Escola de Engenharia
Universidade do Minho

Nuno Alexandre Magalhães Pereira

A Framework for the Timing Analysis of Ethernet-based Factory-floor Networks

Braga, 2004

A Framework for the Timing Analysis of Ethernet-based Factory-floor Networks

by

Nuno Alexandre Magalhães Pereira

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science

Department of Informatics
University of Minho

December 2004

Thesis Supervision:
Dr. Eduardo Manuel de Médicis Tovar
Computer Engineering Department of the
Polytechnic Institute of Porto

Thesis co-Supervision:
Dr. Paulo Martins de Carvalho
Informatics Department of the
University of Minho

UNIVERSITY OF MINHO

ABSTRACT

A FRAMEWORK FOR THE TIMING ANALYSIS OF ETHERNET-BASED FACTORY-FLOOR NETWORKS

by Nuno Alexandre Magalhães Pereira

Throughout the years, researchers have developed and applied a considerable range of theory to the validation of factory-floor distributed real-time systems. Nowadays, some of those systems are based on Ethernet technologies. In fact, a number of characteristics are boosting the eagerness of extending Ethernet to also cover factory-floor distributed real-time applications. Full-duplex links, non-blocking and priority-based switching, bandwidth availability, just to mention a few, are characteristics upon which that eagerness is building up.

In the past few years, it is particularly significant the considerable amount of work that has been devoted to the timing analysis of Ethernet-based technologies. It happens, however, that the majority of those works are restricted to the analysis of subsets of the overall computing and communication system, thus without addressing the system as a whole. In fact, Ethernet technology, by itself, does not include features above the lower layers of the communication stack. Where are the higher layers that permit building real industrial applications? And, taking for free that they are available, what is the impact of those protocols, mechanisms and application models on the overall performance of Ethernet-based distributed factory-floor applications? This dissertation attempts to pave the way towards providing some reasonable answers to these issues.

To this end, a few analysis approaches are explored with the purpose of setting a framework for developing tools suitable to extract temporal properties of Commercial-Off-The-Shelf (COTS), Ethernet-based factory-floor communication systems. The particular case of Ethernet/IP is taken into the research work.

Two models, enabling finding end-to-end response times in Ethernet/IP based distributed systems are provided. The first model is an analytical model, built upon traditional real-time response time analysis, considering a number of worst-case assumptions to derive the end-to-end response time bounds. The second model is a discrete-event simulation model, providing an adequate solution to understand and analyse the timing behaviour of actual systems, also facilitating approaches for timeliness evaluation based on probabilistic measures of meeting deadlines. This may become relevant since modern factory-floor systems tend to be more flexible and adaptive in their nature. Additionally, results from applying both models are presented, and a discussion of the two is provided.

UNIVERSIDADE DO MINHO

RESUMO

ABORDAGENS À ANÁLISE TEMPORAL DE REDES BASEADAS EM ETHERNET PARA AMBIENTES INDUSTRIAIS

por Nuno Alexandre Magalhães Pereira

Ao longo dos anos, diversos investigadores construíram e aplicaram uma quantidade considerável de teoria à validação de sistemas tempo-real distribuídos, para ambientes industriais. Actualmente, alguns destes sistemas são baseados em tecnologias de comunicação Ethernet. Na realidade, existe um conjunto de características que estão a aumentar a veemência para colocar tecnologias de comunicação baseadas em Ethernet em ambientes industriais. Características como ligações *full-duplex*, comutação não bloqueante e baseada em prioridades são apenas alguns exemplos que justificam tal veemência.

É particularmente significante a quantidade considerável de trabalho desenvolvido nos últimos anos dedicado à análise temporal de tecnologias baseadas em Ethernet. No entanto, acontece que a larga maioria destes trabalhos, limitam-se à análise de subconjuntos do sistema de computação e comunicação, não considerando portanto uma visão sistémica. De facto, a tecnologia Ethernet, por si só, não inclui funcionalidades acima das camadas mais baixas da pilha protocolar de comunicações. Onde se encontram então as camadas superiores, que permitem construir aplicações concretas? Adicionalmente, assumindo que estas se encontram facilmente disponíveis, qual é o impacto, a um nível sistémico, da introdução desses protocolos, mecanismos e aplicações no desempenho das aplicações distribuídas baseadas em Ethernet? Esta dissertação empreende esforços no sentido de fornecer algumas respostas razoáveis a estas questões.

Para este fim, algumas abordagens para análise temporal são exploradas com o intuito de formar um enquadramento apropriado para o desenvolvimento de ferramentas e métodos adequados à extracção das propriedades temporais de redes Ethernet para ambientes industriais, baseadas em componentes disponíveis comercialmente. Neste trabalho é tomado o caso particular de sistemas Ethernet/IP distribuídos.

Foram concebidos dois modelos que permitem encontrar o atraso ponto-a-ponto em sistemas Ethernet/IP distribuídos. O primeiro é um modelo analítico, assente em conceitos tipicamente utilizados em análises de sistemas computacionais tempo-real, que consideram um conjunto de suposições sobre os cenários mais pessimistas de utilização dos recursos em causa, para derivar os limites máximos do atraso ponto-a-ponto. O segundo modelo é baseado em simulação discreta de eventos, possibilitando uma solução adequada para a compreensão e análise do comportamento temporal de sistemas concretos. Este segundo modelo facilita também a abordagem do problema de uma forma não determinística, facultando medidas probabilísticas do cumprimento dos atrasos máximos impostos ao sistema. Tais resultados tornam-se relevantes à luz da natureza mais adaptativa e flexível dos sistemas industriais modernos. Adicionalmente, são apresentados resultados obtidos a partir dos dois modelos, juntamente com uma discussão sobre ambos.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ARP	*Address Resolution Protocol*
CAN	*Controller Area Network*
CIP	*Control and Information Protocol*
CORBA	*Common Object Request Broker Architecture*
CoS	*Change-of-State*
COTS	*Commercial-Of-The-Shelf*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
CTDMA	*Concurrent Time Domain Multiple Access*
DCOM	*Distributed Component Object Model*
DEC	*Digital Equipment Corporation*
EDF	*Earliest Deadline First*
EIP	*Ethernet/IP*
FPS	*Fixed Priority Scheduling*
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IGMP	*Internet Group Management Protocol*
IP	*Internet Protocol*
IPC	*Inter-Process Communication*
Java/RMI	*Java Remote Method Invocation*
MAC	*Medium Access Control*
MAP	*Manufacturing Automation Protocol*
NED	*NEtwork Description*
OMNeT++	*Objective Modular Network Testbed in C++*
OPC	*OLE for Process Control*
OSI	*Open Systems Interconnection*
PRNG	*Pseudo-Random Number Generator*
RARP	*Reverse Address Resolution Protocol*
RIO	*Remote Input/Ouptut (IO)*
RM	*Rate Monotonic*
RPI	*Requested Packet Interval*
SNMP	*Simple Network Management Protocol*
TCP/IP	*The Internet protocol suite*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
TDMA	*Time Division Multiple Access*
WCET	*Worst-case Execution Time*
WWW	*World Wide Web*

ACKNOWLEDGMENTS

First of all I want to thank my supervisor, Dr. Eduardo Tovar, for his guidance, advice, support, and for always being available for insightful and enjoyable conversations. I would also like to thank Dr. Paulo de Carvalho for taking interest in my work and for is understanding through the development process of this dissertation.

The work leading up to this thesis was done at the IPP-HURRAY! Research Group, at the School of Engineering of the Polytechnic Institute of Porto. Working at IPP-HURRAY! Has been very stimulating and instructive, and the atmosphere in the group has given rise to several interesting discussions and ideas, on topics related to my work at different levels and otherwise interesting topics. The pleasant research atmosphere at the group is of course a product of all the members of the group. My appreciation to all of you. To Berta Batista, Dr. Mário Alves, Dr. Luís Ferreira, Filipe Pacheco and many thanks to Dr. Miguel Pinho.

Thanks also to Pedro Fortuna, for several interesting discussions and ideas on everything concerning computers.

My gratitude to my Parents, José and Odete, for all the support. And, finally to adorable Eduarda, for standing by me and making life more pleasant.

Chapter 1

OVERVIEW

1.1 Introduction

Today, high innovation-rate companies already make more than 60% of their profit on products less than two years old. Actually, in many industries, product lifecycles are halving every five years [1]. It is becoming less and less viable to sell from stock and have high value finished goods tying up capital. To satisfy the needs from a variety of clients, manufacturing companies are increasingly focusing on agility of operation. To serve these needs, rapid sequencing, configuration and reconfiguration of manufacturing equipment are essential.

In such context, organisational and supply chain agility are becoming key requirements for manufacturing in any sector. From order receipt, through manufacturing and product delivery, an enormous number of variants that need to be handled are introduced, turning information management a vital strategic asset for any manufacturing company today.

The factory-floor, being a central component of every manufacturing enterprise, is the starting point for greater information connectivity. Computer-based factory-floor controls for manufacturing machinery, materials handling systems and related equipment generate a wealth of information about productivity, product design, quality and delivery. Thus, factory-floor networking arises as a prominent building-block for unleashing this information in a cost-effective manner [2].

In a typical automated factory-floor, there will be a controlled system, for example robots or assembling stations and a controlling system that can include the computers and human interfaces that manage and coordinate the activities on the factory-floor. The controller system interacts with the controlled system based on the information available, collected from various sensors. It is imperative that the state of the controlled system, as perceived by the controlling system, is consistent with its actual state. Otherwise, the effects of the controlling systems' activities may lead to serious failures. Hence, periodic monitoring of the environment as well as *timely* processing of the sensed information is necessary. Noticeably, timely delivery support from the network is essential to have distributed automation applications in the factory-floor.

Consider, for example, a sub-system composed of a conveyer belt, where manufactured parts are carried after assembly. A robotic actuator is responsible for distributing the parts between several packaging stations. The information about where to deliver the parts is collected from another sub-system that gathers information about the available packaging stations. In such system, a timely coordination between the information collected from sensors that detect the parts on the conveyer belt, the packing sub-system and the robotic

actuator is essential for this system [3]. Failing to do so may result in the need for human intervention and possibly halting the production system.

A system where its correctness depends not only on the logical result of computation, but also on the time at which the results are produced is defined as a real-time computing system [4].

1.2 Research Context

The timeliness analysis of real-time systems is usually exploited in a framework dominated by the notion of absolute temporal guarantees. In those systems, computational and communication loads are presumed to be bounded and known, and the worst-case (at least believed to be) conditions are assumed. In this way, the problem of engineering distributed real-time systems, of which factory-floor distributed computing systems are a representative example, becomes a problem of devising the appropriate tools and methods to assure that all deadlines are met in all circumstances [5].

To this end there are generally three, usually alternative, approaches. The first consists of building a prototype of the system and perform extensive testing. Although this is conceptually a simple method, in practice, it is usually hindered by many difficulties. To build the prototype may take a considerable amount of resources and even once the prototype is built, it may be impossible to consider and analyse all possible interactions that affect the timing behaviour of the system.

Another option is to develop an analytical model of the worst-case timing behaviour of the system and draw conclusions based on the model. Much research has been done over the years to examine system behaviour based on several real-time system models. However, and for complex distributed systems, analytical models tend to be overwhelmed with simplifications that often lead to very pessimistic assumptions, and therefore to very pessimistic worst-case results. Even knowing that a number of existing techniques may potentially be used and adapted to reduce this pessimism level, the benefit may appear at the cost of adding rather complex abstractions, such as precedence relationships [6], event phasing [7, 8] and inheritance of time characteristics [9, 10]. These, unfortunately, may lead to intractable analytical models, thus making it further difficult to handle and reason the analytical abstractions.

To add on top of this, some characteristics of the system may not lend themselves to deterministic analysis techniques. For example, the arrival pattern of messages or communication times may not be completely deterministic and conversely can be characterised in a probabilistic manner. The emergence of “more complex” distributed systems, with a more flexible and adaptive nature creates the eagerness to approach the timeliness evaluation problem in a different way: instead of using a guaranteed approach, why not tackling the problem by trying to find a probabilistic measure of meeting deadlines?

It is in this context that simulation (the third alternative for the timeliness evaluation of a system) can emerge as an adequate solution to tackle the problem of engineering complex distributed real-time systems. On the other hand, the relatively recent advent of fast and inexpensive computational power allows the approach of trying to model the system as faithfully as possible, and then use simulation to obtain accurate characteristics. The use of

simulation models that mirror the behaviour of the system under analysis may provide a reasonable framework for the timeliness evaluation of such distributed real-time systems. In this dissertation, such approach is applied to a specific COTS technology, Ethernet/IP [11].

1.3 Hypothesis and Objectives

This dissertation explores the development of tools suitable to extract temporal properties of Ethernet-based factory-floor communication systems. This problem can be tackled using different and possibly inter-linked methods.

A major question to be answered is to what extent simulation can be used to extract useful timeliness results about the modelled systems.

1.4 Outline of the Dissertation

This dissertation will proceed by firstly presenting the technological context behind it. Details of the factory-floor technology chosen to support the analysis of an overall communication infrastructure is presented (Chapter 2). Then, in Chapter 3, some of the extant simulation tools for the development of network simulation models are surveyed.

In Chapter 4, the basis for the timeliness approaches followed is presented. Some introductory real-time systems concepts are laid out to introduce the foundations of traditional response time analysis, also applicable to the analysis of distributed systems. Then, the statistical grounds for adequately extracting performance measures from simulation output data are overviewed.

The next two chapters present the models developed for the timeliness analysis of Ethernet/IP-based distributed systems. These models are based on the two approaches introduced in Chapter 4. Therefore, in Chapter 5, a worst-case analytical model is proposed. In Chapter 6, a simulation model for Ethernet/IP, using a specific simulation tool is proposed.

The closing chapter (Chapter 7) of the dissertation presents a brief summary and conclusions of this work, along with a discussion on the results obtained by both models developed. It also introduces a number of topics for future research. Among these topics, the advancement of some options for the extraction of other measures of performance from simulation models, assumes a particular relevance.

1.5 Research Contributions

The main contributions of this dissertation are the following:

- an analytical model for EIP distributed systems, providing the worst-case end-to-end response time of distributed transactions [12];
- a simulation model for EIP distributed systems [13];
- approach to a framework for extracting measures of performance from simulation models, including performance measures other than means [14];
- a framework for the extraction of overall temporal properties of COTS factory-floor communication systems through the combination of different, but potentially integrated, types of analysis [15].

Chapter 2

TECHNOLOGICAL CONTEXT: COMMUNICATION INFRASTRUCTURE

2.1 Introduction

Over the last decade, factory-floor networking has evolved from relatively passive and isolated data collection or reporting roles to feedback control and diagnostics applications, integrated with enterprise-wide information systems. Modern industrial systems must be able to exploit commercial information technologies, including Commercial-Off-The-Shelf (COTS) operating-systems; TCP/UDP/IP based applications and general purpose networks.

However, it is interesting to observe that the development of factory-floor networking has been far slower than the development of office networks. One of the reasons for this delay is related with the timing requirements found in factory-floor applications. To have precise control over the data-sampling task, a common solution used to be employing point-to-point wiring. While being a simple solution, it is important to note that traditional point-to-point wiring is very limited in the information it transmits or receives from the field, because only the process variable is communicated, without any diagnostic or health information. The first step towards an actual factory-floor networking environment was the fieldbus concept. Fieldbusses are a cost-saving solution (being cabling one of the major cost components in any factory-floor installation) that provides much more information and flexibility. Nevertheless, fieldbus technologies present some important drawbacks. The fieldbus market is a small one (when compared to the office network market), where the prices are fairly high and the technology development is rather slow. Conversely, the office network market is rapidly evolving, with enormous data throughput increases and price drops of equal magnitude. Particularly, the high-speed properties of Ethernet, its familiarity and low cost make it a potential candidate for factory-floor communications.

Ethernet was, until recently, generally known for exhibiting unstable performance (e.g. unbounded delay) under heavy load. However, advances in switched-Ethernet made Ethernet more predictable, and have increased the eagerness to introduce Ethernet-based technologies into the factory-floor [16, 17].

Still, there are obstacles to overcome. Indeed, a few research efforts on Ethernet technologies have been focusing on timeliness, trying to find solutions to issues such as bounded response time evaluation, optimal scheduling policies, switching topologies or clock synchronisation [18]. However, they essentially consider the timing characteristics at the Data Link Layer, meaning that an overall approach embracing a fully defined protocol stack is still lacking.

While until a couple of years ago a valid justification for this gap could eventually be the lack of technologies offering an overall ensemble of protocols and mechanisms [19], this justification cannot serve that purpose anymore. In fact, there are already some COTS solutions for Ethernet-based systems providing a fully-defined communication protocol stack. One of such solutions, based on encapsulation technologies, is Ethernet/IP (EIP), where IP stands for "Industrial Protocol".

In the next section some further details will be surveyed on Ethernet technologies, in particular those characteristics enabling its use to support real-time distributed systems. Then, Section 2.3 is devoted to describing the more important details related to Ethernet/IP technologies.

2.2 Ethernet Review

Ethernet is a set of network cabling and signalling specifications originally developed by Xerox, in the late 1970. It was called Ethernet after the *luminiferous ether* as a way of describing an essential feature of the system: the physical medium (i.e., a cable) carries bits to all stations, in a way analogous to the *luminiferous ether* that once was thought to propagate electromagnetic waves through space.

In 1980, Digital Equipment Corporation (DEC), Intel and Xerox began joint promotion of this baseband, Carrier Sense Multiple Access/Collision Detection (CSMA/CD) computer communications network over coaxial cabling, and published the "Blue Book Standard" for Ethernet Version 1. This standard was later enhanced, and in 1985 Ethernet Version 2 was released.

