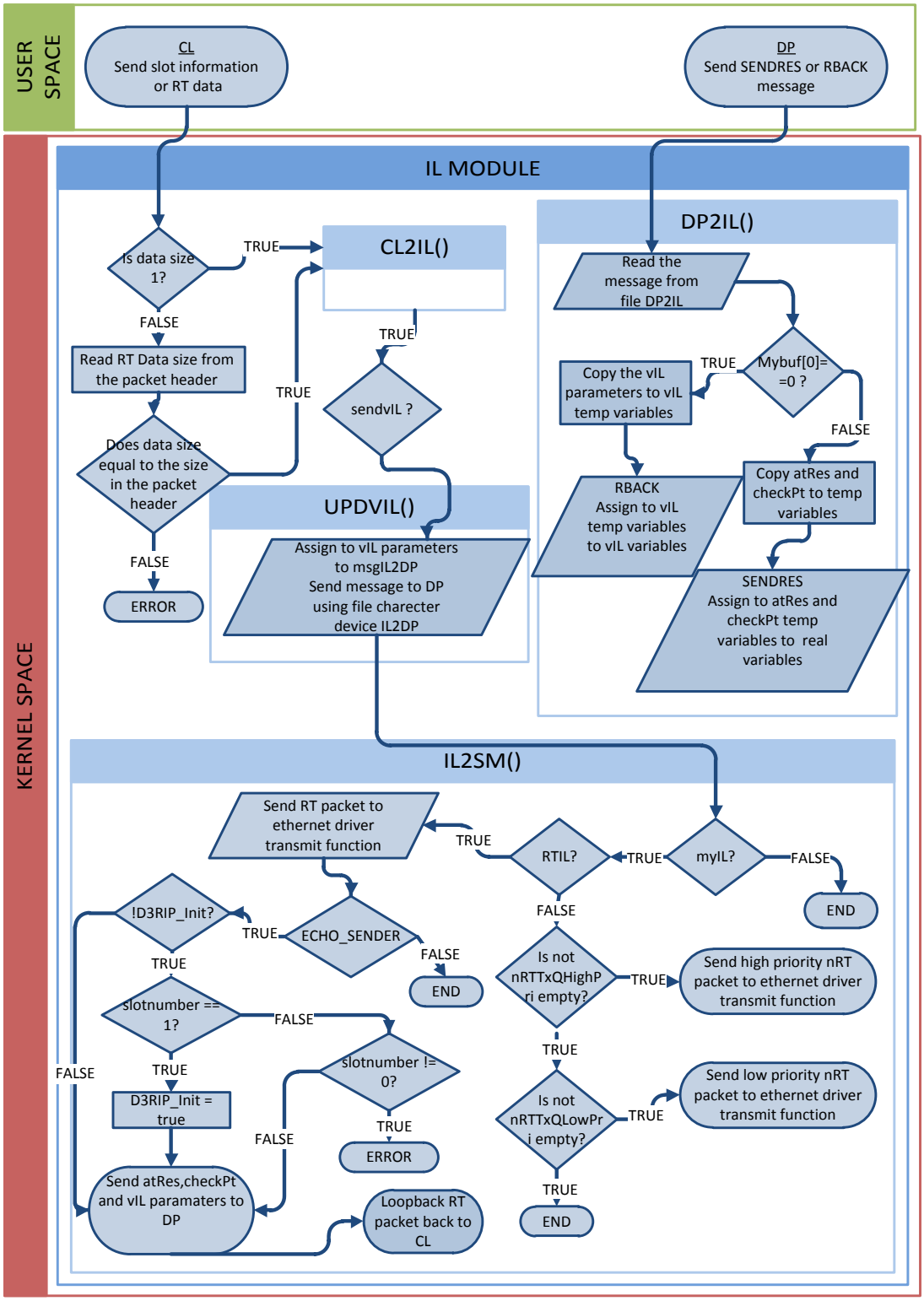

Figure 31: Transmission Part of IL with DP Implementation

Figure 32 shows the reception part of the IL with dependability support. In the e1000e module, `SM2ILDP` function is added to send vIL parameters to DP. After receiving a message from SM, IL continues with the `SM2IL` and `IL2CL` functions. In `IL2CL` function, RT data packet is read and IL sends *atRes, rbSt (checkPt)* and *vCL* parameters to DP. Then IL sends RT packet to CL.

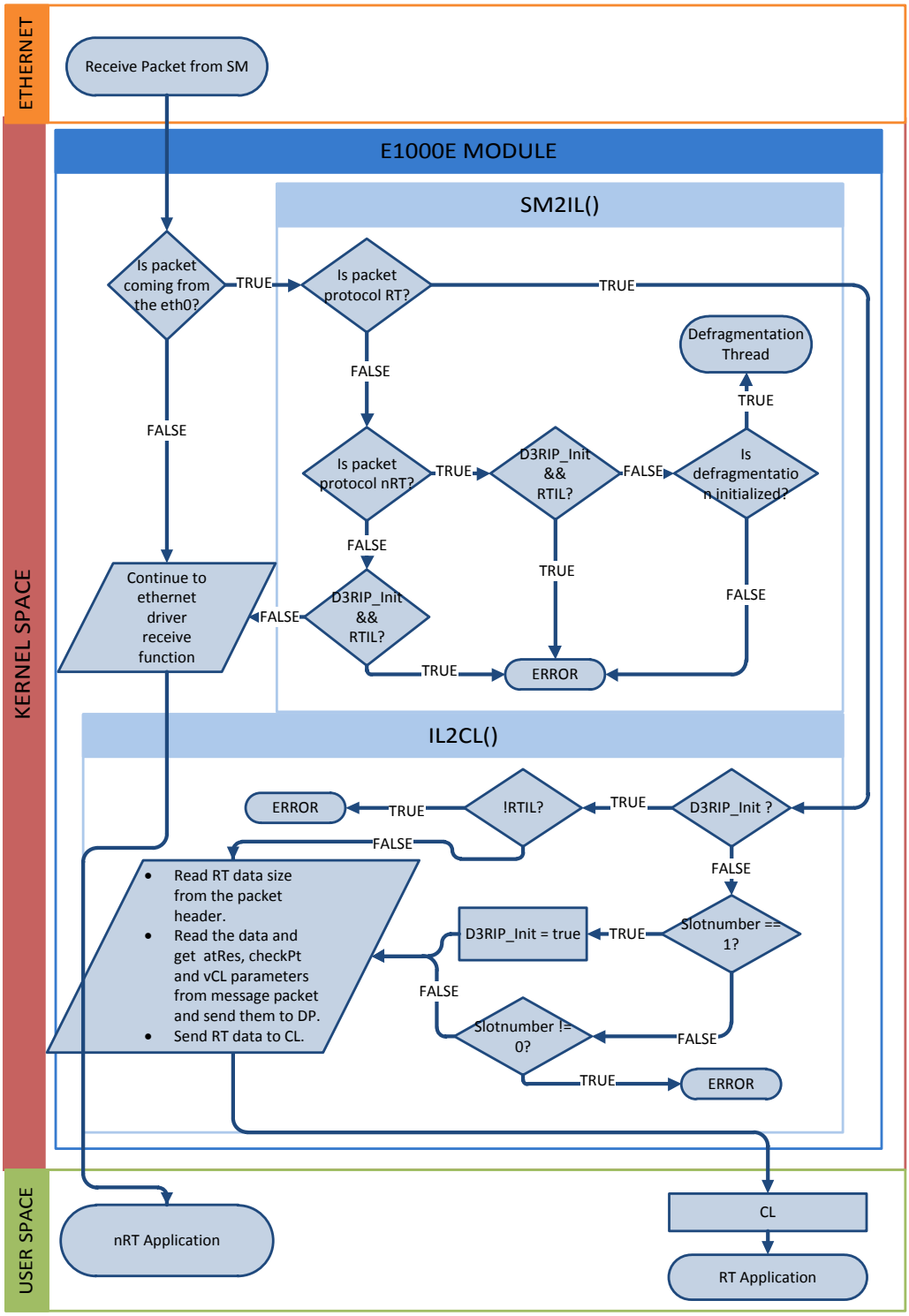

Figure 32: Reception Part of IL with DP Implementation

The third change on the other layers is adding functions to the RT application layer (AP). RT application layer sends RT messages via the CL. However, AP is not expected to resend messages when `RBACK` occurs. Hence, this feature is added to AP to resend messages after `RBACK`. The inclusion of `RBACK` in AP is performed as shown in Figure 33.

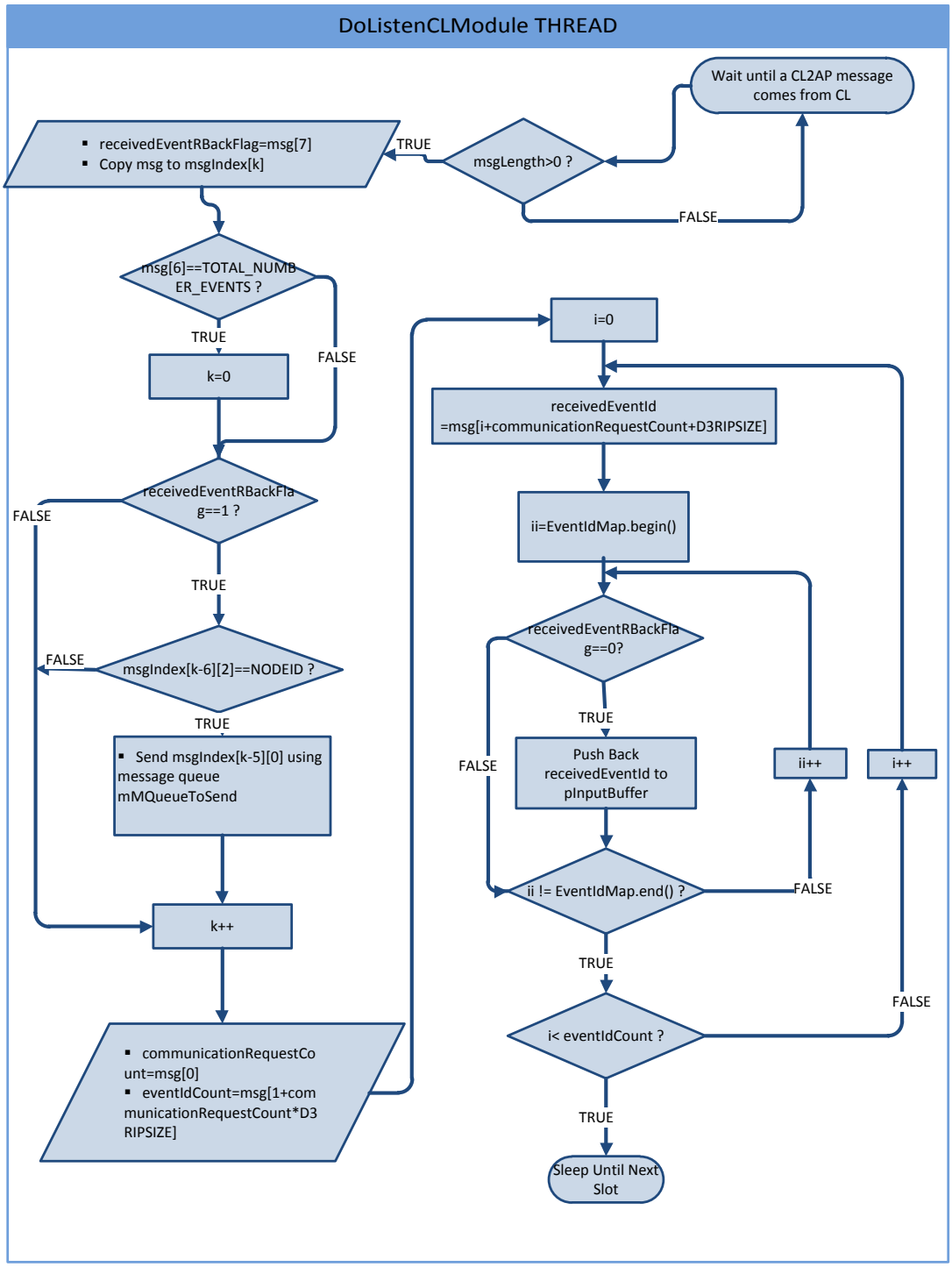

Figure 33: DoListenCLModule Thread in AP

Figure 33 shows DoListenCLModule thread in AP. It waits until action `CL2AP` comes from CL. When it reads the message, it takes the receivedEventRBackFlag from msg and copies the received message into the msgIndex[k]. If receivedEventRBackFlag does not equal to 1, it operates normally. If receivedEventRBackFlag equals to 1, it controls the message which is the last entry in the history of DP. If that message nodeID's equals that nodeId, it sends the message which is sent before and in rollback situation, it does not increment the index of the EventIdMap. So, it sends message using buffer msgIndex until rollback is finished.
Detailed information about DP implementation and their flowchart are mentioned in the following section.

