SIMULATION AND PERFORMANCE EVALUATION OF A DISTRIBUTED REAL-TIME COMMUNICATION PROTOCOL FOR INDUSTRIAL EMBEDDED SYSTEMS

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

BY

GÜRAY AYBAR

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

DECEMBER 2011

Approval of the thesis

SIMULATION AND PERFORMANCE EVALUATION OF A DISTRIBUTED REAL-TIME COMMUNICATION PROTOCOL FOR INDUSTRIAL EMBEDDED SYSTEMS

submitted by **GÜRAY AYBAR** 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. İsmet ERKMEN __________
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Şenan Ece SCHMIDT __________
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Semih BİLGEN ____________
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Şenan Ece SCHMIDT ____________
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Gözde BOZDAĞI AKAR ____________
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI ____________
Electrical and Electronics Engineering Dept., METU

Ms. Sc. Bora KARTAL ___________
REHİS, ASELSAN

Date: 07.12.2011

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	: GÜRAY AYBAR
Signature	:

ABSTRACT

SIMULATION AND PERFORMANCE EVALUATION OF A DISTRIBUTED REAL-TIME COMMUNICATION PROTOCOL FOR INDUSTRIAL EMBEDDED SYSTEMS

AYBAR, Güray
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ece Güran SCHMIDT

December 2011, 77 pages

The Dynamic Distributed Dependable Real-Time Industrial communication Protocol (D^{3}RIP) provides service guarantees for Real-Time traffic and integrates the dynamically changing requirements of automation applications in their operation to efficiently utilize the resources. The protocol dynamically allocates the network resources according to the respective system state. To this end, the protocol architecture consists of an *Interface Layer* that provides time-slotted operation and a *Coordination Layer* that assigns each time slot to a unique transmitter device based on a distributed computation.

In this thesis, a software simulator for D^{3}RIP is developed. Using the D^{3}RIP Simulator, modifications in D^{3}RIP can be easily examined without facing complexities in real implementations and extensive effort in terms of time and cost. The simulator simulates the Interface Layer, the Coordination Layer and additionally, the *Shared Medium*. Hence, using the simulator, the system-protocol couple can be easily analyzed, tested and further improvements on D^{3}RIP can be achieved with the least amount of effort.

The simulator implements the Timed Input Output Automata (TIOA) models of the D^{3}RIP stack components using C++. The resulting code is compiled on GCC (Gnu Compiler Collection). The logs of the simulation runs and the real system with 2 devices connected via cross 100MbE cables are compared. In a 3ms time slot, the simulator and the system incidents differ about 135μs on the average, causing no asynchronousity in their instantaneous operational states. The D^{3}RIP Simulator is useful in keeping track of any variable in the D^{3}RIP system automaton at any instant up to 1μs resolution.

Keywords: Distributed Real-Time Ethernet Systems, Timed I-O Automata (TIOA) Modeling, C++, Simulation of Real-Time Systems, Industrial Control, Computer Applications, Industrial Communication Networks

ÖZ

ENDÜSTRİYEL GÖMÜLÜ SİSTEMLER İÇİN DAĞITILMIŞ GERÇEK ZAMANLI BİR HABERLEŞME PROTOKOLUNUN BENZETİMİ VE BAŞARIM DEĞERLENDİRİLMESİ

AYBAR, Güray
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ece Güran SCHMIDT

Aralık 2011, 77 sayfa

Dinamik Dağıtılmış güvenilir Gerçek Zamanlı Endüstriyel iletişim Protokolü, gerçek zamanlı trafik için servis garantisi sağlamaktadır ve haberleşme kaynaklarından verimli bir şekilde faydalanabilmek için otomatizasyon uygulamalarının işlemlerinde dinamik olarak değişim gösteren gereksinimlerini de birleştirmektedir. Protokol, ağ kaynaklarının ilgili sistem durumuna göre dinamik olarak atanmasını sağlar. Bu amaçla, protokolmimarisi, zaman dilimli işlem sağlayan bir *Arayüz Katmanı*ve her zaman dilimini, dağıtılmış hesaplamalar sonucunda yalnızca tek bir iletici aygıta atayan bir *Koordinasyon Katmanı*'ndan oluşur.

Bu tezde, D^{3}RIP için bir benzetici yazılımı geliştirilmiştir. D^{3}RIP Benzeticisi kullanılarak, D^{3}RIP'teki değişiklikler gerçek uygulamalardaki güçlükler ile karşılaşmadan ve zaman ve maliyet açısından ek yük gerektirmeden kolayca denetlenebilir. Benzetici, *Arayüz Katmanı*'nı, *Koordinasyon Katmanı*'nı ve ek olarak *Paylaşımlı Ortam* benzetimini yapar. Dolayısıyla, benzetici kullanılarak, sistem-protokol ikilisi kolaylıkla analiz ve test edilebilir ve D^{3}RIP'in en az masraf ile daha fazla gelişime ulaşılır.

Benzetici, C++ kullanarak D^{3}RIPyığınındaki parçaların Zamanlı Girdi/Çıktı Otomat (TIOA) modellerini uygular. Ortaya çıkan kod, GCC (GNU Derleyici Koleksiyonu) ile derlenmiştir. Yapılan benzetimlerin sonuçları ile iki aygıtın birbirlerine çapraz 100MbE kablolar ile bağlanmasından oluşan gerçek sistemin kayıtları karşılaştırılmıştır. 3ms zaman aralığında, benzeticide ve sistemde gerçekleşen olaylarının anlık durumlarında zaman uyumsuzluklarına yol açmayan, ortalama 135µs'lik bir fark ortaya çıkmıştır. D^{3}RIP Benzeticisi, D^{3}RIP sistem otomatındaki herhangi bir değişkeni 1µs çözünürlükle herhangi bir anda takip edilmesinde faydalıdır.

Anahtar Kelimeler: Dağıtılmış Gerçek Zamanı Ethernet Sistemleri, Zamanlı Girdi/Çıktı Otomat Modeli, C++, Gerçek Zamanlı Sistemlerin Benzeştirilmesi, Endüstriyel Kontrol, Bilgisayar Uygulamaları, Endüstriyel Haberleşme Ağları

to my beloved family

ACKNOWLEDGEMENTS

This thesis work is one of the most important events throughout my entire life. But I know that without the people I have to thank, it would be impossible for me to achieve.

First of all, I am sincerely grateful and wish to send my hearthful of thanks and gratitude to my advisor, then, supervisor, Assoc. Prof. Dr. Şenan Ece SCHMIDT and Assistant Prof. Dr. Klaus SCHMIDT for their unlimited help, perfect supervision, leading, guidance and understanding from beginning till the end of this thesis work.

I would like to thank my parents Hatice Mesar AYBAR and Adnan AYBAR, and my also-a-colleague brother Bahadır AYBAR for their endless patience, supports, encouragements and their deep belief in me in my thesis work and throughout all my education life.

I have to send my special thanks to the love of my life, yet my fiancee, Sinem İŞBİLİR, who has always encouraged and motivated me on my thesis with her lovely existence, constant patience and understanding.

Finally, I wish to thank all my friends, especially to Ahmet Korhan GÖZCÜ and Ulaş TURAN for their ideas, help and support on completing my thesis.

This work would not have finished if any of them were not there for me.
Really appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATONS

AP2CL	: Control Application to Coordination Layer
AP2ILnRT	: Control Application to Interface Layer
CL	: Coordination Layer
CL2AP	: Coordination Layer to Control Application
CL2ILRT	: Coordination Layer to Interface Layer
CSMA/CD	: Carrier Sense Multiple Access with Collision Detection
D^3RIP	: Dynamic Dependable Distributed Real-time Industrial Protocol
DART	: Dynamic Access Real-Time Protocol
IL	: Interface Layer
IL2APnRT	: Interface Layer to Control Application
IL2CLRT	: Interface Layer to Coordination Layer
IL2SM	: Interface Layer to Shared Medium
LAN	: Local Area Network
nRT	: non-Real-Time
RAIL	: Real-Time Access Interface Layer
REQRT	: Real-Time Request Issued by Interface Layer
RT	: Real-Time
RTE	: Real-Time Ethernet
SM	: Shared Medium
SM2IL	: Shared Medium to Interface Layer
TIOA	: Timed Input/Output Automata
TSIL	: Time-Slotted Interface Layer
UPD	: Update
URT	: Urgency-based Real-Time Protocol

CHAPTER 1

INTRODUCTION

In recent years, distributed electronic Real-Time control systems have become more and more widespread and also essential due to the large demand on hard real time requirements of the technological improvements, such as flight control systems, automotives; signal tracking, aerospace and industrial applications, etc. However, as the technology evolves and develops, it also advances the necessity of the physical distribution of control systems in strict real time. As a result, a need for the network protocols to meet the stringent real time requirements arises, such that the service of operation of real time signals is guaranteed and the network will operate deterministically, meeting the demands.

A hard real-time system must operate within the deadline constraints and does not tolerate unexpected delays. A missed deadline can be disastrous since most of the hard real-time systems are safety critical applications. Ethernet is widely utilized for the communication of industrial automation system components that perform local computations and exchange information via communication networks. But, as defined in IEEE 802.3, the arbitration mechanism is the nondeterministic Carrier Sense Multiple Access with Collision Detection (CSMA/CD) [6], [2], [3]. Thus, it is not convenient to be used for hard real time applications since unpredictable packet transmission delays (back-offs) and transmission errors might take place. To address these issues, there are various ongoing research and development efforts to provide Ethernet-based industrial network solutions with RT support [5], [6] and thus, converge to a single network technology on the different levels of the automation hierarchy.

The D^{3}RIP stack presented in [26] is investigated. The framework has a time-slotted Interface Layer (IL) for RT and nRT; and a Coordination Layer (CL) for

RT signals, such that the system has a guaranteed RT traffic and is dynamically adapting to the changing needs of the application.

The D^3RIP stack is quite an effective and a detailed protocol family. The system state depends on many different variables. But it is hard to be implemented in real life. Especially for each new idea or a modification to be examined, it is impossible to adjust at the setup level, since it would take a lot of effort and time. Moreover, the real system is complicated.

As a result, if a simulator, which simulates the system behaviour as closely as possible, is figured out, it would be possible to investigate new ideas or apply possible changes and modifications to the protocol with the results of the simulation as if they were actually built and run.

Therefore, using a realistic simulator design is quite a way observe the system behaviour, saving a lot of time and effort and minimizes the cost.

In the thesis, the aim of the work is to develop a simulator software for the D^3RIP framework and verify the correctness of both the simulator and the actual implementation in the laboratory that was carried out independently. This verification is carried out by comparing the outcomes of the simulation with those of the actual system in terms of instantaneous states of the system, the values of the variables, i.e. the event times, actions, owner of the time slots, etc.

The simulation is designed as an event-based operation. Hence, in every event, the operation of the event takes place and messages are transmitted and received, the system states are updated, etc. However these actions all occur instantaneously, with no duration, at all. The actual implementation of the D^3RIP stack involves the calculation and reading latencies, line propagation delays, synchronization problems, jitter, etc.

Hence, as well as being hard to be predicted, these communication defects might cause slight differences in the comparison of the real and simulated system behaviours. Our experiments show that the impact of the assumptions and abstractions in the simulator model can lead to discrepancies in the results from the real system.

This thesis totally consists of 6 chapters. The remainder of the thesis is organized as follows.

In Chapter 2, general concepts and some background information on Ethernet, Distributed Systems and finally, Distributed Real-Time Ethernet Systems are given.

In Chapter 3, the protocol in [26] and the work is defined. An example for RTE Systems is given and has been inspected in detail. Then, a solution is proposed; performance specs and challenges are claimed. The Timed Input/Output Automata (TIOA) Models is introduced as a formal modeling tool.

In Chapter 4, The D^3RIP Simulator is defined. Moreover, the models for the Shared Medium, Interface Layer (IL) and the Coordination Layers (CL) both for 2 different types of ILs and CLs are explained in detail. The abstractions and the assumptions are explained and finally, the implementations of these TIOA models for the SM, the IL types and the CL types are elaborated.

In Chapter 5, Simulation results are presented and comparisons to the experimental set-up are discussed.

Lastly, in Chapter 6, the conclusions are given, and results are interpreted with the possible future studies.

References are given in the end.

CHAPTER 2

ETHERNET FOR DISTRIBUTED REAL-TIME COMMUNICATION

We first present some brief information about Ethernet, how it was founded and its application areas, then, we move into the systems that employ Ethernet as the communication protocol family.

2.1 Ethernet

IEEE 802.3 CSMA/CD, namely Ethernet, is a standard embedded protocol family that controls the data delivery technology over a Local Area Network (LAN). It is the most widely employed LAN protocol, today.

Ethernet was first created in 1973 by Robert Metcalfe, being inspired by his studies about Alohanet. Later, DEC (Digital Equipment Corporation), Intel and Xerox worked together for the promotion of Ethernet as a new standard in networking. The group published the standard, which was called the “DIX” standard. The DIX specified Ethernet at the data rate of 10Mbps and addresses of 48-bits. In 1980, for the standardization of local area networks (LAN), the Institute of Electrical and Electronics Engineers (IEEE) launched Project 802. The DIX-group suggested the CSMA/CD protocol in opposition to the Token Ring and the Token Bus.

The connection between nodes is usually provided by RJ-45 jacks. Most commonly, the Ethernet systems are the 10BASE-T systems supporting a data rate of 10Mbps and the devices try to get access to the medium using CSMA/CD protocol via the RJ-45 cables for the connection, shown in Figure 2-1.

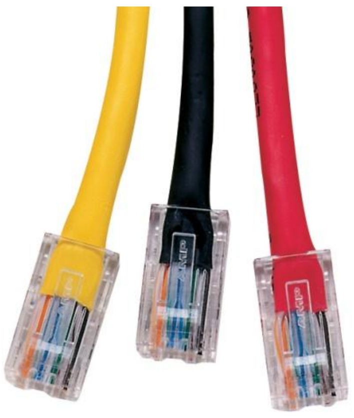


Figure 2-1. The Mostly Used Ethernet Connector: RJ-45.

Often in backbone systems, 10BASE-T cards are used to implement the Fast Ethernet, namely 100BASE-T which provides up to a 100Mbps data rate. Faster Ethernet protocols also exist, such as the Gigabit Ethernet (GbE) supporting a data rate of 1Gbps and 10GbE, which provides 10Gbps of data rate as an invulnerable option for large scale systems requiring a large support of networking infrastructure.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) method is a Medium Access Control method which employs a carrier sensing scheme. The station or device that has the medium access, detects another message transmission attempt from another station. Then, it immediately halts the transmission and after sending a jam signal, it waits a random back-off until next transmission attempt for the same frame. The collision detection feature was added to decrease the time consumed until next attempt, hence increasing the efficiency.

The devices detect the collision in several different ways depending on the medium. For electrical busses as 10BASE-2 or 10BASE-5, by comparing the transmitted data with the received data, collisions can be detected easily. Moreover, the collision might cause higher signal amplitude than the successful transmission on the bus. Hence, the collision can also be detected by checking the amplitude of the signal on the bus.

In the past, the widest use of CSMA/CD was on the Shared Medium Ethernet types that have currently become obsolete, 10BASE-2 or 10BASE-5, and the earlier twisted-pair Ethernet versions. Too many stations transmitting on an Ethernet network might cause an unacceptable level of collisions. This could result in a large amount of reduction on the bandwidth of an Ethernet network due to the lost bandwidth for the re-transmisison. For this reason, Ethernet swithces are employed to reduce these difficulties.

2.2 Distributed and Real-time Systems

A distributed system is composed of a number of autonomous devices. These devices are able to communicate with each other, enabling the cooperation and resource sharing with each other. Hence, the users perceive the network with the devices as a whole integrated system.

In a distributed system, there are multiple control points as well as multiple failure points. A diagram for the difference of a parallel and a distributed system is depicted in Figure 2-2.

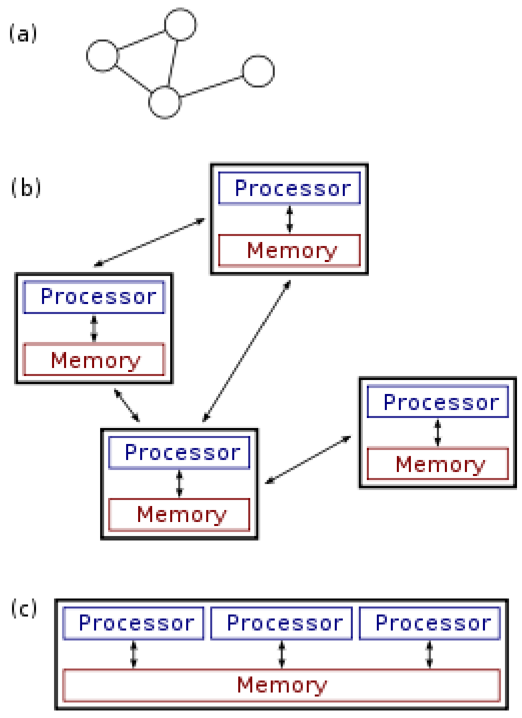


Figure 2-2. (a)-(b) A Distributed System, (c) A Parallel System.

A centralized system malfunctions if a small number of hierarchically high level devices do not work properly. However, a distributed system has no such central nodes. Hence, the system does not have to be dependent on any other node to operate. This is the major advantage of distributed systems.

In addition, as the number of nodes increase, the robustness and resilience of the system increases. Figure 2-3 shows the difference of a distributed system from a centralized system.

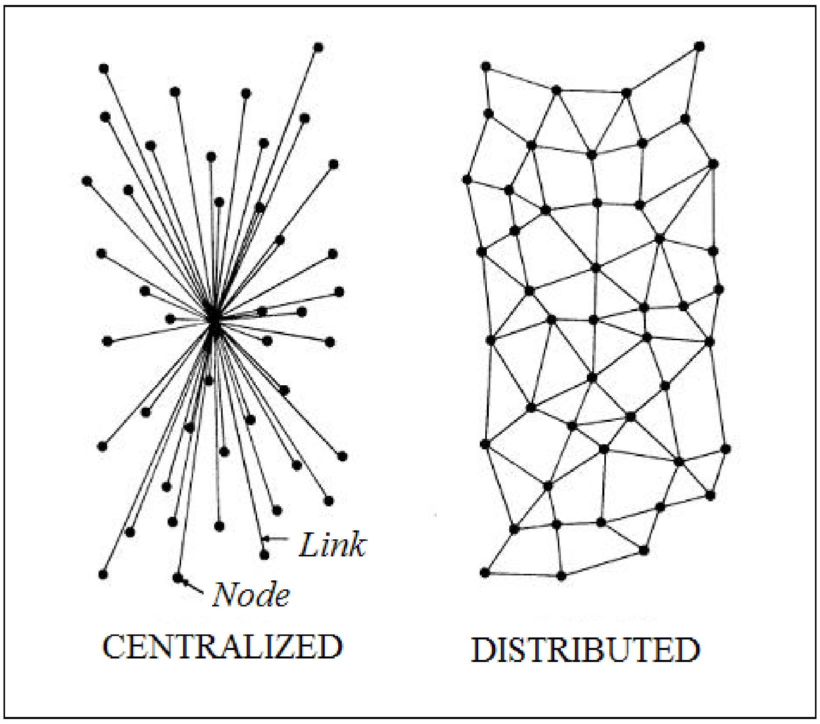


Figure 2-3. Centralized System vs. Distributed System illustration [32].