The Institute of Electrical and Electronics Engineer's (IEEE's) Project 802 then used Ethernet Version 2 as the basis for the 802.3 CSMA/CD network standards. The IEEE 802.3 standard is generally interchangeable with Ethernet Version 2, with the greatest difference being the construction of the network packet header. For the sake of precision, it is important to point out that, in the context of this document, 803.2 would be a more appropriate terminology than Ethernet, when referring to the family of CSMA/CD-based Medium Access Control (MAC) protocols.

A complete description of all Ethernet specifications is far outside the scope of this dissertation, and for further details the reader should refer to the IEEE 802.3 standard [20].

Without going into details, the general idea of the CSMA/CD MAC protocol can be described in the following way. When a station wants to transmit, it listens to the cable. If the cable is busy, the station waits until it goes idle. Otherwise, it transmits immediately. If two or more stations begin transmitting on an idle cable simultaneously, the messages will collide. All colliding stations then terminate their transmission, wait a random time, and repeat the whole process all over again [21].

In a heavily loaded (in terms of traffic volume) network, a station can experience an unbounded number of collisions and, therefore, the time to transmit a frame is also unbounded, justifying the argument of non-deterministic behaviour frequently utilised against the use of Ethernet for distributed control applications in the factory-floor. Determinism enables systems designers to accurately predict the worst-case transmission delay. Another requirement for factory-floor networks is high repeatability (or low jitter);

That is, the guarantee that a periodic message is transmitted successfully almost periodically. This requirement was also difficult to attain with former Ethernet-based technologies.

Developments in Ethernet technology have improved the determinism, repeatability and performance of Ethernet to a great extent. Next, a briefly survey some of those developments is made.

A major step toward deterministic and repeatable behaviour in Ethernet networks resides in the elimination of the random behaviour of CSMA/CD, by avoiding collisions in the network. This can be achieved by using specialised hardware at the heart of the communication infrastructure with an array of *ports* to which all the communicating devices are connected to. This specialised hardware, called *switching hubs*, *layer 2 switches* or simply *switches*, allow traffic to be relayed between any two ports. Current switch technology does this operation at very high speeds and introducing, and extremely low latency.

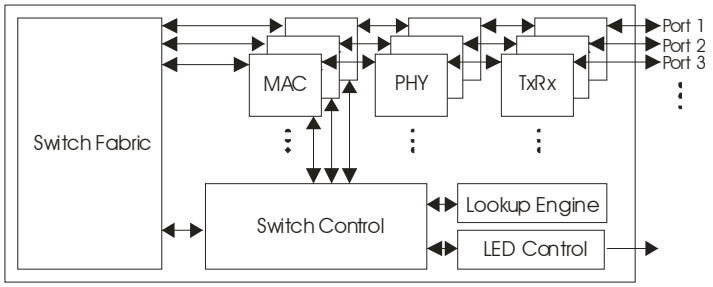


Figure 1. Typical modern switch internals

Most modern Ethernet switches (internals depicted in Figure 1) support full duplex operation, allowing simultaneous two-way transmission over point-to-point links. Since switches provide a separate collision domain for each port, using full-duplex communication, collisions do not exist at all.

Recent switches typically announce wire-speed and non-blocking operation. Wire-speed means that all ports of a switch can simultaneously transmit or receive at their full bit rates. This requires that the switch fabric can operate at a bit rate equalling to the aggregate speeds of all the ports. For example, 24 full-duplex ports operating at 200 Mbps (100 Mbps in each direction) implies a fabric switching at 4.8 Gbps (24×200 Mbps). A switch is non-blocking if it can forward a message to the destination port as long as that port is free, while a blocking switch may not be able to forward a message to a free port due to internal conflicts in the switch fabric.

If traffic is sent to an output port at a higher rate than its capacity, packets must be queued. Queuing exists in any switch, regardless of whether it is full wire-speed or not, and the analysis of the queuing delay depends on knowledge on the input traffic pattern. To alleviate switch queuing problems, support for message prioritisation (IEEE 802.1p) was introduced. The standard specifies a layer 2 mechanism for giving mission-critical data preferential treatment over non-critical data [20, 22]. The concept, driven by the multimedia

industry, is based on priority tagging of packets and implementation of multiple queues to discriminate packets. For tagging purposes, IEEE 802.3q [20] defines an extra field for the Ethernet MAC called *Tag Control Info* (TCI), containing 3 priority bits, thus the standard defines 8 different levels of priority (Figure 2).

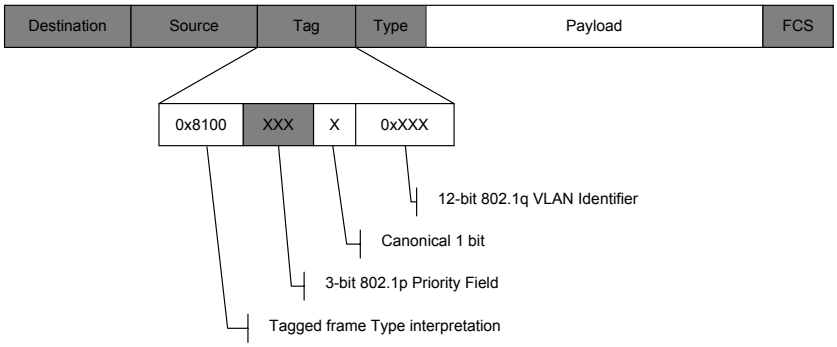


Figure 2. Ethernet MAC header with 802.1q tagging

Taking a closer look at Figure 2, it is possible to observe that the tag contains more than just priority bits. It also contains a 12-bit VLAN identifier. This field is used in advanced switches to allow logically separated networks - virtual LAN (IEEE 802.1q). VLANs permit configuring the switch so that its ports are subdivided into different broadcast groups, such that all packets received on one port of a group will only be transmitted to the ports on the same group, thus isolating broadcast traffic between logically separated networks (each group).

Several fault-tolerance mechanisms are available through spanning trees and port trunking. The Spanning Tree Protocol [20] can be used to provide redundant network paths, still protecting against network loops. Port Trunking establishes backbone links by treating multiple parallel links as a single network pipe. It also provides link redundancy, i.e., traffic on any failed link comprising a network trunk, automatically switches over to the other links in the trunk.

By itself, Ethernet only supports transmission of frames in a LAN. Ethernet lacks more complex features required for a fully functional LAN. Generally, Ethernet networks support one or more communication protocols that run on top of Ethernet and provide sophisticated data transfer and network management functions. It is the communication protocol that determines the level of functionality supported by the network. Protocols (such as AppleTalk, Inter-Process Communication (IPC) or Manufacturing Automation Protocol (MAP)) have been implemented over Ethernet. Of these, TCP/IP is the most popular, due to the emergence of the global *Internet*, including the *World Wide Web* (WWW). Although TCP/IP runs on physical media other than Ethernet, and Ethernet supports other communication protocols, the two have become increasingly linked.

Throughout the years, Ethernet has become a *de facto* standard, supporting many widely spread upper layer protocols like the TCP/IP stack, including the vast range of TCP/IP's stack application protocols such as FTP, HTTP or SNMP. This facilitates the use of Ethernet and allows easily integrating many COTS software components such as OLE for

Process Control (OPC), Microsoft Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), Java/Remote Method Invocation (RMI), and many others.

To summarise, today's Ethernet technology offers the following main interesting features for factory-floor networks:

- generous bandwidth (e.g. 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps);
- deterministic network access delay, due to switching principles and full-duplex links;
- priority handling (IEEE 802.1p), a basic support mechanism for real-time communication;
- broadcast traffic isolation and enhanced security through VLANS;
- reliability improved using Spanning Tree Protocol on redundant links;
- *de facto* standard supporting many widely spread upper protocol stacks.

2.3 Ethernet/IP (EIP)

2.3.1 General Aspects

Ethernet/IP (EIP), is a communication system suitable for use in industrial environments and time-critical applications [11]. It is an open industrial networking standard that takes advantage of COTS Ethernet communication chips and physical media, implementing a full suite of control, configuration and data collection services on top of an Ethernet network.

EIP makes use of an open protocol named Control and Information Protocol (CIP). CIP is an Application Layer protocol that implements a distributed object model. The CIP protocol specification [23] is quite extensive. Mainly, it defines the abstract *object modelling* used to describe the suite of communication services available, the externally visible behaviour of a CIP node and a common means by which information within a CIP-based network is exchanged. It also defines the *messaging protocol* used and the *communication objects* necessary to manage and provide run-time exchange of messages.

In addition, the CIP protocol specification also includes a fairly large collection of commonly used objects such as analogue Input/Output points, position sensor, AC/DC drive, etc, called the *General Object Library*. To avoid having devices of similar functionality from different vendors described with dissimilar object structures, devices of similar functionality are grouped into *device types*, with an associated *device profile* that describes the objects (some required, some optional) and the behaviour associated with that particular type of device.

The CIP protocol specification also made provisions for configuring the devices defining an *Electronic Data Sheet* (EDS) format to provide a full description of all configurable information of a device.

CIP is implemented on top of several different networks: DeviceNet [24], ControlNet [25] and Ethernet, allowing transparent application-level interoperability between factory-floor equipment. Figure 3 depicts CIP's common layering on top of the different networks with the corresponding mappings to the Open Systems Interconnection (OSI) reference model.

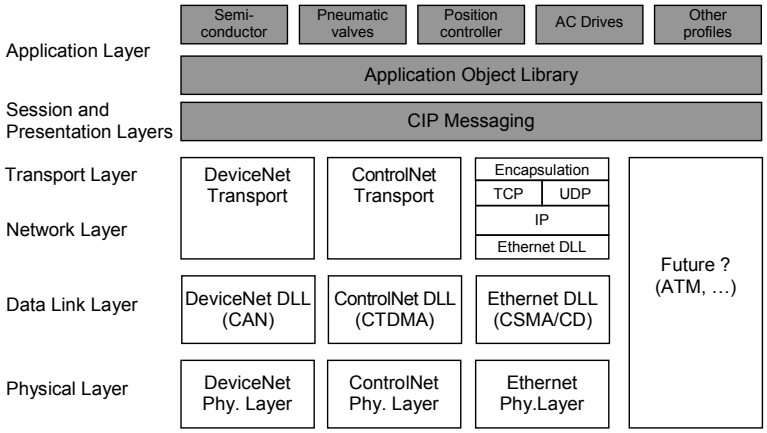


Figure 3. CIP common layering over different networks

DeviceNet [24] was the first member of this protocol family and it is a CIP implementation of the Controller Area Network (CAN) protocol layer. In its typical form, (ISO 11898, [26]) CAN only defines the Data Link Layer (DLL) and Physical Layer of the 7-layer OSI reference model, while DeviceNet covers the upper layers. The low cost of implementation and the ease of use of DeviceNet has led to a considerable popularity, nevertheless, it is limited to the small payload of the CAN protocol (8 bytes) and to the maximum 1 Mbps bandwidth obtainable with CAN.

ControlNet [25], introduced in 1997, essentially implemented the same protocol on a new physical layer, based on a specific method called Concurrent Time Domain Multiple Access (CTDMA). Weighed against CAN, CTDMA allows a higher throughput (5 Mbps), induces strict determinism and repeatability, while extending the length of the bus. ControlNet comes, however, with a fairly high price tag, being its usage restricted to the more demanding applications.

EIP was the latest addition to the CIP family. It is based on Ethernet and implements the CIP distributed object model using TCP/UDP/IP services. Figure 4 presents the relation of CIP to other protocols in a TCP/IP conceptual model.

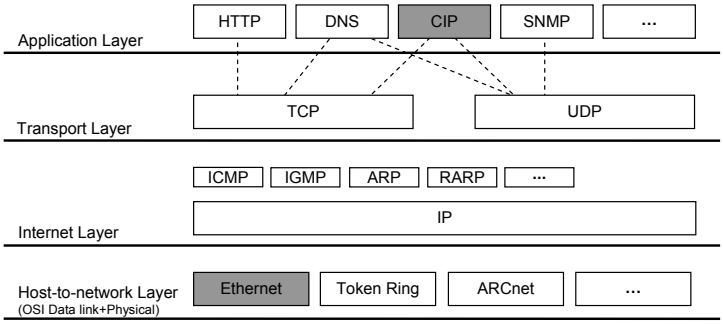


Figure 4. CIP and Ethernet in the TCP/IP layering model

2.3.2 CIP Messaging

In CIP-based networks the majority of the messaging performed is done through connections. *CIP connections* define the packets that will be produced on the network, and these can be of two types: *Explicit* or *Implicit messaging*.

Explicit messaging connections provide generic, multipurpose communication paths between two devices. Explicit messages provide the typical request/response-oriented network communication. Each request contains explicit information that the receiving node decodes, acts upon, and to which generates an appropriate response.

Implicit messaging connections provide dedicated, special purpose communication paths between a producing application object and one ore more consuming application objects. They are called implicit messages because the data that will be exchanged is identified at the time the connection is established and connection identifiers are assigned. Then, each transmission contains only the current values for the application objects that where agreed upon when the connection was established and the connection identifier, thus having a very small overhead.

There are four principal types of Implicit messages: *Polled, Strobed, Cyclic and Change of State (CoS)*. With polled messages, a device assumes the role of master and sequentially queries all of the slave devices by sending their output data and allowing them to reply with their input data. Strobed is a special case of polled in which the master sends out a single multicast request for data and the slaves sequentially reply with their data, requiring no further messages from the master. Cyclic messages are produced on a predetermined rate basis, defined by the *Requested Packet Interval* (RPI) parameter. In Change of State, as the name suggests, messages are only produced in response to an event which caused the data to change. Change of State also maintains a background cyclic rate so that consuming applications know that the node is still online.

Implicit messaging is the messaging used for time critical I/O data, and therefore will receive the focus of our attention, specially the Cyclic Implicit CIP connections.

2.3.3 Producer/Distributor/Consumer Model

As mentioned in the previous section, underlying CIP messaging is a *producer/distributor/consumer* model, replacing the more conventional *source/destination* (master/slave) model. The producer/distributor/consumer model is also usually found in other factory communication networks [27].

In a source/destination model (Figure 5), the source communicates with each destination, one at a time. Real-time data must be adjusted to maintain accuracy as communication takes place with each source, one at a time. Some of the destinations may not need the information, so there is some bandwidth waste. Moreover, the delivery time changes with the number of destination devices.

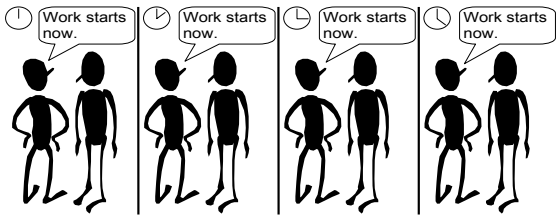


Figure 5. Source/destination model illustration

In the producer/distributor/consumer model (Figure 6) one producer broadcasts (multicasts) the data once to all the consumers. All consumers see the data simultaneously, and may choose whether to consume (receive) the data or not. Delivery time is consistent and bandwidth usage is optimised, no matter how many consumers exist.

Figure 6. Producer/distributor/consumer model illustration

In EIP networks, the distribution of messages is supported upon multicast UDP/IP that, in turn, is mapped onto Ethernet multicast.

In multicast UDP/IP, packets are not transmitted directly to the IP address of the destination node. Instead, they are transmitted to a specific address that identifies a group of nodes – an IP multicast address. Nodes may joint/leave groups using Internet Group Management Protocol (IGMP) messages. Generally, nodes will join multicast groups at connection establishment time, using information exchanged during this process.

The advantages of using multicast UDP/IP are twofold: firstly, it is the lightest transport layer, introducing the least amount of overhead for processing and transmitting each

message; secondly, multicast-based transmission facilitates the distribution of data to multiple destinations.

2.3.4 Ethernet/IP Devices

EIP networks are constituted by three structuring types of nodes: *Remote I/Os*, *Controllers* and interconnecting *Switches*. Diverse modules can compose the Remote I/O and Controller nodes (Figure 7). Typically, a Controller is composed of a number of *I/O modules* (labelled in the Figure 7 as I or O), several *Controller modules* (C) and one or more *Ethernet Adapters* (EA). A Remote I/O node has no Controller modules.

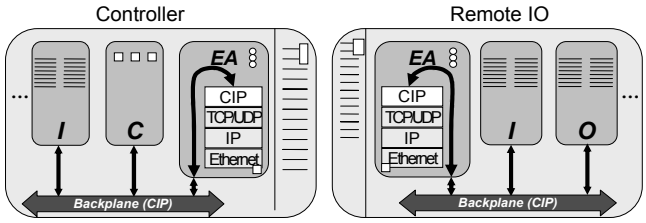


Figure 7. EIP basic nodes

Within each node, the several modules communicate among them via a device-specific backplane. The nodes communicate with each other via Switched Ethernet. Inside the node, communication is based on CIP messages and, when the messages are to be delivered to another node, these messages are encapsulated in TCP/UDP/IP packets by the Ethernet Adapter.

In the case of time critical data, as referred earlier, these messages are encapsulated in UDP/IP packets and delivered using multicast services. The periodicity of time critical data CIP connections is maintained internally in each producing module. Each module maintains a timer for the configured RPI of each connection.

In the following section, the diverse components of the end-to-end latency are introduced, leading to a first delineation of the end-to-end delay in EIP to be analysed. It is important to stress the fact that some of the architectural details and implementations are open to alternative options from technology providers. The descriptions presented are a result of information gathered from a technology provider, and of assumptions taken from the available information. The general assumptions made on the devices and network analysis are also within the next section.