Figure 34 shows thelistenCoordinationLayer thread in DP implementation. In listenCoordinationLayer thread, DP waits until action `UPDVCL` comes from CL. After getting message, DP continues with the `UPDVCL` function. In this function, it creates free space to insert vCL parameters *(nodeid,ch, eT and dT)* into the vCL History Queue. After inserting vCL parameters into the queue, size is controlled to prevent overflow. If it exceeds the *cyc*, queue pops up a node from top of the queue and thread sleeps until next slot.

Figure 35 shows the listenInterfaceLayer thread in DP implementation. In listenInterfaceLayer thread, DP waits until `UPDVIL` or `SM2ILDP` comes from IL. After getting `UPDVIL` or `SM2ILDP`, DP continues according to first bit of message (*msg[0]*). If the first bit of the message is 1, DP continues with `processSM2IL`. In `processSM2IL`, message is read from character device file charDev_DP2IL. *atRes*, *rbSt* and *vCL* parameters in the received message are assigned to the variables in the DP. If atRes equals to 1, DP continues with `ATEST`. In `ATEST()` function, firstly *cnt* is incremented by 1. Then DP calls `fAT()` to make acceptance test. After acceptance test, if *atRes* is true, *rbSt* is determined and result of acceptance test is sent to IL using action `SENDRES`. In `SENDRES`, *atRes* and *rbSt* are assigned to the msgDP2IL and it is sent to IL using character device file charDev_DP2IL. If *atRes* equals to 0, DP continues with `RBACK`.

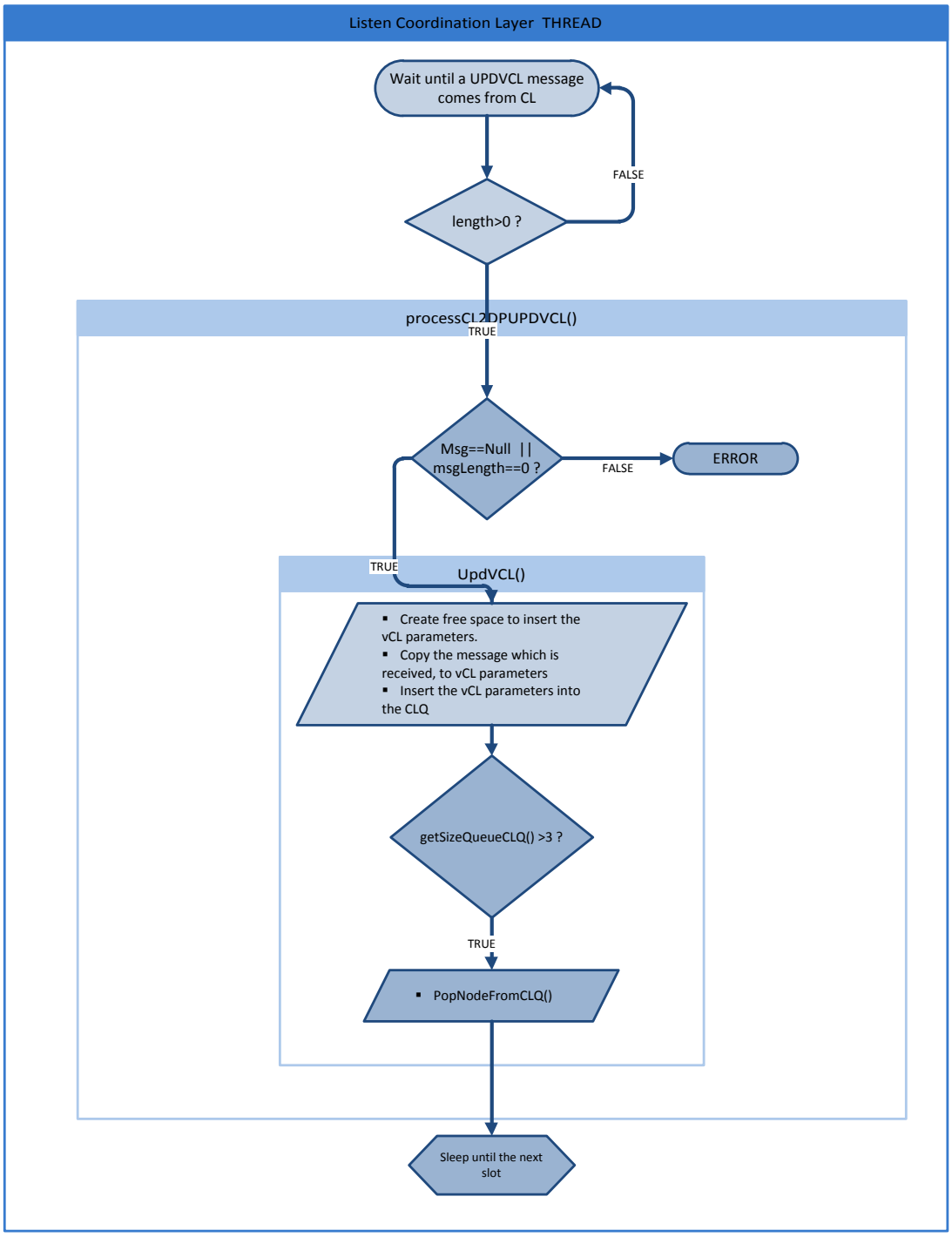

Figure 34: listenCoordinationLayer Thread in DP Implementation

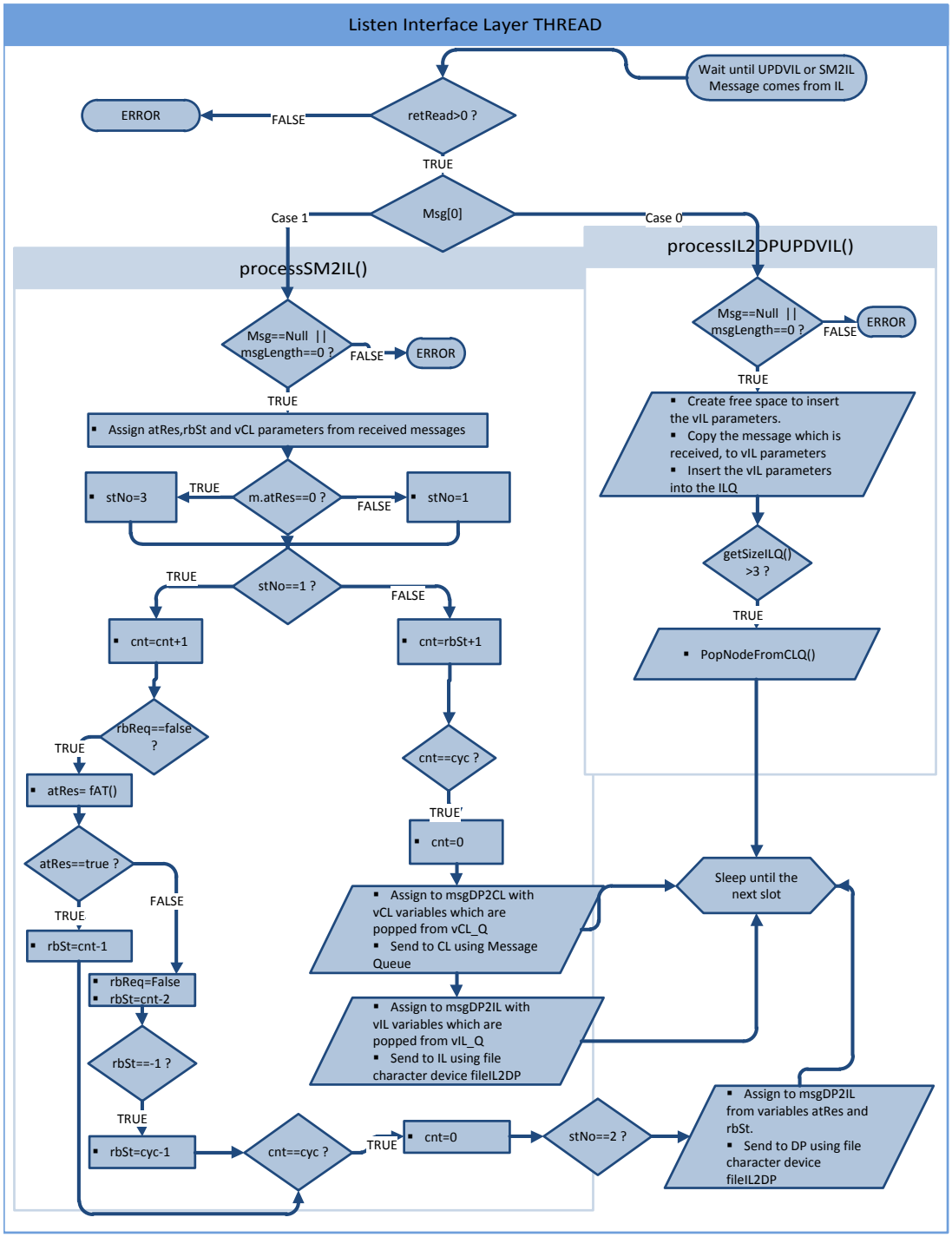

Figure 35: listenInterfaceLayer in DP Implementation

In `RBACK()` function, cnt is determined firstly. Then, vCL parameters which are popped from vCL History Queue, are assigned to msgDP2CL and it is sent to CL using message queue MQUEUE_DP2CLRBACK. After that vIL parameters which are popped from vIL History Queue, are assigned to msgDP2IL and it is sent to IL using character file device charDev_DP2IL. Then if the first bit of the message is 0, DP continues with processIL2DPUPDVIL. In processIL2DPUPDVIL, DP creates free space to insert the vIL parameters. Then it inserts the vIL parameters which are coming from IL using thecharacter

device file charDev_IL2DP, into the vIL History Queue. After inserting vIL parameters into the queue, size is controlled to prevent overflow. If it exceeds the *cyc*, queue pops up a node from top of the queue and thread sleeps until next slot.