Some of the main areas of the distributed systems are listed below [30]:

- Telecommunication networks:
    - Telephone and cellular networks.
    - Computer networks; Internet.
    - Routing algorithms.
    - Wireless Sensor Networks.
- Network applications:
    - World Wide Web and peer-to-peer networks.
    - Interface Protocols employing IEEE1553.
    - Multiplayer games and virtual reality communities.
    - Distributed databases and management systems.
    - Distributed information processing systems
        - Banking systems.
        - Airline reservation systems.

- Real-time process control:
    - Aircraft control systems.
    - Industrial control systems.
- Parallel computation:
    - Scientific computing
    - Distributed rendering in computer graphics.

A Real-time System has two major constraints:

- Successful execution of each command,
- Execution of each command before deadline.

Moreover, the real-time systems can be divided into two groups:

- Hard RT: eg. Aircraft control networks,
- Soft RT: eg. Online multiplayer gaming networks.

2.3 Distributed Real-Time Ethernet Systems

Ethernet has become one of the most popular data networks in recent years. The current Ethernet standards most frequently support up to 1 Gbps of data transmission, which is way higher than DeviceNet (500 Kbps) [31] and ControlNet (5 Mbps) [31]. Ethernet is much easy to maintain and upgrade since it is compatible with its older versions. In addition, it is inexpensive with respect to other network types.

The most important foible of Ethernet in real-time systems is its collision-proneness. The collision occurs when two or more devices try to transmit at the same time. As the utilization of the network increases, i.e. the number of devices connected or the network load increases, the probability of collision dramatically increases. If the collisions are somehow eliminated, with respect to its speed and maintenance ease, Ethernet is the most suitable and efficient network type.

In today's RTE protocols, RT Bandwidth is allocated statically. Hence, due to the operating conditions of the application, except for the FTT-Ethernet [3] which has a master-slave architecture, the dynamically-changing demands of automation systems are not taken into consideration [3], [19]. This is a problem for RTE systems that have hard RT demands.

There are various communication protocols that support RTE [5], [6]. Even though in [7] and [8], soft RT performance is provided using traffic smoothing, this is not suitable at the device level. Because at the device level, hard RT guarantees are required.

Several approaches exist; one of them is to use the full-duplex switched Ethernet with a prioritization scheme similar to Ethernet/IP (EIP) [9]. With this approach, an extra parameterization is made and a possible complicated scheduling in the switch [10], [11] is faced but the collision is vanished. Moreover, the existence of switches add the network some extra timing delays, decreasing the hard RT performance [12].

A majority of automation systems require synchronization between devices and hence employ synchronization protocols, such as IEEE 1588 [13]. In addition, the synchronization precision is another issue in switched network designs. If the devices are interconnected via several switches cascaded, the end-to-end delay guarantees and precise synchronization is tough [14]. Some industrial protocols such as ProfiNet [15], employ special hardware to handle the synchronization problems but this enhances the cost of the system.

In some protocols, there are interface layers that are responsible for the controlling or shaping the traffic transmitted to the Ethernet layer; thus, avoiding collision manually. Some of these protocols are FTT-Ethernet [3], Ethernet Powerlink (EPL) [16], Time Critical Control Network (TCNet) [17] and Ethernet for Plant Automatization (EPA) [18]. But, in EPL and FTT-Ethernet, the IL is a slave in the master-slave configuration. Similarly, the TCNet has a token-passing mechanism and in EPA, TDMA is employed. If these protocols are investigated to be used in industrial automation systems, since the automation systems are distributed, the master-slave operation is useless. In addition, the maximum utilization in EPL [6] is observed to be 25%, which is insufficient. Likewise, the tokens take time to be passed between devices; the token-passing mechanism slows the network speed. The TDMA uses exclusive and static allocated bandwidth for each device.

Today's RTE protocols use static allocation of the RT bandwidth or static configuration of possible transmission times to provide real-time supports on Ethernet. But, these protocols require that the demands of the system do not alter due to the state of the application [3], [19]. The sole protocol that adapts the dynamically changing demands of the application is the master-slave protocol, FTT-Ethernet [3].

The previous works on protocols that can be parameterized dynamically according to the communication needs of automation system networks are given in [22], [19]. A detailed comparison between RTE protocols is given in Table 2-1 [33].

Table 2-1. Some Other Real Time Ethernet Protocol Proposals [33].

Protocol	Academic / Industrial	Approach	Disadvantages	Max Number of Nodes Supported	Theoretical Limit	Topology	Synchronization Support	Applications	Links
MODBUS/RTPS (Real-Time Publisher Subscriber)	Industrial (Schneider Electric Company)	Publisher-Subscriber & Composite State Transfer	Depends on master slave operation, so no peer-to-peer no sophisticated communication possible	254	Not available	Master slave	Can support IEEE-1588	field bus	http://www.modbus.org/
Ethernet Powerlink	Industrial (Bernecker & Rainer Company)	TDMA with polling	Master-slave based operation (no distributed synch) and homogeneous solution	240	Not available	Any topology can be formed by using hubs.	Master based IEEE 1588 (jitter < 1us can be reached)	fieldbus	http://www.automation.com/pdf_articles/Introduction_to_ETHERNET_Powerlink.pdf
Ethernet/IP	Industrial (Rockwell Automation)	Offers producer-consumer services	Hypothetical (everything is done according to a schedule, no delay is handled)	-	Not available	Based on Common Industrial Protocol (CIP)	IEEE-1588	field bus	http://www.odva.org/
PNET	Danish national committee	Master-slave (up to 32	Master dependent operation, no	125	300 transactions per second	Multi-Master-slave bus	No synchronization, master based	field bus	http://www.p-net.dk/

		masters)	distributed synchronization, low theoretical data rates.		76,800 bps	topology.	asynchronous data transfer		
Vnet/IP	Industrial (Yokogawa Electric Corporation)	Uses UDP for real-time traffic	Not a deterministic protocol. Just uses UDP for real-time messages no control	-	1 Gbps	Client-Server OR Publisher-Subscriber	10ms scheduling cycle for real-time traffic.	field bus	http://www.yokogawa.com/
TCnet	Industrial (Toshiba)	Virtual-circuit for real-time traffic	Low speed synchronization based on broadcast messages	-	Not available	Publisher-Subscriber	-	field bus	-
EPA (Ethernet for Plant Automation)	- (Chinese proposal)	MAC layer TDMA based	Scheduling is done via periodic message broadcast.	-	Not available	Publisher-subscriber topology	-	field bus	-
PROFINET CBA	Industrial (Siemens – PROFIBUS)	-	Homogeneous solution. No standard nodes are allowed	Unlimited	Not available	-	-	field bus	http://www.profibus.com/
SERCOS (Serial Real-		Master-slave	Switch use is prohibited.	254	Not available	Daisy-chain or	Master based synchronization	Motion control bus	http://www.sercos.com/

Time Comm. Interface)			Homogeneous solution.			Ring	messages.	(not and does not compete with field bus)	
EtherCAT	Industrial (BeckHoff & EtherCat Technology Group)	Master-Slave mode of operation	Homogeneous Solution. Modified Ethernet topology		Not available	Line, tree or star	IEEE 1588 based distributed clocks (accuracy < 1us)	field bus	http://www.ethercat.org/
FTT Ethernet	Academic (Electronic Systems Laboratory at IEETA /University of Aveiro)	TDMA based. For both real-time and non-real time traffic	Layer over MAC Compatible with standard Ethernet	-	Not available	-	-	field bus	-

CHAPTER 3

DYNAMIC DISTRIBUTED DEPENDABLE REAL-TIME INDUSTRIAL COMMUNICATION PROTOCOL (D^3RIP)

Dynamic Distributed Dependable Real-Time Industrial communication Protocol (D^3RIP) family, introduced in [26] as a new Real Time Ethernet protocol, provides a collision-free industrial communication with an efficient use of the bandwidth since the behaviour of the D^3RIP adapts itself due to the changing specifications of dynamic systems. This adaption mechanism is based on the previously known communication requirements, such as the timing and scheduling of messages and needs, that is broadcast to every node in the system, dynamically. Hence, it is possible to provide Real-Time guarantees and dependability.

As the system, i.e. the control application, behavior is assumed to be known from the system design; D^3RIP allocates the bandwidth due to the needs of the application, so that the remaining bandwidth could be present for non-real-time communications.

D^3RIP is a framework for the distributed shared-medium communication architectures, that is applicable to similar environments including the well-known protocols IEEE 802.11 and WirelessHART.

The framework dwells mainly on the real-time communication of distributed devices that control the system through a network based on Ethernet that adapts to the communication needs of the automation structure on the lower levels, i.e. the machine, the cell and the subsystem levels [26], provided in Figure 3-1.

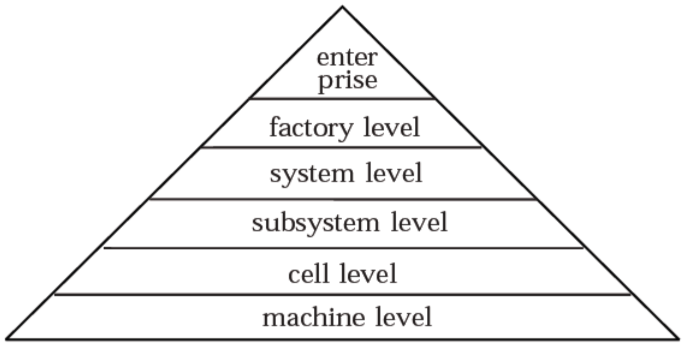


Figure 3-1. Automation Hierarchy [26].

In general, the needs include various types of messages to be transmitted, so the requirements change dynamically. For instance, in the system, there can be signals with real-time restrictions. Some of these signals could be periodic with small delay and jitter constraints such as position control signals, as well as sporadic or event-based signals with limited delay permissions, such as limit switches. In addition, there can be non-real-time signals for that time is not critical. Maintenance or diagnosis related data signals are also members of nRT traffic.

3.1 Example System and Communication Scenario

The workcell given in Figure 3-2 is modified from a manufacturing system in [23] and [24]. The devices are as follows:

Robot (R): moves the arm of the robot

Conveyor (C): carries the parts

Painting Device (PD): paints the manufactured parts using spray paint guns

On top of these devices, there also are 4 controller devices. These are DevR, DevC, DevPD and DevS. Each controller device except DevS is responsible for the corresponding local device, i.e. DevR, DevC and DevPD control the actions of Robot (R), Conveyor (C) and Painting Device (PD), respectively. DevS is responsible for the coordination of the devices.

In the framework, the model of the operation is given in an event-based fashion; each action is defined by an event. These events constitute commands for DevS, notifications given by controller devices, and the local discrete actions or start/stop actions of the closed-loop control processes performed by them [26].

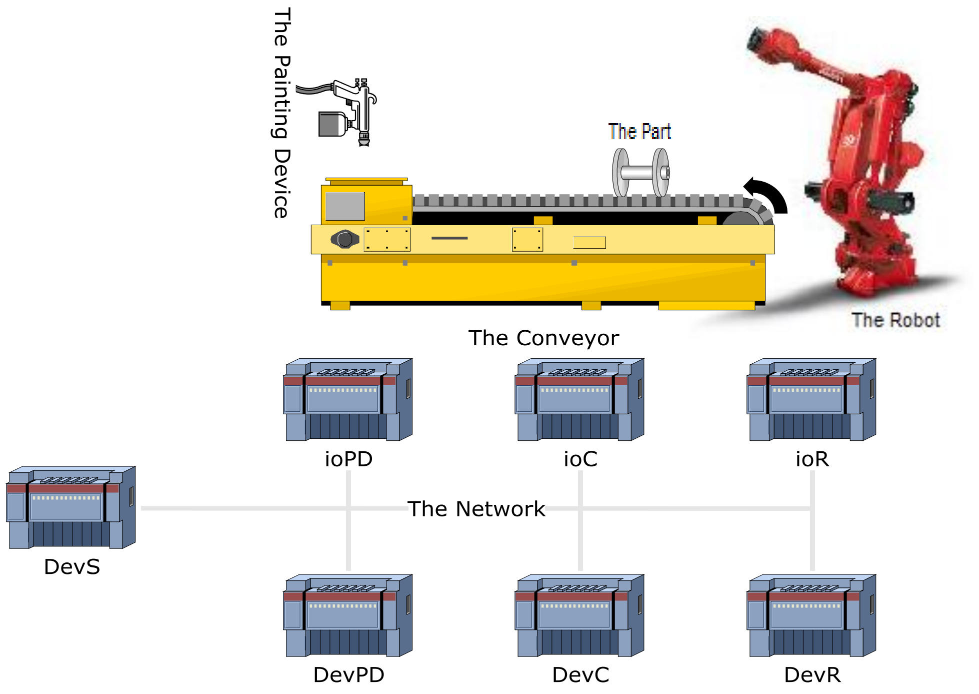


Figure 3-2. Workcell: Robot-Conveyor-Painting Device.

In the beginning, the system is in initial state that is the robot has put no parts on the conveyor belt, yet.

DevS sends an `mvC` command to the network and DevR receives it and powers up the robot to place a part on the conveyor, which is the `sC` event in DevR. Then, DevR executes the robot's closed loop control action and processes the position information in `posR` (position) and `actR` (actuator) signals, before the conveyor is reached by the robot, the stopping signal `stpR`, and the arrival signal `arC`, informs the DevS about the arrival of the part. Similarly, the signals `mvI`, `sI` and `arI` refer to the same actions in the reverse direction.

When `mvPd` signal is received by DevS, the conveyor is responsible for the transportation of a part into the Painting Device (PD), by the signal, `sPD`. Whenever the signal `posC` which indicates the position of the conveyor signals the arrival to the PD, the conveyor stops (`stpC`) issuing a notification action of its arrival to DevS (`arPD`). This is the one-way motion. The signals during the return of the part from the PD to the Robot are `mvR`, `sR` and `arR`.

Moreover, the PD could be operational, i.e. it could be switched on and off due to its current operation. If it is to be set on, DevS transmits a `PDon` signal, initiating the painting process by `iPD` immediately after the cover of the PD is locked (`lPD`). Then, it unlocks its cover (`ulPD`) whenever it finished the painting action (`fPD`). Finally, the Painting Device is switched off, notifying DevS by the `PDoff` signal. During process, the Painting Device carries out the position control for the spray gun via `posPD` and `actPD`.

Basically, for overall communication, the position signals, arrival notifications, commands must be shared between devices. Moreover, the local controller devices must acquire signals from the plant components; robot, conveyor and painting device and must apply actuators to them, using their I/O interfaces, `ioR`, `ioC` and `ioPD`.

First, in industrial automation networks, there exist various signals of different RT requirements, i.e. some signals (periodic sensor or actuator signals – `posR`, `actR`, `posC`, `posPD`, `actPD`) might be delivered in hard RT messages, while some others (event based notifications or commands in DevS, local start/stop signals – `sC`, `stpR`, `sI`, `sPD`, `stpC`, `sR`, `lPD`, `ulPD`, `iPD`, `fPD`) may fit into sporadic messages, that have soft RT needs.

In addition, all of the devices and components of the workcell are supposed to deliver diagnostic data via nRT messages.

The logical behaviour of the sample system in an automated painting factory is depicted in Figure 3-3.

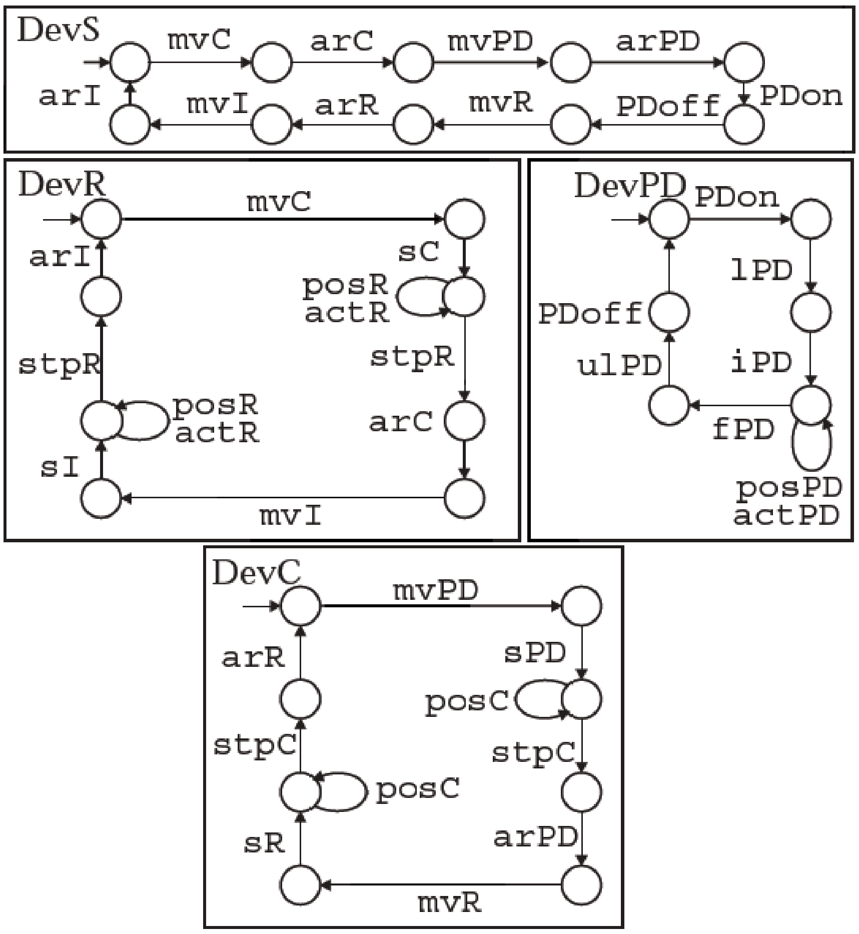


Figure 3-3. Illustration of the Behaviour of the workcell [26].