2.3.5 Defining an End-to-End Transaction

The several components considered to makeup an end-to-end EIP transaction are illustrated using the following simple EIP network (Figure 8).

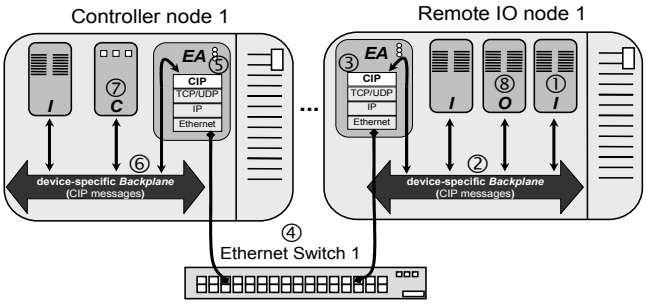


Figure 8. EIP end-to-end transaction

The EIP transaction considered is an independent transaction, starting at the input module of the Remote I/O (①). After the hardware delay to energise the input and a user defined filter delay, a message with the input data will be generated, at the periodicity defined for the input data connection. This message will then suffer the contention caused by the device backplane (②), and will arrive at the Ethernet Adapter, where it is processed, encapsulated and transmitted via the Ethernet communication interface (③). With this, the message arrives at the Ethernet switch, where it is relayed to the corresponding output port(s), and later will arrive at the Controller Ethernet Adapter (④). At the Ethernet Adapter (⑤), the message is processed, in order to be passed to the Controller module, passing through the Controller backplane (⑥). At the controller the input data will be processed by a controller task, characterised by a worst-case response time, that generates the corresponding output data (⑦). The output data will be transmitted at a defined periodicity and will go back through the inverse path (⑥,⑤,④), until it reaches the Ethernet Adapter of the Remote I/O (③), is processed and delivered to the output module that will, in result, energise the corresponding output(s) (⑧).

2.3.6 Assumptions

Before continuing, a few words on assumptions are worthy to be provided. Throughout the development of the EIP models to be later described in this dissertation, a number of assumptions where introduced in order to narrow, in some reasonable way, the number of variables needed to be accounted for. Additionally, these assumptions also reflect some of the implementation details specific of the considered EIP devices.

Thus, the following assumptions will be considered:

- traffic in the network is restricted and isolated;
- processing and network traffic related to the network setup period is negligible;
- only Cyclic Implicit CIP connections are assumed to exist in the network, and their periods are known;

- the controller tasks are independent, execute periodically and have a bounded worst-case response time;
- during task execution, the input data is processed and corresponding output data is generated once;
- packet processing time in the Ethernet Adapter is characterised;
- time to transfer a frame in the Backplane is characterised;
- the input filter delay is a known variable, defined by the user;
- output module hardware delay is negligible;
- network data rate is known;
- switch latency and other switch processing delays are characterised;
- propagation delays are ignored.

2.4 Summary

Characteristics like generous bandwidths, switching technologies, priority handling and support for widely spread upper protocol stacks are driving an increasing eagerness for extending Ethernet to also cover factory-floor distributed real-time applications. This chapter exposed the basics of Ethernet technology and the developments of factory-floor communication systems driving that eagerness.

Additionally, the Commercial-Off-The-Shelf (COTS) factory-floor communication system to be investigated – Ethernet/IP (EIP) – is introduced in this chapter. The fundamentals of CIP, the application layer protocol used by EIP, were brought out, and some details of the EIP devices considered for the analysis were conveyed. Finally, the assumptions included in the analysis to be presented later in this dissertation were enumerated.

Chapter 3

TECHNOLOGICAL CONTEXT: SIMULATION SOFTWARE

3.1 Introduction

Simulation is basically the imitation of the operation of a real-world system over time. The availability of special-purpose simulation languages, increasing computing capabilities at a decreasing cost per operation and advances in simulation methodologies, have made simulation one of the most accepted tools in operations research and systems analysis [28].

Simulation, for the study of any system, usually involves the development of a model, where the details and behaviour that affect the system under study are represented. A model of a system can be classified, according to its nature and system modelled, into several different types. Simulation models can be classified as being *static* or *dynamic*, *deterministic* or *stochastic*, and *discrete* or *continuous*. A static simulation model represents a system at a particular point in time, while dynamic simulation models represent systems as they change over time.

Simulation models that contain no random variables are classified as deterministic. Deterministic models have a known set of inputs, which will result in a unique set of outputs. On the other hand, a stochastic model has one or more random variables as inputs, which result in random outputs.

A system can be classified as discrete or continuous, according to the way its state variables change. The state of a system is defined to be the collection of variables necessary to describe the system at any time, relative to the objectives of the study. When the state variables change only at a discrete set of points in time, the system is classified as discrete. Conversely, in a continuous system, the state variables change continuously over time. In practice, very few systems are strictly discrete or strictly continuous, but since one type predominates for most systems, it is usually acceptable to classify a system as either being discrete or continuous [28].

In the case of discrete-event simulation, the model is analysed by numerical rather than by analytical methods. Analytical methods employ the deductive reasoning of mathematics to *solve* the model. Numerical methods employ computational procedures to *solve* analytical models. In the case of simulation models, which employ numerical methods, models are *executed* rather than *solved*. That is, an artificial history of the system is generated based on the model assumptions, and observations are collected to be analysed and to estimate the true system performance measures.

Predominantly, computer systems and communication networks are described by state variables that change discretely, justifying the choice for developing discrete-event simulation models.

The considerable amount of established techniques for its development and analysis and the large number of software packages readily available for this type of simulation, are just some of the additional characteristics that further corroborate this option. Some simulation tools will be overviewed throughout this chapter.

Generally, the implementation of discrete-event simulation models encompasses a number of common features such as:

- generation of random numbers;
- advancing simulated time;
- maintaining a list of events;
- determining the next event from the list;
- passing control to the appropriate block of code;
- collecting output data;
- detecting error conditions.

These features are, in reality, so common in the implementation of discrete-event simulation models that they have led to the emergence of special-purpose simulation software tools that provide these common facilities for the implementation of simulation models. These software tools can be generally classified in two distinct categories: simulation languages and simulation packages. The latter can be divided in application-oriented and general-purpose simulation packages.

A deeper debate of this matter shall not be attended. As this dissertation is focusing on the simulation of a communication network, a discussion of application-oriented simulation packages for this purpose will suffice.

3.2 Simulation Languages

Simulation languages provide maximum flexibility for the simulation developer who wants to construct simulators by means of programming. Because most simulation languages have expressive power equivalent to a general-purpose programming language, the simulation developer has great flexibility in designing and implementing the simulator. The trade-off for this flexibility is the development effort required to program the simulator.

Much work has been done at the simulation language level, either in the form of true languages or as function libraries. Some examples of freely available simulation languages in use today are briefly addressed below.

PARSEC (PARallel Simulation Environment for Complex systems) [29] is a C-based simulation language developed by the Parallel Computing Laboratory from the University

of California, Los Angeles, for sequential and parallel execution of discrete-event simulation models. It is available in binary form only for academic institutions.

SMURPH (System for Modelling Unslotted Real-time PHenomena) [30] is intended for simulating communication protocols at the medium access control (MAC) level. SMURPH can be viewed as a combination of a protocol specification language based on C++ and an event-driven, discrete-time simulator that provides a virtual and controlled environment for protocol execution. SMURPH can be used for designing low-level communication protocols and for investigating their quantitative and qualitative properties.

SIMSCRIPT [31] is a simulation language with both declarative and procedural features, designed for discrete-event and hybrid discrete/continuous modelling. It has been in continuous use and development since its invention in 1962. The syntax and semantics of SIMSCRIPT II are designed to make simulation programs easy to write and understand. The language syntax is "English-like" and fairly high-level. Today, SIMSCRIPT II.5 is more than a simulation language, being a commercial product offered by CACI Products Company. An important contribution from SIMSCRIPT is its considerable impact on the development of SIMULA (SIMUlation LAnguage), through its list processing, time scheduling mechanisms, random drawing and other utility routines.

The SIMULA [32] programming language was designed and built at the Norwegian Computing Centre (NCC) in Oslo between 1962 and 1967. It was originally designed and implemented as a language for discrete event simulation, but was later expended and re-implemented as a full scale general purpose programming language. Although SIMULA never became widely used, the language has been highly influential on modern programming methodology.

CSIM/C++SIM [33] is a programming tool for simulation of discrete processes. It is an extension of the C language obtained by including SIMULA-like possibilities by means of C macros and functions. CSim uses a special C-functions and C-macros library. The typical application area of CSim is functional validation of distributed, parallel and fault-tolerant systems and programs. Similarly, C++SIM is a collection of C++ libraries.

3.3 Network Simulation Packages

Network simulation packages provide a more comprehensive support than simulation languages. They include the basic constructs for the development of network simulation, typically require less programming effort and have a smoother learning curve, when compared to simulation languages. Many network simulation packages include some type of pre-built and reusable models of networking protocols, devices and applications. Additionally, they also provide means for using and creating user interfaces to the simulation models, facilitating their development, debugging and understanding.

There exist several examples of such simulation packages. Some characterisation to a number of these is provided next.

OPNET [34] is widely held as the state-of-art in network simulation. It is a suite of products that combines predictive modelling and a comprehensive understanding of networking technologies to enable design, deployment, and management of network infrastructures, network equipments, and networked applications. In particular, OPNET

Modeller is a development environment, allowing to design and study communication networks, devices, protocols, and applications. OPNET is a commercial product, although it provides some academic licensing programmes, albeit with some restrictions.

NetSim [35] is intended to offer a very detailed simulation of Ethernet, including realistic modelling of signal propagation, the effect of the relative positions of stations on events on the network, the collision detection and handling process and the transmission deferral mechanism. However, its development has stagnated and it is infeasible its extension in order to address modern networks.

CNET [36] is a discrete-event network simulator enabling experimentation with various data-link layer, network layer, routing and transport layer networking protocols. It has been specifically developed for, and used in, undergraduate computer networking courses taken by thousands of students worldwide.

Ns-2 (Network Simulator 2) [37] is a discrete event simulator targeted at networking research. Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Ns began as a variant of the REAL [38] network simulator in 1989, and has evolved substantially over the past few years. The full source code of ns-2 can be downloaded and it can be compiled in multiple platforms, including the most popular UNIX flavours and Windows.

OMNeT++ (Objective Modular Network Testbed in C++) [39], is a public-source, object-oriented modular discrete event simulation package that can be used for modelling communication protocols, computer networks, traffic modelling, multiprocessors and distributed systems. OMNeT++ also supports animation and interactive execution.

The previous list is not, by any means, comprehensive. It presents the several options considered for the development of the work presented in this dissertation. Among the simulation packages described, only ns-2 and OMNeT++ were assessed has possible solutions for the use in the work described by this dissertation. While ns-2 is a network simulation classic, it has many drawbacks, when compared with OMNeT++, which is a more modern and structured simulation package. The following summarises a number of advantages from OMNeT++ over ns-2.

- the OMNeT++ simulation kernel is a class library: the components are developed as any other class library, and then linked with the executable library. There is no need to modify OMNeT++ sources anywhere. In contrast, ns-2 tends to be a bit monolithic: to add implementations to it, it is necessary to download the full source and modify it in several places;
- OMNeT++ follows a modular approach: the model is assembled from self-contained building blocks. These components are reusable “as is” in other simulations;
- ns-2 has some considerably detailed built-in concepts about nodes, agents, protocols, links, packet representation, network addresses, etc. This often increases the difficulty in developing models that include even slightly different concepts. OMNeT++ is completely flexible and generic: it is possible to simulate anything that can be mapped to active components that communicate by passing messages;

- in OMNeT++, it is possible to fight model complexity by using hierarchical design: any complex component can be implemented as one unit or built out of several smaller components. In ns-2, models are "flat":
- OMNeT++ has a powerful interactive graphical environment, where it is possible to examine nearly everything during execution. Ns-2 only includes Network AniMator (NAM), which is little more than a playback tool.

3.4 A Few More Details on OMNeT++

OMNeT++ is a discrete event simulation package written in C++ with a primary application area in the simulation of computer networks and other distributed systems. The OMNeT++ simulation models are composed of hierarchically nested modules that communicate with message passing. Modules at the lowest level are programmed using C++, while the model structure is defined by a topology description language. Using this topology description language, modules can be combined and reused flexibly.

The package contains the C++ simulation kernel library, a manual, a simulation kernel API reference, a graphical topology editor, a graphical runtime environment with interesting animation and tracing capabilities, as well as a command-line runtime environment for batch execution. It also includes several other tools and sample simulations.

One of the strengths of OMNeT++ is that one can execute the simulation under a graphical user interface with interesting features. The GUI makes the internals of a simulation model fully visible to the person running the simulation: it displays the network graphics, animates the message flow and lets the user peek into objects and variables within the model. The use of the tracing/debugging capabilities does not require extra code to be written by the simulation programmer.

OMNeT++ already contains detailed IP, TCP, and FDDI protocol models and several other simulation models (file system simulator, Ethernet, a framework for simulation of mobility, etc.). However, the simulation model suite for OMNeT++ has not crystallised yet and many of these models are still under development.

OMNeT++ is open source, free for non-profit usage, and has an active user community. It has been tested on Linux, Solaris, Windows and Mac OSX. The Web site [40] provides source code, binaries, documentation, mailing lists, a Web-based discussion forum and information on workshops. OMNeT++ modelling concepts will be briefly described now.

An OMNeT++ model is composed of hierarchically nested modules which communicate with messages. The top level module is the *system module* (often called *network*). The system module contains *sub-modules*. These sub-modules may be of two different types: *compound modules* and *simple modules*. Modules that can contain sub-modules are termed compound modules. These may contain an unlimited nesting of sub-modules. Conversely, modules that do not contain any sub-modules are called simple modules and are at the lowest level of the module hierarchy (Figure 9).

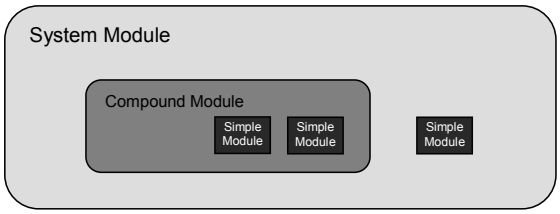


Figure 9. OMNeT++ module hierarchy

Simple OMNeT++ modules contain the algorithms of the models. These are implemented as C++ classes derived from a simple module base class, by redefining the virtual function that contains the algorithm. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library.

Modules communicate by exchanging messages. In a simulation, messages can represent frames or packets in a communication network, jobs or customers in a queuing network, or other types of mobiles entities. OMNeT++ class library includes a message base class. This class can be extended to arbitrarily represent any type of mobile entity needed for the simulation model.

Simple modules can send messages either directly to their destination or along a predefined path, through *gates* and *connections*. Gates are the input and output interfaces of modules. Messages are sent out to *output gates* and received through *input gates*. Each connection is created within a single level of the module hierarchy. In a compound module one can connect the corresponding gates of two sub-modules, or a gate of one sub-module and a gate of the compound module (Figure 10). Due to the hierarchical structure of the model, messages typically travel through a series of connections, to start and arrive in simple modules.

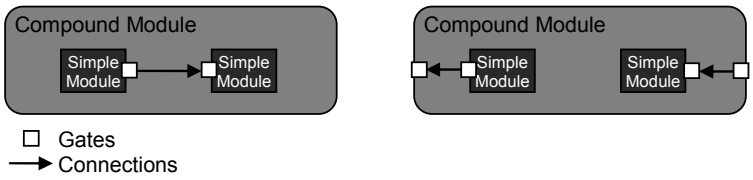


Figure 10. OMNeT++ gates and connections

To help modelling transmission channels and packet transmissions, connections can be assigned three parameters: *propagation delay*, *bit error rate* and *data rate*. All these three are optional. One may specify link parameters individually for each connection, or define link types and use them throughout the whole model. For example, by defining the data rate of a connection, it is possible to model the transmission time of a packet by using the size attribute from the message class that represents the packet.

The structure of the modules (both simple and compound) is defined using NEtwork Description (NED). The NED language supports the definition of the network's topology in a modular fashion. A network description consists of a number of component descriptions (channels, simple/compound module types). The channels, simple modules and compound

modules of one network description can be reused in another network description. As a consequence, the NED language allows users to build their own module libraries.

All modules can have parameters that can be used to parameterise the module topology, customise simple module behaviour, or for module communication. Parameters can be numeric values, expressions using other parameters, calling of C functions, random variables from different distributions, and values input interactively by the user.

3.5 Summary

Simulation may present itself as an appealing option for analysing the timing properties of EIP-based distributed systems. This chapter introduced some basic concepts and options on simulation software. The progress of the chapter sustained the choice for the OMNeT++ simulation package, to which some further details were provided.

Chapter 4

APPROACHES FOR TIMELINESS ANALYSIS

4.1 Introduction

In this chapter, basic concepts of real-time systems are laid out to introduce the foundations of traditional real-time response time analysis, some of which may also be applicable to the analysis of distributed systems. The next two sections addresses concepts firstly associated with single processor systems, which are then adapted and extended to be applied in the analysis of distributed systems.

In this chapter some aspects related to the application of simulation-based approaches to perform timeliness analysis will be also covered. An essential component of a simulation-based analysis is the development of accurate simulation models and the adequate exploration of the produced output data. In section 4.4, both of these issues are addressed.

4.2 Basic Concepts of Real-time Systems

Real-time computing systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [41]. There are various examples of real-time computing systems, such as command and control systems, flight control systems or robotics.