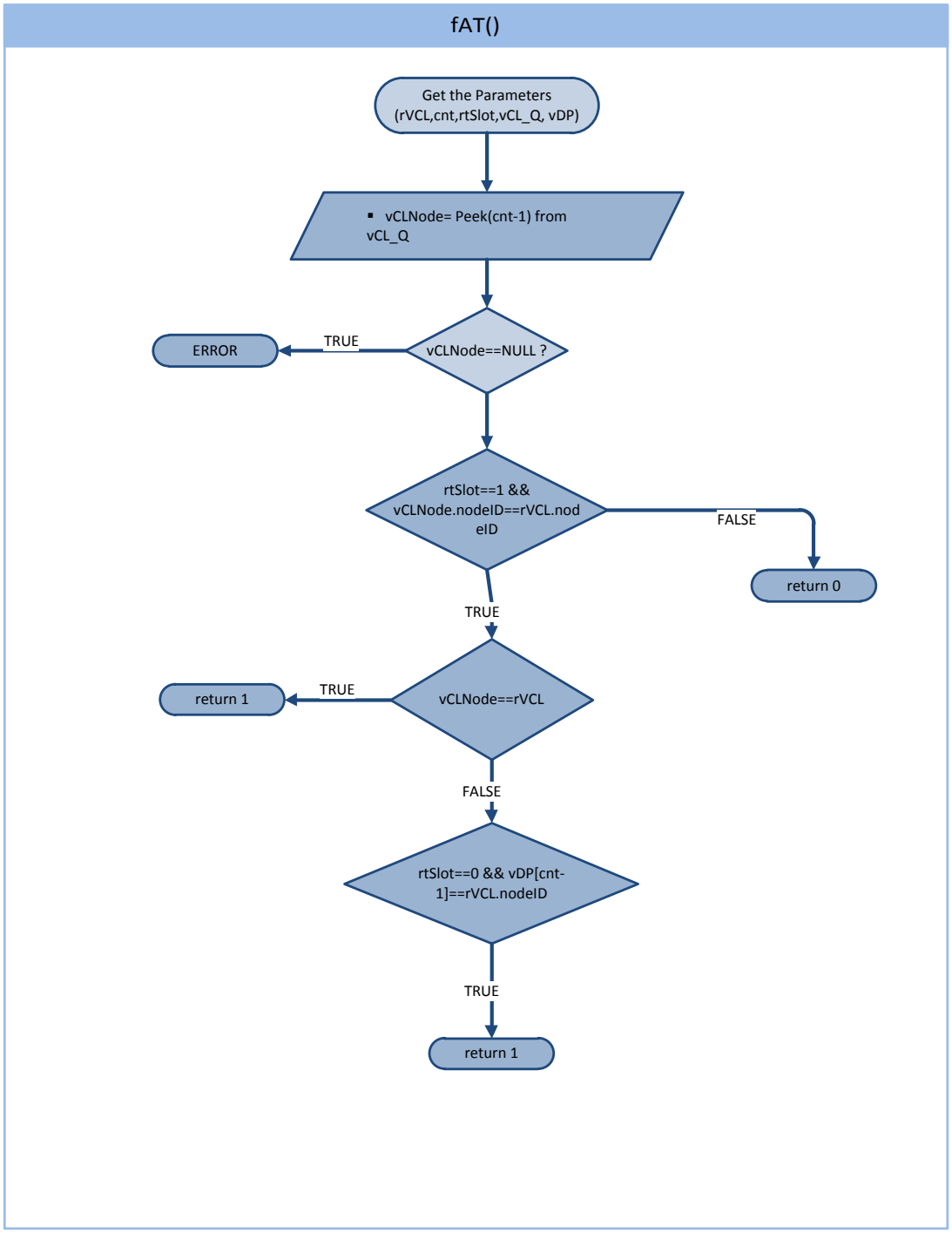

Figure 36: f_{AT} Implementation in DP

Figure 36 shows the f_{AT} implementation in DP. $f_{AT}()$ function is called by DP in `processSM2IL`. In f_{AT}, DP gets *rvCL, cnt, rtSlot* and *m.nodeid* parameters. Then it takes a node from the top of the vCL_Q. vCL parameters which are received from `SM2ILDP` and vCL parameters which are taken from vCL_Q are compared to each other. If they are equal to each other, f_{AT} returns 1. If they are not equal to each other, f_{AT} returns 0. Also for nRT comparison, if the received nodeid equals the node id which is stored in vDP, f_{AT} returns 1. If the received nodeid is not equal to the node id which is stored in vDP, f_{AT} returns 0.

4.2 Data Structures

Figure 37 illustrates the UML Class Diagram for the dependability plane. DPProtocol class is an abstract class that controls the operations of DP and interfaces to other layers (CL and IL). URT class is derived from DPProtocol that consist of functions of process in DPProtocol. vDPURT class stores the vCL and vIL parameters and makes operations on them.

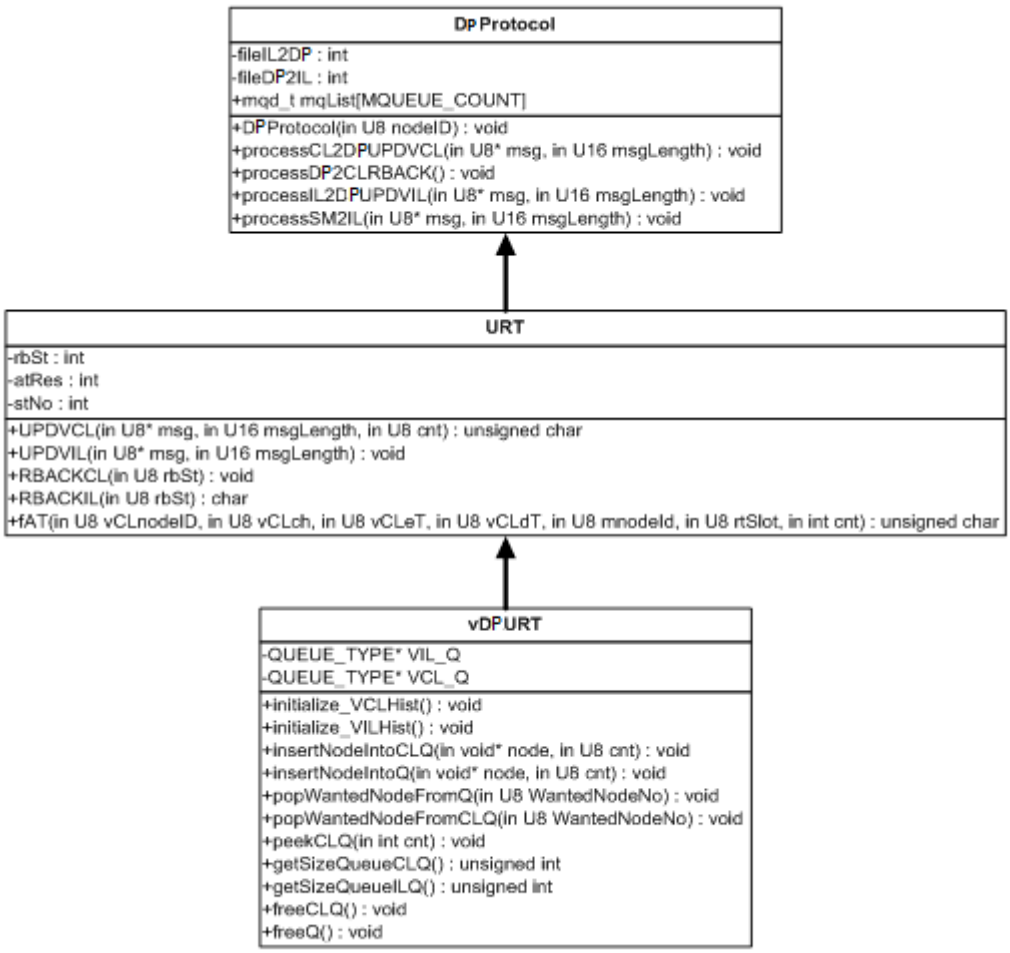

Figure 37: Dependability Plane UML Class Diagram

In vDPURT class, vCL_Q_TYPE, vIL_Q_TYPE and QUEUE_TYPE data structures are used. Table 3 shows the vCL_Q_TYPE. There are 4 variables defined in vCL_Q_TYPE data structure: *nodeID, ch, eT and dT*. In vCL parameters, *nodeId* shows the id of the node which sends the RT message in that time slot. *ch* shows the channel of the RT communication. *eT* and *dT* show the eligibility time and deadline time of the RT message.

Table 3: vCL_Q_TYPE

nodeID: U8 (unsigned char) ch:U8 (unsigned char) eT:U8 (unsigned char) dT:U8 (unsigned char)

Table 4 shows the vIL_Q_TYPE. There are 3 variables defined in vIL_Q_TYPE data structure: *cnt, cyc and slots*. In vIL parameters, *cnt* shows the count which is incremented by 1 with modulo of *cyc* at the each nRT time *slot.cyc* indicates the cycle of the nRT. In other words, it shows the total node number. Slots show the nRT time slot owner's node Id.

Table 4: vIL_Q_TYPE

cnt: U8 (unsigned char) cyc:U8 (unsigned char) slots: U8 (unsigned char)

Table 5 shows the QUEUE_TYPE. There are 3 variables defined in QUEUE_TYPE data structure: *currentSize, availableSize and nodes*.

Table 5: QUEUE_TYPE

currentSize: U32 (unsigned int) availableSize:U32 (unsigned int) nodes: void **

4.3 Actions and Operation

In the DP operation, there are many actions that are shared with other layers. Flowcharts of DP and the functions in the DP are mentioned in the previous section. In this section, actions and operations of the DP are described.

4.3.1 Actions

4.3.1.1 Actions of DP

ListenCoordinationLayer Thread: It listens to messages coming from CL. When it takes action UPDVCL from CL, it calls processCL2DPUPDVCL function.

processCL2DPUPDVCL Function: It calls UPDVCL(*msg,msgLength*) function.

UPDVCL() Function: It takes the vCL parameters from a received message and calls insertNodeIntoCLQ () function.

ListenInterfaceLayer Thread: It listens to the message coming from IL. When it takes the action UPDVIL from IL, it calls processIL2DPUPDVIL function. When it takes the action SM2ILDP from IL, it calls processSM2IL function.

processIL2DPUPDVIL Function: It calls UPDVIL(*msg,msgLength*) function.

UPDVIL() Function: It takes the vIL parameters from a received message and calls insertNodeIntoQ () function.

processSM2IL Function: It takes the atRes, rbSt (*checkPt*), vCL parameters and assigns the variables in DP. If atRes is true, it calls the fAT function to make ATEST and it sends the results to IL using the SENDRES function. .If atResis false, it calls RBACKCL and RBACKIL functions.

fAT() Function: It compares popped up vCL parameters from vCLHistoryQueue and received vCL parameters from IL. If they are equal, it returns true. If they are not equal, it returns false.

RBACKCL() Function: If there is a fault, it calls popWantedNodeFromCLQ() function. After popWantedNodeFromCLQ() function returns vCL parameters, it sends them to the CL.

RBACKIL() Function: If there is a fault, it calls popWantedNodeFromQ() function. After popWantedNodeFromQ() function returns vIL parameters, it sends them to the IL.

Initialize_VCLHist() Function: It starts the vCLHistoryQueue.

Initialize_VILHist() Function: It starts the vILHistoryQueue.

insertNodeIntoCLQ() Function: This function is called by UPDVCL() function. It inserts the vCL parameters into the vCLHistoryQueue.

insertNodeIntoQ() Function: This function is called by UPDVIL() function. It inserts the vIL parameters into the vILHistoryQueue.

popWantedNodeFromQ() Function: This function is called by RBACKIL() function. It pops up the vIL parameters at the top of the vILHistoryQueue.

popWantedNodeFromCLQ() Function: This function is called by RBACKCL() function. It pops up the vCL parameters at the top of the vCLHistoryQueue.

peekCLQ() Function: This function is called by fAT() function. It shows the vCL parameters at the top of the vCLHistoryQueue.

getSizeQueueCLQ() Function: This function is called by UPDVCL() function. It returns the size of the vCLHistoryQueue.

getSizeQueueILQ() Function: This function is called by UPDVIL() function. It returns the size of the vILHistoryQueue.

freeCLQ() Function: It releases the vCLHistoryQueue.
freeILQ() Function: It releases the vILHistoryQueue.