When we examine the workcell given above, as long as the components are known from system design never to operate at the same instant, it could easily be observed that the static allocation of the bandwidth would be overly conservative. The effective allocation algorithm should allocate the bandwidth only to the active device; which cannot be provided by current static RTE protocols.

For instance, considering a conventional industrial network, assume that all hard real-time signals have a deadline of 5ms and the soft real-time signals have a deadline of 10ms. As long as the allocation of the bandwidth in conventional Ethernet is static, we have to allocate the required bandwidth according to the worst case scenario, in terms of communication channels. Assume that the channel requirements of such a system are given in Table 3-1.

Table 3-1. Channel Requirements of a Sample System [26].

	Required number of channels		
	Channel periods		
	5ms	10ms	Related Signal
DevS	-	1	soft RT signals
DevR	2	1	`posR`–`actR`-sporadic signals
DevC	1	1	`posC`-soft RT signals
DevPD	2	1	`posPD`–`actPD`-sporadic signals

If each signal is transmitted in a single Ethernet Frame of minimum size, 576bits, the net RT bandwidth of

$$4 * \frac{576\ bits}{10\ ms} + 5 * \frac{576\ bits}{5\ ms} = 0.806\ Mbps$$

is needed to handle the requirements.

On the other hand, if the medium access could be shifted among devices by the command from DevS, it would be possible to achieve a relatively much less bandwidth, but today’s static Real Time Ethernet protocols cannot actualize such allocation.

3.2 D^{3}RIP Protocol Stack

D^{3}RIP (Dynamic Distributed Dependable Real-time Industrial Protocol) framework's software architecture includes two different protocol layers operating upon the conventional MAC layer as presented in Figure 3-4. The IL runs the conventional TDMA Protocol for RT and nRT traffic, where each CL coordinates the ownership of the current RT slot, due to distributed computations, as previously claimed.

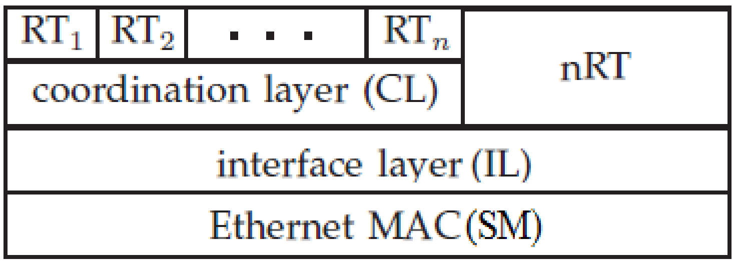


Figure 3-4. D^{3}RIP Software Architecture [26].

The operative details and functionalities of IL and CL will be examined in detail in the further sections. The formal definition of the model we use, TIOA first introduced in [21], will be presented in the first section of the next chapter.

The protocol, D^{3}RIP is presented to run as a dynamic and dependable protocol operating on Real Time Systems via Ethernet. The dependability proposed here presents a collision-free protocol. Hence, the protocol should be able to adapt the changing requirements of the system while keeping its collision-free structure, as well.

The D^{3}RIP Protocol Family is composed of a Shared Medium (SM) which is the MAC layer, an Interface Layer (IL) that implements TDMA on the SM and a Coordination Layer (CL) enabling instantaneous allocations of Real-Time messages on the TDMA scheduling via ILs.

The system is modelled using TIOA Formalism as previously stated. The overall system structure is provided in Figure 3-5.

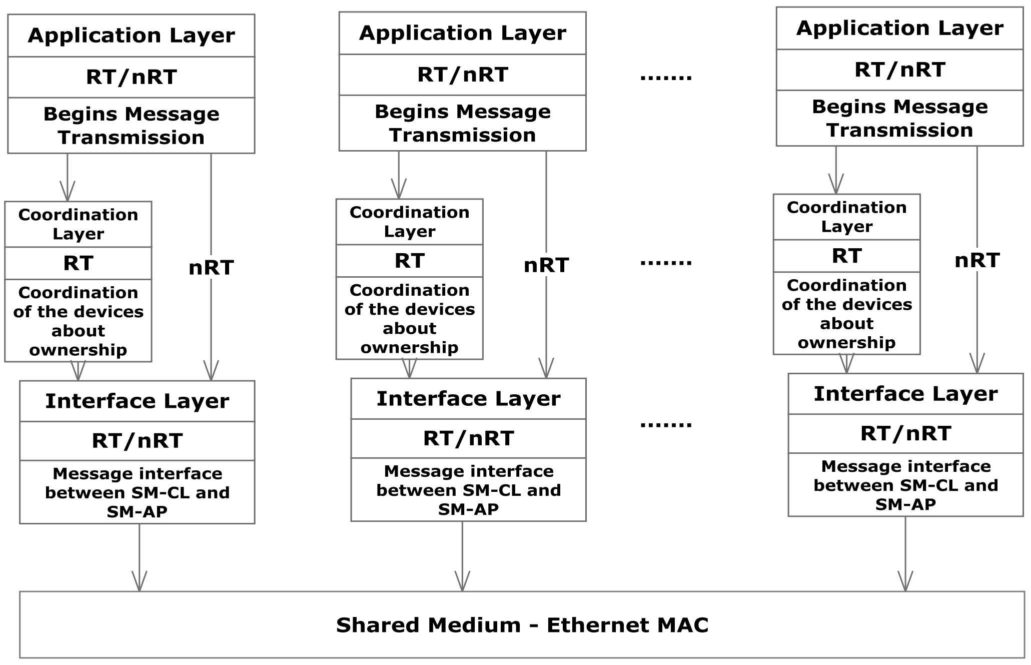

Application Layer
RT/nRT
Begins Message Transmission
Coordination Layer
RT
Coordination of the devices about ownership
nRT
Interface Layer
RT/nRT
Message interface between SM-CL and SM-AP
....
Shared Medium - Ethernet MAC

Figure 3-5. The Structure of the Framework Proposed for a Distributed Real-Time System [26].

The CL in each device in the automation makes the distributed computation and a single device is assigned for transmission for each time slot of IL. On the other hand, the remaining bandwidth, i.e. the time slots that are not used in RT communication, are allocated for non-real-time message (nRT) transmissions.

The framework proposes interface layers and coordination layers, and some distributed computations in distributed devices; not a particular real-time protocol is proposed. Thereby, the protocol family is general.

3.3 TIOA Formalism

The Timed I/O Automata (TIOA), whose states alter by discrete trajectories or by trajectories evolving over time, is introduced for the analysis and modeling of RT systems in the mathematical frameworks of [21] and [25]. Each TIOA belongs to a component in the system and the composition of TIOA describes the behavior of a system with respect to time. The timed systems are considered with components changing instantaneously and as well as continuously in time. So, TIOA is used for the description of the framework [26].

For the representation of the states of a TIOA, a variable set is used. Each variable has two types: a static type and a dynamic type. The static type of a variable defines the possible range of values; whereas the dynamic type is responsible for holding the possible evolutions of time for the variable. Depending on the dynamic type of variables, they are either analog or discrete. The dynamic types of discrete variables are in step function forms and those of analog variables are assumed to be piecewise continuous functions.

A function *val(X)* for a set *X* of variables is called a *valuation* of *X* if it matches each variable of *X* with a value in the static type of the variable, in turn, a function is called a *trajectory* if it is a mapping of time instants into [0, *val(X)*] time interval, so that the time evolution of the variables of *X* will be described.

A TIOA is defined as an eight-tuple $A = \ X, Q,\ Q_0, I, O, H, D, T$

- The variable set X
- The state set Q $\subseteq val(X)$
- Non-empty set $Q_0 \subseteq Q$ of start states
- I is the input action set describing inputs from external world, O is the output action set representing outputs, H is the internal action set with all actions given as $I \cup O \cup H$.

- $D \subseteq Q \times A \times Q$ is the discrete transition set, for the $(q, a, q') \in D$ transition, $q \overset{a}{\quad} q'$ is used. Each discrete transition can be *input*, *output* or *internal* according to the action which labels the transition.
- T is the trajectory set with $\tau\ \ t\ \ \in Q\ \forall\tau \in T$ with t in the domain of τ.

For convenience, the *external action set* which is in interaction with the external world is given as $E = I \cup O$, with the *locally controlled actions* under the control of A is defined as $L = O \cup H$.

CHAPTER 4

D^{3}RIP SIMULATOR

The D^{3}RIP Simulator is needed to simulate the D^{3}RIP system behavior, instead of running a complicated and slow test in the actual system. Namely, the simulator runs identically to the actual system, hence, making it possible to examine the system states at any time instant for any protocol modifications.

It takes a lot of time to make arrangements for the tests in the real system and also it is not possible to monitor every single variable in the automata. The overall system modeled, monitored and analyzed easily. As a result, new ideas and changes in the D^{3}RIP family will be integrated much more easily and inexpensively at both time and cost.

In the previous sections, it is mentioned that the D^{3}RIP protocol stack is composed of a common Shared Medium, the Interface Layer (IL) and the Coordination Layer (CL) of each device, in co-operation with the Application Layer (AP). Hence, in the D^{3}RIP simulator, each of these instances has to be simulated.

The simulator incorporates the class definitions of SM, IL and CL. In addition, as given in [26], the actions and transitions are implemented.

First of all, the actions are to be handled as events, such that, in the actual system, if an action is supposed to be active at an instant, the D^{3}RIP simulator is responsible of the creation of that action as an event and its service according to the D^{3}RIP Protocol Stack.

We considered several different models for the simulator. The time could be polled or the actions could be assumed to occur periodically. However, our D^{3}RIP simulator is designed to simulate the dynamic adaptation of our D^{3}RIP Protocol Stack to the changing needs of the industrial automation systems, hence; we cannot make early predictions and assume periodicity. In addition, polling the time or having an infinite loop in the simulation would cause our D^{3}RIP simulator to behave in a time slotted manner. But, since the real time functioning automation system is not time slotted, we cannot make such assumptions. Hence, the D^{3}RIP simulator is designed as an event-based simulator.

There is a Priority Queue (PQ) of Events. The events are created and pushed into the PQ. The D^{3}RIP simulator then serves the event with the highest priority.

Each event is identified by its occurrence time, the device id that the corresponding action belongs to and its type. Hence, once the D^{3}RIP simulator pops an event out of the PQ, it will be informed about the details of the event it will serve. The priority order is defined first by its occurrence time, then its type, finally the device number that the event occurs at. Hence, the events are queued with priorities.

The Priority Queue of Events is depicted in Figure 4-1.

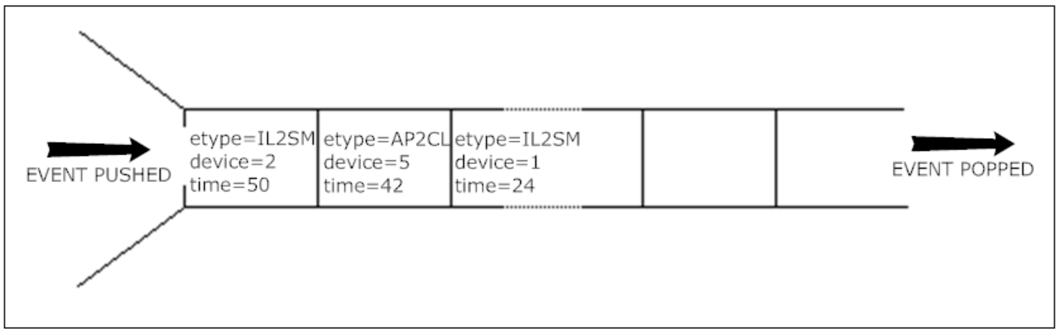


Figure 4-1. PQ Structure – Events Are Enqueued with respect to their Occurrence Times.

The next event that is invoked by the current one is pushed into the PQ of events.. During some events, it will be observed that more than one event will get pushed into the PQ, where in some, no event will.

It is required to note that, according to the D^{3}RIP Stack design in [26], in every time slot, an `UPDATE` event is pushed into the PQ for the update of the state of the system. Hence, the PQ never becomes empty.

4.1 Abstractions and Assumptions in the D^{3}RIP Simulator

During the simulation of the system, there are some assumptions and abstractions made.

The assumptions are listed below.

- *Line delays*: The real system is connected through 100Mbps Ethernet cables. The messages transmitted is standard and of size 150B = 1200bits. In a 100Mbps line, 1200 bits takes 12µs to be transmitted. In the simulation, `m.length` is taken as 12µs. Hence, the time `now+m.length` becomes `now+12` (in µs), as in real systems. The SM experiences no line delays, so, the line delay has been created as the message length for more precise results.

- *Synchronization latencies*: In the devices in the real implementation, RT Linux system is operating and the machines are synchronized at less than 500µs. Hence, the timings of the events regarding the SM might differ at this amount. The simulator assumes that all devices are synchronized. As a result, no pings or IEEE 1588 Synchronization packets are transmitted or received. In the D^{3}RIP Simulator, exactly the same as the synchronization packets, the nRT packets are being transmitted at the same time instants as they are in the real system, but they are not treated as synchronization packets or any other packet than regular nRT packets, since it is assumed that all the devices are already synchronized.

- *Transmission Errors*: In the devices, there might be transmission errors or line failures, but these are not considered in the simulation.

- *Device Related Lags*: The devices that operate in the automation network might cause themselves delays due to internal calculations and/or any other computer related lags such as the stack delays.

- *Guard Periods*: The guard periods [28] and [29], i.e. the time intervals that a device waits before transmission until it is guaranteed in the real system to be in the same time slot synchronously with its neighboring devices, are not taken into account since there is no synchronization error in the simulation.

- *ISR Delays*: The RT Linux operating system in the real implementation is not purely Real-time. Hence, it supports RT up to a limit. One consequence of this lack is in the interrupt service routine calls. The OS provides an ISR service with a 100μs guarantee. This boundary is calculated as 25μs on the average. As a result, the experimental and simulation results are possible to be at least that much in difference.

- *Timer delays*: The real implementation uses timers to call the ISR causing extra delays, which are not present in the simulator.

- *NIC Delays*: The Network Interface Card adds additional delays to the communication, the simulator neglects it.

In the following sections, having these abstractions and assumptions, the SM, IL and CL implementations are explained.

4.2 Shared Medium (SM)

In this subsection, the Timed I/O Automata of the shared medium of a generic broadcast channel in networks that the communication is based on a shared medium is presented.

4.2.1 Shared Medium Operation and TIOA Model

We know that the conventional Ethernet employs a shared medium, called the MAC Layer, in which CSMA/CD algorithm runs. But in our framework, such an algorithm is not applied since only a single device is guaranteed to have access to the SM. Thus, the SM only has to get the message transmitted and broadcast it on the right time. Besides, it will check the collision situation, i.e. if a message is held at the time another message transmission begins.

The SM operation is given in Figure 4-2 [26].

```
TIOA $SM(dNumber, M)$ where $dNumber \in \mathbb{N}$

Variables $X$
  $\texttt{mess}^{d} \in \mathcal{M}$ (empty)
  $\texttt{coll}^{d} \in \mathbb{B}$ (false)
  $\texttt{next}^{d} \in \mathbb{R}$ (0)
  $\texttt{now}^{a} \in \mathbb{R}$ (0)

Actions $A$
  input $\text{IL2SM}(m)_i,\ m \in M,$
  $1 \le i \le dNumber$
  output $\text{SM2IL}(m),\ m \in M$

Transitions $D$

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

output $\text{SM2IL}(m)$
  precondition:
  $\texttt{now}^{a} = \texttt{next}^{d}$
  effect:
  $m = \texttt{mess}^{d}$
  set $\texttt{mess}^{d}$ empty
  $\texttt{next}^{d} = 0$

Trajectories $\mathcal{T}$

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

evolve
$d(\texttt{now}^{a}) = 1$
```

Figure 4-2. TIOA Model of the Shared-Medium [26].

The devices connected to the SM is represented by the variable `dNumber` and the messages flowing in the SM by the variable $m \in M$ each of whose data is characterized by $m.data \in \mathbb{R}^{+}$ and length by where M is the message class.

The variables used whose initial values are given in parenthesis are of two types. For the discrete variables, a superscript "*d*" is used while a superscript "*a*" for analog variables. The variables used in the TIOA of SM are $mess^{d}$, the message that is being transmitted in SM, $coll^{d}$, the Boolean variable indicating collision status, $next^{d}$, the next reception time, now^{a}, which holds the current time evolution, i.e. keeping track of the time spent in message transmission in SM.

There are two actions given in SM, that are IL2SM(m) and SM2IL(m). IL2SM is responsible for the reception of the message transmitted from the interface layer

(IL) of the transmitting device into the SM. Then, in the appropriate time, an SM2IL action will happen in order to broadcast the recently received message.

Note that, IL2SM is declared as an input action for SM, whereas SM2IL an output action. This is there must be an incoming message, so that the SM will get the message by its action IL2SM. Thus, IL2SM is an input action since it is initiated by a message reception from an IL of a device from outside world, i.e. the ILs of connected devices. Unlikely, the IL2SM activates a SM2IL to broadcast the message to the outside world; hence, it is an output action. In Section 4.3, these actions will be observed as output and input actions in the TIOA models of ILs, respectively.

As explained above, the action IL2SM_i takes the message from the IL of device *i*. Then, the $mess^d$ variable is checked if it holds any other message. If not, it means that the SM is idle and ready. If this is the case, the message taken from the IL of device *i* is saved in $mess^d$ with the update of the $next^d$ variable. Otherwise, i.e. if the $mess^d$ is not empty, an overwrite will occur, resulting in collision and the reset of $next^d$. As a result, the messages will be discarded.

Like IL2SM, SM2IL_i occurs when the current time is equal to the $next^d$, which was updated in IL2SM, previously. As SM2IL_i action happens, each device connected to the SM invokes its respective input SM2IL_i actions in it IL in order to receive the message being transmitted in the SM. Then, $mess^d$ is saved in *m* and reset. The next reception time $next^d$ is also reset.

The variable that holds the current time, now^a, evolves with time with a time derivative of 1. That is

$$\frac{d(now^a)}{dt} = 1.$$

In addition, the current time now^a, stops if now^a equals $next^d$ and $mess^d$ is not empty, else is the halting condition.

It is clear that for SM, on input transitions, there are no preconditions present. Furthermore, in IL2SM transitions, the stop condition is invalidated; hence the time can evolve further. Hence, SM is input action and time passage enabling [26].

4.2.2 SM Implementation and Related Class Definitions

The shared-medium has various variables given in the previous subsection; each will constitute a member of SM class variable family. The SM Class definition can be found in Table 4-1.

Table 4-1. The `SM` **Class for the Shared Medium Object.**

SM
int next int now bool collision int dNumber M mess
Represents the class of the Shared Medium

The next reception variable *next*^{d}, the number of connected devices, `dNumber` and current time variable *now*^{a} are declared as integers, with the Boolean *coll*^{d} as bool. The message currently transmitted in SM, *mess*^{d}, is a member of class M.