A typical real-time computing system has a real-time program running on the system, which reads inputs from input devices, processes these inputs, and often produces outputs to be sent to output devices. The time between the arrival of an input from a device and the completion of the processing for that input is called the *response time* for the device [42]. The *relative deadline* for the device can be defined as the maximum admissible interval between the instant of the input arrival and the completion of the processing for that input. Hence, the response time for a device must be smaller or equal to its relative deadline.

Assume that each input device is assigned a task (process) of the application program and that the tasks share a same processor. The problem of determining whether the system will meet its peak processing load, or in other words, whether no input from any device will be lost, becomes one of schedulability analysis of tasks [43].

A round-robin scheduling policy ensures that each task gets a share of the processor. However, such an approach may not be suitable for real-time systems. Assume the following example [44]: “Consider a computer controlling an aircraft. Among its tasks are maintaining stability and keeping the cabin temperature within acceptable limits. Suppose the aircraft encounters turbulence that makes it momentarily unstable. The computer is then supposed to adjust the control surfaces to regain stability. If we use round-robin scheduling

for this application, the computer may switch context partway through making the control adjustments in order to spend time making sure the cabin temperature is just right. The result may well be a crash, and the fact that the cabin is being maintained at optimum temperature will be scant consolation to the passengers as the airliner falls out the sky. What we want is to give the stability-maintenance task a very high priority, which ensures that when stability is threatened, all other interfering tasks are elbowed out of the way to allow this all-important task enough computer cycles."

It follows that the consideration of priority levels is crucial to a real-time computing system. If different inputs have different response time requirements, we need to consider different *priority levels* to *schedule* the related processing tasks. Consider a real-time system, within which several devices are connected at different priority levels to a single processor computer system. An input being processed will be preempted when another input of higher priority arrives, and will only be resumed when there is no processing remaining at higher priorities.

Assume that the input from a device is saved in a buffer, until it is overwritten by the next input of the same device. The problem is to determine whether for a given assignment of priority levels, the system will meet its peak processing load (i.e. no input from any device will be lost). A more basic problem is how to assign devices to priorities in order to meet the system-processing load.

4.2.1 Characterisation of Tasks

There is a number of attributes related to a task in a real-time system, typically including the following:

- C, the worst-case execution time (WCET) of the task;
- T, for periodic tasks it is the minimum time between arrivals of instances of the same task;
- D, the relative deadline of the task, i.e., the maximum time allowed between the release of the task and completion of its execution;
- P, priority level assigned to the task;
- B, longest time a task may be blocked by a lower priority task;
- R, worst-case response time of a task (most schedulability analyses try to verify if $R < D$).

To illustrate these attributes, consider a task (Γ_1) that is released periodically (every 9 time units) to perform some kind of processing. The worst-case execution time (C) of the task is 3 time units, and its relative deadline (D) is 6. Several instances ($\Gamma_{1,1}$, $\Gamma_{1,2}$,..., $\Gamma_{1,n}$) of task Γ_1 are depicted in Figure 11.

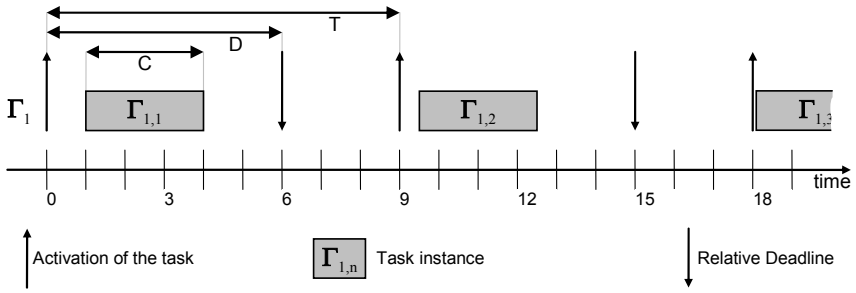


Figure 11. Illustration of task attributes

Tasks can also be characterised according to their predictability. This characteristic of the tasks affects their schedulability analysis. Concerning the predictability, three basic types of tasks can be defined: *periodic*, *aperiodic* and *sporadic*.

Periodic tasks, as their name implies, are released on a regular basis. They are characterised by their period, their deadline and their required execution time per period. The deadline is often assumed to be equal to the period, i.e. the processing of an input must be completed, at most, before the next input from the same device.

Aperiodic tasks are released only occasionally, and are usually triggered by an external event. To allow worst-case calculations to be made, a minimum period between any two aperiodic inputs (from the same device) is often defined. If this is the case, the task involved is said to be sporadic, and its period corresponds to its minimum inter-arrival time.

Tasks can also be characterised according to their criticality, depending on the consequences of not being executed before their deadlines. Concerning their criticality real-time tasks can be *soft*, *hard* or *safety-critical*.

Real-time tasks are said to be soft is meeting its deadline is desirables for performance reasons, but missing a deadline does not cause serious damage and does not jeopardise the correct system behaviour. Conversly, hard real-time tasks are those whose timely execution is critical. If deadlines are missed, severe faults may occur in the system. If the fault is catastrophic, the task is said to be a safety-critical real-time task.

4.2.2 Scheduling Tasks in Real-time Systems

Scheduling involves the allocation of time (and resources) to tasks, in such a way that timing requirements (or other performance requirements) are met. Scheduling has been perhaps the most widely research topic within real-time systems. As a consequence, there are multiple taxonomies for the *scheduling schemes* and for the methodologies for the *schedulability analysis*.

In a single processor computing system, a set of tasks shares a common resource: the processor. Schedulability analysis has to be performed to predict whether the tasks will meet their timing constraints.

The schedulability analysis can be performed *online* or *offline*. In the first case, the schedulability of the task set is analysed at run-time, whereas in the latter it is performed prior to run-time (pre-run-time schedulability analysis).

The offline scheduling has several advantages over the online scheduling: it requires little run time overhead and the schedulability of the task set is guaranteed before execution. However, it requires a prior knowledge of the tasks' characteristics, which fortunately is possible in most of real-time systems. If the tasks' characteristics are not known prior to run time, schedulability analysis must be performed online.

The most used type of offline scheduling is the priority-based approach, where no explicit schedule is constructed. At run-time, tasks are executed in a highest-priority-first basis. Priority-based approaches are much more flexible and accommodating than other approaches.

4.2.3 Priority Assignment Schemes

One of the most used priority assignment schemes is to give the tasks a priority level based on its period: the smaller the period (T), the higher the priority (P); that is, $T_i < T_j \Rightarrow P_i > P_j$. This assignment is intuitively explained by the fact that more critical devices will provide inputs more frequently (via asynchronous interrupts), or will be polled more frequently. Thus, if they have smaller periods, their worst-case response time must also be smaller. This type of priority assignment is known as the *rate monotonic* (RM) assignment, and the related pre-run-time schedulability analysis was firstly introduced in [45].

If some of the tasks are sporadic, it may not be reasonable to consider the relative deadline equal to the period. A different priority assignment can then be to give the tasks a priority level based on its relative deadline: the smaller the relative deadline (D), the higher the priority; that is, $D_i < D_j \Rightarrow P_i > P_j$. This type of priority assignment is known as the *deadline monotonic* (DM) assignment [46].

Both RM and DM priority assignments belong to the group of *fixed priority scheduling (FPS)* mechanisms, in the sense that priorities do not vary along time. At run-time, tasks are dispatched highest-priority-first. A similar dispatching policy can be used if the task, which is chosen to run, is the one with the earliest deadline. This also corresponds to a priority-driven scheduling, where the priorities of the tasks vary along time. Thus, the *earliest deadline first* (EDF) is a dynamic priority assignment scheme. Pre-run-time schedulability analysis for tasks dispatched according to the EDF assignment scheme was also introduced in [45].

In all three cases, the dispatching phase will take place either when a new task is released or the execution of the running task ends.

In a priority-based scheduler, a higher-priority task may be released during the execution of a lower-priority one. If the tasks are being executed in a preemptive context, the higher-priority task will preempt the lower-priority one. Contrarily, in a non preemptive context, the lower-priority task will be allowed to complete its execution before the higher-priority task starts execution. This situation can be described as a priority inversion due to non preemption (a higher-priority task is delayed by a lower-priority one). This is also known as blocking.

To illustrate, both RM and EDF scheduling, consider the following task set:

Table 1. Example task set

Task	*C (ms)*	*T=D (ms)*
1	1	5
2	5	12
3	4	14

Figure 12 illustrates a time-line of the schedule for this task set, assuming that all of them share a common initial release time (at time instant 0), and the tasks are preemptable.

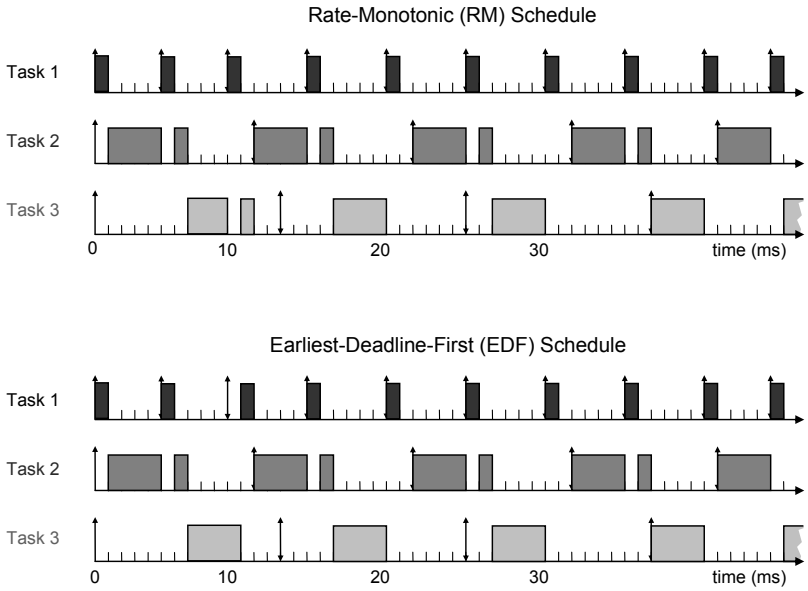


Figure 12. RM and EDF schedule examples on 1 Processor

4.3 Analytical-Based Timing Analysis

Real-time computing systems with tasks dispatched according to a priority-based policy (only RM and DM will be considered) must be tested *a-priori* in order to check if, during run time, no deadline will be lost. This feasibility test is called the *pre-run-time schedulability analysis* of the task set.

It can be shown that for periodic tasks, a set of tasks is schedulable if and only if there is a feasible schedule for the LCM (least common multiple) of the periods [47]. Moreover, it can also be shown that if the tasks share a common request time (known as the critical instant), it is a pre-run-time schedulability sufficient condition that the tasks are schedulable for the longest of the periods [45]. This suggests that a time-line could be used to perform the schedulability analysis. For instance, and concerning the example shown in the previous section, where the longest period is 14, Figure 12 shows that the schedule generated by both RM and EDF schemes are feasible for the task set (if all the tasks share a common initial

release time). However, time-line approaches may not be effective for systems with a large number of tasks. Hence, analytical methods are preferable.

There are mainly two types of analytical methods to perform pre-run-time schedulability analysis. One is based on the analysis of the *processor utilisation*. The other is based on the *response time analysis* for each individual task. In [45], the authors demonstrated that by considering only the processor utilisation of the task set, a test for the pre-run-time schedulability analysis could be obtained. Contrarily, a response time test must be performed in two stages. First, an analytical approach is used to predict the worst-case response time of each task. The values obtained are then compared, trivially, with the relative deadlines of the tasks.

The utilisation-based tests have a major advantage: it is a simple computation procedure, which is applied to the overall task set. By this reason, they are very useful for implementing schedulers that check the feasibility online. However, utilisation-based tests have also important drawbacks, when compared with their response-time counterparts. They do not give any indication of the actual response times of the tasks. More importantly, and apart from particular task sets, they constitute sufficient but not necessary conditions. This means that if the task set passes the test, the schedule will meet all deadlines, but if it fails the test, the schedule may or may not fail at run-time (hence, there is a certain level of pessimism). It is also worth mentioning that the utilisation-based tests cannot be used for more complicated task models [48].

In the next sub-sections, the most relevant feasibility tests for task sets scheduled with fixed priority schemes for both preemptive and non preemptive contexts will be surveyed. Depending whether the tests are applied to the overall task set or individually to each task, they are classified as utilisation-based tests or response time tests, respectively.

4.3.1 Utilisation-Based Tests

For the RM priority assignment, Liu and Layland [45] introduced a utilisation-based pre-run-time schedulability test, which, when satisfied, guarantees that tasks will always be completely executed before their deadlines:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \times \left(2^{1/N} - 1\right) \qquad (1)$$

with N being the number of tasks in the system.

This utilisation-based test is valid for periodic independent tasks, with relative deadlines equal to the period, and for preemptive systems. As mentioned in the previous section, typically the utilisation-based tests are sufficient but not necessary conditions. For instance, for the task set shown in Table 1, the test fails (from Equation (1), $0.90 < 0.78$ is false), but the task set is schedulable, as can be seen by the time-line of Figure 12.

Formulations for the utilisation-based tests with deadlines smaller than periods are not available, to our best knowledge. It is however possible to formulate a simple utilisation-based test for the case of non preemptive tasks.

In [49], the authors update the basic utilisation based test (1) to include blocking periods, during which higher-priority tasks are blocked by lower-priority ones, to solve the

problem of non-independence of tasks (for instance tasks that share resources which are protected by mutual exclusion):

$$\left(\sum_{i=1}^{i}\frac{C_i}{T_i}\right)+\frac{B_i}{T_i}\le i\times\left(2^{1/i}-1\right),\quad \forall_{i,\ 1\le i\le N} \tag{2}$$

where B_i is the maximum blocking a task i may suffer [49]. Equation (2) assumes that $P_{i+1} \le P_i$, $\forall_{i<N}$; that is, tasks are ordered by decreasing priority.

In a non preemptive context, a higher-priority task can also be "blocked" by a lower-priority task. Assuming that the tasks are completely independent, the maximum blocking time a task may suffer is given by:

$$\begin{cases} B_i = 0, & \text{if } P_i = \min\limits_{j=1,..,N}\{P_j\} \\ B_i = \max\limits_{j\in lp(i)}\{C_j\}, & \text{if } P_i \ne \min\limits_{j=1,..,N}\{P_j\} \end{cases} \tag{3}$$

where *lp(i)* denotes the set of lower-priority tasks (than task *i*).

Therefore, Equation (2) can be used as an utilisation-based test for a set of non preemptable but independent tasks, with the blocking for each task as given by Equation (3).

4.3.2 Response Time Tests

In [42] the authors proved that the worst-case response time R_i of a task i is found when all tasks are synchronously released at their maximum rate. This is known as the *critical instant*. In such case, R_i is defined as:

$$R_i = I_i + C_i \tag{4}$$

where I_i is the maximum interference that task i can experience from higher-priority tasks in any interval $[t, t + R_i]$. The maximum interference (I_i) occurs when all higher-priority tasks are released synchronously with task i (the critical instant). Without loss of generality, it can be assumed that all processes are released at time instant 0.

Consider a task j with higher-priority than task i. Within the interval $[0, R_i]$, it will be released $\lceil R_i/T_j \rceil$[1] times.

Therefore, each release of task j will impose an interference of C_j. Hence, the overall interference is given by:

---

[1] The ceiling function $\lceil x \rceil$ returns the smallest integer greater than or equal to x. Similarly, the floor function $\lfloor x \rfloor$ is used to denote the larger integer smaller than or equal to x.

$$I_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \quad (5)$$

where $hp(i)$ denotes the set of higher-priority tasks (than task i). Substituting this value back in Equation (4), the worst-case response time R_i of a task τ_i is given by:

$$R_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad (6)$$

Equation (6) embodies a mutual dependence, since R_i appears in both sides of the equation. In fact, all the analysis underlay this mutual dependence, since in order to evaluate R_i, I_i must be found, and vice-versa. To solve such equation, a recurrence relationship must be formed [50]:

$$W_i^{m+1} = \sum_{j \in hp(i)} \left(\left\lceil \frac{W_i^m}{T_j} \right\rceil \times C_j \right) + C_i \quad (7)$$

The recursion ends when $W_i^{m+1} = W_i^m = R_i$, and can be solved by successive iterations starting from $W_i^0 = C_i$. Indeed, it is easy to show that W_i^m is non-decreasing. Consequently, the series either converges or exceeds T_i (in the case of RM) or D_i (in the case of DM). If the series exceeds T_i (or D_i), the task Γ_i is not schedulable.

In [50] the authors updated the analysis by Joseph and Pandya to include blocking factors introduced by periods of non preemption, due to the non-independence of the tasks. The worst-case response time is then updated to:

$$R_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad (8)$$

which can also be solved using a similar recurrence relationship. B_i is also as given by Equation (3).

Some care must be taken using Equation (8) for the evaluation of the worst-case response time of non preemptable independent tasks. In the case of preemptable tasks, with Equation (6) we are finding the processor's level-i busy period preceding the completion of task i; that is, the time during which task i and all other tasks with a priority level higher than the priority level of task i still have processing remaining. For the case of non preemptive tasks, there is a slight difference, since for the evaluation of the processor's level-i busy period we cannot include task i itself; that is, we must seek the time instant preceding the execution start time of task i.