4.3.1.2 Actions of CL related with DP

ListenDependableLayer Thread: It listens to messages coming from DP. When it takes RBACK from CL, it calls processDP2CLRBACK function.
processDP2CLRBACK Function:It calls RBACK(*msg,msgLength*) function.
RBACK() Function: It takes the vCL parameters from a receivedmessage and inserts them into the priority queue PQ_dT in the CL.
UPDVCL() Function: It is called by CL in the processCLUPDATE function and it sends vCL parameters of the time slot to the DP using message queue.

4.3.1.3 Actions of IL related with DP

UPDVIL() Function: It sends vIL parameters of the time slot to the DP using character device file charDev_IL2DP.
RBACK() Function: It takes the vIL parameters from a receivedmessage and assigns them to the variables in the IL.

4.3.2 Operations

Figure 38 illustrates the operation and timing of D^{3}RIP, if there is no fault before the considered slot. In the figure, AP2CL is sent by RT application. CL takes this message, after that it sends the RT message to IL. Before sending RT message to IL, CL sends the vCL parameters (*nodeid, ch, eT, dT*) of that time slot to DP using UPDVCL. When IL takes the RT message from CL, it sends the vIL parameters (*cnt, cyc, slots*) of that time slot to DP using UPDVIL. Then IL sends the RT message to SM using IL2SM. Destination node and that node get this RT message using SM2IL. IL sends this RT message to CL immediately and sends SM2ILDP to DP to provide the parameters (*atRes, rbSt, rvCL.nodeid, rvCL.ch, rvCL.eT, rvCL.dT*) which come from the sender node. If the received *atRes* is true, DP performs itsacceptance test using the internal action ATEST. In ATEST, vCL which comes with UPDVCL and vCL which comes with SM2ILDP are compared. If they are equal to each other, atRes returns true. If they are not equal to each other, *atRes* returns false. After that, DP sends SENDRES to IL to provide *atRes* and *rbSt*. IL sends these parameters to other nodes when an RT message comes from CL in the next time slot. While DP is making ATEST, CL sends the received RT message to AP using CL2AP.

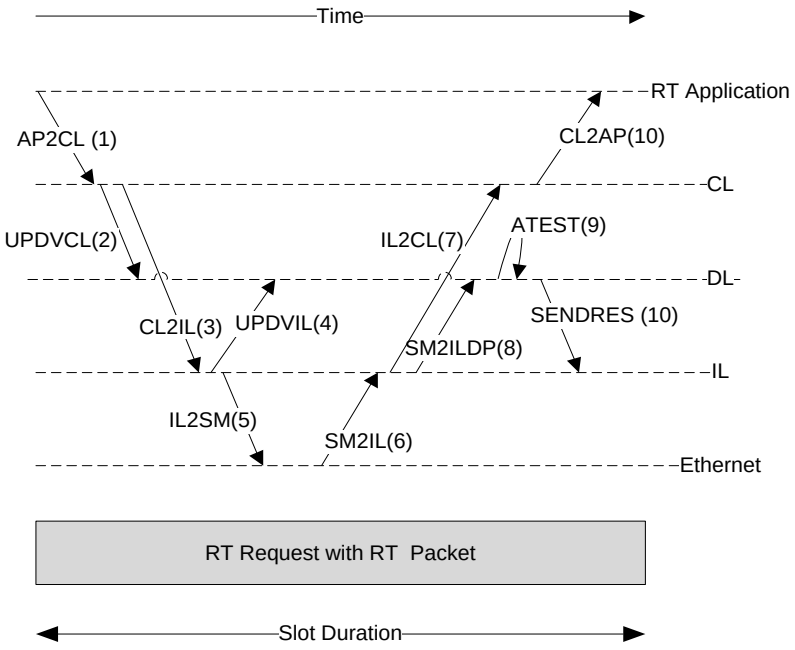

Figure 38: The Timing of the D^{3}RIP without Any Fault

Figure 39 shows the operation and timing of D^{3}RIP, if a fault is communicated in the considered slot. The difference between the conditions with fault and without fault is that DP behaves differently when it takes `SM2ILDP`. If there is a fault before that time slot, DP sends `RBACK` to CL and IL to bring back the recovery point. In `RBACK`, it pops out the recovery points in vCL and vIL History queues. When CL takes the `RBACK`, it sends a flag with RT message using `CL2AP`. Thus, AP knows that there is a `RBACK` situation and sends RT message which is earliest in the history of DP in the next time slot. Then nodes send the RT message each other according tohistory of DP. The length of history of DP is calculated using formula below:

$$\Delta F=\Delta I+\Delta S+\Delta C+\Delta N$$

ΔF: Longest fault detection time.
ΔI: Maximum number of time slots between sending two packets by the same node is calculated by using maximum number of time slots which is assigned to a node between two time slots.
ΔS: Consecutive time slots which are not assigned to any node in the system use all the time.
ΔC: Possible maximum consecutive collision number on the shared medium.
ΔN: The maximum number of the consecutive message which is sent by one node.

It is assumed that these parameters are known from the operation of the respective application. Also, there are assumptions and necessary conditions for DP to work properly. [43]

- There is not any fault that occurs successively in the system. There can be only one fault after the previous fault is resolved.

- In the system, each node has a slot to transmit its message to the other nodes. So, there has to be a bounded interval of at most ΔI time slots between two successive transmission slots.
- There is not any fault in the first time slot.
- There has to be at least 3 nodes that are connected to the system.

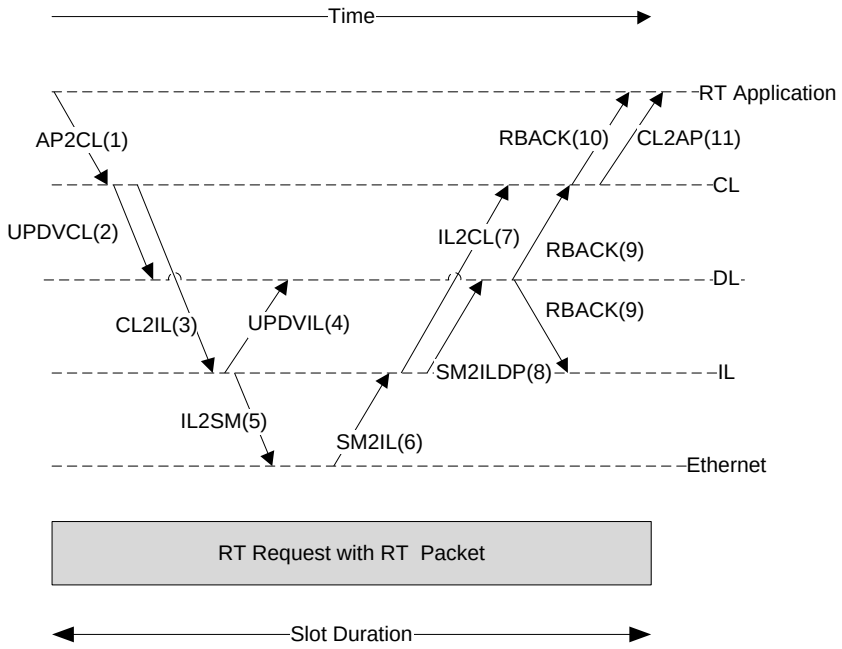

Figure 39: The Timing of the D[3]RIP with Fault

CHAPTER 5

EVALUATION OF THE DEPENDABILITY PLANE

5.1 Example Description

In this thesis, the manufacturing cell in Figure 41 is considered as an application example [44]. In Figure 40, there are 4 controllers working with each other synchronously, these controllers are: PLC-S, PLC-R, PLC-C, and PLC-PD. PLC-S coordinates the other controllers. PLC-R, PLC-C and PLC-PD control the 3 parts of the system namely robot (R) that moves a robot arm, conveyor (C) that carries parts and apainting device (PD) that paints parts using a spray gun.

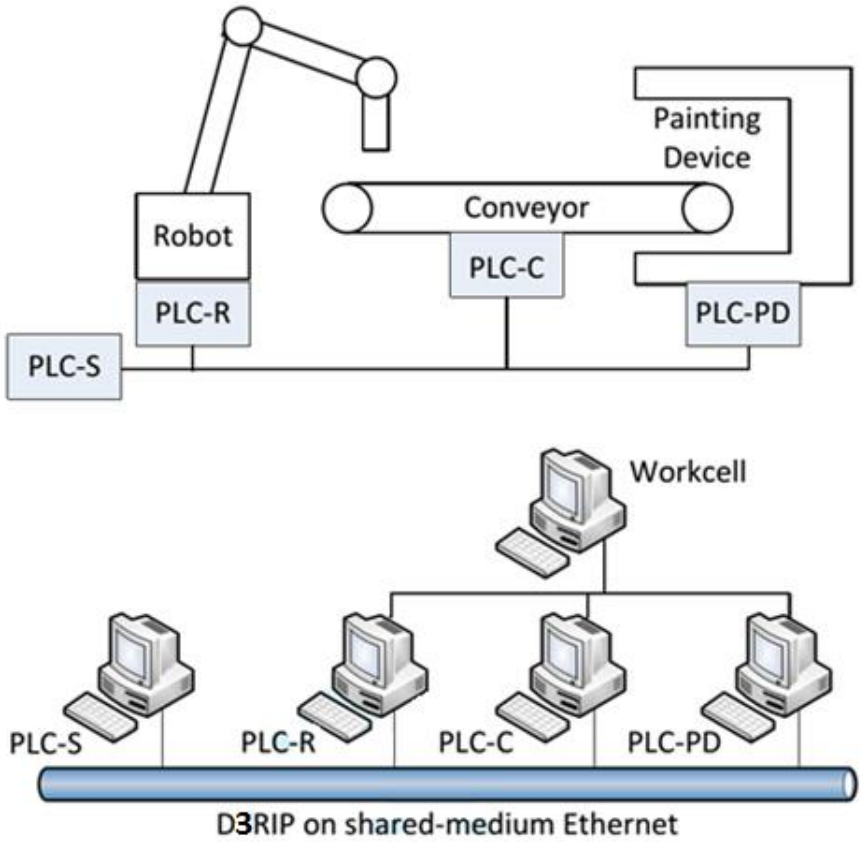

Figure 40: A Manufacturing System [44]

This workcell has an event-based operation. All controllers have a state machine and the states show the state of the controller, arrows between states indicate transitions, whereby all transitions are labeled with event names. Figure 41 shows the state machines of the workcell. Synchronized actions among the different controllers are represented by transitions with equal names. For example, the transition mvC occurs synchronously in PLC-S and PLC-R.

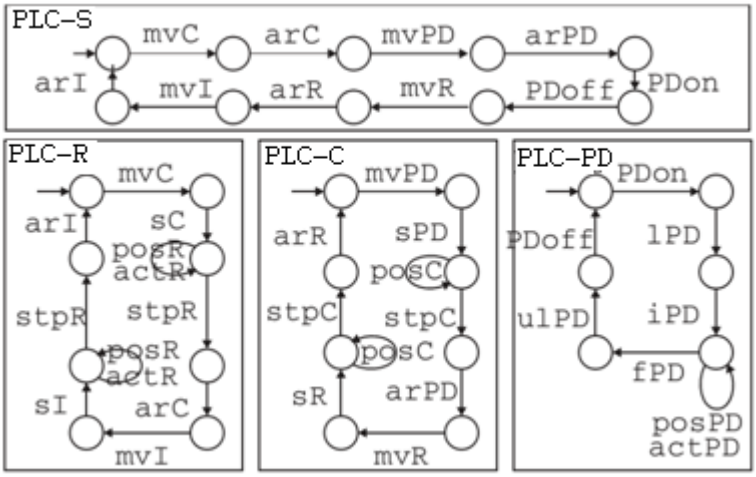

Figure 41: State Machines of Workcell [12]

In the state machines of the workcell, PLC-S initiates the system start by triggering mvC that is shared withPLC-R. PLC-R sends a signal to the robot arm to move a part to the conveyor (Event sC is occurred). Closed-loop control posR which gives the position of robot arm and actR which indicates the actuator signal of robot arm, are sent to move the robot arm to the conveyor. When robot arm reaches the conveyor, stpR signal is sent to stop the robot arm and arC signal notifies PLC-S about the arrival of the robot arm at the conveyor. PLC-S sends mvPD command to PLC-C to move the part to the painting device. PLC-C sends signal to the conveyor to move part to the painting device (Event sPD occurs). Position control signal posC is sent to the conveyor to inform the position of conveyor. When conveyor reaches the painting device, stpC signal is sent to stop the conveyor and arPD signal notifies PLC-S controller about the arrival of the robot arm at the conveyor. PLC-S sends PDon command to the PLC-PD to start operation of the painting device. lPD signal locks the painting device for painting operation, iPD initiates the painting process. Then, posPD which gives the position of painting device and actPD which indicates the actuator signal of painting device are sent to operate the painting device. After that, fPD signal which shows that the painting device finishes the painting operation is sent by painting device. After finishing painting operation, unlock painting device ulPD signal is sent and painting device is turned off by PLC-S with PDoff signal. After that, conveyor moves back to the robot arm with signal mvR, sR, arR (These signals are same with mvPD, sPD and arPD) and robot arm gets part from conveyor and put it on its old place with signal mvI, sI, arI (These signals are same with mvC, sC and arC).