Unlike the variables, the actions are not declared as members of SM. We implement IL2SM$_i$ and SM2IL actions as events.

The simulation process is not based on polling *now*^{a} and checking if it is equal to some values. The idea is to update *now*^{a} in the beginning of every event and if there is, add the invoked event whose time is increased from the current *now*^{a} at the required amount.

Every time slot of length `dSlot` has 3 portions reserved for data transmission (`data`), protocol related computations (`cmp`) and operation for the upper layer protocol (`rem-cmp`). The structure of time slots is given below in Figure 4-3.

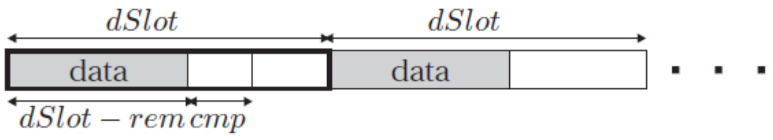


Figure 4-3. The Time Slot Structure [26].

The events are declared as objects of class `Event`. The class has `event` type, time and device as members. The event type is of type `Etype`, which is the enumeration of different event types. Hence, a switchable main loop will be obtained depending on the type of the popped event. The time variable constitutes the time that the event is supposed to happen. The events will be pushed into the `PQ` and sorted with respect to their times. Hence, the event whose time is the closest to the current time will be popped as the next event. The `Event` Class is depicted in Figure 4-4.

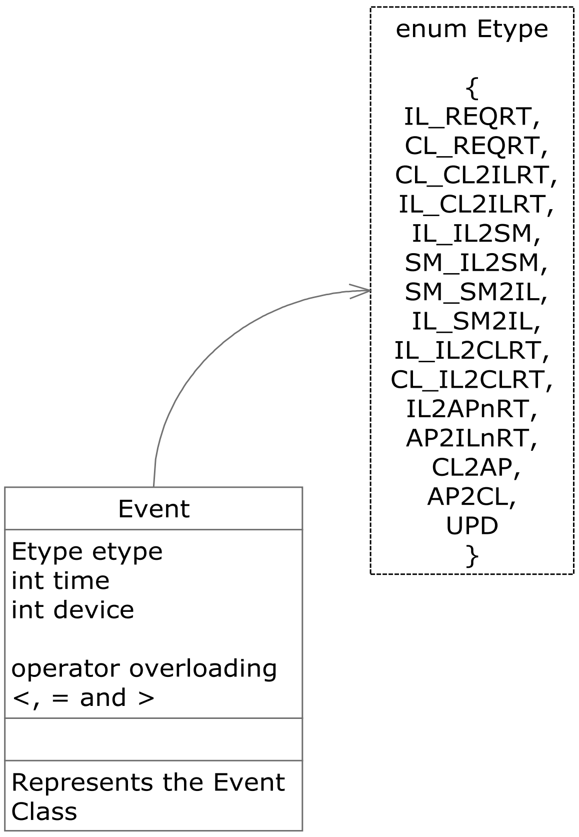


Figure 4-4. `Event` Class for the Event objects.

The brief explanations of the enumerated events are:

- IL_REQRT: The request issued by IL to the CL
- CL_REQRT: The request received by CL from the IL
- CL_CL2ILRT: The response of CL to the request from IL (might contain a message)
- IL_CL2ILRT: The response of CL is received by IL (might contain a message)
- IL_IL2SM: IL transmits message to the SM
- SM_IL2SM: The SM receives the message transmitted by IL
- SM_SM2IL: The SM broadcasts the message it currently holds to IL
- IL_SM2IL: IL receives the message broadcast by the SM
- IL_IL2CLRT: IL forwards the message received from the SM to the CL
- CL_IL2CLRT: CL receives the message forwarded by IL

- IL2APnRT: The application gets the nRT message from IL
- AP2ILnRT: The application puts an nRT message to IL
- CL2AP: The application gets the RT message from CL
- AP2CL: The application puts an RT message to CL
- UPD: The update of the state of the automaton; i.e. includes update of the message buffers, timings, calculations of decision variables, etc.

The events possible to be popped out of the PQ are depicted in Figure 4-5 [29].

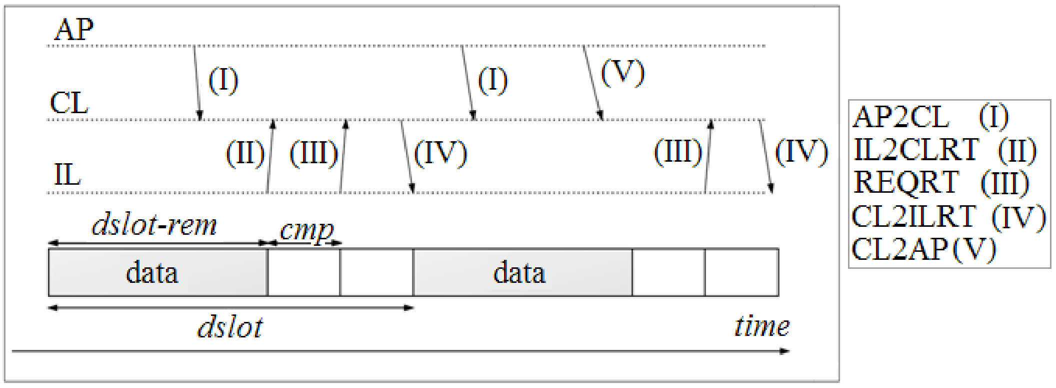


Figure 4-5. Possible Events in a Time Slot [29].

There is only one SM which is common for all connected devices. The system architecture and the SM-IL-CL interfaces can be seen in Figure 3-4. So the common variables should be used and altered by all devices in various events. Hence, in SM, the default constructor constructs the shared-medium with the variables of their initial values; i.e. (now^a, $next^d$, $coll^d$, $mess^d.data/mess^d.length$) = (0, 0, false, 0/0). We shall now investigate the class `M` for message type. As given in Table 4-2, the class `M` has 8 members.

Table 4-2. The `M` Class Definition.

M
int length int data int nodeID int packetID int framenum int frameseq int framelength PAR par
Represents the Message Class

The first seven, `int length`, `int data`, `int nodeID`, `int packetID`, `int framenum`, `int frameseq` and `int framelength`, are integers representing the message length, the data being transmitted, the ID of the node that transmits or receives the message, the number of frames that the original non-real-time packet is fragmented into, the sequence of the frame being transmitted and the length of the frame as a fragment of the packet, respectively. The eighth member is the `PAR par` indicating the protocol parameters of the message, which are of `PAR` class, to be determined later.

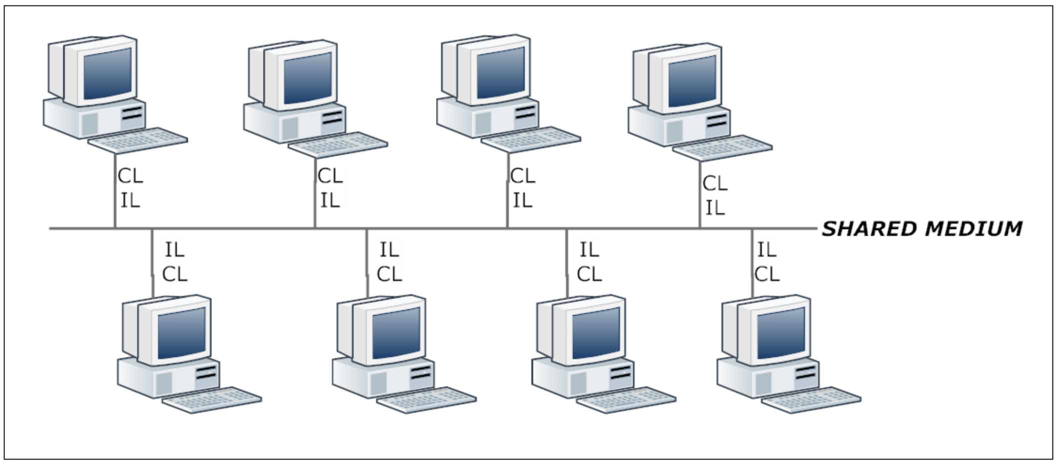


Figure 4-6. The system: SM-IL-CL and the Control Application Running in Devices.

In the event IL2SM$_i$ of SM, as depicted in Figure 4-2, if no collision occurred, that is *mess*^{d} is empty, *next*^{d} is updated. So, we know from the behavior that when *now*^{a} equals *next*^{d}, the precondition of SM2IL$_i$ will be satisfied and the output action SM2IL must happen for all devices, since broadcast messages will be received by all connected devices. This structure is built by pushing an event of type SM2IL for each device, with the event time of $now^a + m.length$ at the end of IL2SM$_i$ reception as the transmission of the message ends *m.length* unit time later than the current time *now*^{a}.

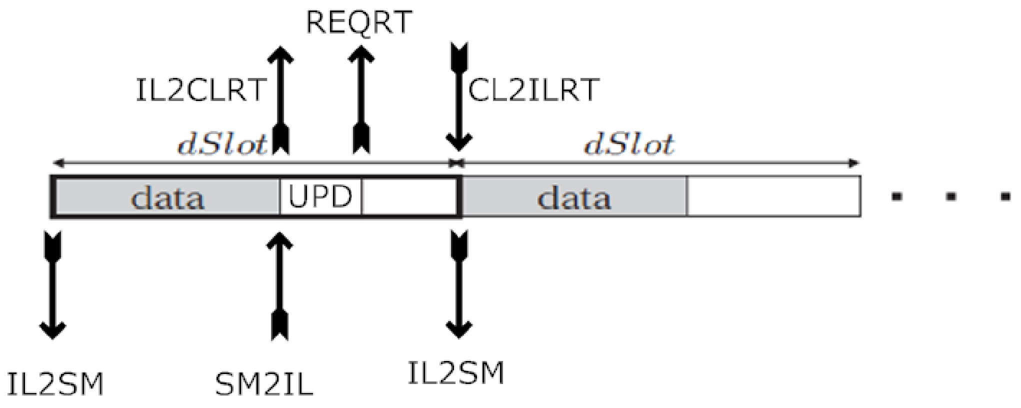


Figure 4-7. Timing Diagram between IL-SM.

In the SM2IL that occurs after *m.length* unit time after the IL2SM$_i$, the message in the SM is copied into *m* and *mess*^{d} is reset using the `reset()` function of class `M`. Later, *m* will be used to deliver the message that was received by SM to the ILs of all devices, as seen in Figure 4-8.

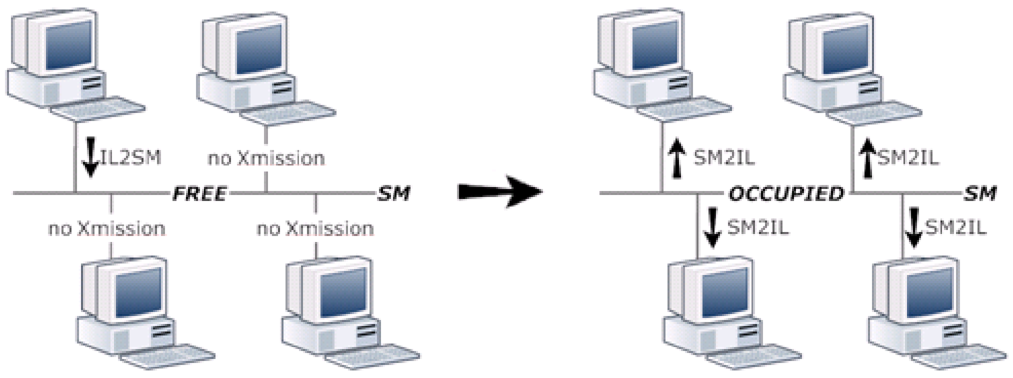


Figure 4-8. IL2SM Transmission and SM2IL Broadcast.

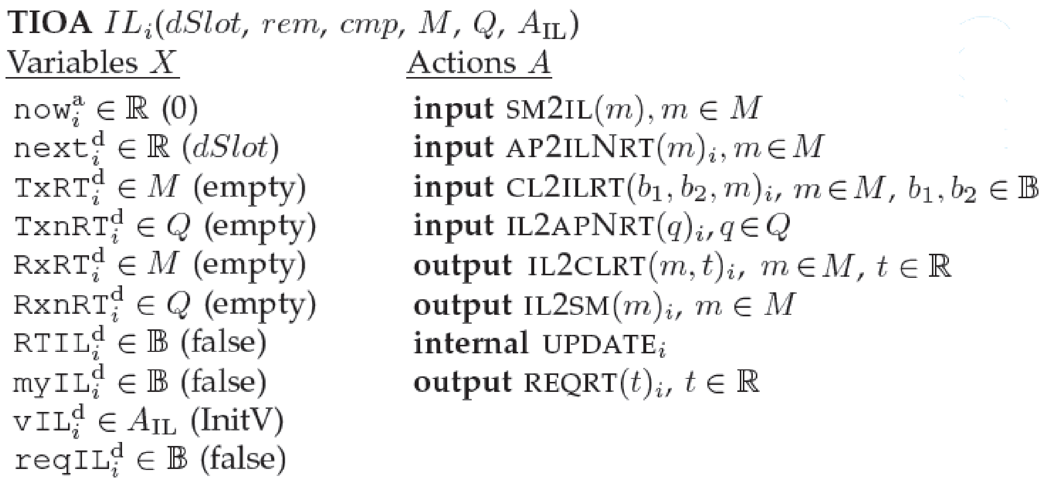
At that instant, the output action SM2IL invokes its respective input action $\text{SM2IL}_{i,}$ in the IL, namely, a SM2IL_i event of time *now*^{a} is pushed into PQ since there is no time spent between the events SM2IL of SM and SM2IL_i of IL of device *i*. This is as mentioned before, due to the input actions being enabled without any precondition. The case is the same in IL. The details of operation and implementation of IL will be examined in the Subsection 4.3.1.

4.3 Interface Layer (IL)

In this subsection, the Interface Layer (IL) protocol family is introduced. As given before, IL is operating on top of the SM as a broadcast channel and under a Coordination Layer (CL), to be discussed later. The operation of IL provides a collision-free communication for both RT and nRT traffic by allocating each time slot to a unique owner based on information that are locally stored or requested from the CL_i for each time slot.

4.3.1 Interface Layer (IL) TIOA Model

First, the TIOA model of IL is presented generically; then, the two types will be analyzed. The TIOA Model definition is given in Figure 4-9.



TIOA $IL_i(dSlot,\ rem,\ cmp,\ M,\ Q,\ A_{\mathrm{IL}})$

Variables X	Actions A
$\mathtt{now}_i^{\mathrm{a}} \in \mathbb{R}$ (0)	**input** $\mathrm{SM2IL}(m), m \in M$
$\mathtt{next}_i^{\mathrm{d}} \in \mathbb{R}$ $(dSlot)$	**input** $\mathrm{AP2ILNRT}(m)_i, m \in M$
$\mathtt{TxRT}_i^{\mathrm{d}} \in M$ (empty)	**input** $\mathrm{CL2ILRT}(b_1, b_2, m)_i,\ m \in M,\ b_1, b_2 \in \mathbb{B}$
$\mathtt{TxnRT}_i^{\mathrm{d}} \in Q$ (empty)	**input** $\mathrm{IL2APNRT}(q)_i, q \in Q$
$\mathtt{RxRT}_i^{\mathrm{d}} \in M$ (empty)	**output** $\mathrm{IL2CLRT}(m, t)_i,\ m \in M,\ t \in \mathbb{R}$
$\mathtt{RxnRT}_i^{\mathrm{d}} \in Q$ (empty)	**output** $\mathrm{IL2SM}(m)_i,\ m \in M$
$\mathtt{RTIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	**internal** UPDATE_i
$\mathtt{myIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	**output** $\mathrm{REQRT}(t)_i,\ t \in \mathbb{R}$
$\mathtt{vIL}_i^{\mathrm{d}} \in A_{\mathrm{IL}}$ (InitV)	
$\mathtt{reqIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	

Figure 4-9. TIOA Model Definition of the Interface Layer [26].

For each device *i*, the IL functionalities are parameterized by several variables. These variables are $dSlot \in \mathbb{R}^+$ that represents the slot duration, the First-In-First-Out type queue, Q of messages, time intervals, $rem \in \mathbb{R}^+$ and $cmp \in \mathbb{R}^+$, Booleans $RTIL^d$, $myIL^d$, $reqIL^d$ and the message type M. In addition to the variables presented, we also use an abstract data type A_{IL}, each of whose choice will select a type of the interface layer protocol family. This abstract data type will be discussed in the Subsection 4.3.4.

As depicted in Figure 4-3, each time slot is of length *dSlot*. Each slot can be divided into two time intervals: for data transmission, and the remaining. *rem* is the interval remaining from the message transmission, hence, used for other purposes. *cmp* is a part of *rem*, which is reserved for protocol related computations of the interface layer, and the rest of *rem* consists of the time for the operation of the upper layer protocol.

Similar to SM, the only analog variable now_i^a holds the time evolution of each IL, IL$_i$. The variables except now_i^a are discrete. $next_i^d$ holds the beginning of the next slot as well as the end of the current one, $TxRT_i^d$, $TxnRT_i^d$, $RxRT_i^d$, $RxnRT_i^d$ are the transmit and receive buffers. The RT buffers are single-message buffers that hold the active message, where the nRT buffers are FIFO queues of type `Q` that buffers messages of type `M`.

For the IL of device i, the variable $RTIL_i^d$ shows the type of the message being transmitted, that is RT of nRT. If $RTIL_i^d = true$, the message is RT. Similarly, $myIL_i^d$ indicates if the current time slot is assigned to device i. If the assignment is to be done by the upper layer, $reqIL_i^d$ is $true$ to issue a request to the CL in order to calculate $RTIL_i^d$ and $myIL_i^d$. The abstract data type class object $vIL_i^d \in A_{IL}$ contains extra information. The updates of $RTIL_i^d, myIL_i^d$ and vIL_i^d are performed by f_{RT}, f_{my}, f_{req} and f_{upd}, respectively.

In the TIOA model of IL, an internal action called UPDATE$_i$ is introduced. The UPDATE action is responsible for the computation of the state of the protocol for the next slot. This computation takes place after the transmission of the message in the current time slot is finished. The computations include the update of $vIL_i^d$and the computation of $reqIL_i^d$ according to the updated vIL_i^d. The preconditions of the UPDATE$_i$ are that now_i^a is at the end of the time interval reserved for data transmission or in the beginning of the time interval remaining from data transmission, and the RT receive buffer $RxRT_i^d$ is empty. Then, if $reqIL_i^d$ is false, then $RTIL_i^d$ becomes false and $myIL_i^d$ calculation is made locally by f_{my}. Regardless of $reqIL_i^d$, $next_i^d$ is incremented by `dSlot`, i.e. updated.