Therefore, Equation (4) can be used to evaluate the task's response time of a task set in a non preemptable context and independent tasks, where the interference must be now re-defined:

$$I_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{I_i}{T_j} \right\rceil \times C_j \right) \quad (9)$$

4.3.3 From Task to Message Schedulability Analysis

Communication between processes on different machines in a distributed system requires messages to be transmitted and received on the underlying communication sub-system. In general, these messages will have to compete with each other to gain access to the network medium.

In order for hard real-time processes to meet their deadlines (in general), the access to the communication subsystem will be scheduled in a manner which is consistent with the scheduling of processes on each processor. Although the communication link is just another resource, there are some issues which distinguish the link scheduling problem from processor scheduling that are summarised below. In fact, unlike a processor, which has a single point of access, a communication channel has multiple points of access. While preemptive algorithms are appropriate for scheduling processes on a single processor, preemption during message transmission will mean that the entire message will need retransmitting. Typically, message transmissions are considered non preemptive. In addition to the deadlines imposed by the application processes, deadlines may also be imposed by buffer availability.

Considering some analogies between, for example, task execution time and message transmission time or task blocking time and message blocking time, it is possible to adapt the tests available for the schedulability analysis of non preemptable tasks in single processor systems to the message scheduling for some type of networks. Examples can be found in [27, 51-55].

Holistic[2] Approach

A reasonably large distributed real-time system may contain tens of processors and several distinct communication channels. In these systems, one of the most challenging problems consists in finding end-to-end timing characteristics. For each couple of communicating tasks (in different processor units) there is an end-to-end timing constrain: the maximum time available for producing a message at the sender side, transmitting the message over the network and processing it at the receiver side.

[2] Holistic, *adj*. relating to or concerned with wholes or with complete systems rather than with the analysis of, treatment of, or dissection into parts; Holism, *n*. a theory that the universe and especially living nature is correctly seen in terms of interacting wholes (as of living organisms) that are more than the mere sum of elementary particles [From Merriam-Webster's Collegiate Dictionary]

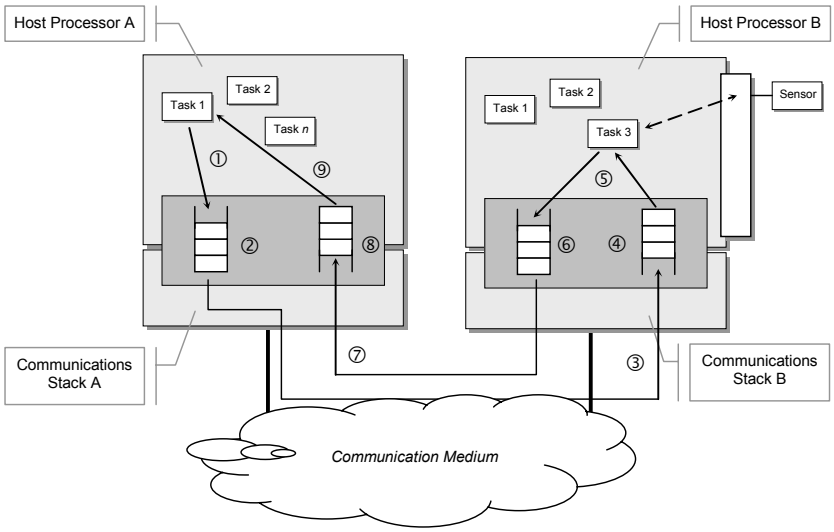


Figure 13. End-to-End communication delay

Both the processor and the communication subsystems can be analysed so that the worst-case timing behaviour is predictable. To depict the several delay components typically involved in such analysis, consider the following example of a remote I/O reading, illustrated in Figure 13: the end-to-end communication delay starts when the sending task is released and starts competing with other running tasks on the sender-hosting computer. The task may suspend as soon as the message request is passed to the communications stack (①). Then, the message request waits in a queue (assuming a simplified protocol stack) until it gains access to the communication medium. This queuing delay depends on how the queue is implemented (first-come-first-served queue, priority queue, etc.) and how the MAC level behaves (②). The message request is then transmitted. This time interval depends on the data rate and length of the transmission unit and also depends on the propagation delay (③). (Note that if intermediate systems (e.g., switches) are used, the propagation delay component is more complex).

The message indication is then queued in the remote communication stack (④). The receiving task processes the message indication, and performs the actual reading of the required data. The response frame is produced and queued (⑤). The message response will suffer similar types of delays. A queuing delay (again assuming a simplified protocol stack) in the remote transmitting queue (⑥), a transmission delay (⑦), a queuing delay in the local receiving queue (⑧), and finally the time for the local task to process the response (⑨).

In terms of the response time analysis of communicating tasks, distribution brings the need to include the end-to-end communication delays, as one of the components of the overall task's response time. This is a quite complex approach to real-time analysis, and it involves the provision of methodologies for the evaluation of the worst-case messages' response times in the communication network, which are then "embedded" with the communicating task, operating system and communication stack models.

It is possible to reduce the difficulty of a global distributed system analysis by means of a very simple concept: attribute inheritance. The overall analysis can be decoupled in several simpler analyses of smaller subsystems. Given their attributes, these subsystems can be analysed by means of exact procedures that let us find the worst-case response times. By suitably combining these values, we can find tight bound for end-to-end computations and we can then compare them with the relative constrains, in order to establish the feasibility of the whole system. This type of analysis is called *holistic* analysis and has been addressed previously by several researchers [9, 10].

4.4 Simulation-Based Timing Analysis

Developing a simulation model that accurately portray the timing behaviour of a distributed system, bears a number of issues that must be correctly handled. The steps needed for a correct simulation model development are depicted in Figure 14.

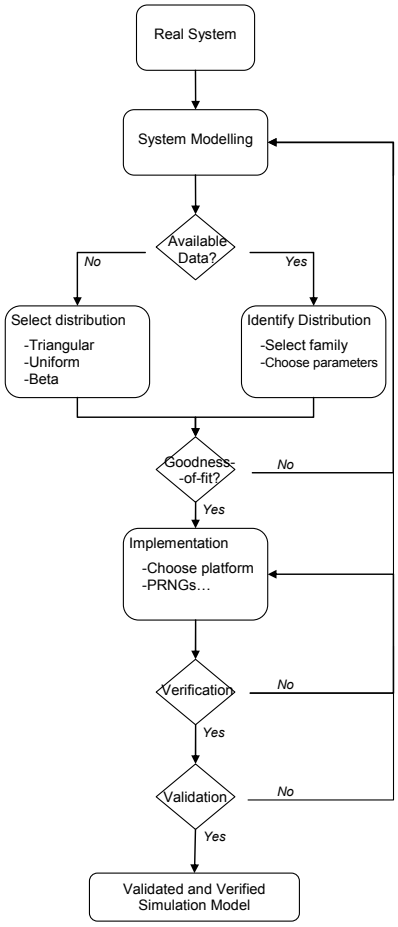


Figure 14. Simulation Development

The process begins with the development of a conceptual model, representative of the abstractions and assumptions about the real system. Here, special care should be taken with the level of detail introduced and the overall model complexity.

The next step is to properly characterise the simulation model's inputs, usually achieved by collecting data of the actual system. Using this data, the developer may choose an appropriate probability distribution, representative of the input process and select the proper distribution parameters. If it is not possible to collect system data, by using system expert knowledge and other available information about the nature of the process, it is usually possible to use one of the triangular, uniform or beta distributions to model the input process.

After modelling the input process, proper goodness-of-fit evaluation of the chosen distributions should be performed, either by simple observations of the modelled input process and of the actual system, or by using statistical tests.

With a conceptual model built, the next step is to perform the implementation of the model. It is advisable to use one of the many available simulation packages available, rather than build a simulation from scratch. As pointed out in [56], special care should be taken with the Pseudo-Random Number Generators (PRNGs) used, as some common implementations have short generation cycles, for today's computational resources.

The implementation of a simulation model should entail a verification step for backing up the correctness of the implementation of the model. This is whether the implementation reflects properly the details and assumptions included in the model.

The validation of the model basically consists of the corroboration of the input-output transformations of the model. This may comprise comparing simple observations of the system with the model, verification of assumptions, using existing theory and other relevant results or using quantitative techniques (like distribution fitting, homogeneity tests or sensitivity analysis) to validate the simulation model components. The validation of the simulation model may also include the comparison of the results from the simulation model with the results of an analytical model and investigate if the two are coherent.

Adhering to the steps described above do not guarantee a successful simulation exploitation, but are indeed a good sequence of actions towards it.

4.4.1 Meaningful Results from Simulation Output Data

One important aspect of any simulation study is the careful analysis of the output data results, which is often an overlooked aspect [56]. Since this dissertation will describe simulation models that have one or more random variables as input, these are said to be *stochastic simulation models*, and, by its nature, a stochastic simulation model will also produce random outputs. Thus, simulation has to be regarded as a computer-based statistical experiment, and to have any meaning, appropriate statistical techniques must be employed to analyse the simulation experiments.

Moreover, the data resulting from a simulation cannot be directly analysed using traditional statistical methods, since most of these apply to Independent and Identically Distributed (IID) data. This is an important topic of concern for the remainder of this text.

Let us consider a simple example of a waiting queue, with a random service time. The waiting time experienced by the first user will always be zero. On the other hand, the waiting time of the second user will depend on the departure of the first one, and so on. If we are interested in studying the waiting time in the queue, it is easy to observe that the distribution of these times is neither identically distributed nor independent.

A method commonly used to overcome this problem is to make observations from the results of multiple and *independent simulation runs* (or *simulation replicas*). Typically this is achieved by making multiple simulation runs with the same initial conditions and parameters, but yet different seeds for the random numbers used to drive the simulation through time. In this way, it is possible to obtain independent and identically distributed variables. Hence, it is possible to make estimates for the variables of interest, such as the average delay observed, the number of messages dropped, or the maximum response time, just to roll a few examples.

Next, a brief survey of the typical formulations for obtaining an estimator for a mean value, as well as its respective confidence interval will be presented. Another aspect of concern is how to get confidence intervals with some specified precision. These are crucial pieces of basic statistical reasoning used in the majority of the approaches for simulation output data analysis.

4.4.2 Statistical Ground for the Analysis of Simulation Output Data

Suppose we would like to obtain an estimate for the mean of an output variable. By the way of example, let us say it is the mean message delay in queue to access a communication medium. For a matter of simplicity, consider that we would like to observe this delay during a defined period, because the system is shutdown or restarted after that period (e.g. the case of a system that is disconnected at the end of a working day) – a terminating simulation.

One run of the simulation will produce one estimate for the mean message delay. Noticeably, the value of just one sample of a random process has no significance by itself. However, executing multiple runs of the simulation will provide a set of mean delay values, characterised by some distribution. Moreover, it will be IID, as pointed out. The samples mean (remember that, in this example, our samples are the set of mean delays observed for each simulation replica) is a natural estimator of the (unknown) true mean message delay.

But, how reliable is this estimate?

If we would make another set of simulation replicas, the result would, most likely, be different. Indeed, an estimate without an indication of its precision is of little value.

However, to come up with this type of conclusions, one would have to know something about the distribution of the sample. There is a basic, but very useful and important concept in statistics, called the *central limit theorem*. This theorem basically states that the sum and the average of many random values present a distribution close to normal. Typically, a normal approximation is sufficiently good if about 30 or more values are used in the sum (or average) [57]. Then, well-known methods can be used to draw confidence intervals from normal distributions.

There is, however, an important aspect to point out. The standard procedures for inference are developed for situations where the standard deviation for the entire population is known. As usually the entire population is unknown, it is also need to estimate the standard deviation from the available data, in which case, the statistic will not have a normal distribution, but a *t-distribution*.

For the sake of completeness, let us now lay down some basic statistics, applied to the estimation of the model true characteristics.

Suppose that $X_1, X_2, \ldots X_n$ are IID random variables with a mean μ (in our example, the mean message delay in queue to access a communication medium) and a variance σ^2. Our primary objective is to estimate μ. The sample mean ($\overline{X}(n)$), is an unbiased (point) estimator of μ, and is defined by:

$$\overline{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n} \tag{10}$$

That is, the expected value of $\overline{X}(n)$ is μ: $E\left[\overline{X}(n)\right] = \mu$. If we perform a very large number of independent experiments, each resulting in a $\overline{X}(n)$, their average will be μ.

While $\overline{X}(n)$ is the estimator of μ, in a similar way, the sample variance ($S^2(n)$) is an unbiased estimator of σ^2:

$$S^2(n) = \frac{\sum_{i=1}^{n} \left[X_i - \overline{X}(n)\right]^2}{n-1} \tag{11}$$

As discussed, it is important to have an assessment of the estimation precision. The usual way to do this is to construct a confidence interval. An approximate 100(1 - α)% confidence interval for μ is given by:

$$\overline{X}(n) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{S^2(n)}{n}} \tag{12}$$

The estimate ($\overline{X}(n)$, in our case) represents the guess for the value of interest. The margin of error (terms after the ± sign) gives a measure on how accurate the estimation is, based on the variability of the estimation.

The confidence level reflects the amount of confidence that, in the long run, this approach will be able to approximate the true value of interest. As we increase the confidence level, the confidence interval gets wider. It can be shown that to cut the length of the confidence interval in half, four times more samples are required.

4.4.3 Non-Terminating Simulations

So far, we have been concerned with a finite set of samples extracted from a terminating simulation. Nevertheless, *non-terminating simulations* are an important class that must be target of our attention. Indeed, often our systems of interest will not have a terminating event, and we will probably be interested in analysing the system's behaviour in the long run.

There are several subtypes of non-terminating simulations. In this case, a subtype where the outputs of the simulation model tend to stabilise; that is, the system reaches a *steady state* will be considered. A measure of performance for such simulation is said to be a *steady-state parameter*. We will then focus our attention on analysing non-terminating, steady-state parameters and thus assuming that, as the amount of data becomes large, distributions will converge to a common distribution in the steady-state.

The analysis of steady-state parameters raises a very important problem, which is how to choose the simulation data that actually represents the steady-state. Mostly due to the choice of starting conditions, the initial output data of the simulation is usually not very representative of the steady-state behaviour. This period, affected by the initialisation bias, is usually referred to as the *warm-up period*. Using data from this period for the estimation of system's steady-state parameters may yield deceptive results.

To circumvent the warm-up period problem, one may simply resort to very long runs, such that the data from the initial phase has a negligible impact, or to start the simulation in a state supposed to be close to the steady-state. Effectively, these methods have some serious practical impairments, thus somewhat more elaborate methods are commonly used. These methods typically ignore data from the warm-up period, utilising some techniques, based on the assumption that the variance of the samples is substantially lower in the steady-state than in the warm-up period, to detect when it ends.

The replication approach described earlier may still be used in the context of non-terminating simulations. All what is necessary is to define how to extract the steady-state means from each simulation replica (X_i used to compute the sample mean in Equation (10)). Suppose that we make n simulation replicas, each of length m, where m is much larger than l (the length of deleted data used to eliminate the impact from initial conditions). As a rule of thumb, m-l should be at least 10 times the size of l. In the context of non-terminating simulations, this method is commonly called *replication/deletion*.

Let X_i be IID variables given by the mean in each simulation replica i, from the set of values collected between l and m (Y_{ij}):

$$X_i = \frac{\sum_{j=l+1}^{m} Y_{ij}}{m-l} \quad for \;\; i = 1,2,...n \tag{13}$$

Similarly to the terminating case, Equation (10) gives an approximately unbiased point estimation for the steady-state mean μ, and a confidence interval may be obtained with Equation (12).

An informal description of the method may be as follows:

1. define the size of the initial phase *l* from test simulation runs;
2. perform *n* independent simulation replicas of length *m* (with *m* much larger than *l*);
3. for each simulation replica *i*, compute the mean of all observations after the initial phase *l*;
4. apply usual point estimate and confidence intervals on the IID means obtained (given by Equations (10), (11) and (12)).

As referred previously in Section 4.4.2, the confidence interval depends on the variance of X_i, which will be unknown when the first *n* simulation replicas are performed. If we make a fixed number of replications, the resulting confidence interval may be too wide for our particular purpose. However, also as pointed out in Section 4.4.2, we can decrease the length of the confidence interval by a factor of 2, by performing 4 times as many replications.

There are other methods that apply some variations. Instead of achieving independence trough multiple simulation runs, one can perform one long simulation run and try to obtain independent observations from subsets of data. The method of *batch means* [58], similarly to the replication/deletion, attempts to obtain independent observations, but, in this case, the single simulation run is divided into batches, where a batch takes the role of a single replica. It can be shown that, for a sufficiently large number of batches, the mean of the several batches will be approximately IID normal.

One of the most relevant advantages of this method is that it only has to go through one warm-up phase, on other hand, a major problem is on choosing the batch size m, or equivalently, the number of batches k. A number of guidelines extracted from research literature, and a general recommended strategy may be found in [58].

In the group of methods based on one long simulation, other methods may be encountered [28]. These methods, such as the *autoregressive* method or *spectral analysis*, try to use estimates of the autocorrelation structure of the underlying stochastic process to obtain an estimate of the variance of the sample and then to construct a confidence interval. For sake of simplicity, the reader is referred to the literature [28, 58] for further information on other methods.

All the procedures described to this point are, usually, classified as fixed-sample procedures, where the sample sizes taken (the whole simulation, in the case of replication/deletion or the batch, in batch means) are of a fixed size. Generally, some conclusions may be established for all of these fixed-sample procedures [28]:

- if the total sample size is chosen too small, the actual coverage may be lower than the desired;
- the appropriate choice of the total sample size is extremely model dependent and impossible to choose arbitrarily.