Figure 42 illustrates the timing diagram for the PLC communication of the example workcell. PLC-S sends event message to the related controller with queries (?). Then the other PLC controller responds that event message with a notification (!) to PLC-S whenever it is ready to execute the event. When PLC-S gets the event message with a notification (!), it sends a message with a single command (√) to execute the event.

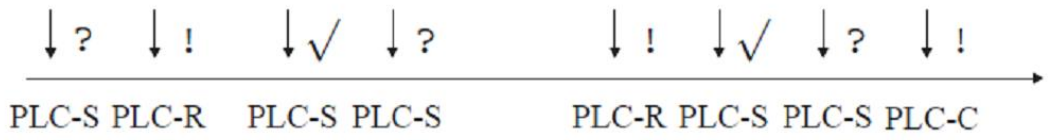

Figure 42: The Timing Diagram for the PLC Communication of the Example Workcell [44]

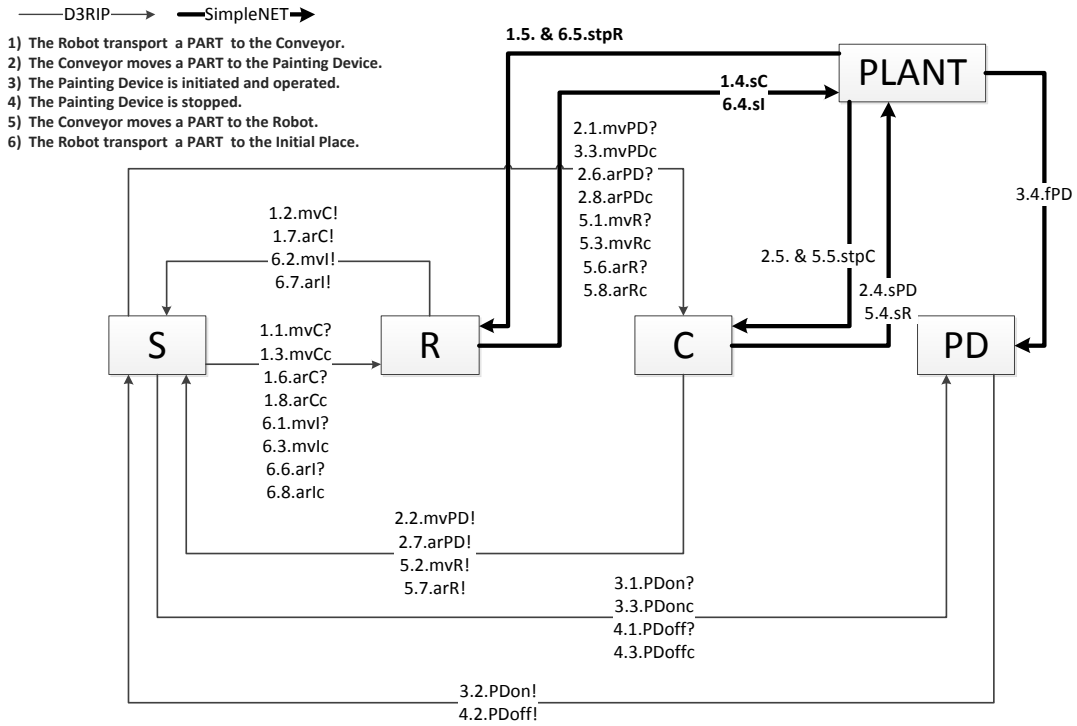

Figure 43: The Connection of the Controllers and Plant

In the system simulation and implementation, each computer behaves like controller in the system. There are 4 controllers in the system: PLC-S, PLC-R, PLC-C and PLC-PD. Also, these controllers are connected to another computer which runs as a plant to simulate the actual system operation. So, there are 5 computers in the system which are connected to each other for all system simulation and implementation. Figure 43 shows the connection of the controllers and plant computer. There are 4 controllers and plant computer in the system. Plant and other controllers are connected with Ethernet 802.3. This is called Simplenet

communication. Simplenet communication carries on the bold black connections. Controllers exchange signal data (sensor/actuator) with the plant. This is done using the simplenet protocol which is specific to libfaudes and fits to our application.
Also our RT Ethernet protocol D^{3}RIP is used for industrial communication between controllers. D3RIP communication carries on the black connections. In D^{3}RIP communication, signal data (sensor/actuator) between controllers are carried RT. The numbers of the signals show the order of event occurrence. A part is taken, carried, painted and moved back to the old places with events in Figure 41.

On the plant computer, simfaudes which is a simulator for the example control application, runs and it controls the system operation. Simfaudes uses XML files to configure the system operation. There are different XML files which are used as input files for D^{3}RIP, simplenet connection definition and simulator description. Figure 45 below shows the D^{3}RIP XML configuration file for controller C**.** Event name, event type, event id, channel transmit value, parameter record, destination node id, destination channel value, eligibility and deadline time of events are defined and configured in D^{3}RIP XML file according to the connection of the controllers.

Communication Example between controllers (Figure 43-44):
Controller S sends event "mvPD?" to Controller C to move the part to the painting device (PD). Controller C sends back event "mvPD!" to ask the confirmation of the action and then Controller S sends event "mvPDc" to confirm that action. These actions are defined in D^{3}RIP XML file for each controller. In Figure 44 below, event "mvPD?" is defined. It is "output" event, its eventid is "14" which shows the order of occurance in the system. Its channel value is "1" and parameter record is "11". The destination node of event is "1". In other words, it is sent to Controller S by Controller C (Controller S's node id equals 1). The destination channel value is "1". The eligibility time of event is "8" which shows that how much time that event is eligible for the system. Also in that definition, deadline time of event is defined. For event "mvPD!", the deadline time is "10".

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE D3RIP SYSTEM "controllerC_d3rip.dtd">
<D3RipUrtDevice  name="controllerC_d3rip">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>

<Event name="mvPD?" iotype="input">
<EventId value="13"/>
</Event>

<Event name="mvPD!" iotype="output">
<EventId value="14"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="8" />
<DeadlineTime value="10"/>
</ParameterRecord>
```

```
</Event>

<Event name="mvPDc" iotype="input">
<EventId value="15"/>
</Event>

<Event name="arPD?" iotype="input">
<EventId value="16"/>
</Event>

<Event name="arPD!" iotype="output">
<EventId value="17"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="8" />
<DeadlineTime value="10"/>
</ParameterRecord>
</Event>

<Event name="arPDc" iotype="input">
<EventId value="18"/>
</Event>

<Event name="mvR?" iotype="input">
<EventId value="19"/>
</Event>


<Event name="mvR!" iotype="output">
<EventId value="20"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="8" />
<DeadlineTime value="10"/>
</ParameterRecord>
</Event>


<Event name="mvRc" iotype="input">
<EventId value="21"/>
</Event>


<Event name="arR?" iotype="input">
<EventId value="22"/>
</Event>


<Event name="arR!" iotype="output">
<EventId value="23"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
```

```
<EligibilityTime value="8" />
<DeadlineTime value="10"/>
</ParameterRecord>
</Event>


<Event name="arRc" iotype="input">
<EventId value="24"/>
</Event>
</EventConfiguration>
</D3RipUrtDevice>
```

Figure 44: D[3]RIP XML Configuration File for Controller C

Simplenet XML files show the connection between controller and plant computer and the physical connection of simplenet is over ethernet. Figure 45 shows the Simplenet XML configuration file for controller C**.** Network topology is defined with network name and nodes which are in the network and events are configured with event names, event types according to the connection of the controllers and plant in Figure 43.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE SimplenetDevice SYSTEM "controllerC.dtd">
<SimplenetDevice name="controllerC">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>

<!-- Network topology -->
<Network name="paintingNet">
<Node name="plant"/>
<Node name="controllerR"/>
<Node name="controllerC"/>
<Node name="controllerPD"/>
</Network>

<EventConfiguration>
<Event name="sPD" iotype="output"/>
<Event name="stpC" iotype="input"/>
<Event name="sR" iotype="output"/>
</EventConfiguration>

</SimplenetDevice>
```

Figure 45: SimpleNet XML Configuration File for Controller C

Simulator XML files define the protocol parameters and properties of the controller. Simulator files help the generation of simulator in the simfaudes. Figure 46 shows the simulator XML configuration file for controller C**.** Alphabet tag shows the commands which

are related that node. States tag shows the state status and definition. TransRel tag shows the state transition diagrams members and description how the states are connected to each other. InitStates tag indicates the initial state of that state diagram. MarkedStates tag shows the marked state in the state diagram. SimEventAttributes tag shows the event properties and priorities in the system.

```
<Executor>
<Generators>
<Generator>
controllerC
<Alphabet>
mvPD? mvPD! mvPDc mvPD arPD? arPD! arPDc arPD sPD stpPD
mvR? mvR! mvRc mvR sR arR? arR! arRc arR
</Alphabet>
<States>
<Consecutive> 1 28 </Consecutive>
</States>
<TransRel>
1 mvPD? 2
2 mvPD! 3
3 mvPDc 4
4 arPD? 5
4 mvPD 6
5 mvPD 7
6 arPD? 7
6 sPD 8
7 sPD 9
8 arPD? 9
8 stpPD 10
9 stpPD 11
10 arPD? 11
11 arPD! 12
12 arPDc 13
13 arPD 14
13 mvR? 15
14 mvR? 16
15 arPD 16
16 mvR! 17
17 mvRc 18
18 arR? 19
18 mvR 20
19 mvR 21
20 arR? 21
20 sR 22
21 sR 23
22 arR? 23
22 stpPD 24
23 stpPD 25
24 arR? 25
25 arR! 26
26 arRc 27
27 arR 1
27 mvPD? 28
28 arR 2
</TransRel>
```

```
<InitStates>          1      </InitStates>
<MarkedStates>   1     </MarkedStates>
</Generator>
</Generators>
<SimEventAttributes>
"stpR"     <Priority>-1 </Priority>
"fPD"        <Priority>-1 </Priority>
"mvC"      <Priority> 1 </Priority>
"sC"         <Priority>      1 </Priority>
"arC"  <Priority>1 </Priority>
"mvl"       <Priority>       1 </Priority>
"sl"         <Priority>       1 </Priority>
"arl"        <Priority>       1 </Priority>
"mvPD"    <Priority>       1 </Priority>
"sPD"      <Priority>       1 </Priority>
"arPD"     <Priority>       1 </Priority>
"mvR"      <Priority>       1 </Priority>
"sR"         <Priority>       1 </Priority>
"arR"        <Priority>       1 </Priority>
"PDon"     <Priority>       1 </Priority>
"PDoff"      <Priority>     1 </Priority>
</SimEventAttributes>
</Executor>
```

Figure 46: Simulator XML Configuration File for Controller C

D^3RIP, Simplenet and simulator XML configuration files of all controllers and plant can be found in the appendix part of this thesis.