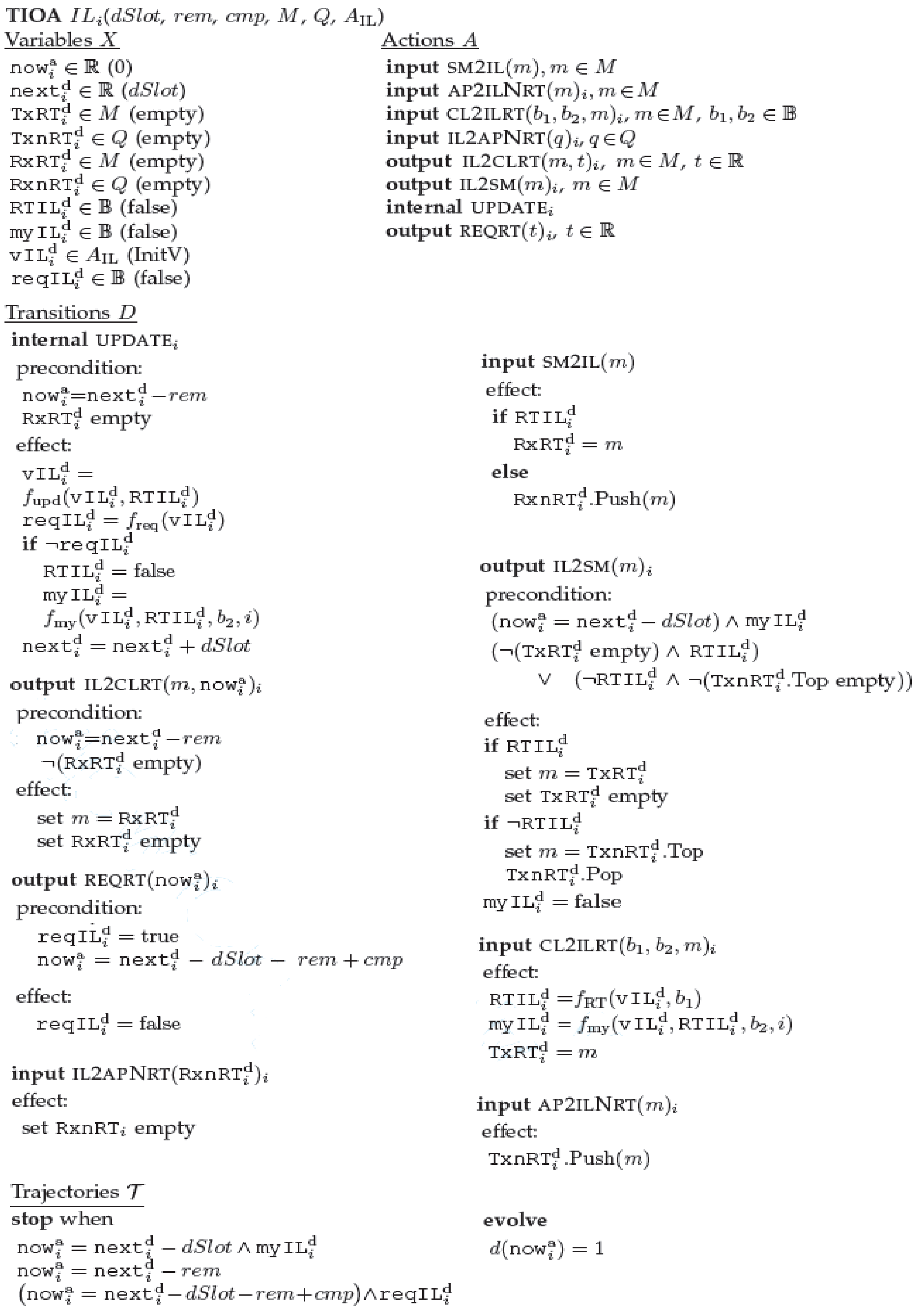
Here, we examine the output action, REQRT(t)$_i$. According to the preconditions, we see that REQRT(t)$_i$ can only happen after the $reqIL_i^d$ returns true in UPDATE$_i$ at the right interval. As $next_i^d$ is updated in the UPDATE$_i$, in precondition checks of the upcoming actions, the updated version of will be used. Hence, the cmp interval in the current time slot now becomes $next_i^d - dSlot - rem + cmp$.

When now_i^a is in the beginning of cmp interval, the time has come for REQRT(t)$_i$. If $reqIL_i^d$ turned true, at that instant a REQRT(t)$_i$ will be called. In that case, the $myIL_i^d$ calculations are not made locally, but instead, a request for the calculations to the upper layer is issued. It provides the current time now_i^a to CL and requests an RT message with the ownership information of the next slot. The response is delivered from CL in CL2ILRT$_i$ (`b1`,`b2`,`m`)$_i$. The `b1` and `b2` are used for the determination of the new values of $RTIL_i^d$ and $myIL_i^d$. The argument `m` is the RT message in CL if present.

The message broadcast from a device to every device begins with the transmission of a message from IL$_i$ to SM in the output transition, IL2SM(m)$_i$. The action in device i whose $myIL_i^d$ is true, takes place at the time when now_i^a equals the beginning of the next time slot, that is $next_i^d - dSlot$ since $next_i^d$ was updated before. The type of the message transmitted is based on the valuation of$RTIL_i^d$. If true, then the message is an RT message. On the other hand, the input transition SM2IL(m) is responsible for the message reception and its storage in $RxRT_i^d$ or $RxnRT_i^d$.

We select RT messages of length less `m.length` less than the transmission window, i.e. $next_i^d + dSlot - rem$. For such selection of RT messages, upon reception, each RT message is immediately forwarded to the CL$_i$ of the device by the transition IL2CLRT($RxRT_i^d$, now_i^a).

In addition, as presented in Figure 4-11, the input transitions AP2ILnRT(m)$_i$ and IL2APnRT($RxnRT_i^d$)$_i$ give access to the application layer (AP) to the transmit and receive buffers of the IL at any time instant, since the application determines when to transmit or get the nRT messages. But it must be noted that the case is different for RT messages. The IL2CLRT$_i$ is an output transition since IL controls the RT message transmission from IL to CL at the appropriate time slot and time interval. The whole TIOA Model of IL is presented in Figure 4-10 [26].



```
TIOA $IL_i(dSlot,\ rem,\ cmp,\ M,\ Q,\ A_{\mathrm{IL}})$
Variables $X$
  $\mathtt{now}_i^{\mathrm{a}} \in \mathbb{R}$ (0)
  $\mathtt{next}_i^{\mathrm{d}} \in \mathbb{R}$ ($dSlot$)
  $\mathtt{TxRT}_i^{\mathrm{d}} \in M$ (empty)
  $\mathtt{TxnRT}_i^{\mathrm{d}} \in Q$ (empty)
  $\mathtt{RxRT}_i^{\mathrm{d}} \in M$ (empty)
  $\mathtt{RxnRT}_i^{\mathrm{d}} \in Q$ (empty)
  $\mathtt{RTIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)
  $\mathtt{myIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)
  $\mathtt{vIL}_i^{\mathrm{d}} \in A_{\mathrm{IL}}$ (InitV)
  $\mathtt{reqIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)

Actions $A$
  input SM2IL$(m), m \in M$
  input AP2ILNRT$(m)_i, m \in M$
  input CL2ILRT$(b_1, b_2, m)_i, m \in M,\ b_1, b_2 \in \mathbb{B}$
  input IL2APNRT$(q)_i, q \in Q$
  output IL2CLRT$(m, t)_i,\ m \in M,\ t \in \mathbb{R}$
  output IL2SM$(m)_i,\ m \in M$
  internal UPDATE$_i$
  output REQRT$(t)_i,\ t \in \mathbb{R}$

Transitions $D$
internal UPDATE$_i$
  precondition:
    $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
    $\mathtt{RxRT}_i^{\mathrm{d}}$ empty
  effect:
    $\mathtt{vIL}_i^{\mathrm{d}} =$
    $f_{\mathrm{upd}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}})$
    $\mathtt{reqIL}_i^{\mathrm{d}} = f_{\mathrm{req}}(\mathtt{vIL}_i^{\mathrm{d}})$
    if $\neg\mathtt{reqIL}_i^{\mathrm{d}}$
      $\mathtt{RTIL}_i^{\mathrm{d}} =$ false
      $\mathtt{myIL}_i^{\mathrm{d}} =$
      $f_{\mathrm{my}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}}, b_2, i)$
    $\mathtt{next}_i^{\mathrm{d}} = \mathtt{next}_i^{\mathrm{d}} + dSlot$

output IL2CLRT$(m, \mathtt{now}_i^{\mathrm{a}})_i$
  precondition:
    $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
    $\neg(\mathtt{RxRT}_i^{\mathrm{d}}$ empty)
  effect:
    set $m = \mathtt{RxRT}_i^{\mathrm{d}}$
    set $\mathtt{RxRT}_i^{\mathrm{d}}$ empty

output REQRT$(\mathtt{now}_i^{\mathrm{a}})_i$
  precondition:
    $\mathtt{reqIL}_i^{\mathrm{d}} =$ true
    $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot - rem + cmp$
  effect:
    $\mathtt{reqIL}_i^{\mathrm{d}} =$ false

input IL2APNRT$(\mathtt{RxnRT}_i^{\mathrm{d}})_i$
effect:
  set $\mathtt{RxnRT}_i$ empty

input SM2IL$(m)$
  effect:
    if $\mathtt{RTIL}_i^{\mathrm{d}}$
      $\mathtt{RxRT}_i^{\mathrm{d}} = m$
    else
      $\mathtt{RxnRT}_i^{\mathrm{d}}$.Push$(m)$

output IL2SM$(m)_i$
  precondition:
    $(\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot) \wedge \mathtt{myIL}_i^{\mathrm{d}}$
    $(\neg(\mathtt{TxRT}_i^{\mathrm{d}}$ empty$) \wedge \mathtt{RTIL}_i^{\mathrm{d}})$
        $\vee \quad (\neg\mathtt{RTIL}_i^{\mathrm{d}} \wedge \neg(\mathtt{TxnRT}_i^{\mathrm{d}}$.Top empty$))$
  effect:
  if $\mathtt{RTIL}_i^{\mathrm{d}}$
    set $m = \mathtt{TxRT}_i^{\mathrm{d}}$
    set $\mathtt{TxRT}_i^{\mathrm{d}}$ empty
  if $\neg\mathtt{RTIL}_i^{\mathrm{d}}$
    set $m = \mathtt{TxnRT}_i^{\mathrm{d}}$.Top
    $\mathtt{TxnRT}_i^{\mathrm{d}}$.Pop
  $\mathtt{myIL}_i^{\mathrm{d}} =$ false

input CL2ILRT$(b_1, b_2, m)_i$
  effect:
  $\mathtt{RTIL}_i^{\mathrm{d}} = f_{\mathrm{RT}}(\mathtt{vIL}_i^{\mathrm{d}}, b_1)$
  $\mathtt{myIL}_i^{\mathrm{d}} = f_{\mathrm{my}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}}, b_2, i)$
  $\mathtt{TxRT}_i^{\mathrm{d}} = m$

input AP2ILNRT$(m)_i$
  effect:
  $\mathtt{TxnRT}_i^{\mathrm{d}}$.Push$(m)$

Trajectories $\mathcal{T}$
stop when
  $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot \wedge \mathtt{myIL}_i^{\mathrm{d}}$
  $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
  $(\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot - rem + cmp) \wedge \mathtt{reqIL}_i^{\mathrm{d}}$

evolve
  $d(\mathtt{now}_i^{\mathrm{a}}) = 1$
```

Figure 4-10. TIOA Model of IL [26].

As explained before, due to the choice of the variable of the abstract data type, vIL_i, the IL protocol family divides into two:

1. Real-Time Access Interface Layer (RAIL) Protocol
2. Time Slotted Interface Layer (TSIL) Protocol.

The protocols are designed such that they do not use any different functions from each other, but the functions operate differently according to the choice of the protocol that is made before the simulation is run.

In the following subsections, the protocol operations and differences with respect to each other with the implementation details are given.

4.3.2 Real-Time Access Interface Layer (RAIL) Protocol

The RAIL Protocol operates in a time-slotted manner. In each time slot, access is granted to a single device while the time slots that are used for RT and nRT messages are separated statistically. For this separation, the variable vIL_i must be defined.

As a member of the abstract data type, *cyc* variable is introduced. The ownership arbitration schedule repeats itself each *cyc* slots. There is also a set defined for every device that determines the reserved nRT slots in each slot cycle containing a total of *cyc* slots for that device only, and another set which is general and the same for all devices, which determine the RT slots in the *cyc* slots. The sets are the *RTSet* and the *nRTSet* with. On the other hand, $vIL_i^d.nRTSet$ is also a subset of the same set, but $vIL_i^d.nRTSet$ of device i and $vIL_j^d.nRTSet$ of device j has no intersection with each other, i.e. $vIL_i^d.nRTSet \cap vIL_j^d.nRTSet = \emptyset$, with non-equal device numbers, *i* and *j*.

The functions f_{RT}, f_{my}, f_{req} and f_{upd} that take place in the updates of variables are to be defined finally.

- f_{upd} is responsible for the update of the slot counter, in mod cyc, as defined previously.

- f_{req} issues the request to the upper layer.

- f_{RT} determines if the slot is assigned as an RT Slot or not

- f_{my} determines the owner of the slot.

The RAIL Protocol suggests the f_{upd}, f_{req}, f_{RT} and f_{my} functions as follows:

- In every single time slot, the *cnt* variable of each device *i* is incremented by 1. That is,

$$f_{upd}\ \mathrm{vIL_i^d}\ = (\mathrm{vIL_i^d.cnt} + 1)_{\mathrm{mod\ vIL_i^d.cnt}}.$$

- f_{req} issues the request to the upper layer if the slot counter is an element of the $vIL_i^d.RTSet$, i.e. returns 1, hence the request will be made in UPDATE$_i$.

$$f_{req}\ \mathrm{vIL_i^d}\ = \begin{matrix} true, & \mathrm{vIL_i^d.cnt} \in \mathrm{vIL_i^d.RTSet} \\ false, & otherwise \end{matrix}$$

- The reply of the request issued by f_{req} If the upper layer announces the slot to be an *RT* slot, becomes $CL2ILRT(true, b1, b2)$. As $b1$ is true, then f_{RT} returns true, else false.

$$f_{RT}\ \mathrm{vIL_i^d}, b_1\ = \begin{matrix} true, & \mathrm{vIL_i^d.cnt} \in \mathrm{vIL_i^d.RTSet} \\ & \wedge b_1 = true \\ false, & otherwise \end{matrix}$$

- f_{my} shows if device i is the owner of the next time slot or not, using $b2$coming from the upper layer with $RTIL_i^d$ and also if the slot counter is an element of the $vIL_i^d.nRTSet$, then, returns 1.

$$f_{my}\ \mathrm{vIL_i^d}, \mathrm{RTIL_i^d}, b_2, i) = \begin{array}{rl} b_2, & \mathrm{RTIL_i^d} \\ true, & (!\,\mathrm{RTIL_i^d}) \land \mathrm{vIL_i^d.cnt} \in \mathrm{vIL_i^d.nRTSet} \\ false, & otherwise \end{array}$$

4.3.3 Time Slotted Interface Layer (TSIL) Protocol

The TSIL Protocol is also based on a time-slotted structure. But as explained before, what is different is the operation of the functions.

Different from RAIL, the slot counter of the IL of the devices are updated, i.e. incremented by f_{upd} in nRT slots and f_{req} returns true at each call, hence the upper layer will make the choice of the type of the slot via f_{RT}.

The main difference is that the slot ownership decision of the RT Slots is made by the upper layer, where for the nRT slots, the decision is made locally, by the TSIL protocol.

The TSIL Protocol suggests the f_{upd}, f_{req}, f_{RT} and f_{my} functions as follows:

- The *cnt* variable of each device *i* is incremented by 1 if the slot is an RT Slot.. That is,

$$f_{upd}\ \mathrm{vIL_i^d}, \mathrm{RTIL_i^d} = \begin{array}{rl} (\mathrm{vIL_i^d.cnt} + +)_{\mathrm{mod\ vIL_i^d.cnt}}, & \mathrm{RTIL_i^d} \\ \mathrm{vIL_i^d.cnt}, & otherwise \end{array}$$

- f_{req} issues the request to the upper layer in each slot, hence it returns true.

$$f_{req}\ \mathrm{vIL_i^d} = true$$

- f_{RT} depends only on b_1. As b_1 is true, then f_{RT} returns true, else false.

$$f_{RT}\left(\mathrm{vIL}_{\mathrm{i}}^{\mathrm{d}}, b_1\right) = \begin{cases} \text{true}, & b_1 = true \\ \text{false}, & otherwise \end{cases}$$

- f_{my} does not differ from RAIL Protocol. It again shows if device i is the owner of the next time slot or not, using *b2* coming from the upper layer with $RTIL_i^d$ and also if the slot counter is an element of the $vIL_i^d.nRTSet$, then, returns $true$.

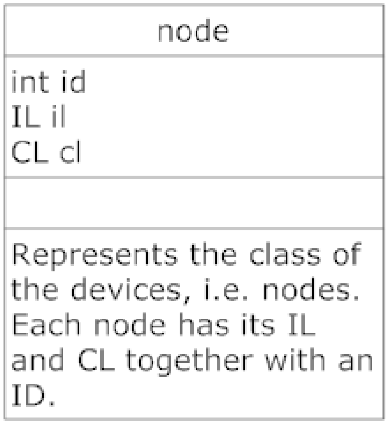
$$f_{my}\left(\mathrm{vIL}_{\mathrm{i}}^{\mathrm{d}}, \mathrm{RTIL}_{\mathrm{i}}^{\mathrm{d}}, b_2, i\right) = \begin{cases} b_2, & \mathrm{RTIL}_{\mathrm{i}}^{\mathrm{d}} \\ true, & !\,\mathrm{RTIL}_{\mathrm{i}}^{\mathrm{d}} \wedge \mathrm{vIL}_{\mathrm{i}}^{\mathrm{d}}.\mathrm{cnt} \in \mathrm{vIL}_{\mathrm{i}}^{\mathrm{d}}.\mathrm{nRTSet} \\ false, & otherwise \end{cases}$$

We can imply here that, since the slot type is dynamically controlled and decided by the upper layer, the TSIL protocol can adapt to the immediate needs of the RT communication networks, better than RAIL protocol.