Evidently, no procedure that sets the run length before the simulation begins will always produce a satisfactory confidence interval. A *sequential* scenario where the simulation's end is determined by a relative statistical error that is verified in consecutive checkpoints is

a more interesting approach. Sequential methods are commonly based on the same methods for non-terminating simulations as batch means or spectral analysis, in conjunction with absolute or relative-error stopping rules. These procedures are more complex, requiring computing the estimates at several points of the simulation to check if the stopping rule has been satisfied, which can be computationally very expensive. Additionally, these procedures may not be easily applicable when multiple measures of performance are needed, and, because of random nature of simulation, the relative stopping rule can be accidentally satisfied, resulting in premature termination of the simulation, and on wrong estimation results.

Another typical problem with sequential procedures is that they are not very popular among existent software packages. A simulation package supporting sequential procedures is Akaroa2, designed at the University of Canterbury, New Zealand [56]. Besides the problems described, sequential procedures are recognised as a practical approach allowing control on the error of the final results of stochastic simulations [56].

4.4.4 Other Measures of Performance

All the previous methods seek to obtain a mean value for the output point estimator. What about other kind of measures? Consider that we would like to estimate the probability of a value belonging to an interval, for example, imagine the case of investigating the probability that a queue length is greater than k messages. Another different performance measure is a quantile. Quantiles describe the level of performance that can be delivered with a given probability p. The next sub-section outlines some procedures to extract such measures of performance like proportions, probabilities or quantiles.

Probabilities and Quantiles

Suppose we need to estimate the steady-state probability (p) of the mean message delays in queue to access a communication medium being less than a value x. The variable under analysis may be represented by 1 if the queue delay exceeds the value x, and 0 otherwise.

Making $p = P(Y \in B)$, where B is a set of real numbers smaller than x, and Y is the original steady-state random variable, we are just in the presence of a special case of estimating the mean, by letting the random variable Z be defined by:

$$Z = \begin{cases} 1, if\ Y \in B \\ 0, otherwise \end{cases} \quad \textbf{(14)}$$

It can be shown that estimating p is equivalent to estimating the steady-state mean for the expected value of Z ($E(Z)$).

A performance measure that does not fit in the same reasoning is a quantile. For instance, if the variable represents the delay in queue that a client experiences, the 0.90-quantile is the value x such that 90% of all messages experienced a delay shorter than x.

Estimating quantiles is both conceptually and computationally (in terms of number of observations required) a more difficult problem than estimating the steady-state mean. Additionally, most of the procedures for estimating these performance measures are based

on order statistics and require storage and sorting of the observations. Nevertheless, the general reasoning is similar to the one for obtaining the interval estimator for a steady-state mean.

An example taken from [59] points-out one major problem with quantile estimation describes that, for the steady-state estimation of 0.99 quantile of waiting times, an estimate with relative a precision of 10% required about 500.000 observations, and a 0.999 quantile needed a sampling size of approximately 2.300.000. Because quantile estimation require storage and sorting of observed values, obtaining small quantile estimations, with a good accuracy is often impractical. However, this is a problem under investigation, and several techniques already exist that do not require storage and sorting have already been attained and implemented. In [60], a number of such approaches are presented and evaluated.

4.5 Summary

The basis for the timeliness approaches addressed in this dissertation was presented in this chapter. Basic concepts of real-time systems were laid out to introduce the foundations of traditional real-time response time analysis applicable also to the analysis of distributed systems. The first two sections addressed concepts associated with single processor systems, which are then adapted and extended to be applied to the analysis of distributed systems.

The second part of this chapter addressed issues related to the exploitation of a correct simulation-based study. Both the proper development of simulation models and the grounds for extracting performance measures from the produced output data were approached.

The concepts brought forward in this chapter will be the basis for the models to be introduced in the subsequent chapter of this dissertation, for the analytical worst-case and for simulation-based analysis in Chapter 5 and Chapter 6, respectively.

Chapter 5

WORST-CASE BASED ANALYTICAL MODEL

5.1 Introduction

An effort to formulate an analytical formulation to find end-to-end response times in EIP based distributed systems is provided in this chapter. This model builds upon the response time analysis presented in the previous chapter. Using the concept of attribute inheritance, it considers a number of worst-case assumptions to derive the end-to-end response time bounds.

While this is a very interesting and useful approach to start with, it basically leads to an additive formulation built on top of several worst-case assumptions, thus potentially exacerbating the levels of pessimism. This level of pessimism is easily foreseen in a distributed system, where the probability of concurrence of independently generated worst-case situations is realistically extremely low. Nevertheless, it is important to stress that the decoupling of the diverse latency components brought by the producer/distributor/consumer model underlying EIP corroborates the validity of tackling the problem in such a guaranteed worst-case fashion.

5.2 End-to-End Latency Formulation

The decoupling of the diverse latency components brought by the producer/distributor/consumer model underlying EIP makes possible to break down the overall transaction, described in Section 2.3.5, into two independent transactions. Considering the assumptions outlined in Section 2.3.6, the analytical formulation for computing the worst-case end-to-end delay of a transaction can thus be defined as follows:

$$R_i = fd_i + \sum_{m \in \{input,\ output\}} (Li_m^{sn \to sw} + Li_m^{sw} + Li_m^{sw \to dn}) + R\tau_i \qquad \textbf{(15)}$$

In brief, the delay associated to an end-to-end transaction results from the delay components associated to two independent transactions ($\sum$ term), added to the worst-case controller task response time ($R\tau_i$) and the input filter (fd_i).

In Equation (15) $Li_m^{sn \to sw}$ denotes the worst-case time that a message m takes to arrive from a source node sn to a switch sw. In the case $m = input$, sn is considered to be the node that contains the input module related to the overall transaction i. In the case $m = output$, then sn is considered to be the node that includes the controller responsible for processing the output related to overall transaction i. Thus:

$$Li_m^{sn\to sw} = \begin{cases} T_{i\,RPIinput} + Qb_m^{sn} + Qea_m^{sn}, & \text{if } m = input \\ Qb_m^{sn} + Qea_m^{sn}, & \text{if } m = output \end{cases} \quad (16)$$

where $T_{iRPIinput}$ denotes the time span corresponding to the periodicity defined for the message m connection (*input RPI*) related to overall transaction i, Qb_m^{sn} denotes the worst-case delay caused by access contention in node sn backplane, and Qea_m^{sn} denotes the worst-case delay for message dispatching at the Ethernet adapter of node sn.

Similarly, $Li_m^{sw\to dn}$ corresponds to the worst-case delay a message m may experience from the switch sw to the destination node dn. In the case $m = input$, dn is considered to be the node that contains the controller module related to the overall transaction i. In the case $m = output$, dn is considered to be the node that includes the output module related to overall transaction i. Therefore, it follows that:

$$Li_m^{sw\to dn} = \begin{cases} Qb_m^{dn} + Qea_m^{dn}, & \text{if } m = input \\ T_{iRPIoutput} + Qb_m^{dn} + Qea_m^{dn}, & \text{if } m = output \end{cases} \quad (17)$$

where $T_{iRPIinput}$ denotes the time span corresponding to the periodicity defined for the message m connection (*input RPI*) related to overall transaction i.

In (15), Li_m^{sw} denotes the worst-case relaying delay a message may experience at Ethernet switch sw. This latency includes the time taken by the switch to relay message m to the corresponding output port, and the queuing delay the message may suffer at the output port. This latency will be reasoned out later on in a separate sub-section.

Finally, and before going through further details, a few words on the computation of $R\tau_i$. Typical EIP controller modules support fixed priority scheduling. Therefore, it is possible to obtain the worst-case response-time for the task associated to overall transaction i ($R\tau_i$) by applying well-known response time analysis.

5.3 Latency Introduced by the EA (Qea_m)

For the sake of simplicity, a rough characterisation is adopted for analysing the latency introduced by the Ethernet Adapters (EA). Messages are assumed to be handled in an on-demand fashion: as soon as a packet fully arrives at the network interface, a "packet arrived" interrupt is raised on the host processor. The interrupt handler releases a task which copies the data from the network buffer, performs the necessary delivery operations to a task that, in turn, will encapsulate the data and transmit it to the remote Ethernet address. Some delay components (Figure 15) are considered, such as the delivery delay of the message (④), the generation delay of the encapsulated message (⑤), the possible queuing to deliver it to the Ethernet network interface (⑥), and, finally, a transmission and a propagation delay (⑦). The worst-case aggregating all these latencies will be denoted as D^{ea}; that is, the worst-case processing delay for any message being processed at the particular Ethernet adapter of node ea. Note that ea will correspond to sn or dn, depending on the formulations under consideration (Equations (16) or (17)).

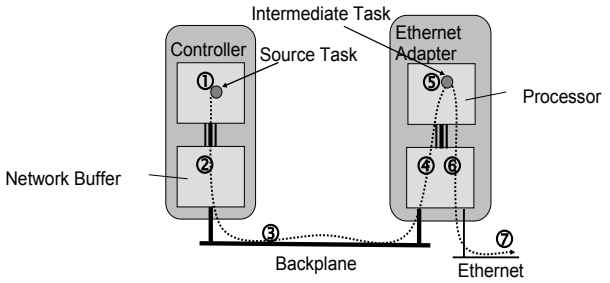


Figure 15. Controller: Message delay components

Therefore, a simple worst-case formulation of the delay introduced by the EA considers that a particular message *m* will be processed only after all possible contending messages (nc_m^{ea}) are processed:

$$Qea_m^{ea} = nc_m^{ea} \times D^{ea}, \quad \text{with } ea \in \{sn, dn\} \tag{18}$$

5.4 Latency Introduced by the Backplane (Qb_m)

Figure 15 also illustrates other components contributing to the overall worst-case latencies. The generation delay introduced by the task processing the related output object to execute and generate the packet (①) (this corresponds to $R\tau_i$). The access delay, when sending the message to the backplane of the Controller (②) (Qb_m) and the propagation delay in the backplane (③). The latter will be, for now, neglected and some reasoning about the second will now be exposed.

The Backplane access medium is based on a time division concept (TDMA-Time Division Multiple Access). In this case, it divides the transmission time between each connection producing data to the backplane. The access to the media is ordered by time, such that each connection is assigned a time-slot in a cyclic schedule.

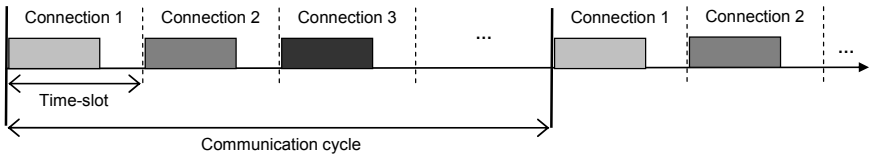


Figure 16. Backplane Medium Access Control Scheme

Considering that the Backplane in each node *nod* is divided in fixed time-slots, with duration ts^{nod}, one for each connection producing data to the backplane. These time-slots are assumed to be larger than the time needed to transmit the largest message transferred in the node's backplane, including overhead.

Considering the worst-case situation when a message arrives just a fraction of time after its connection was served. In such case, the message must wait an entire communication cycle, until it is transmitted. This can be effortlessly represented by:

$$Qb_m^{nod} = ncb_m \times ts^{nod}, \quad \text{with } nod \in \{sn, dn\} \tag{19}$$

where the queuing time in the backplane is defined the number of connections producing data to the backplane (ncb_m), multiplied by the time taken to transmit each message (ts^{nod}).

5.5 Latency Introduced by the Switch (Li^{sw})

In a preliminary analysis a switch that implements priorities will be contemplated, based on classification of the Ethernet frames. All the traffic in the network is assumed to be characterised, with well defined periodicities. It is also considered that, under a controlled load the switch will introduce constant switching delay. If traffic is sent to an output port at a higher rate than its capacity, packets must be queued. The following formulation may give the worst-case queuing time in a switch:

$$Li_m^{sw} = Is_m + \sum_{j \in cp(m)} Cs_j \tag{20}$$

where, $cp(m)$ is the set of messages from connections going out through the same switch port as message m. Cs_j is the time to transmit a message j, including the inter-frame delay (being j from the set of messages given by $cp(m)$). This also includes the time to transmit a message m itself (Cs_m).

Is_m will be the sum of the maximum blocking time a message may experience, including the blocking by messages of equal priority, and the interference from higher priority messages:

$$Is_m = Dsl \times (1 + neq(m)) + \sum_{j \in hp(m)} \left\lceil \frac{Is_m}{T_j} \right\rceil \times Dsl \tag{21}$$

This formulation considers a first-come-first-served policy between messages of equal priority, to account with the possible aggregation of priorities due to the reduced number of priorities supported in standard implementations. In Equation (21), $neq(m)$ is the number of messages with priority equal to m, and Dsl includes the latency introduced by the switch to classify and relay the frame to an output port. More sophisticated formulations have been tackled previously, which could be considered into this analysis [61, 62]. The formulations used here are for the sake of simplicity.

5.6 Numerical Example

For the purpose of instantiating the proposed formulation, a scenario with eight end-to-end transactions in a 100Mbps Ethernet network was setup (Figure 17).

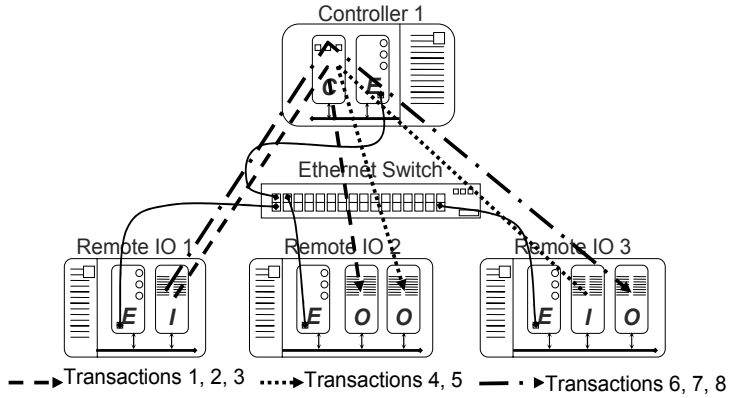


Figure 17. Example scenario

The Controller module has two tasks processing their input data. Task τ_1, ($R\tau_1$=2ms) processes input data from connections 1, 2, 3, 4, 5, while τ_2 ($R\tau_2$=4ms) is processing the input data from connections 6, 7, 8. Table 2 includes some parameters necessary for the calculations. The worst-case latencies resulting from this scenario are given in Table 3.

Table 2. Assumptions for device parameters

Parameter	*Dea*	*Dsl*	*Interframe delay*	$ts^{\forall nod}$
Value (ms)	0.20	0.011	9.6E-04	0.05

Table 3. Transactions response time results

Tr.	*RPI (ms)*	*Size (bytes)*	*Input*			*Output*			*Ri (ms)*
			$Li^{sn\to sw}$ (ms)	Li^{sw} (ms)	$Li^{sw\to en}$ (ms)	$Li^{sn\to sw}$ (ms)	Li^{sw} (ms)	$Li^{sw\to en}$ (ms)	
1	5	48	6.50	0.05	4.00	9.00	0.05	1.25	22.85
2	7	46	8.50	0.06	4.00	11.00	0.06	1.25	26.87
3	10	50	11.50	0.07	4.00	14.00	0.07	1.25	32.90
4	25	48	26.25	0.08	4.00	29.00	0.07	1.25	62.65
5	30	48	31.25	0.09	4.00	34.00	0.08	1.25	72.67
6	45	55	46.50	0.11	4.00	49.00	0.08	1.25	104.94
7	75	46	76.50	0.12	4.00	79.00	0.09	1.25	164.96
8	150	60	151.5	0.13	4.00	154.00	0.10	1.25	314.98

5.7 Summary

This chapter provided an effort to formulate a mathematical model enabling to find end-to-end response times in Ethernet/IP based distributed systems. This model builds upon the response time analysis presented earlier in Chapter 4, considering a number of worst-case assumptions to derive the end-to-end response time bounds. The analytical formulations to compute each of the delay components were presented along with an example scenario where the overall approach is applied.

Chapter 6

SIMULATION BASED TIMING ANALYSIS OF EIP NETWORKS

6.1 Introduction

An EIP simulation environment was developed using the OMNeT++ discrete-event simulation platform. The simulation model developed from scratch for EIP is composed of three basic components (nodes), mapping on the main EIP devices: a Remote IO, a Controller and an Ethernet Switch, described earlier in Section 2.3.4. Each of these basic nodes can be instantiated into several different device models, with different particular characteristics, since modularity and parameterisation are considered into the design to a sufficient extent. In the next subsections, further details are provided concerning the model implementation.

6.2 The Remote IO Node

The Remote IO (RIO) is composed of several IO modules and an EIP Adapter, which communicate through a backplane, using CIP packets. The IO modules contain the several input/output connections of the device. Typically, each IO module will act as an Input or Output module, but not as both at the same time. The Ethernet Adapter is responsible for relaying messages between the Backplane and the Ethernet network. CIP packets are eventually (for the case of a consumer outside the node) encapsulated into UDP packets inside the EIP Adapter (*ethIPAdapter* in Figure 18).

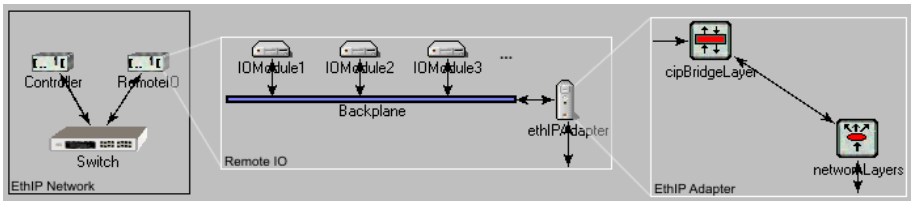


Figure 18. EIP simulation model hierarchy in OMNeT++.