5.2 Performance Parameters

The duration of the time slot is calculated using the timing of the D^3RIP operation. In Figure 39 the timing of the D^3RIP without any fault, there are 11 sequential actions. In Table 6, the total number of sequential actions time duration shows the duration of the time slot without any fault. Also, 900 sample results (30 loop tests are made) are taken to show the duration of the time slot in Table 6.

In Figure 40 the timing of the D^3RIP with fault, there are also 11 sequential actions. In Table 7, the total number of sequential actions time duration shows the duration of the time slot with fault. Also, in Table 7, 900 sample results (30 loop tests are made) are taken to show the the duration of the time slot.

Table 6: Synchronization Accuracy and Sequential Actions in D[3]RIP Operation Without Any Fault

IEEE 1588 Synchronization			
	Mean (µs)	Max.(µs)	+/- % CI (99%)
Path Delay [44]	1.6	1.7	%0
Accuracy [44]	1.4	4.2	%3.71
Sequential Actions			
	Mean (µs)	Max. (µs)	+/- % CI (99%)
CL Thread Wake-up	0.7	23.5	%0.21
AP2CL	3,2	6.1	%3.23
UPDVCL	17.9	36.4	%2.47
CL2IL	1.3	6.5	%0.45
IL2SM	15.4	27.9	%0.38
UPDVIL	20.4	41.5	%1.96
SM2IL	15.4	27.9	%0.38
IL2CL	12.6	64.2	%1.22
SM2ILDP	15.6	34.6	%2.37
ATEST	1.4	4.2	%2.63
SENDRES	6.3	16.6	%2.85
CL2AP	2.3	12.8	%4.07
Completion Time	189.1	502.2	%1.58

Table 7: Sequential Actions in D[3]RIP Operation With Fault

Sequential Actions			
	Mean (µs)	Max. (µs)	+/- % CI (99%)
CL Thread Wake-up	0.7	23.5	%0.21
AP2CL	3.2	6.1	%3.23
UPDVCL	17.9	36.4	%2.47
CL2IL	1.3	6.5	%0.45
IL2SM	15,4	27.9	%0.38
UPDVIL	20.4	41.5	%1.96
SM2IL	15.4	27.9	%0.38
IL2CL	12.6	64.2	%1.22
SM2ILDP	15.6	34.6	%2.37
RBACKCL	56.5	90.5	%1.19
RBACKIL	58.2	98.2	%1.13
CL2AP	2.3	12.8	%4.07
Completion Time	339.5	770.3	%0.93

Table 8: Synchronization Accuracy and Sequential Actions in D^{2}RIP Operation Before Adding Dependability Plane [44]

IEEE 1588 Synchronization			
	Mean (µs)	Max. (µs)	+/- % CI (99%)
Path Delay [44]	1.6	1.7	% 0
Accuracy [44]	1.4	4.2	%3.71
Sequential Actions			
	Mean (µs)	Max. (µs)	+/- % CI (99%)
CL Thread Wake-up	0.7	23.5	%0.21
AP2CL	3,2	6.1	%3.23
CL2IL	1.3	6.5	%0.45
IL2SM	15.4	27.9	%0.38
SM2IL	15.4	27.9	%0.38
IL2CL	9.2	46.1	%1.22
CL2AP	2.3	12.8	%4.07
Completion Time	94.1	231.1	%1.82

After implementation of dependability plane, sample measurements were made for the system dependability. Table 6 and Table 7 show these measurements. In these measurements, duration of time slot is chosen as 1ms to cover all operations for dependability. Table 8 shows the timing of sequential actions in D^{2}RIP operation before adding dependability plane. In Table 6, 7, 8, the completion time includes the duration of sequential action, the spending time in hub and the transmission delay.

In the example (Figure 41), maximum number of time slots (denoted as ΔI) between sending two packets by the same node is calculated by using maximum number of time slots which is assigned to a node between two time slots. This parameter is $\Delta I=6$ time slots for the example system. Consecutive time slots which are not assigned to any node in the system use all the time slots. So $\Delta S=0$. Possible maximum consecutive collision number on the shared medium is assumed as $\Delta C=4$. The maximum number of the consecutive message which is sent by one node is $\Delta N=2$. For these assumptions the longest fault detection time ΔF is calculated below.

$$\Delta F=\Delta I+\Delta S+\Delta C+\Delta N=12$$

So Dependablity Plane stores at least $\Delta F=12$ state variables in its memory. Also if there is a fault (when acceptance test is made) at time slot x, rollback time slot is $x-\Delta I$ [45]. In this example, rollback time slot equals to the 6. When there is a fault, node rolls back 6 messages before and determines the state variables for this time slot. Application layer sends the related message.

5.3 Experiments and Results

According to performance parameters (Section 5.2), three experiments are made to show the Dependability Plane functionality and performance.

In the first experiment, there is a problem at the vCL data structure in the PLC-S component. In this example, there is a vCL fault (eT parameters fault) at the 9th time slot (belongs to PLC-S if the system is started up correctly). DP of another node detects the fault, assigns atRes to 0 at that time slot and it sends the result to all nodes in the 10th time slot. Then, DP of all nodes sends RollBack to CL and IL. Application layer sends 5th event message instead of 11th event message in the PLC-S component. S component rolls back the system 6 time slots before. Figure 47 shows the experiment result of the first experiment.

Figure 47: The Experiment Result of the First Experiment

In the second experiment, there is a problem at the message transmission when message is taken from the shared medium. In this example, there is a fault at the 16th time slot (belongs to PLC-S). PLC-S component sends 16th event message to all components in the system. That message cannot reach the destination node. DP of another node detects the fault, assigns atRes to 0 at that time slot and it sends the result to all nodes in the 17th time slot. Then, DP of all nodes sends RollBack to CL and IL. Application layer sends 12th event message instead of 18th event message in the PLC-S component. PLC-S component rolls back the system 6 time slots before. Figure 48 shows the experiment result of the second experiment.

Figure 48: The Experiment Result of the Second Experiment

In the third experiment, there is a problem at the vCL data structure in the PLC-R (Robot Arm) component. In this example, there is a vCL fault (eT parameters fault) at the 25th time slot when R component gets the message from PLC-S component. DP realizes the fault, assigns atRes to 0 at that time slot and it sends the result to all nodes in the 26th time slot. Then, DP of all nodes sends RollBack to CL and IL. Application layer sends 21th event message instead of 27th event message in the PLC-S component. PLC-S component rolls back the system 6 time slots before. Figure 49 shows the experiment result of the third experiment.

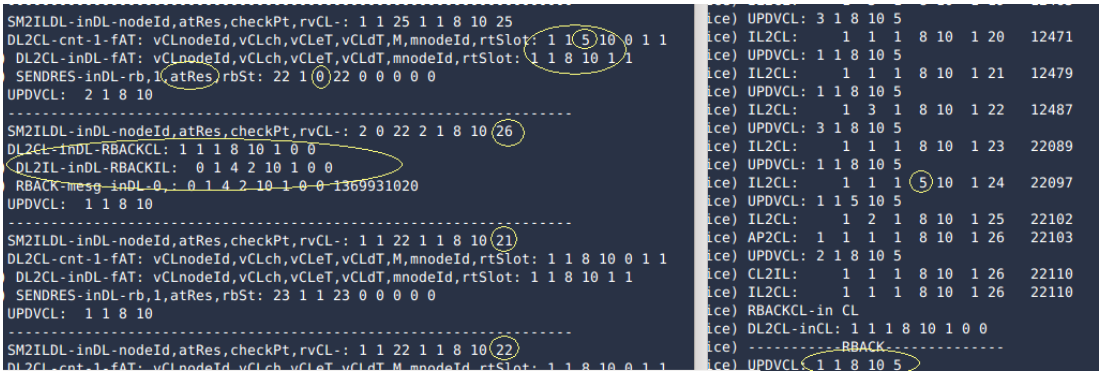

Figure 49: The Experiment Result of the Third Experiment

Also, another experiment was wanted to be applied on the system. It was about the packet loss and packet collision. The reason why it was not applied is that IL-CL communication is implemented with blocking read in the previous and current implementation. Thus, this method give us guarentee not to miss any packet while transmission between IL and CL. But

when the experiment is wanted to implement, IL-CL communication method must be changed to non-blocking read method. Also, this modification affects the flow of the program operation of CL and IL. When communication method, the flow of the program operation of CL, reading data from charecter device file in main CL thread, writing data to charecter device file in IL section are changed and extra communication between CL-DP is added, this experiment can be implemented. This change is intended for future work.

CHAPTER 6

CONCLUSION & FUTURE WORK

6.1 Conclusion

With the development of technology, the industrial RT Ethernet networks have become an important subject in academia and industry. In the literature, there are different types of the protocols and solutions to implement industrial RT Ethernet networks. In our solution, the Distributed, Dependable and Dynamic Real Time Industrial Protocol (D^3RIP) is proposed for the industrial RT Ethernet network.

In this thesis, the implementation of the dependability plane for D^3RIP and its evaluation are studied. First, generic interface, coordination layers and dependability plane of D^3RIP are explained and formally represented by timed input output automata models. Then, based on an existing implementation of the predecessor protocol D^2RIP, the implementation of the dependability support for D^3RIP is discussed. The operation of D^3RIP is demonstrated by a manufacturing cell example with 4 controller nodes. In summary, the following main tasks are performed in this thesis:

- Dependability plane implementation,
- Integration of the dependability plane with coordination layer, interface layer and application layer (AP),
- Testing overall structure with all layers in the D^3RIP with 4 controllers real scenario,
- Measurement of the dependability plane performance,
- Different experiments over dependability plane to show the functionality and performance of the dependability plane.

In this thesis, the dependability plane is implemented over a specific D^2RIP structure that is implemented in [41], whereby implementation of DP is independent of the D^2RIP structure. While implementing the DP, the interface rfunctions between CL-DP and IL-DP are implemented in IL and CL sides. Also a buffer structure is added to the AP to recover messages when the RBACK event occurs. Other than these modifications, there are not any extra modifications on D^2RIP structure. After adding the dependability plane over specific D^2RIP structure [41], the slot duration is increased to 1ms because of new actions that have to be performed in each time slot.

Finally it can be seen that with dependability plane, possible D^3RIP failures are prevented and if there is a failure in the system, the system can recover and continue its operation.

6.2 Future Work

In D^3RIP implementation and test, system and protocol are observed.The following works make the framework much better when it runs in real operation:

- Implementation on Different Operating System: The framework might show improved performance on RT operating systems such as VxWorks, RTLinux, LynxOS.
- Extended Implementation to cover the packet loss and collision scenario: It can be implemented when the communication method between IL and CL, the flow of the program operation of CL are modified and extra communication between CL-DP is added.

REFERENCES

[1] (2013) Bosch CAN Specification Version 2.0. [last accessed on 31/07/2013]. [Online]. Available: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[2] (2013) Profibus. [last accessed on 31/07/2013]. [Online]. Available: http://www.profibus.com

[3] (2013) Lonworks Protocol Specifications. [last accessed on 31/07/2013]. [Online]. Available: http://www.echelon.com/technology/lonworks/

[4] D. Jansen, H. Buttner, "Real-time Ethernet the EtherCAT solution", *Computing & Control Engineering Journal*, 15, 16–21, 2004.

[5] E. Schemm, "Sercos to link with Ethernet for its third generation", *Computing & Control Engineering Journal*, 15, 30–33, 2004.