4.3.4 IL Implementation and Related Class Definitions

The Interface Layer exists in every device, i.e. node. Hence, every device is a node in the system. First, the `node` class shall be defined. The node class is defined as illustrated in Table 4-3.

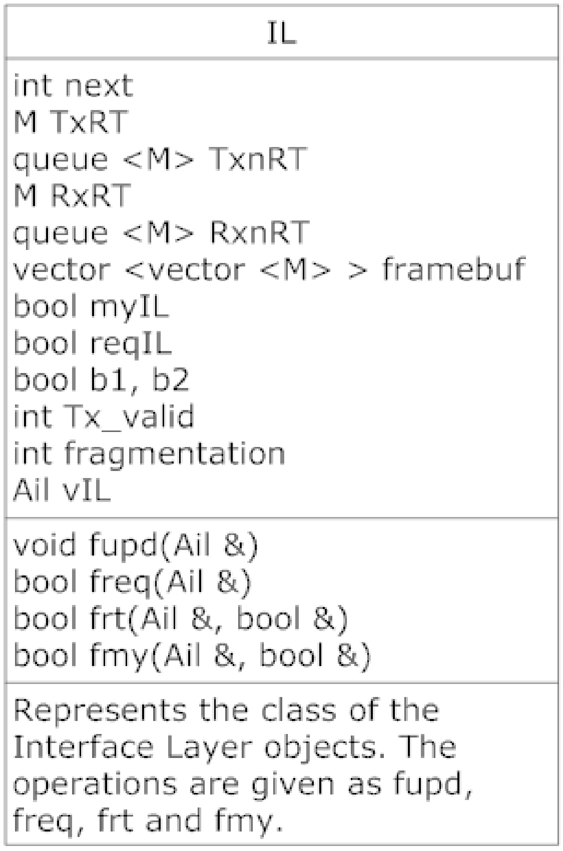
Table 4-3. The `node` Class Definition.



node
int id IL il CL cl
Represents the class of the devices, i.e. nodes. Each node has its IL and CL together with an ID.

There are 3 members of each node: integer id representing the device ID, and IL and CL which are IL and CL class objects. So, it is clear that, every node has its own IL and CL. The `IL` class definition is given in Table 4-4.

Table 4-4. The Interface Layer (`IL`) Class Definition.



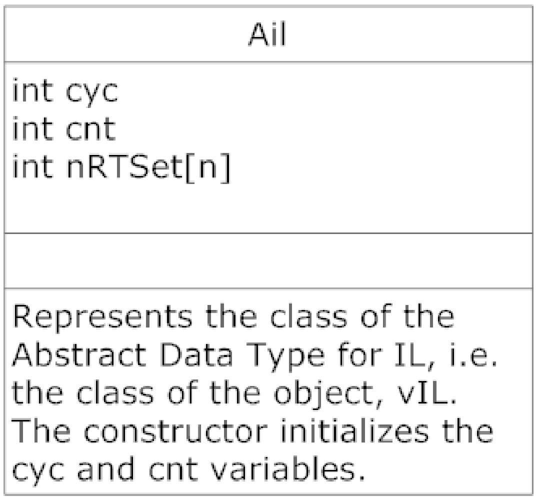
IL
int next M TxRT queue <M> TxnRT M RxRT queue <M> RxnRT vector <vector <M> > framebuf bool myIL bool reqIL bool b1, b2 int Tx_valid int fragmentation Ail vIL
void fupd(Ail &) bool freq(Ail &) bool frt(Ail &, bool &) bool fmy(Ail &, bool &)
Represents the class of the Interface Layer objects. The operations are given as fupd, freq, frt and fmy.

The Interface Layer has two nRT message queues for non-real-time transmission and reception and two message buffers for real-time transmission and reception. These queues and buffers hold the message until delivered to SM, CL or Control Application.

The Boolean variables, `myIL`, `reqIL`, `b1` and `b2` are as they have been explained in Section 4.3. The integer `Tx_valid` shows if the message being transmitted is valid, and the other integer `fragmentation` shows if there is a fragmentation on the packet currently being delivered.

The abstract data class object, `Ail vIL` is for the protocol related computations. The class `Ail` can be shown in Table 4-5.

Table 4-5. The Abstract Data Class `Ail` Definition.



Ail

int cyc
int cnt
int nRTSet[n]

Represents the class of the Abstract Data Type for IL, i.e. the class of the object, vIL. The constructor initializes the cyc and cnt variables.

The integer array `nRTSet[]` of the IL in each device keeps the slot numbers that are reserved for non-real-time communication for that device. Hence, when the update functions are called at each node, the node then comprehends if the nRT slot is assigned for itself by finding the *cnt* variable in `nRTSet[]`.

The approach is similar for `nRTSet[]`. But in this case, the array `nRTSet[]` is common for all nodes, hence, it is not a member of the `vIL` objects of the IL of any node.

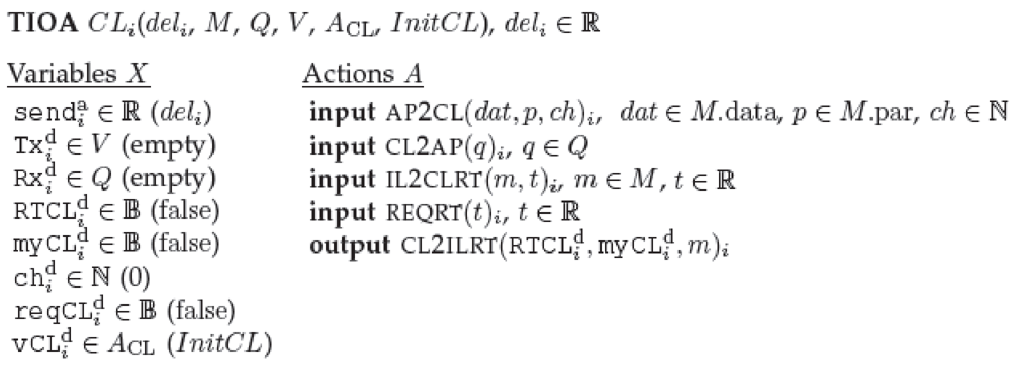
4.4 Coordination Layer (CL)

In this section, the Coordination Layer (CL) protocol family that is connected to IL and its TIOA Model is explained [26].

Basically, the CL of each device is responsible of broadcasting and processing information between the control application and its IL, using RT messages. So, using this information, the CL's adjust their RT operating behavior according to the distributed computations made in each device.

4.4.1 Coordination Layer (CL) TIOA Model

The TIOA Model definition of `CL` is shown below in Figure 4-11.



TIOA $CL_i(del_i,\ M,\ Q,\ V,\ A_{\mathrm{CL}},\ InitCL),\ del_i \in \mathbb{R}$

Variables X	Actions A
$\mathtt{send}_i^{\mathrm{a}} \in \mathbb{R}\ (del_i)$	**input** $\mathrm{AP2CL}(dat, p, ch)_i,\ \ dat \in M.\mathrm{data},\ p \in M.\mathrm{par},\ ch \in \mathbb{N}$
$\mathtt{Tx}_i^{\mathrm{d}} \in V$ (empty)	**input** $\mathrm{CL2AP}(q)_i,\ q \in Q$
$\mathtt{Rx}_i^{\mathrm{d}} \in Q$ (empty)	**input** $\mathrm{IL2CLRT}(m, t)_i,\ m \in M,\ t \in \mathbb{R}$
$\mathtt{RTCL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	**input** $\mathrm{REQRT}(t)_i,\ t \in \mathbb{R}$
$\mathtt{myCL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	**output** $\mathrm{CL2ILRT}(\mathtt{RTCL}_i^{\mathrm{d}}, \mathtt{myCL}_i^{\mathrm{d}}, m)_i$
$\mathtt{ch}_i^{\mathrm{d}} \in \mathbb{N}$ (0)	
$\mathtt{reqCL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)	
$\mathtt{vCL}_i^{\mathrm{d}} \in A_{\mathrm{CL}}\ (InitCL)$	

Figure 4-11. TIOA Model Definition of CL [26].

Brief explanation for each variable is as follows:

- $send_i^a$: the time that passes after a request was issued from the interface layer
- Tx_i^d: the vector of RT messages
- Rx_i^d: the queue of messages
- $RTCL_ş^d$: the Boolean decision variable that indicates whether the slot is decided to be a real time slot
- $myCL_i^d$: the Boolean decision variable that indicates whether the slot is owned by the i-th device
- ch_i^d: the decision variable that indicates the channel to be used for communication
- $reqCL_i^d$: the flag indicating whether a request is issued from the interface layer
- del_i: the delay between $REQRT_i$and the $CL2ILRT_i$ following afterwards, with $del_i < rem - cmp$
- dat: the data being transmitted
- par: the protocol information kept in the message

As claimed, the Coordination Layer is responsible of several duties. One is the message transmission between Application Layer and the Interface Layer. The messages are kept in the Tx_i^d and Rx_i^d buffers. The CL is capable of transmissions of messages of different protocol parameters via various communication channels. The Boolean variables are the decision variables for communication related calculations and they are updated due to the abstract type variable, vCL.

The control of the protocol is handled by the CL. Upon receiving a request from the IL, the CL, together with the timing information, assigns a single transmitter device for the subsequent time slot, such that, there will be only one unique transmitter device for each slot, avoiding collision.

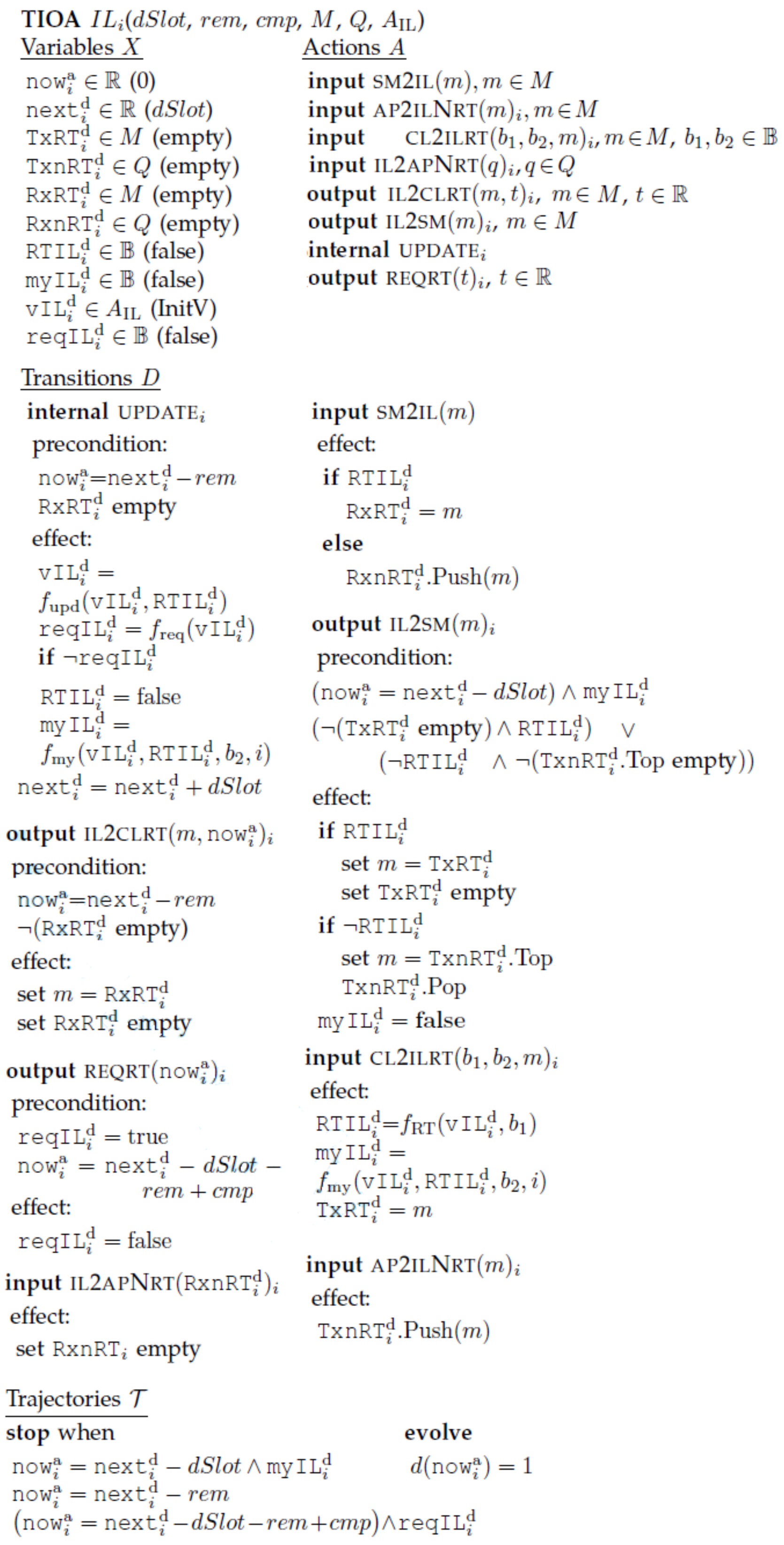
Whenever a request from IL arrives at the CL via $REQRT(t)_i^d$, the CL updates the real time status flag, $RTCL_i^d$ with the ownership details of the next slot included in $myCL_i^d$ and ch_i^d. After $send_i^a$amount of time which is less than or equal to del_i , via $CL2ILRT_i\ \ RTCL_i^d, myCL_i^d, m$, these detail are transferred to IL. But to note, the message *m* is non-empty for only the *i*-th device whose $myCL_i^d$ is *true*.

The Coordination Layer receives RT messages from the Application Layer via $AP2CL_i$ and from the Interface Layer via $IL2CLRT_i$ actions. When $AP2CL_i$ occurs, the message kept in $Tx_i^d\ \ ch$ is built from the data, `dat`, the protocol parameters, *par*, using the channel `ch`. For the reception from the IL, the received message is kept in Rx_i^d, and using the parameters in the received message, *par*, and the timing information *t* received from the Interface Layer, the decision variables are updated. The received message is forwarded to the upper layer, i.e. the control application, via $CL2AP_i$ on demand.

The current values of the decision variables vCL_i^d, the slot type $RTCL_i^d$ and the channel ch_i^d with its owner, device *i* where $myCL_i^d$ is *true*, are determined by the functions, g_{upd}, g_{RT} and g_{my}. The Coordination Layer definition is given based on these functions. The operations of these functions vary slightly according to the selected Coordination Layer protocol.

- $g_{upd}(vCL_i^d, m.par, t)$ is responsible for the update of the slot counter, in mod $vCL_i^d.cyc$ as defined previously, as well as protocol related updates for two different protocols.
- $g_{RT}\ \ vCL_i^d, RTCL_i^d, t$ determines if the slot is assigned as an RT Slot or not.
- $g_{my}(vCL_i^d, RTCL_i^d, t, i)$ decides about the owner of the slot.

The TIOA Model of the Coordination Layer [26] is presented in Figure 4-12.



```
TIOA $IL_i(dSlot,\ rem,\ cmp,\ M,\ Q,\ A_{\mathrm{IL}})$
Variables $X$
  $\mathtt{now}_i^{\mathrm{a}} \in \mathbb{R}$ (0)
  $\mathtt{next}_i^{\mathrm{d}} \in \mathbb{R}$ ($dSlot$)
  $\mathtt{TxRT}_i^{\mathrm{d}} \in M$ (empty)
  $\mathtt{TxnRT}_i^{\mathrm{d}} \in Q$ (empty)
  $\mathtt{RxRT}_i^{\mathrm{d}} \in M$ (empty)
  $\mathtt{RxnRT}_i^{\mathrm{d}} \in Q$ (empty)
  $\mathtt{RTIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)
  $\mathtt{myIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)
  $\mathtt{vIL}_i^{\mathrm{d}} \in A_{\mathrm{IL}}$ (InitV)
  $\mathtt{reqIL}_i^{\mathrm{d}} \in \mathbb{B}$ (false)

Actions $A$
  input $\mathrm{SM2IL}(m), m \in M$
  input $\mathrm{AP2ILNRT}(m)_i, m \in M$
  input $\mathrm{CL2ILRT}(b_1, b_2, m)_i, m \in M,\ b_1, b_2 \in \mathbb{B}$
  input $\mathrm{IL2APNRT}(q)_i, q \in Q$
  output $\mathrm{IL2CLRT}(m, t)_i,\ m \in M,\ t \in \mathbb{R}$
  output $\mathrm{IL2SM}(m)_i,\ m \in M$
  internal $\mathrm{UPDATE}_i$
  output $\mathrm{REQRT}(t)_i,\ t \in \mathbb{R}$

Transitions $D$
  internal $\mathrm{UPDATE}_i$
    precondition:
      $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
      $\mathtt{RxRT}_i^{\mathrm{d}}$ empty
    effect:
      $\mathtt{vIL}_i^{\mathrm{d}} = f_{\mathrm{upd}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}})$
      $\mathtt{reqIL}_i^{\mathrm{d}} = f_{\mathrm{req}}(\mathtt{vIL}_i^{\mathrm{d}})$
      if $\neg\mathtt{reqIL}_i^{\mathrm{d}}$
        $\mathtt{RTIL}_i^{\mathrm{d}} = \mathrm{false}$
        $\mathtt{myIL}_i^{\mathrm{d}} = f_{\mathrm{my}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}}, b_2, i)$
    $\mathtt{next}_i^{\mathrm{d}} = \mathtt{next}_i^{\mathrm{d}} + dSlot$

  output $\mathrm{IL2CLRT}(m, \mathtt{now}_i^{\mathrm{a}})_i$
    precondition:
      $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
      $\neg(\mathtt{RxRT}_i^{\mathrm{d}}$ empty$)$
    effect:
      set $m = \mathtt{RxRT}_i^{\mathrm{d}}$
      set $\mathtt{RxRT}_i^{\mathrm{d}}$ empty

  output $\mathrm{REQRT}(\mathtt{now}_i^{\mathrm{a}})_i$
    precondition:
      $\mathtt{reqIL}_i^{\mathrm{d}} = \mathrm{true}$
      $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot - rem + cmp$
    effect:
      $\mathtt{reqIL}_i^{\mathrm{d}} = \mathrm{false}$

  input $\mathrm{IL2APNRT}(\mathtt{RxnRT}_i^{\mathrm{d}})_i$
    effect:
      set $\mathtt{RxnRT}_i$ empty

  input $\mathrm{SM2IL}(m)$
    effect:
      if $\mathtt{RTIL}_i^{\mathrm{d}}$
        $\mathtt{RxRT}_i^{\mathrm{d}} = m$
      else
        $\mathtt{RxnRT}_i^{\mathrm{d}}.\mathrm{Push}(m)$

  output $\mathrm{IL2SM}(m)_i$
    precondition:
      $(\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot) \wedge \mathtt{myIL}_i^{\mathrm{d}}$
      $(\neg(\mathtt{TxRT}_i^{\mathrm{d}}$ empty$) \wedge \mathtt{RTIL}_i^{\mathrm{d}}) \vee$
          $(\neg\mathtt{RTIL}_i^{\mathrm{d}} \wedge \neg(\mathtt{TxnRT}_i^{\mathrm{d}}.\mathrm{Top}$ empty$))$
    effect:
      if $\mathtt{RTIL}_i^{\mathrm{d}}$
        set $m = \mathtt{TxRT}_i^{\mathrm{d}}$
        set $\mathtt{TxRT}_i^{\mathrm{d}}$ empty
      if $\neg\mathtt{RTIL}_i^{\mathrm{d}}$
        set $m = \mathtt{TxnRT}_i^{\mathrm{d}}.\mathrm{Top}$
        $\mathtt{TxnRT}_i^{\mathrm{d}}.\mathrm{Pop}$
      $\mathtt{myIL}_i^{\mathrm{d}} = \mathrm{false}$

  input $\mathrm{CL2ILRT}(b_1, b_2, m)_i$
    effect:
      $\mathtt{RTIL}_i^{\mathrm{d}} = f_{\mathrm{RT}}(\mathtt{vIL}_i^{\mathrm{d}}, b_1)$
      $\mathtt{myIL}_i^{\mathrm{d}} = f_{\mathrm{my}}(\mathtt{vIL}_i^{\mathrm{d}}, \mathtt{RTIL}_i^{\mathrm{d}}, b_2, i)$
      $\mathtt{TxRT}_i^{\mathrm{d}} = m$

  input $\mathrm{AP2ILNRT}(m)_i$
    effect:
      $\mathtt{TxnRT}_i^{\mathrm{d}}.\mathrm{Push}(m)$

Trajectories $\mathcal{T}$
  stop when
    $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot \wedge \mathtt{myIL}_i^{\mathrm{d}}$
    $\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - rem$
    $(\mathtt{now}_i^{\mathrm{a}} = \mathtt{next}_i^{\mathrm{d}} - dSlot - rem + cmp) \wedge \mathtt{reqIL}_i^{\mathrm{d}}$
  evolve
    $d(\mathtt{now}_i^{\mathrm{a}}) = 1$
```

Figure 4-12. TIOA Model of the Coordination Layer [26].