The *Backplane* is a simulation module that exists both at Controller and RIO nodes. For simulation performance, at initialisation time the Backplane uses the information about the data connections produced/consumed at each module to build a table with information on the gates where to deliver each of the configured connections. Figure 19 provides a sample of NED code defining the Backplane OMNeT++ simple module. A simple OMNeT++ module is declared with the keyword simple, followed by the module's name. Included in

the declaration are the OMNeT++ simple module's parameters and gates. The gates of an OMNeT++ module define the entry points of the module. For the example of the Backplane module, an array of input and output gates are defined, where each pair of input and output represents a Backplane interface connecting to a node's module.

```
simple Backplane
    parameters:
            tTableTime : numeric,
            frameTime  : numeric,
            timeDivison: bool;
    gates:
        in: in[];
        out: out[];
endsimple
```

Figure 19. Backplane NED definition.

The Backplane simple module has the parameter *tTableTime*, which defines the transmit table time, used for the time division multiple access (TDMA) protocol used as backplane's MAC protocol. The parameter *frameTime* concerns the time a message takes to be transmitted in the backplane, and the parameter *timeDivision* specifies whether the time division protocol behaviour should be precisely simulated or simplified. The Backplane module simulates the behaviour of a TDMA contention schema where access to the communication medium is equally distributed to the several producing connections delivering data to the backplane. Nevertheless, and because this simulation approach of the backplane can introduce a great amount of events, it is possible to disable this behaviour. The alternative will then be to insert a variable delay, as a function of the number of connections that send messages to the backplane.

The EIP Adapter is responsible for relaying messages to/from the Ethernet network. It receives the CIP messages from the Backplane and, in the CIP Bridge Layer (*cipBridgeLayer* in Figure 18) encapsulates them into UDP packets which are passed down to the Network Layer of the UDP/IP stack. In the opposite direction the packets are retrieved from the UDP/IP packet and delivered to the Backplane.

The EIP Adapter encloses the delays introduced to perform the encapsulation of the messages, to access the network and the delays resulting from the concurrent access to the adapter resources. Figure 20 illustrates the NED definition of the Ethernet Adapter OMNeT++ module (a compound module). Like an OMNeT++ simple module, a compound module is composed of the module's parameters and gates. Additionally, it has to include its sub-modules and the connections between the sub-modules and gates.

Two parameters are used. The *connectionIDProducedList* and the *connectionIDConsumedList* parameters are used for listing the CIP connection identifiers of the connections produced and consumed in the node's modules connected to the backplane. The sub-modules of an *EthIPAdapter* module are the CIP Bridge Layer (*cipBridgeLayer*

sub-module) and Network Layer (*networkLayers* sub-module). The connections implemented (refer to the NED code sample in Figure 20) are between these two layers and the input/output gates from the backplane and the Ethernet network.

```
module EthIPAdapter
    parameters:
        connectionIDProducedList : string,
        connectionIDConsumedList : string;
    gates:
        in: from_backplane;
        out: to_backplane;
        in: from_eth;
        out: to_eth;
    submodules:
        cipBridgeLayer: CIPBridgeLayer;
        networkLayers: NetworkLayers;
    connections:
        from_backplane --> cipBridgeLayer.from_bp[0];
        to_backplane <-- cipBridgeLayer.to_bp[0];
        networkLayers.to_application --> cipBridgeLayer.from_ntw;
        networkLayers.from_application <-- cipBridgeLayer.to_ntw;
        from_eth --> networkLayers.from_phy;
        to_eth <-- networkLayers.to_phy;
endmodule
```

Figure 20. EthIPAdapter NED definition.

Each of the IO modules (labelled *IOModule1*, *IOModule2*, *IOModule3*, etc., in Figure 18) inside a node and connected to the Backplane contains a CIP Layer, responsible for managing data transfers to/from the IO Connections. The IO Connection can behave either as an output or input connection, and each IO Module may have several input or output connections connected to its CIP Layer (Figure 21).

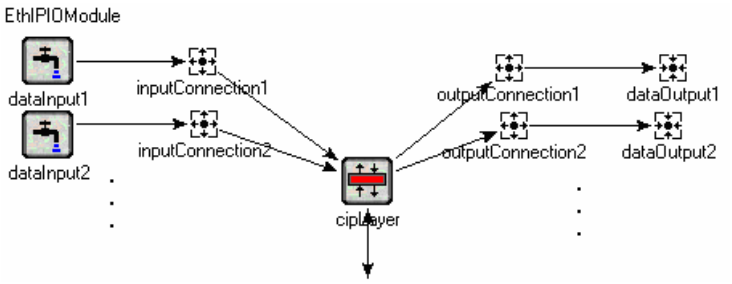


Figure 21. OMNeT++ EthIPIOModule composition.

When an IO Connection is doing the task of an input connection, it receives data from a data input, which generates input data at a defined periodicity (this data input models the input signals of an input connection). At a defined Requested Packet Interval (RPI), the IO Connection constructs a CIP data item from the last received data, and sends it to the CIP Layer. When an IO Connection is acting like an output connection, it receives data from the CIP Layer, which is delivered to a data output, after a parameterised hardware delay. This is illustrated in Figure 22, which provides the C++ code of the message handler from the *IOConnection* class.

```
void IOConnection::handleMessage(cMessage *msg) {
 if (msg->isSelfMessage() == true && inputModule == true) {
  // at rpi, send input data and schedule next rpi
  sendInputData();
  if (((simtime_t)*rpi) > 0)
   scheduleAt(simTime()+((simtime_t)*rpi), msg);
  } else {
   if (inputModule == true) { // acting as an input
    // discard previous dataItem and store new one
    if (dataItem != NULL) delete dataItem;
     dataItem = (CIPDataItem*) msg->dup();
    } else // acting as an output
    sendDelayed(msg->decapsulate(),((simtime_t)*asicDelay), "out");
    delete msg; // After finishing with a message, it is released
  }
}
```

Figure 22. IOConnection class message handler C++ code.

The data input generators (*dataInput1*, *dataInput2*, ..., in Figure 22) impersonate the signals applied at the input pins of the IO. They are parameterised by the length of the data generated and the periodicity of the data generation, and by two delays introduced after the

generation of the input (a hardware delay and a filter delay). OMNeT++ supports defining any of these parameters as a user-defined randomly distributed function. These parameters can be either defined in the NED code of a compound module, in which case it will be the same for all instances of this compound module, or defined in a special initialisation file that may assign the parameters individually for each module in the simulation.

Figure 23 exemplifies the definition of the *dataInput* (NED code) parameters in an IO module: a random variable with a uniform distribution in the interval [100, 150] milliseconds (highlighted code line in Figures 23 and 24).

```
module EthIPIOModule
…
    submodules:
        dataInput: Input[numInputs];
            parameters:
                hwDelay = 200 us,
                dataLength = 22,
                filterDelay = 0 ms,
                period = uniform (0.1, 0.15);
…
endmodule
```

Figure 23. OMNeT++ EthIPIOModule NED code for parameter configuration.

Figure 24 illustrates the alternative setting of the same parameters through an initialisation file, for a particular IO module instantiation (*ioModule1*), inside of a RIO node (*ethIPIO1*), within a network (*ethIPNetwork1*).

```
ethIPNetwork1.ethIPIO1.ioModule1.dataInput[0].hwDelay = 200 us
ethIPNetwork1.ethIPIO1.ioModule1.dataInput[0].dataLength = 22
ethIPNetwork1.ethIPIO1.ioModule1.dataInput[0].filterDelay = 0 ms
ethIPIO1.ioModule1.dataInput[0].period = uniform(0.1,0.15)
```

Figure 24. EthIPIOModule parameter configuration through initialisation file.

6.3 The Controller Node

The Controller node is, in its structure, similar to the RIO node. The *Backplane*, the *EIP Adapter* and *IO modules* are exactly the same modules as described previously for the Remote IP node. Of course, it is possible to parameterise each of the modules differently, and therefore manipulate their actual behaviour.

There is however a module that must be specified for the particular case of Controller nodes: the *Controller* module (Figure 25). In an actual EIP system, the controller module is responsible for executing the tasks performing the control functions.

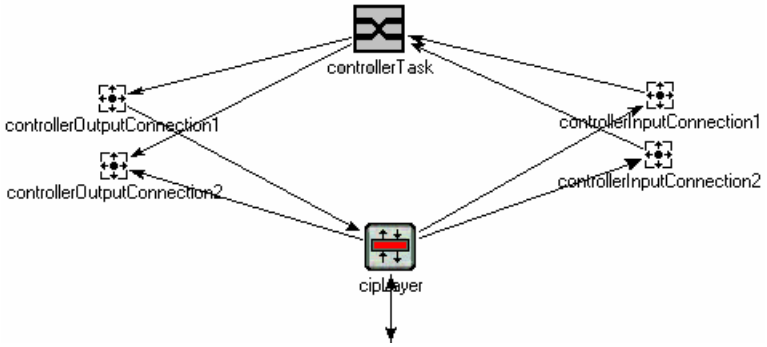


Figure 25. OMNeT++ Controller module composition

The Controller was modelled reusing some OMNeT++ modules described earlier: the IO Connection modules and the CIP Layer. The *controllerInputConnection* module receives the data to be delivered to the *ControllerTask* module, corresponding to an output connection at the remote source node. The output data generated by the controller task is delivered to the *controllerOutputConnection* module. The *ControllerTask* (worst-case) response time is a parameter which is a time span introduced between the reception and the generation of data. This parameter can be defined has a random function that best models the response time for each controller task.

6.4 The Switch Node

The Switch node models the delays introduced by an Ethernet Switching component. For the purpose of this simulation, it is only necessary that the Switch recognises multicast groups and deliver the frames received in an appropriate manner. The Switch model is composed of several ports that connect to the nodes in the network. Because there is a port in each direction, the Ethernet medium is assumed to be full-duplex.

The Switch node is a simple OMNeT++ module. The NED definition of the Switch OMNeT++ module is rather simple, and is given in Figure 26. It is similar to the Backplane OMNeT++ module, since it has an array of input and output gates, in which each pair represents the interface with each connecting modules (the switch port).

```
simple Switch
    parameters:
        nodename : string,
        switchDelay : numeric;
    gates:
        in: in[];
        out: out[];
endsimple
```

Figure 26. Ethernet Switch NED definition.

OMNeT++ offers a rather convenient manner of defining channel transmission characteristics. It is possible to define the characteristics of the connection between any two modules by using a predefined channel. A channel is defined with its name, preceded by the keyword channel. A channel may be assigned with the attributes *delay*, *error* and *datarate*. The example code depicted in Figure 27 corresponds to the definition of a 100 Mbit/sec Ethernet channel with a normally distributed delay, with mean value of 150 μs and a standard deviation of 50 μs. The connecting channels model the transmission delays and queue the messages whenever concurrent access to the medium occurs.

```
channel ethernet
    delay normal(0.00015,0.00005);
    datarate 100*10^6;
endchannel
```

Figure 27. Ethernet Channel definition in OMNeT++.

To simplify the multicast delivering process, the connection identifier of a producing connection is directly mapped into the last octet of an IP Multicast Address. For example, for a connection with the identifier 128, the IP Multicast Address would be constructed with a user defined prefix and the last octet being 128; that is, for a prefix of 239.0.0., the connection with identifier 128 would be mapped to the multicast group with address 239.0.0.128.

Because the rules defined by multicast Ethernet MAC address mapping are applied [63], the Ethernet frames actually contain the connection identifier mapped into the multicast groups. In this way, it is possible for the Switch to simply construct, at initialisation time, a list of all producing/consuming connection IDs for each connected node. At run time, the Switch module will merely compare the connection identifiers of the received frames with the ones in the list for each node, swiftly delivering copies of the received frame to all nodes that belong to the multicast group. The Switch is parameterised by a delay that represents the time taken to process the frames, which can also be defined as a random function.

In order to provide some insight into the obtainable results with this modelling and simulation approach for EIP-based distributed systems, an example is presented. The results of its simulation and how they could be analysed are then discussed in this section. Note that we are aiming at obtaining an estimation of the worst-case end-to-end response time for a number of transactions. A primary goal is to consider some fundamental aspects about the analysis of the simulation results.

6.5 Example Scenario

The example system is constituted of three RIOs, one Controller and an interconnecting switch (Figure 28).

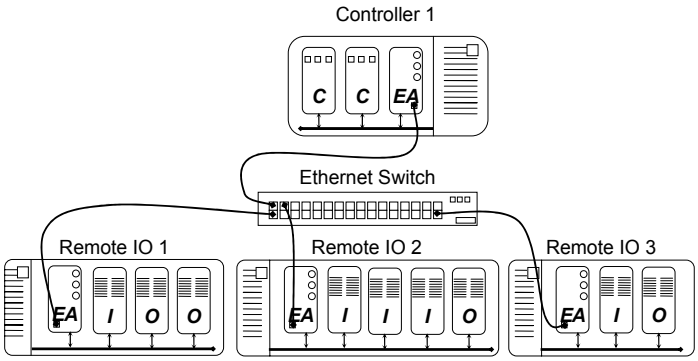


Figure 28. Simulated system depiction.

The Controller node is composed of one IO module and two Controller modules. The first RIO includes four IO modules, two for output and two for input. The second RIO also includes four IO modules, three for input and one output. Finally, the last RIO contains three IO modules, two for input and one output.

The system has nine end-to-end transactions (analogous to the end-to-end transaction described in Section 2.3.5) between the RIOs and the Controller. This results in a total of eighteen connections, half from the RIOs to the Controller (Input direction) and the other half, from the Controller to the RIOs (Output direction).

Table 4. End-to-End transactions

Transaction	*Size (Bytes)*	*RPI (ms)*	*Connection: Input Direction*	*Connection: Output Direction*
1	46	10	131	141
2	46	7	132	142
3	46	25	133	143
4	46	20	134	144
5	46	55	151	161
6	46	80	152	162
7	46	75	153	163
8	46	200	171	181
9	46	350	172	182

Table 4 presents the size, periodicity and identifiers of the system's connections, whereas Tables 5-7 provide the details about the mapping of connections to the system modules.

As an example, Transaction 8 is initiated at the IOModule3 of RemoteIO1 (connection 171) with an RPI of 200 ms (Table 5). It is delivered to module 2 of the Controller (Table 6), where the data is processed, and the corresponding output is generated (connection 181). This connection is then sent to the IOModule2 of RemoteIO3 (Table 7). The RPI of the output connections is equal to the corresponding input connection.

Table 5. Input Connections

Node	*Module*	*Input Connection*	
		ID	*RPI (ms)*
RIO 1	*IO module1*	171	200
		172	350
RIO 2	*IO module 1*	131	10
	IO module 2	132	7
	IO module 3	133	25
		134	20
RIO 3	*IO module1*	151	55
		152	80
		153	75

Table 6. Connections at the controller

Module	*Input Connection*	*Output Connection*
Controller 1 module 1	131	141
	132	142
	133	143
	153	163
Controller 1 module 2	134	144
	151	161
	152	162
	171	181
	172	182

Table 7. Output connections

Node	*Module*	*Output Connections*
RIO 1	*IO module 1*	141; 142; 14; 162
	IO module 2	163
RIO 2	*IO module 4*	144
RIO 3	*IO module 2*	161; 181; 182

6.5.1 Statistical Results of the Simulation

Table 8 provides the results of the application of such approach to the analysis of the simulation output data. In this, we will attempt to construct a confidence interval for the

end-to-end response time that can be expected in the long run. This estimation is based on the observation of successive end-to-end response time values verified across simulation replications and the variance of these observations. The number of replications performed was 50, which was a number of replications that allowed obtaining an error below 25-26% of the estimate for all transactions.

The *X* in the table below represents the estimation for the end-to-end response time of the transactions. The margin of error (ε) gives a measure on how accurate the estimation is, based on the variability of the estimation. The confidence level (99.9%) reflects the amount of confidence that, in the long run, this approach will be able to approximate the true end-to-end response time mean. With these values, it is possible to construct the confidence intervals displayed.

Table 8. Results of simulation output using replication/deletion (sorted by transaction periodicity)

Transaction	*Estimation for 99.9% confidence interval ($X \pm \varepsilon$ ms)*	*99.9% Confidence interval (ms)*
2	11.30 ± 2.53	[8.77,13.83]
1	15.81 ± 4.05	[11.76,19.86]
4	30.77 ± 3.79	[26.98,34.57]
3	38.36 ± 6.82	[31.53,45.18]
7	113.01 ± 18.86	[94.15,131.87]
5	83.13 ± 17.23	[65.90,100.36]
6	120.88 ± 15.13	[105.76,136.01]
8	300.70 ± 23.83	[276.87,324.53]
9	526.64 ± 34.53	[492.11,561.17]

This evaluation of the behaviour of a concrete system may be of relevance to the systems designer, when a probabilistic analysis of the system is being carried out.

Note that this evaluation is more suitable for means and variance behaviour. Its applicability for values on the tail of distributions (such as worst-case) is still object of current work, thus the reader is referred to the next sections of this dissertation for further discussion on these issues.

Some additional remarks that might be raised towards this analysis include the fact that the simulation data needed to produce such results may be at a prohibitive computation cost. This time actually depends on a number of variables. The complexity of the system influences the number of events generated during the simulation, the variance of the variables under study affect the size needed for each individual simulation replication, and the margin of error desired, which is also influenced by the variation of the variables of interest, may be controlled by the number of simulation replications. A close investigation of these matters is beyond the scope of dissertation, nevertheless, it can be advanced that, for the example presented, each replication took less than 2 minutes to run on a fairly old machine (PIII 1GHz).