[6] "Real-time Ethernet: Profinet IO: Proposal for a publicly available specification for real-time Ethernet", *Doc. IEC 65C/359/NP*, 2004.

[7] F.B. Carreiro, J.A.G. Fonseca, P. Pedreiras, "Virtual Token-Passing Ethernet – VTPE" , *IFAC*, 2003.

[8] "Real-time Ethernet: EPL (Ethernet powerlink): Proposal for a publicly available specification for real-time Ethernet", *Doc. IEC 65C/356a/NP*, 2004.

[9] (2013) Schneider automation – modbus messaging on TCP/IP implementation guide. [last accessed on 31/07/2013]. [Online]. Available: http://www.modbus.org/

[10] "Real-time Ethernet: EPA (Ethernet for plant automation): Proposal for a publicly available specification for real-time Ethernet", *Doc. IEC 65C/357/NP*, 2004.

[11] P. Pedreiras, P. Gai, L. Almeida, and G. C. Buttazzo, "FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems" , *IEEE Transactions on Industrial Informatics*, vol 1, no. 3, 2005.

[12] K.W. Schmidt, E.G. Schmidt, "Distributed Real-Time Protocols for Industrial Control Systems: Framework and Examples", *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 10, pp. 1856 - 1866, 2012.

[13] (2013) IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. [last accessed on 31/07/2013]. [Online]. Available: http://ieee1588.nist.gov

[14] J. Moyne, D. Tilbury, "The emergence of industrial control networks for manufacturing control, diagnostics, and safety data", *Proceedings of the IEEE*, 95, 29–47, 2007.

[15] J. P.Thomesse, "Fieldbus technology in industrial automation", *Proceedings of the IEEE*, 93, 1073–1101, 2005.

[16] M. Felser, R. Zurawski ,"Real-time Ethernet for automation applications*", Embedded Systems Handbook, Second Edition: Networked Embedded Systems*, pp. 21–1–21–20 2nd edition, 2009.

[17] J. C. Eidson, "Measurement, Control, and Communication Using IEEE 1588", *Springer*, 2006.

[18] K. W. Schmidt, E. G.Schmidt, "Distributed real-time protocols for industrial control systems: Framework and examples", *Parallel and Distributed Systems, IEEE Transaction*s on, 23, 1856–1866, 2012.

[19] "1st IFAC workshop on dependable control of discrete systems", *Proceedings of a meeting held in Cachan, France*, 2007.

[20] A. Avizienis, J.C Laprie, B.Randell, C.Landwehr, "Basic concepts and taxonomy of dependable and secure computing", *Dependable and Secure Computing, IEEE Transactions on*, 1, 11–33, 2004.

[21] M. Felser, T. Sauter, "Standardization of industrial Ethernet - the next battlefield", *Factory Communication Systems, Proceedings. IEEE International workshop on*, 413–420, 2004.

[22] J. D. Decotignie, "The many faces of industrial Ethernet [past, present]", *Industrial Electronics Magazine, IEEE*, 3, 8–19, 2009.

[23] S.K. Kweon, K. G. Shin, "Statistical real-time communication over Ethernet, Parallel , Distributed Systems", *IEEE Transactions on*, 14, 322–335, 2003.

[24] S.K. Kweon, M.G. Cho, K. G. Shin, "Soft real-time communication over Ethernet with adaptive traffic smoothing", *Parallel and Distributed Systems, IEEE Transactions on*, 15, 946–959, 2004.

[25]J.D. Decotignie, "Ethernet-based real-time and industrial communications", *Proceedings of the IEEE*, 93, 1102–1117, 2005.

[26] M. Felser, "Real-time Ethernet - industry prospective", *Proceedings of the IEEE*, 93, 1118–1129, 2005.

[27] Z. Wang, Y. Song, J. Chen, Y. Sun, "Real time characteristics of Ethernet and its improvement", *4th World Congr. Intelligent Control, Automation*, pp. 1311–1318, 2002.

[28] "Real-time Ethernet: Ethernet/IP with time synchronization: Proposal for a publicly available specification for real-time Ethernet", *Doc. IEC 65C/361/NP*, 2004.

[29] U.Turan, "Implementation and Evaluation of a New Protocol for Industrial Communication Networks" , M.Sc. Thesis, METU Sept.2011.

[30] “Real-time Ethernet: TCnet (Time-Critical Control Network): Proposal for a publicly available specification for real-time Ethernet”, *Doc. IEC 65C/353/NP*, 2004.

[31] T. Akima and K. Shibata, “Development of Real-Time Ethernet Based I/O Network, SICE Annual Conference”, *The University Electro-Communications*, Japan, 2008.

[32] (2013) Flexible Time-Triggered (*FTT*) paradigm. [last accessed on 05/01/2013] [Online].Available: *http*:// http://www.ieeta.pt/lse/ftt/

[33] V. Nelson, “Fault-tolerant computing: fundamental concepts”, Computer, 23, 19 –25, 1990.

[34] A. Avizienis, J.C. Laprie, B. Randell, “Fundamental concepts of Dependability”, *Technical Report Series University Of Newcastle*, 1145, 7–12, 2001.

[35] B.Meyer, “Every little bit counts: toward more reliable software”, Computer, 32, 131 – 135, 1999.

[36] R.Chillarege, I. Bhandari, J. Chaar, M.Halliday, D. Moebus, B. Ray, M.Y. Wong, “Orthogonal defect classification-a concept for in-process measurements”, *IEEE Transactions on Software Engineering*, 18, 943–956, 1992.

[37] M. C.Paulk, B. Curtis, M. B.Chrissis, C. V. Weber, “Capability maturity model, version 1.1”, *IEEE Softw.*, 10, 18–27, 1993.

[38] B.Randell, “System structure for software fault tolerance”, *SIGPLAN Not*., 10, 437–449, 1975.

[39] K. G.Shin, Y. H. Lee, “Evaluation of error recovery blocks used for cooperating processes”, *IEEE Trans Soft Eng*, SE-10, 692–700, 1984.

[40] P.Ramanathan, K. G.Shin, “Use of common time base for checkpointing and rollback recovery in a distributed system”, *IEEE Trans. Softw. Eng.*, 19, 571–583, 1993.

[41] A. Kaya, “Implementation and Evaluation of the Dynamic Distributed Real Time Industrial Protocol (D^2RIP)” , M.Sc. Thesis, METU Sept. 2013.

[42] A. K. Gözcü, “Implementation and Evaluation of a Synchronous Time-Slotted Medium Access Protocol for Networked Industrial Embedded Systems”, M.Sc. Thesis, METU Sept. 2011.

[43] Y. B. Kartal, “Dependable Framework Design for Distributed Real-Time Network Protocols Running On Shared Medium: Design, Simulation and Verification”, Phd. Thesis, METU, 2013. (Under Preparation).

[44] K. Schmidt , E. Schmidt, A. Kaya, “Dynamic Distributed Real-time Industrial Ethernet Protocol (D^2RIP): Architecture, Implementation and Experimental Evaluation”, *Submitted to IEEE Transactions on Industrial Informatics*, 2013.

[45] Y. B.Kartal, K. W. Schmidt, E. G. Schmidt, “Dependability design for a distributed real-time protocol family, Parallel and Distributed Systems”, *IEEE Transactions on*, (to be submitted), 2013.

APPENDIX

XML FILES

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE D3RIP SYSTEM "controllerPD_d3rip.dtd">
<D3RipUrtDevice  name="controllerPD_d3rip">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>


<!-- Event configuration -->
<Event name="PDon?" iotype="input">
<EventId value="25"/>
</Event>


<Event name="PDon!" iotype="output">
<EventId value="26"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>


<Event name="PDonc" iotype="input">
<EventId value="27"/>
</Event>


<Event name="PDoff?" iotype="input">
<EventId value="28"/>
</Event>


<Event name="PDoff!" iotype="output">
<EventId value="29"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="PDoffc" iotype="input">
<EventId value="30"/>
</Event>
```

```
</EventConfiguration>

</D3RipUrtDevice>
```

D3RIP XML Configuration File for Controller PD

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE SimplenetDevice SYSTEM "controllerPD.dtd">

<SimplenetDevice name="controllerPD">
<TimeScale value="10"/>

<!-- Ip address of this node, incl. server tcp port -->
<ServerAddress value="localhost:40000"/>

<!-- Network topology -->
<Network name="paintingNet">
<Node name="plant"/>
<Node name="controllerR"/>
<Node name="controllerC"/>
<Node name="controllerPD"/>
</Network>

<!-- Event configuration -->
<EventConfiguration>
<Event name="fPD" iotype="input"/>
</EventConfiguration>

</SimplenetDevice>
```

SimpleNet XML Configuration File for Controller PD

```
<Executor>
<Generators>
<Generator>
controllerC
<Alphabet>
PDon?  PDon!   PDonc  PDon    fPD      PDoff?  PDoff!   PDoffc  PDoff
</Alphabet>
<States>
<Consecutive>
1        12
</Consecutive>
</States>
<TransRel>
1        PDon?  2
2        PDon!  3
3        PDonc  4
4        PDoff? 5
4        PDon   6
5        PDon   7
6        PDoff? 7
6        fPD    8
7        fPD    9
```

```
8          PDoff?  9
9          PDoff!  10
10         PDoffc  11
11         PDoff   1
11         PDon?   12
12         PDoff   2
</TransRel>

<InitStates> 1       </InitStates>

<MarkedStates> 1    </MarkedStates>
</Generator>
</Generators>

% specify event attributes
<SimEventAttributes>
% Sensor Events
"stpR"    <Priority>-1</Priority>
"stpC"    <Priority>-1</Priority>
"fPD"     <Priority>-1</Priority>
% Actuator Events
"mvC"   <Priority>1</Priority>
"sC"     <Priority>1 </Priority>
"arC"    <Priority>1 </Priority>
"mvI"    <Priority>1  </Priority>
"sI"     <Priority>1  </Priority>
"arI"    <Priority>1</Priority>
"mvPD" <Priority>1</Priority>
"sPD"    <Priority>1</Priority>
"arPD"   <Priority>1 </Priority>
"mvR"    <Priority>1 </Priority>
"sR"     <Priority>1 </Priority>
"arR"    <Priority>1</Priority>
"PDon"   <Priority>1</Priority>
"PDoff"  <Priority>1</Priority>

</SimEventAttributes>

</Executor>
```

Simulator XML Configuration File for Controller PD

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE D3RIP SYSTEM "controllerR_d3rip.dtd">
<D3RipUrtDevice  name="controllerR_d3rip">
<TimeScale value="10"/>
<!-- Ip address of this node, incl. server tcp port -->
<ServerAddress value="localhost:40000"/>
<Event name="mvC?" iotype="input">
<EventId value="1"/>
</Event>

<Event name="mvC!" iotype="output">
<EventId value="2"/>
<ChannelToTransmit value="1"/>
```

```
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvCc" iotype="input">
<EventId value="3"/>
</Event>

<Event name="arC?" iotype="input">
<EventId value="4"/>
</Event>

<Event name="arC!" iotype="output">
<EventId value="5"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arCc" iotype="input">
<EventId value="6"/>
</Event>

<Event name="mvI?" iotype="input">
<EventId value="7"/>
</Event>

<Event name="mvI!" iotype="output">
<EventId value="8"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvIc" iotype="input">
<EventId value="9"/>
</Event>

<Event name="arI?" iotype="input">
<EventId value="10"/>
</Event>

<Event name="arI!" iotype="output">
<EventId value="11"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
```

```
<DestinationNode value="1"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arlc" iotype="input">
<EventId value="12"/>
</Event>
</EventConfiguration>
</D3RipUrtDevice>
```

D3RIP XML Configuration File for Controller R

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE SimplenetDevice SYSTEM "controllerR.dtd">

<SimplenetDevice name="controllerR">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>

<!-- Network topology -->
<Network name="paintingNet">
<Node name="plant"/>
<Node name="controllerR"/>
<Node name="controllerC"/>
<Node name="controllerPD"/>
</Network>

<!-- Event configuration -->
<EventConfiguration>
<Event name="sC" iotype="output"/>
<Event name="stpR" iotype="input"/>
<Event name="sl" iotype="output"/>
</EventConfiguration>

</SimplenetDevice>
```