Similar to the Interface Layer design, due to the choice of the variable of the abstract data type, vCL_i, the protocol family divides into two:

1. Dynamic Allocation Real-Time (DART) Protocol
2. Urgency-Based Real-Time (URT) Protocol.

The protocols are designed such that they do not use any different functions from each other, but the functions operate differently according to the choice of the protocol that is made before the simulation is run.

In the following subsections, the protocol operations and differences with respect to each other with the implementation details are given. Further details of two different types of Coordination Layer (CL) are given in the Subsections 4.4.2 and 4.4.3.

4.4.2 Dynamic Allocation Real-Time Protocol (DART)

The DART Protocol holds the variables and information as allocated real-time slots assigned to specific controller devices. These variables and information depend on the state of the control application such that, they are dynamically updated and modified relative to the instantaneous state of the application.

Up to now, $vCL_i^d.cyc$ and $vCL_i^d.cnt$ are present as the decision variables constituting the cycle and the slot counter variables, respectively. Furthermore, the *allocation data object ado* is introduced with members $ado.num \in \mathbb{N}$, $ado.slots \subseteq \{0, 1, 2, \dots, cyc - 1\}$ and $ado.used \subseteq \mathbb{N} \times \mathbb{N}$, where $ado.num$ holds the RT Slot number allocated in every *cyc* RT slots, $ado.slots$ is an ordered list of $ado.num$ allocated RT Slots and finally, $ado.used$ is a tuple indicating the device that uses the ADO together with its channel id ((0,0) if not used). The class definition of the *allocation data objects* is given in Table 4-6.

Table 4-6. The ADO Class Definition.

ADO
int num
vector<int> slots
pair<int,int> used
Represents the class of the Allocation Data Objects.

The CL of each device has a vector of allocation data objects (ado), $vCL_i^d.alloc[\,]$. But for $k \neq l$, it has to hold that $vCL_i^d.alloc[k] \neq vCL_i^d.alloc[l]$. So that, every `ado` is different from each other.

The protocol parameter of the messages *m*.par in DART has two members: $\mathrm{par.free}$ and $\mathrm{par.new}$. $\mathrm{par.free}$ is a vector of 2-tuples ($\mathrm{par.free}\ \mathrm{l}\ \in \mathbb{N} \times \mathbb{N}$) indicating the ado with member $\mathrm{alloc[\,k\,].used} = \mathrm{par.free[\,l\,]}$ should be freed, i.e. the allocation data objects that exist in the par.free vector will be deleted from $\mathrm{alloc[k].used}$ vector. $\mathrm{par.new}$ is a vector of triples $(\mathrm{a}, \mathrm{b}, \mathrm{c}) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$, indicating the new allocation data object to be assigned such that, an `ado` with *a* slots is to be assigned to the channel *c* for node *b*.

The update functions for DART are defined in [26] as:

- $g_{upd}\ vCL_i^d, m.par, t\ =\ vCL_i^d.cnt + 1\ _{mod\ vCL_i^d.cyc}$

and updates the allocation data objects such that each entry in `m.par.free` is removed from `ado.used` and the first suitable unused `ado` in $\mathrm{vCL_i^d.alloc[\,]}$ is assigned.

- $g_{RT}\ vCL_i^d, RTCL_i^d, t\ =\ \begin{matrix} true,\ vCL_i^d.cnt \in vCL_i^d.alloc\ k\ .slots, \exists k \\ false,\ otherwise \end{matrix}$

- $g_{my}\ \ vCL_i^d, RTCL_i^d, t, i\ \ =$

$$\begin{array}{ll} true, c\ \ , & vCL_i^d.cnt \in vCL_i^d.alloc\ \ k\ \ .slots \\ & \qquad\qquad \wedge \\ & vCL_i^d.alloc\ \ k\ \ .used =\ \ i, c\ \ ,\ \exists k \\ false, 0\ \ , & otherwise \end{array}$$

4.4.3 Urgency-Based Real-Time Protocol (URT)

In the Urgency-Based Real-Time Protocol, there are *communication* r*equests* that contain information about the transmission rights for each device in the network.

The decision variable is a *priority queue* vCL_i^d.PQ of 4-tuple requests. The members of this priority queue are sorted according to their `eT` values such that the device having the most urgent available request gets the access. A request could be explained as:

$$b, c, eT, dT$$

$$b \rightarrow\ device$$

$$c \rightarrow channel$$

$$eT \rightarrow eligibility\ time$$

$$dT \rightarrow deadline.$$

Note that, the deadline is not absolute. The deadline of each request is relative to the time instant that the request is issued. The meaning of $\ b, c, eT, dT\ $ is, device *b* can send its next message via channel *c* after `eT` and must finish sending before `dT`. Moreover, the URT suggests that each message being transmitted contain a set of requests `m.par.req` as its protocol related parameter. The requests that exist in the message received are stored in vCL_i^d.PQ.

The update functions of URT are given as the followings:

- If $g_{upd}\ vCL_i^d, m.par, t$ is called and $RTCL_i^d = true$, the first request that is the request that was available, i.e. eligible, in the previous RT slot, is popped out from vCL_i^d.PQ if *m*.par is non-empty as a result of a successful reception of a valid RT message. If the received message is not valid, the first request is re-pushed into the vCL_i^d.PQ since its transmission has not been successfully completed. In addition, all the requests kept in `m.par.req` are pushed in vCL_i^d.PQ after their `dT` members are made absolute by adding it the current time.

- $$g_{RT}\ vCL_i^d, RTCL_i^d, t \ = \ \begin{array}{l} true,\ vCL_i^d.PQ.Top.eT \leq t \\ false,\ otherwise \end{array}$$

- $$g_{my}\ vCL_i^d, RTCL_i^d, t, i \ = \ \begin{array}{ll} true, a\ , & \begin{array}{c} vCL_i^d.PQ.Top.b = i \\ \wedge \\ RTCL_i^d = true \\ \wedge \\ vCL_i^d.PQ.Top.c = a \end{array} \\ false, 0\ , & otherwise \end{array}$$

4.4.4 CL Implementationand Related Class Definitions

In every node in our system, as well as the Interface Layer, there is a Coordination Layer. Hence, every node should have the object `cl` of class `CL`. The `CL class` is presented in Table 4-7.

Table 4-7. The `CL` Class Definition.

CL
M msg
double send
vector<M> Tx
queue<M> Rx
bool RTCL
bool myCL
int ch
bool reqCL
Acl vCL
void gupd(Acl &, M)
bool grt(Acl &)
pair<int,int> gmy(Acl &, bool &, int)
Represents the class of the Coordination Layer objects. The operations are given as gupd, grt and gmy.

The CL class has a `M` type `msg` to keep the real-time message being held at the CL momentarily and a `double send` as given in the definitions, previously. In addition, two vectors of messages, one for transmission and reception each. These vectors contain RT messages until removal by transmission down to IL or CL2AP actions. The Boolean variables are `RTCL`, `myCL` and `reqCL`, as explained before. Moreover, the integer `ch` is added to keep the active channel. Finally, the abstract class object; `vCL`. The abstract class `Acl` can be declared as in Table 4-8.

Table 4-8. The `Acl` Class Definition.

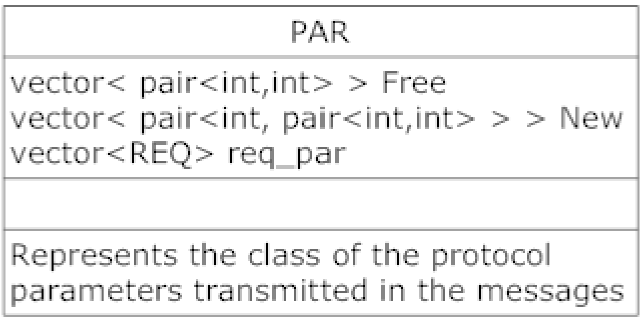
Acl
int cyc
int RTcnt
vector<ADO> alloc
priority_queue <REQ, vector<REQ>, greater<REQ> > PQ
PAR par_acl
Represents the class of the Abstract Data Type for CL, i.e. the class of the object, vCL. The constructor initializes the ADOs.

The abstract class contains the `cyc` variable and the `RTcnt` variable, which is the analogue of `cnt` variable in the `Ail` class.

The CL definition is unique, i.e. there is only one CL that can run both of the two suggested protocols, DART and URT. Hence, the vector of `ADO`s `alloc`, is declared to be used in case the CL protocol is DART as well as the priority queue of requests `PQ`, exists for URT.

Furthermore, there is a `par` variable in the `Acl` class. The `PAR` class that represents the protocol parameters and the `REQ` class that is the type of the requests are declared as in Table 4-9 and Table 4-10, respectively.

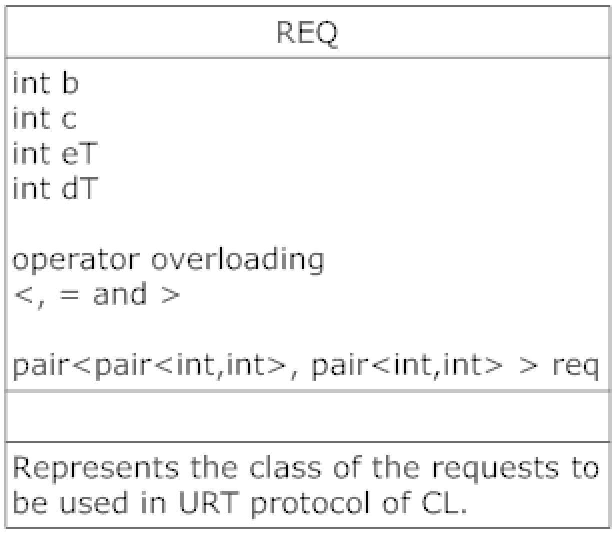
Table 4-9. The `PAR` Class Definition.



PAR
vector< pair<int,int> > Free
vector< pair<int, pair<int,int> > > New
vector<REQ> req_par
Represents the class of the protocol parameters transmitted in the messages

The vectors `Free` is a 2-tuple where `New` is a 3-tuple. The operational functions are given in Section 4.3.1. The vector of requests `req_par` holds the requests to be used later.

Table 4-10. The REQ Class Definition.



REQ
int b int c int eT int dT operator overloading <, = and > pair<pair<int,int>, pair<int,int> > req
Represents the class of the requests to be used in URT protocol of CL.

The `REQ` class objects are 4-tuple requests. The entries are the `b` for the node, `c` representing the channel and `eT` the eligibility time and `dT` the deadline time, as previously mentioned. These integers constitute the `req` as a 4-tuple request object.

CHAPTER 5

COMPARISON RESULTS

In Chapter 4, The D^3RIP Simulator is examined in detail. The reason the simulator is created is to simulate the actual system as realistically and identically as possible, such that the states and values of the variables and times that simulator calculates are identical to those of the real system; i.e. it is checked if a packet that is created at the same time in both the experiment and simulation, delivered at the same time instants, if these packets are transmitted to the SM at the same times, etc.

We compare the output of the D^3RIP simulator to the output of the real implementation of the D3RIP stack [28], [29]. The laboratory test bed system is composed of 2 devices; Controllers G1 and G2. These controllersare identical PCs equipped with Intel Core i3 550@3.20GHz Processors and 4GB of RAMs.

The system runs the Real-time Access Interface Layer Protocol (RAIL) as the IL protocol and on top of it, the Urgency-based Real-Time Protocol (URT) is employed as the CL protocol.

As given in [29], there are 3 message transmission as the Real-Time communication, i.e. ?μ, !μ and μ. The message ?μ is the first request that a controller sends to the other controllerover a cross Ethernet cable. In each comparison, there will be 6 message transmissions. For RT communication, these are ?μ, !μ, μ, ?μ, !μ and μ, in order. Where for nRT traffic, the messages are just ordinary nRT messages.

The test bed is run for more than 900 seconds. 31 RT packets are transmitted and received. For RT communication, the message order is the same, i.e. ?µ, !µ and µ as illustrated in Figure 5-1. Hence, transmissions of these 3 messages always repeats itself. In the comparisons, two cycles are taken as comparison data. The average, maximum and minimum delays are presented in Table 5-2.

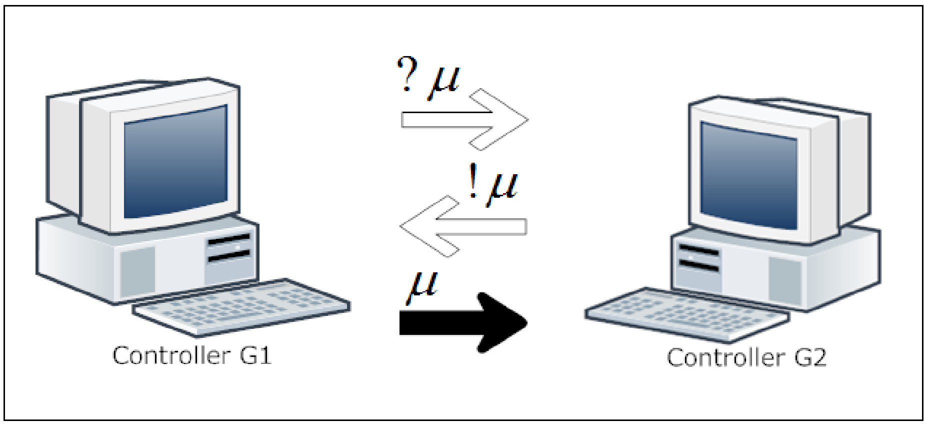


Figure 5-1. RT Communication.

For nRT communication, as claimed in Section 4.1, the synchronization packets are generated and transmitted every second according to IEEE 1588. The experiment in the test bed was run for 90 seconds, so, more than 90 nRT packets are transmitted and received. The average, maximum and minimum delays are presented in Table 5-4.

We set `dSlot` = 3ms, `rem` = 0.5ms and `cmp` = 0.1ms. In the simulator, `dSlot` is taken as 3000, making each time unit equal to 1µs. Hence, `rem` = 500 and `cmp` = 100 time units. According to Figure 4-3, the `rem` is less than `dSlot`, similarly, `cmp` is less than `rem`. Different selections of these variables are possible, but every selection has boundaries for the system performance. For instance, depending on `dSlot`, the maximum message length that could be transmitted within the time slot would decrease; if it is too high, then, the bandwidth would be wasted.

The other issue to be noted is the scheduling while tests. The `cycle` variable is chosen as 4, and the scheduling is made as depicted in Figure 5-2.

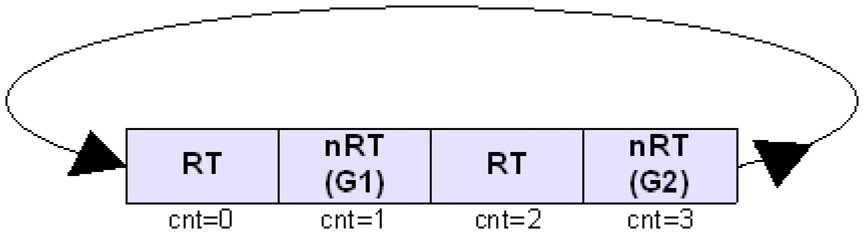


Figure 5-2. Fair Scheduling Used During the Test.

For RT communications, the occurrence times of AP2CL events are obtained from the laboratory as real values, and then, input to the simulation. Then, the only thing to examine is the flow of the simulation, i.e. check and trace in the simulation results, if the packet, which had been created at a specific known time in the real system, pass to the IL via CL2IL at the source and are taken by the application via CL2AP at both controllers at same instants with the real time logs. Likewise, for the nRT traffic, the AP2ILnRT, IL2SM at the source and IL2APnRT events are logged and will be compared.

We are now ready to investigate the simulation results for the RT and nRT communication.

5.1 Real-Time Packets

The comparison of the simulation and the obtained logs for the RT communication are given in Table 5-1. The RT messages, ?μ, !μ and μ, are created by the control application running in the transmitter node at different times via AP2CL_Tx. First, the experimental system is run and the timings are noted. Then, at these times, the D3RIP Simulator creates an identical messageat the corresponding time instant for every RT message in the real system. Hence, the initialization of the simulation is triggered and CL2IL and CL2AP timings are noted to be compared with the real logs.

In Table 5-1, the AP2CL_Tx and CL2IL_Tx events represent the AP2CL and the CL2ILRT events that take place at the transmitting node, likely, the CL2AP_Tx and CL2AP_Rx events denote the reception of these messages by the control applications on the transmitting and the other devices, respectively.