SimpleNet XML Configuration File for Controller R

```
<Executor>
<Generators>
<Generator>controllerR
<Alphabet>
mvC?   mvC!   mvCc   mvC    arC?   arC!   arCc   arC    sC     stpR
mvl?   mvl!   mvlc   mvl           sl     arl?   arl!   arlc   arl
</Alphabet>
<States>
<Consecutive>
1      28
</Consecutive>
```

```
</States>
<TransRel>
1          mvC?    2
2          mvC!    3
3          mvCc    4
4          arC?    5
4          mvC     6
5          mvC     7
6          arC?    7
6          sC      8
7          sC      9
8          arC?    9
8          stpR    10
9          stpR    11
10         arC?    11
11         arC!    12
12         arCc    13
13         arC     14
13         mvl?    15
14         mvl?    16
15         arC     16
16         mvl!    17
17         mvlc    18
18         arl?    19
18         mvl     20
19         mvl     21
20         arl?    21
20         sl      22
21         sl      23
22         arl?    23
22         stpR    24
23         stpR    25
24         arl?    25
25         arl!    26
26         arlc    27
27         arl     1
27         mvC?    28
28         arl     2
</TransRel>

<InitStates>1        </InitStates>
<MarkedStates>1    </MarkedStates>
</Generator>
</Generators>
<SimEventAttributes>
% Sensor Events
"stpR"    <Priority>-1              </Priority>
"stpC"     <Priority>-1              </Priority>
"fPD"      <Priority>-1              </Priority>
% Actuator Events
"mvC"      <Priority>1               </Priority>
"sC"      <Priority>1                </Priority>
"arC"      <Priority>1               </Priority>
"mvl"      <Priority>1               </Priority>
"sl"      <Priority>1                </Priority>
"arl"      <Priority>1       </Priority>
"mvPD"      <Priority>1              </Priority>
"sPD"      <Priority>1               </Priority>
```

```
"arPD"      <Priority>1              </Priority>
"mvR"       <Priority>1              </Priority>
"sR"      <Priority>1                </Priority>
"arR"       <Priority>1              </Priority>
"PDon"      <Priority>1              </Priority>
"PDoff"     <Priority>1              </Priority>

</SimEventAttributes>
</Executor>
```

Simulator XML Configuration File for Controller R

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE D3RIP SYSTEM "controllerS_d3rip.dtd">
<D3RipUrtDevice  name="controllerS_d3rip">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>

<!-- Event configuration -->
<!-- Communication with robot (node 1) -->
<Event name="mvC?" iotype="output">
<EventId value="1"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvC!" iotype="input">
<EventId value="2"/>
</Event>

<Event name="mvCc" iotype="output">
<EventId value="3"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arC?" iotype="output">
<EventId value="4"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>
```

```
<Event name="arC!" iotype="input">
<EventId value="5"/>
</Event>

<Event name="arCc" iotype="output">
<EventId value="6"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvI?" iotype="output">
<EventId value="7"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvI!" iotype="input">
<EventId value="8"/>
</Event>

<Event name="mvIc" iotype="output">
<EventId value="9"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arI?" iotype="output">
<EventId value="10"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arI!" iotype="input">
<EventId value="11"/>
</Event>

<Event name="arIc" iotype="output">
<EventId value="12"/>
```

```
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="2"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<!-- Communication with conveyor (node 3) -->
<Event name="mvPD?" iotype="output">
<EventId value="13"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvPD!" iotype="input">
<EventId value="14"/>
</Event>

<Event name="mvPDc" iotype="output">
<EventId value="15"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arPD?" iotype="output">
<EventId value="16"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arPD!" iotype="input">
<EventId value="17"/>
</Event>

<Event name="arPDc" iotype="output">
<EventId value="18"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
```

```
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvR?" iotype="output">
<EventId value="19"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="mvR!" iotype="input">
<EventId value="20"/>
</Event>

<Event name="mvRc" iotype="output">
<EventId value="21"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arR?" iotype="output">
<EventId value="22"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="arR!" iotype="input">
<EventId value="23"/>
</Event>

<Event name="arRc" iotype="output">
<EventId value="24"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="3"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<!-- Communication with painting device (node 4) -->
<Event name="PDon?" iotype="output">
```

```
<EventId value="25"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="4"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="PDon!" iotype="input">
<EventId value="26"/>
</Event>

<Event name="PDonc" iotype="output">
<EventId value="27"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="4"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="PDoff?" iotype="output">
<EventId value="28"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="4"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>

<Event name="PDoff!" iotype="input">
<EventId value="29"/>
</Event>

<Event name="PDoffc" iotype="output">
<EventId value="30"/>
<ChannelToTransmit value="1"/>
<ParameterRecord name="11">
<DestinationNode value="4"/>
<DestinationChannel value="1"/>
<EligibilityTime value="2" />
<DeadlineTime value="5"/>
</ParameterRecord>
</Event>
</EventConfiguration>
</D3RipUrtDevice>
```

D3RIP XML Configuration File for Controller S

```
<Executor>
<Generators>
<Generator>controllerS
<Alphabet>
mvC?   mvC!   mvCc   arC?   arC!   arCc
mvl?   mvl!   mvlc   arl?   arl!   arlc
mvPD?  mvPD!  mvPDc  arPD?  arPD!  arPDc
mvR?   mvR!   mvRc   arR?   arR!   arRc
PDon?  PDon!  PDonc  PDoff?  PDoff!  PDoffc
</Alphabet>
<States>
<Consecutive>1 30</Consecutive>
</States>
<TransRel>
1        mvC?   2
2        mvC!   3
3        mvCc   4
4        arC?   5
5        arC!   6
6        arCc   7
7        mvPD?  8
8        mvPD!  9
9        mvPDc  10
10       arPD?  11
11       arPD!  12
12       arPDc  13
13       PDon?  14
14       PDon!  15
15       PDonc  16
16       PDoff?  17
17       PDoff!  18
18       PDoffc  19
19       mvR?   20
20       mvR!   21
21       mvRc   22
22       arR?   23
23       arR!   24
24       arRc   25
25       mvl?   26
26       mvl!   27
27       mvlc   28
28       arl?   29
29       arl!   30
30       arlc   1
</TransRel>
<InitStates>1       </InitStates>
<MarkedStates>1    </MarkedStates>
</Generator>
</Generators>
<SimEventAttributes>
% Sensor Events
"stpR"    <Priority>-1 </Priority>
"stpC"     <Priority>-1</Priority>
"fPD"      <Priority>-1</Priority>
% Actuator Events
"mvC"      <Priority> 1          </Priority>
"sC"      <Priority>   1          </Priority>
"arC"      <Priority> 1          </Priority>
```

```
"mvl"      <Priority>1          </Priority>
"sl"      <Priority>   1        </Priority>
"arl"      <Priority>  1        </Priority>
"mvPD"      <Priority>1         </Priority>
"sPD"      <Priority>1          </Priority>
"arPD"      <Priority>1         </Priority>
"mvR"      <Priority>1          </Priority>
"sR"      <Priority>1           </Priority>
"arR"      <Priority>1          </Priority>
"PDon"      <Priority>1         </Priority>
"PDoff"     <Priority>1         </Priority>
</SimEventAttributes>
</Executor>
```

Simulator XML Configuration File for Controller R

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE SimplenetDevice SYSTEM "plant.dtd">
<SimplenetDevice name="plant">
<TimeScale value="10"/>
<ServerAddress value="localhost:40000"/>
<Network name="paintingNet">
<Node name="plant"/>
<Node name="controllerR"/>
<Node name="controllerC"/>
<Node name="controllerPD"/>
</Network>

<EventConfiguration>
<Event name="sC" iotype="input"/>
<Event name="stpR" iotype="output"/>
<Event name="sl" iotype="input"/>
<Event name="sPD" iotype="input"/>
<Event name="stpC" iotype="output"/>
<Event name="sR" iotype="input"/>
<Event name="fPD" iotype="output"/>
</EventConfiguration>
</SimplenetDevice>
```

SimpleNet XML Configuration File for Plant

```
<Executor>
<Generators>
<Generator>plantConveyor
<Alphabet>
mvPD   sPD      stpC     arPD     mvR     sR       stpC     arR
</Alphabet>
<States>
1          2
3   <Invariant> "cMove"   "LT" 10 </Invariant>
4          5          6
7  <Invariant> "cMove"   "LT" 10 </Invariant>
8
</States>
```

```
<TransRel>
1          mvPD    2
2          sPD      3
<Timing>
<Resets>"cMove"   </Resets>
</Timing>3        stpC      4     <Timing>
<Guard>    "cMove" "GT" 5 </Guard>
</Timing>
4          arPD     5
5          mvR      6
6          sR        7
<Timing>
<Resets>"cMove"</Resets>
</Timing>7        stpC      8<Timing>
<Guard>    "cMove" "GT" 5</Guard>
</Timing>8        arR       1
</TransRel>
<InitStates>1        </InitStates>
<MarkedStates>1     </MarkedStates>
<Clocks>"cMove" </Clocks>
</Generator>
<Generator>plantPaintingDevice
<Alphabet>PDon            fPD       PDoff</Alphabet>
<States>
1
2          <Invariant> "pdMove"   "LT" 10 </Invariant>
3
</States>
<TransRel>1      PDon     2<Timing>
<Resets>"pdMove"</Resets>
</Timing>2        fPD                 3<Timing>
<Guard>    "pdMove" "GT" 5</Guard>
</Timing>3        PDoff     1</TransRel>
<InitStates>1        </InitStates>
<MarkedStates>1     </MarkedStates>
<Clocks>"pdMove" </Clocks>
</Generator>
<Generator>plantRobot
<Alphabet>mvC sC         stpR      arC       mvI       sI          stpR      arI</Alphabet>
<States>
1          2
3  <Invariant> "rMove"   "LT" 10 </Invariant>
4          5          6
7          <Invariant> "rMove"   "LT" 10 </Invariant>
8
</States>
<TransRel>
1          mvC      2
2          sC        3
<Timing>
<Resets>"rMove"</Resets>
</Timing>3        stpR      4<Timing>
<Guard>    "rMove" "GT" 5</Guard>
</Timing>
4          arC       5
5          mvI       6
6          sI         7
<Timing>
```

```
<Resets>"rMove"</Resets>
</Timing>7        stpR      8<Timing>
<Guard>    "rMove" "GT" 5</Guard>
</Timing>8        arl         1
</TransRel>
<InitStates>1        </InitStates>
<MarkedStates>1     </MarkedStates>
<Clocks>"rMove" </Clocks>
</Generator>
</Generators>
<SimEventAttributes>
"stpR"     <Priority>-1          </Priority>
"stpC"      <Priority>-1</Priority>
"fPD"       <Priority>-1</Priority>
"mvC"       <Priority>1           </Priority>
"sC"       <Priority>1           </Priority>
"arC"       <Priority>1           </Priority>
"mvl"       <Priority>1           </Priority>
"sl"       <Priority>1           </Priority>
"arl"       <Priority>1           </Priority>
"mvPD"       <Priority>1           </Priority>
"sPD"       <Priority>1           </Priority>
"arPD"       <Priority>1           </Priority>
"mvR"       <Priority>1           </Priority>
"sR"       <Priority>1           </Priority>
"arR"       <Priority>1           </Priority>
"PDon"       <Priority>1           </Priority>
"PDoff"       <Priority>1           </Priority>
</SimEventAttributes>
</Executor>
```

Simulator XML Configuration File for Plant