As shown in Figure 5-1, ?µ, !µ and µ are the RT messages. These messages are randomly selected six of all the messages created by the control applications in the real system. The numbers are the time instants in µs at which each corresponding event is observed to occur in the experiment and in simulation. Then, the difference between the occurrence times of each event in obtained from real system and the simulation are also given in µs. The difference between the experimental and the simulation timings yield the simulation precision.

Table 5-1. Simulation Results and Comparisons of the RT traffic.

	EXPERIMENT (µs)	SIMULATION (µs)	Δt(µs)
?µ	**Message 1**		
AP2CL_Tx	2469	2469	0 (Input to Simulator)
CL2IL_Tx	2535	2550	15
CL2AP_Tx	3078	3024	54
CL2AP_Rx	3204	3024	180
!µ	**Message 2**		Δt(µs)
AP2CL_Tx	3733	3733	0
CL2IL_Tx	8536	8550	14
CL2AP_Tx	9051	9024	27
CL2AP_Rx	9253	9024	229

µ	Message 3		Δt(µs)
AP2CL_Tx	9890	9890	0
CL2IL_Tx	14551	14550	1
CL2AP_Tx	15092	15024	68
CL2AP_Rx	15351	15024	327

?µ	Message 4		Δt(µs)
AP2CL_Tx	64324	64324	0
CL2IL_Tx	66581	66550	31
CL2AP_Tx	67090	67024	66
CL2AP_Rx	67387	67024	363

!µ	Message 5		Δt(µs)
AP2CL_Tx	67906	67906	0
CL2IL_Tx	72553	72550	3
CL2AP_Tx	73063	73024	39
CL2AP_Rx	73372	73024	348

µ	Message 6		Δt(µs)
AP2CL_Tx	73769	73769	0
CL2IL_Tx	78578	78550	28
CL2AP_Tx	79079	79024	55
CL2AP_Rx	79312	79024	288

As a result, we observed that for a total of 31 RT packet transmissions. For each packet,six events that occur in both systems are timestamped. The timestamps obtained from the real system are then compared with the corresponding ones taken from the simulator.

Finally, for a total of 186 events, the average differences between the occurrence times of the same events in the simulator and the test bed is found as 122,6 µs, the maximum difference as 401 µs, where the minimum difference is 0.

Table 5-2. The Differences Between the Real System and the Simulation for RT Packets.

RT Communication	Average Difference (μs)	Min (μs)	Max (μs)
TOTAL (31 Packets)	122,6	0	401

5.2 Non-Real-Time Packets

Similarly to Table 5-1, the timings are obtained for random six nRT message transmissions from the experiment and in simulation are presented. In the simulation, the messages are created and delivered to the IL by AP2ILnRT_Tx and the timings of the corresponding IL2SM and IL2APnRT events are observed.

The AP2ILnRT_Tx and IL2SM_Tx events represent the AP2ILnRT and the IL2SM events that take place at the transmitting node, similarly, the IL2APnRT_Tx and IL2APnRT_Rx events denote the reception of these messages by the control applications on the transmitting and the other devices, respectively.

Note that, in the simulation, the messages are forwarded to the control application via IL2APnRT at both stations as soon as the IL2SM action ends. Hence, the IL2SM and the IL2APnRT events are assumed to occur at the same time instants.

Table 5-3. Simulation Results and Comparisons of the nRT traffic.

	EXPERIMENT (μs)	SIMULATION (μs)	Δt(μs)
	Message 1		
AP2ILnRT_Tx	6509100	**6509100**	0 (Input to Simulator)
IL2SM_Tx & IL2APnRT_Tx	6515016	6515012	4
IL2SM_Rx &IL2APnRT_Rx	6515302	6515012	290

	Message 2		Δt(μs)
AP2ILnRT_Tx	6515354	6515354	0
IL2SM_Tx & IL2APnRT_Tx	6521016	6521012	4
IL2SM_Rx &IL2APnRT_Rx	6521300	6521012	288

	Message 3		Δt(μs)
AP2ILnRT_Tx	14509111	14509111	0
IL2SM_Tx & IL2APnRT_Tx	14519021	14519012	9
IL2SM_Rx &IL2APnRT_Rx	14519323	14519012	311

	Message 4		Δt(μs)
AP2ILnRT_Tx	14519369	14519369	0
IL2SM_Tx & IL2APnRT_Tx	14525017	14525012	5
IL2SM_Rx &IL2APnRT_Rx	14525334	14525012	322

	Message 5		Δt(μs)
AP2ILnRT_Tx	22509104	22509104	0
IL2SM_Tx & IL2APnRT_Tx	22511017	22511012	5
IL2SM_Rx &IL2APnRT_Rx	22511318	22511012	306

	Message 6		Δt(μs)
AP2ILnRT_Tx	22511384	22511384	0
IL2SM_Tx & IL2APnRT_Tx	22517018	22517012	6
IL2SM_Rx &IL2APnRT_Rx	22517225	22517012	213

It is observed that in a total of 95 nRT packets, the average difference between the simulation and the real timestamps is found to be 139,6μs, the maximum difference between the simulation and the logs obtained in the lab is 344μs, where the minimum is found as 0.

Table 5-4. The Differences Between the Real System and the Simulation for nRT Packets.

nRT Communication	Average Difference (μs)	Min (μs)	Max (μs)
TOTAL (95 Packets)	139,6	0	344

The selection of `dSlot` length changes the collisions in the test bed. Due to the constant synchronization error that is at most 500μs, if the slot length is not big enough, the real system would face with collisions and retransmissions. Hence, all the timings would be shifted ambiguously. But in our selection as 3ms, the safety margin is three times the minimum required. That is why no such errors occur.

It is interesting that when we examine Table 5-3 and Table 5-4, we observe that the transmitting device receives its message earlier than the other device. Actually, for the six messages each for RT and nRT traffic, three of them were sent by Controller G1 and the other three were sent by Controller G2. But it is found that the message is always received earlier by the transmitting device.

In fact, this is not the case. This is only the synchronization error between the devices. In the beginning, it was claimed to be less than 500μs.

The delays due to the operating system running on the controllers, which are because of the ISR call lags, are also a major instance since for a single transmission, in each event that occurs, i.e. IL2SM and SM2IL, the interrupt service routine is called once. Hence for nRT transmissions, the ISR is called twice and for RT traffic, the ISR is called 4 times, additionally for CL2IL and IL2CL. Hence, since the average ISR delay is provided as 25μs with a maximum of 100μs, on the average, an additional 50μs for nRT and 100μs for RT traffic is included in the experimental results as another main reason for the differences between the results of the simulation and the experiment.

Moreover, the experimental setup uses two timers for *rem* and *dSlot* values. With the help of these timers, the ISR is called for actions. This is also an additional two ISR calls, 50μs on the average, at each slot beginning and for each transmission.

It can be inferred that since the maximum delayed action within all actions in the system is 401μs, the packets and the events are guaranteed to be transmitted or served at the same time slots without any asynchronization between service of events such that, the simulation is serving the same event that the two controllers are also serving, at the same time slot. As a result, the non-real-time message queues, the slot numbers, the event times, every variable, i.e. the state of the TIOA representing the overall system in the simulator and the system are at the same state.

To sum up, the simulation is claimed to be consistent with the real system under some assumptions and abstractions. They are mainly the ISR and timer delays (each 100µs max) and the synchronization errors when the timestamps in real systemare obtained at the maximum synchronousity of 100µs due to current RT Linux capabilities.

The timestamps in the experiment began to be taken while the machines are in synchronization up to 500µs, such that, in the results, we observed a 401µs latency. But in the best case, this would not be the case. Generally, if the timestamps are obtained while the machines are at maximum synchronization, i.e. less than 100µs, it can be inferred that the simulation is reliable at about a maximum of 300µs error.

To conclude. for 35,3µs standard deviation, for 126 packets in total, i.e. 756 event comparisons, the Confidence Intervals of the Standard Deviations are given in Table 5-5.

Table 5-5. The Confidence Interval of the Standard Deviation.

CI	SD
90%	33.871 to 36.866
95%	33.605 to 37.176
99%	33.096 to 37.794

Namely, if the comparison process is repeated on different samples, the calculated CI would contain the true parameter 90, 95 and 99% of the time with respect to the given SD.

For a total of 756 events, the average difference is calculated as 135.42μs. For constant SD, the 0.95 confidence interval is then calculated as (134,994-135,841) in μs. Hence, we can claim that in 95% of different simulations, even with the assumed and abstracted cases, the simulation would be confident within these limits.

One last addition is about the complexity of the simulator. In the worst case, we can assume that in every time slot a device will be ready for transmission. Consequently, for every RT packet of the transmitting device, AP2CL – CL2ILRT – IL2SM events will be pushed into the PQ of events. For the reception of there RT packets in every receiving device, SM2IL – IL2CLRT – CL2AP events will be pushed. For nRT messages, the events are onl AP2ILnRT – IL2SM and SM2IL – IL2APnRT, in transmitting and receiving devices, respectively. Namely, for each RT transmission 3, for each RT reception 3, for each nRT transmission 2 and for each nRT reception 2 events are pushed.

As a result, in the 4 time slots as given in Figure 5-2, 2 RT and 2 nRT messages will be transmitted. These messages will be received by both devices.

So, totally,

$$2\ RT\ transmission * 3\frac{events}{RT\ transmission} + 2\ RT\ reception * 3\frac{events}{RT\ reception}$$
$$+$$
$$2\ nRT\ transmission * 2\frac{events}{nRT\ transmission} + 2\ nRT\ reception * 2\frac{events}{nRT\ reception}$$

$$= 20\ events$$

will be pushed and served in 4 time slots. That is, if the load is not more than the amount discussed above, the size of the PQ remains the same after each cycle represented in Figure 5-2.

If the load is much more, such as if the control application transmits many more messages, then, the PQ will begin to queue the events. Depending on the machine capabilities, the PQ size may cause space inadequacies in hard disks. This issue may constitute a boundary for the simulator.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, the Dynamic Distributed Dependable Real-Time Industrial communication Protocol, the D^{3}RIP [26], that adapts to the dynamically changing demands of industrial real-time Ethernet automation networks with an *Interface Layer* and a *Coordination Layer* that cooperate between the MAC layer and the control application, is simulated using GCC compiler and the simulation results including the packet transmission and arrival times are discussed.

The major contribution of this thesis is the D^{3}RIP Protocol Simulator that can be used to try and analyze new ideas and the changes in the D^{3}RIP Family, without any real instrument, extensive effort, with no cost and time.

The major works accomplished in the thesis are listed as follows:

- The Shared Medium (SM) as the MAC Layer is implemented.
- The Interface Layer (IL) that is connected to CL for RT traffic and the Control Application for the nRT traffic, providing the time-slotted access to the SM for RT and nRT communication is implemented.
- The Coordination Layer (CL), that deterministically arbitrates the access to the MAC Layer that is provided by the IL for the RT traffic, is implemented.

- The Real-time Access Interface Layer (RAIL), that provides time-slotted access to the MAC Layer with exclusively and specifically distinguished the time slots for RT and nRT traffic, is implemented.
- The Time-Slotted Interface Layer (TSIL), that for each time slot, allows the CL make the decision about the type, is implemented.
- The Dynamic Allocation Real-time Protocol (DART) is implemented
- The Urgency-based Real-Time Protocol (URT) is implemented.
- The Shared Medium is integrated with both ILs and both CLs. The user chooses the types of IL and CL and then the D^{3}RIP Simulator runs.
- A real sample system with RAIL as IL protocol and URT as CL protocol is run in the laboratory, and the experimental results are logged. Then, the logs are compared with the D^{3}RIP Simulator results. The assumptions made, the similarities and the differences are discussed.

For the slot length of 3ms, the real system and the D^{3}RIP Simulator are run. The data are logged after the synchronization difference between Controller G1 & G2 was 500µs, and the D^{3}RIP Simulator results have come out to be less than 401µs for both RT and nRT traffic, using RAIL as the IL protocol and URT as the CL protocol.

The simulator behaviour fits to the D^{3}RIP framework. So, the simulator can be plugged and used in any D^{3}RIP module. Namely, making no change in the simulator but the subtle modifications in the update functions, new versions of IL and CL layers could be modelled. As a result, keeping in mind that as long as the assumptions and abstractions made do not constitute problematic issues and they are reasonable for such systems, it can be claimed that the simulator for the D^{3}RIP Protocol using RAIL-URT protocol couple implemented and delivered is precise.

For D^{3}RIP systems whose behaviour and parameters are known from the design, the simulator can be used for modelling. Hence, using this simulator, innovative ideas and further investigations with modications would be much more easier to be applied and tested rather than setting up a real system. Moreover, if a new protocol type for IL and CL is created, the modification required in the simulator is quite simple. Eventually, with the simulator, the D^{3}RIP framework will also be developed quickly, easily and smoothly.

6.2 Future Work

In addition to the RAIL as the IL protocol and URT as the CL protocol choices, the D^{3}RIP Simulator is capable of simulating the 4 different RTE Protocols presented in [26] as the D^{3}RIP Protocol Family. In each case, the simulator can analyze every variable in every state or time instant.

In the future, for a more dependable simulator, a real system that can implement the other offered protocols, TSIL and DART, can be set up, and the simulator and the system could be cross-checked and investigated. So that, the simulator would be more trustworthy since it would be guaranteed for all of the protocol couples to simulate correctly. Additionally, the D^{3}RIP Simulator will be integrated with a simulator for the control system to simulate the overall system behaviour, from head to toe.

In addition, the delays experienced in the Network Interface Card (NIC) can be modelled and added to the simulator for more realistic results. In general, for more precise and realistic behaviour, the assumptions can be included in the simulator.

REFERENCES

[1] J. Baillieul and P. Antsaklis, "Control and communication challenges in networked real-time systems," Proceedings of the IEEE, vol. 95, no. 1, pp. 9–28, Jan. 2007.

[2] J. Moyne and D. Tilbury, "The emergence of industrial control networks for manufacturing control, diagnostics, and safety data," Proceedings of the IEEE, vol. 95, no. 1, pp. 29–47, Jan. 2007.

[3] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo, "FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems," Industrial Informatics, IEEE Transactions on, vol. 1, no. 3, pp. 162–172, Aug. 2005.

[4] P. Neumann, "Communication in industrial automation – what is going on?" Control Engineering Practice, vol. 15, pp. 1332–1347, 2007.

[5] M. Felser, "Real-time Ethernet - industry prospective," Proceedings of the IEEE, vol. 93, no. 6, pp. 1118–1129, June 2005.

[6] J.-D. Decotignie, "The many faces of industrial ethernet [past and present]," Industrial Electronics Magazine, IEEE, vol. 3, no. 1, pp. 8–19, March 2009.

[7] S.-K. Kweon and K. G. Shin, "Statistical real-time communication over ethernet," Parallel and Distributed Systems, IEEE Transactions on, vol. 14, no. 3, pp. 322–335, 2003.

[8] S.-K. Kweon, M.-G. Cho, and K. G. Shin, "Soft real-time communication over Ethernet with adaptive traffic smoothing," Parallel and Distributed Systems, IEEE Transactions on, vol. 15, no. 10, pp. 946–959, 2004.

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

[10] H. Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched real-time Ethernet with earliest deadline first scheduling protocols and traffic handling," in Parallel and Distributed Processing Symposium, 2002, pp. 94 –99.

[11] J. Wang and B. Ravindran, "Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis," Parallel and Distributed Systems, IEEE Transactions on, vol. 15, no. 2, pp. 119–133, 2004.

[12] (2010) Ethernet powerlink standardization group. [Online]. Available: http://www.ethernet-powerlink.org/index.php?id=17, last visited on 16.08.2011.

[13] (2002, Nov.) IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. [Online]. Available: http://ieee1588.nist.gov,last visited on 16.08.2011.

[14] R. Zarick, M. Hagen, and R. Bartos, "The impact of network latency on the synchronization of real-world ieee 1588-2008 devices," in Precision Clock Synchronization for Measurement Control and Communication, International IEEE Symposium on, 2010.

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

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

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

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

[19] K. Schmidt, E. Schmidt, and J. Zaddach, “Safe operation of distributed discrete-event controllers: A networked implementation with real-time guarantees,” in IFAC World Congress, 2008.

[20] T. Sauter, “The three generations of field-level networks-evolution and compatibility issues,” Industrial Electronics, IEEE Transactions on, vol. 57, no. 11, pp. 3585 –3595, 2010.

[21] D. E. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, “The theory of timed I/O automata,” MIT Laboratory for Computer Science, Cambridge, MA, Tech. Rep. MIT-LCS-TR-917, 2003.

[22] K. Schmidt, E. Schmidt, and J. Zaddach, “A shared-medium communication architecture for distributed discrete event systems,” Control & Automation, Mediterranean Conference on, pp. 1–6, June 2007.

[23] M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham, “Multitasking supervisory control of discrete-event systems,” Journal on Discrete Event Dynamic Systems: Theory and Applications, vol. 15, pp. 375–395, 2005.

[24] K. Schmidt, M. de Queiroz, and J. Cury, “Hierarchical and decentralized multitasking control of discrete event systems,” Decision and Control, IEEE Conference on, pp. 5936–5941, Dec. 2007.

[25] D. E. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, “Timed I/O automata: A mathematical framework for modeling and analyzing real-time systems,” in 24th IEEE International Real-Time Systems Symposium, 2003.

[26] K. Schmidt, E. Schmidt, and J. Zaddach, Distributed real-time protocols for industrial control systems: Framework and examples, IEEE Transactions on Parallel and Distributed Systems, Accepted for publication,2011.

[27] Gerald W. Brock (2003-09-25). The Second Information Revolution. Harvard University Press. p. 151.ISBN 0674011783.

[28] A. K. Gozcu, Implementation and Evaluation of a Synchronous Time-Slotted Medium Access Protocol for Networked Industrial Embedded Systems, MSc. Thesis, Middle East Technical University, 2011.

[29] U. Turan, “Implementation and evaluation of a new protocol for industrial communication networks,” MSc Thesis, Middle East Technical University, 2011.

[30] Andrews (2000), p. 10–11. Ghosh (2007), p. 4–6. Lynch (1996), p. xix, 1. Peleg (2000), p. xv. Elmasri & Navathe (2000), Section 24.]

[31] Lian, F., Moyne, J. R., and Tilbury, D. M., “Performance Evaluation of Control Networks: Ethernet, ControlNet and DeviceNet”, IEEE Control Systems Magazine, pp. 66-83, Feb 2001.

[32] Paul Baran, 1964.

[33] Yusuf Bora Kartal, Ph.D Thesis Monitoring Committee Report, 2011.