6.6 Summary

In this chapter, the modelling and simulation of EIP-based distributed systems performed with the purpose of extracting temporal properties was delineated. The basic blocks enabling to build EIP discrete-event simulations are detailed. Similar to what was done in Chapter 5, the simulation is instantiated with an example scenario that allows obtaining some results by applying the reasoning outlined in Chapter 4.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary and Conclusions

Ethernet-based technologies have already gained a strong position in the factory-floor. For many years, deemed non determinist, Ethernet has gone through some evolution which enables its use in real-time applications. Nevertheless, Ethernet technology, by itself, does not include features above the lower layers of the OSI communication model. Although lots of attention has been devoted to the timing analysis of Ethernet-like technologies and solutions, most of the work on Ethernet has been restricted to the Data Link Layer level. This dissertation investigates the extraction of overall temporal properties of COTS factory-floor communication systems based on a concrete Ethernet-based COTS technology (EIP), which provides a fully defined communication protocol stack.

Primarily, de inner workings of the distributed system to analyse are overviewed. The main components of EIP systems are outlined and the major delay components in a defined distributed transaction are identified.

In order to tackle the problem of devising appropriate tools for the timeliness analysis of EIP systems, two main lines are followed.

One builds upon traditional real-time response time analysis to provide an analytical formulation enabling to find end-to-end response times in EIP based distributed systems. Although this analysis can still be further enhanced, it suffers of some pitfalls, being the most important of these, the inherent pessimistic results. *Per se* it contains a relevant contribution for the holistic analysis of Ethernet-based COTS systems.

The other approach promotes the idea that simulation is a useful tool for analysing and understanding complex systems. Indeed, the use of discrete-event simulation models can be a powerful tool for the timeliness evaluation of the overall system, but particular care must be taken with the results provided by traditional statistical analysis techniques. Therefore, some discussion was also introduced on the use of simulation results to perform statistical timeliness analysis.

A closing discussion, based on the preceding models and results presented, can be commenced by evaluating the distance between the worst-case of the analytical model results and the average and worst-case that actually can be verified within a considerable life-time of the simulation.

Besides the efforts made in the validation and analysis of the results correctness from both the simulation and analytical models, comparing results from both models provides a further step in their validation. As the analytical model developed is based on a set of

worst-case assumptions, its results should always bound the results given by the simulation model.

Table 9. Analytical model results for previously simulated scenario (Figure 28) (sorted by transaction periodicity)

Tr.	*Input*			*Output*			*Ri (ms)*
	$Li^{sn \to sw}$ (ms)	Li^{sw} (ms)	$Li^{sw \to en}$ (ms)	$Li^{sn \to sw}$ (ms)	Li^{sw} (ms)	$Li^{sw \to en}$ (ms)	
2	8.25	0.05	4.50	11.50	0.03	1.75	28.09
1	11.25	0.05	4.50	14.50	0.03	1.75	34.09
4	21.25	0.06	4.50	24.50	0.03	1.25	53.59
3	26.25	0.07	4.50	29.50	0.06	1.75	64.13
7	56.50	0.09	4.50	59.50	0.06	1.50	124.14
5	76.50	0.10	4.50	79.50	0.08	1.75	164.42
6	81.50	0.11	4.50	84.50	0.09	1.75	174.45
8	201.75	0.12	4.50	204.50	0.09	1.50	414.46
9	351.75	0.13	4.50	354.50	0.10	1.50	714.48

Table 9 exhibits the analytical model results for the scenario put forward earlier (Figure 28, Section 6.5). The previous data (from Table 9) is now set side by side, in Table 10, with the simulation model results. The table displays both the estimate average and the maximum value observe within all replications. The difference between the maximum value observed and the worst-case predicted by the analytical model is exhibited, in percentage, in the last column.

Table 10. Comparison of simulation with analytical model results (sorted by transaction periodicity)

Trans.	*Simulation model results*		*Math. model results (ms)*	*Difference (1-Math.Model/Sim. max)*
	Estimated Average (ms)	*Maximum observed (ms)*		
2	11.30	11.4	28.09	59%
1	15.81	16.0	34.09	53%
4	30.77	30.9	53.59	42%
3	38.36	38.9	64.13	39%
7	113.01	116.2	124.14	6%
5	83.13	85.7	164.42	48%
6	120.88	123.0	174.45	29%
8	300.70	305.3	414.46	26%
9	526.64	537.3	714.48	25%

A first observation to be made from the data in Table 10 is that the analytical model indeed bounds the end-to-end response times observed in the simulation. The expected pessimist inherent to the analytical formulation, based on a number of worst-case assumptions is confirmed, weighing the difference between the analytical model results and

the maximum observed through the set of simulation replications. Even more important is the fact that as the system becomes more complex, the pessimism increases, thus increasing the necessity to consider a stochastic representation of the events.

Actually, it is also noticeable that the maximum values observed are much closer to the estimated long-run average behaviour than the analytical model predictions. This is also justifiable by the fact that the analytical model aggregates a number of worst-case scenarios for the several delay components that, in reality, hardly ever occur in conjunction. This further motivates the need for different types of simulation and simulation output analysis, adequate to evaluate such issues correctly, rather than only extraction average case performance measurements. These matters are already currently being assessed [14] and will be overviewed in the Future Work Section.

The problem of developing methods to correctly introduce and handle probabilistic assumptions in analytical models has already been tackled by several researchers [64-69]. Nevertheless, even assuming the existence of a probabilistic characterisation of the system components, it is also clear that the correct characterisation, in statistical terms, of a system is very much sensitive and dependent on the concrete system and on the concrete application of the system. This characterisation becomes a problem with greater relevance when the complexity of the system is increasingly higher and an *a priori* evaluation of the system is required. Additionally, the correct results of a probabilistic analysis are, in great magnitude, dependent on these inputs.

The use of discrete event simulation models is thus an appealing approach for the analysis of intricate systems. Being a very practical tool and because of its approximation to the real world, discrete-event simulation presents itself as an attractive method to acquire knowledge of elaborate distributed systems, recurring to the statistical background already established for the analysis of simulation data.

As the complexity of systems increases (perhaps to the point where worst-case analytical-based techniques will fail to be useful), simulation or a combination of simulation with other techniques may be essential. It is however noted that simulation can be very good at modelling the middle of distributions, but there are numerous problems when trying to extract other performance measures, like worst-case end-to-end response time. The Future Work Section of this dissertation unveils some approaches to extract other measures of performance than means.

7.2 Future Work

In the course of the development of this dissertation a number of possible ramifications for further work have sprung. Some of these are already in the process of investigation, while others are little more than suggestions for further development. The most relevant of these are:

- fine-tuning and thorough validation of the models;
- development of a wrapping layer on top of the simulation model;
- introduction of less pessimistic assumptions into the analytical model;
- exploration of different performance measures for simulation.

7.2.1 Fine-tuning and Thorough Validation of the Models

The models presented are, to a considerable degree, tuned and validated. However, it was not possible to perform experimentation and comparison with large real systems. It is necessary to point out that such large systems are difficult to build, because of the high costs involved.

Further validations and fine-tuning could certainly be accomplished by comparing the models results with real data, especially in order to have a better understanding of how these models scale up.

7.2.2 Development of a Wrapping Layer

At this moment, the configuration of the simulation model is done directly in OMNeT++ configuration files. This is actually a very laborious task. It comprises the definitions of nodes, connections and their topology in the network. Each of these possesses a great number of parameters to configure, and, most of these configurations are repetitive and quite obtuse to do by hand.

The development of an application to hide these details from users, and enable them to easily build different simulations, reducing the repetitive tasks needed and providing a more intuitive interface to the system designer, would be of great value.

7.2.3 Introduction of Less Pessimistic Assumptions

Chapter 5 provided an effort to formulate an analytical solution enabling to find end-to-end response times in EIP based distributed systems. This formulation includes some rather pessimistic assumptions. Further developments on this model can possibly relax some of these assumptions and therefore reduce the pessimism of the overall model.

Still, with the development of simulation models for the same system, another perspective can be pursued. Discrete event simulation can also provide results enabling less pessimistic assumptions (e.g. on precedence or offsets of events) for the analytical response time formulations.

7.2.4 Different Measures of Performance in Simulation Studies

Simulation model are usually a very good tool for evaluation of average behaviour. Earlier, in Section 4.4.2, some well known approaches for estimating distributions out of simulation output, and the confidence which can be applied to its mean values where presented and applied. *But, what about worst-case behaviour?*

This is the basis for a discussion on the applicability of such approaches to derive confidence on the tail of distributions, where the worst-case is expected to be. Some options to extract worst-case performance measures that could be subject of future advancements are now briefly explored.

Extreme Value Theory

Goodness-of-fit tests may be used to evaluate the likeness between the sample data distribution and a theoretical distribution. If it is possible to obtain a good approximation from the theoretical distribution, then it is usually feasible to obtain good estimates of the output variables. However, for the purpose of drawing worst-case estimates from these distributions we are considering the tails of the probabilistic distributions and it is known that these are the areas where less accuracy exists. Considering that we capture a sufficient number of values close to the worst-case value during the simulation runs, we will probably end up with a heavy-tailed distribution. A distribution function or random variable is said to be heavy-tailed if it presents a high coefficient of variance. For example, in [70], the authors found that the distribution of execution times was better represented by a Gumbel distribution (a heavy tail-distribution). Other examples of heavy tail distributions include all extreme values distributions (Gumbel, Fréchet and Weibull), t-student or Pareto distributions.

An important property related to heavy tail distributions is that they are (essentially) invariant under maximisation (extreme value theory) [71]: This means that, if X(1), X(2), … are independent and identically distributed with common distribution function F that is heavy-tailed and M(n) = max(X(1),…,X(n)), then $M(n)/\sqrt[\alpha]{n} \to G$, as $n \to \infty$, in which G is the Fréchet distribution. This may suggest a generalisation for extreme values, similar to the central limit theorem for means, when *n* is large enough.

Heavy-tail distributions have been object of recent study in the fields of load balancing (CPU, network), job scheduling (Web servers) and complex system studies. Particularly, there are some proposals for modelling and analysing heavy-tail distributions for estimation of rare event probabilities with computable tractable techniques [72, 73].

Average Maximums

Derived from the methods referred in Section 4.4 for simulation output analysis, an intuitive approach, for trying to obtain an estimator for the worst-case value of the output variable is to pick the maximum value in the set of data from each simulation replica, instead of calculating a mean value. The problem of such approach is the assumption of a normally distributed variable, needed for the applicability of the previously mentioned methods for estimating means. A possible solution could be to group the values obtained in batches and apply the assumption of a normally distributed average over the means of each batch, in a similar way to the batch means procedure. Doing this could result in an

additional statistical error introduced by this second grouping. Additionally, the results obtained in this way, would not be exactly worst-case values, but average maximums, which can be a rather different thing and, to achieve the conditions of the central limit theorem, much more data would be necessary, most likely making this an impractical approach.

Rare Event Simulation

It is possible to view the event of a worst-case as a rare event, and the average system behaviour tends to be far apart from the worst-case. Obtaining precise estimates of such rare event probabilities using classical simulation can require prohibitively long run lengths.

A popular technique applied for the simulation of rare events is called *importance sampling*. Basically, importance sampling comprises of to different approaches. One, that attempts to modify the probability dynamics, in such a way that rare events will occurs more frequently. An alternative importance sampling technique is trajectory splitting, based on the assumption that there exist some well identifiable intermediate system states that much more often that the rare events of interest. The idea is to detect these intermediate states during simulation execution and split the simulation execution into several independent sub-trajectories, simulated from that state. Naturally, to obtain the final estimator, the results must be adjusted accordingly to the modification introduced. See [74] and references within for further information about importance sampling techniques.

Importance sampling may indeed obtain a significant reduction in the amount of observations required to obtain the same estimator precision as would be obtained in a simulation that does not use importance sampling, however, this requires a considerable amount of problem-specific knowledge from the simulation designer and how the modified distributions introduced will affect the distribution of the target events of interest. Reducing the simulation length, while simultaneously retaining the ease and flexibility of simulation is an important issue, receiving increasing considerable attention from researchers.

REFERENCES

[1] K. Mashford, "Next generation manufacturing", *IEEE Manufacturing Engineer*, vol. 82(6), pp. 30-40, 2003.

[2] Rockwell Automation, "Making Sense of e-Manufacturing: a Roadmap for Manufacturers", Rockwell Automation Inc., Cleveland, Ohio, Industry White Paper, 2000. Available online at www.rockwellautomation.com.

[3] E. Tovar, L. M. Pinho, and L. Almeida, "Position Paper on Time and Event-triggered Communication Services in the Context of e-Manufacturing", in proceedings of the IEEE Workshop on Large Scale Real-Time and Embedded Systems (LARTES'02), Austin, Texas, US, 2002.

[4] John Stankovic and K. Ramamritham, "Real-Time computing systems: The next generation", in *Tutorial: hard real-time systems*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1989, pp. 14-37.

[5] Giorgio C. Buttazzo, *Hard real-time computing systems predictable scheduling algorithms and applications*. Boston: Kluwer Academic Publishers, 1997.

[6] J. C. Palencia Gutierrez and Michael Gonzalez Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems", in proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99), Phoenix, Arizona, pp. 328-339, 1999.

[7] K. Tindell, "Adding time-offsets to schedulability analysis", University of York Dept. of Computer Science, Heslington, York, England, Technical Report YCS-94-221, 1994. Available online at ftp://ftp.cs.york.ac.uk/reports/YCS-94-221.ps.Z.

[8] J. C. Palencia Gutierrez and Michael Gonzalez Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", in proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98), Madrid, Spain, pp. 26-37, 1998.

[9] K. Tindell, "Holistic schedulability analysis for distributed hard real-time systems", *Microprocessors and Microprogramming, Elsevier Science Publishers*, vol. 50(No. 2-3), pp. 117-134, 1994.

[10] Marco Spuri, "Analysis of Deadline Scheduled Real-Time Systems", INRIA, Technical Report 2772, April 1996. Available online at http://www-rocq.inria.fr/reflecs/research_reports/RR-2772.pdf.

[11] ODVA, "Ethernet/IP Specification (Release 1.0)", ControlNet International and Open DeviceNet Vendor Association, Ethernet/IP Specification, June 5 2001. Available online at http://www.odva.org/.

[12] N. Pereira and E. Tovar, "Ethernet-based Systems: Contributions to the Holistic Analysis", in proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'4) - WIP Session, Catania, Italy, pp. 25-28, 2004.

[13] N. Pereira, E. Tovar, and L. M. Pinho, "INDEPTH: Timeliness Assessment of Ethernet/IP-based Systems", in proceedings of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), Volendam, The Netherlands, pp. 192-201, 2004.

[14] N. Pereira, E. Tovar, B. Baptista, L. M. Pinho, and I. Broster, "A Few What-Ifs on Using Statistical Analysis of Stochastic Simulation Runs to Extract Timeliness Properties", in proceedings of the 1st International Workshop on Probabilistic Analysis Techniques for Real-time and Embedded Systems (PARTES'04), Piza, Italy, 2004.

[15] N. Pereira, E. Tovar, and L. M. Pinho, "Timeliness in COTS Factory-Floor Distributed Systems: What Role for Simulation", in proceedings of the 5th IEEE International Workshop on Factory Communication Systems (WFCS'04), Vienna, Austria, pp. 13-21, 2004.

[16] Haydn A. Thompson, "Wireless and Internet communications technologies for monitoring and control", *Control Engineering Practice, Elsevier Science*, vol. 12(6), pp. 781-791, 2004.

[17] G. Kaplan, "Ethernet's winning ways", *IEEE Spectrum*, vol. 38(1), pp. 113-115, 2001.

[18] Berta Batista, "Industrial Ethernet: Building Blocks for a Holistic Approach", in proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02) - WIP Session, Vasteräs, Sweden, 2002.

[19] M. Volz, "Quo Vadis Layer 7", *The Industrial Ethernet Book*, vol. 5, 2001, 8-10. Available online at http://ethernet.industrial-networking.com/.

[20] ANSI/IEEE, "Telecommunications and information exchange between systems-Local and metropolitan area networks - Common Specification", ANSI/IEEE Std 802, 1998. Available online at http://standards.ieee.org/getieee802/802.3.html.

[21] Andrew S. Tanenbaum, *Computer Networks*, 3rd ed: Prentice Hall, Inc, 1996.

[22] T. Skeie, S. Johanssen, and O. Holmeide, "The Road to and End-to-End Deterministic Ethernet", in proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02), Vasteras, Sweden, pp. 3-9, 2002.

[23] ODVA, "CIP Common Specification (Release 1.0)", ControlNet International and Open DeviceNet Vendor Association, CIP Specification, 2000. Available online at http://www.odva.org/.

[24] ODVA, "DeviceNet Specification (Release 2.0)", ControlNet International and Open DeviceNet Vendor Association, DeviceNet Specification, April 1 2001. Available online at http://www.odva.org/.

[25] ODVA, "ControlNet Specification (Release 2.0)", ControlNet International and Open DeviceNet Vendor Association, ControlNet Specification, December 31 1999. Available online at http://www.odva.org/.

[26] ISO, "Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication", International Organization for Standardization, ISO 11898, 1993. Available online at http://www.iso.org/.

[27] L. Almeida, E. Tovar, J. Fonseca, and F. Vasques, "Schedulability Analysis of Real-Time Traffic in WorldFIP Networks: an Integrated Approach", *IEEE Transactions on Industrial Electronics*, vol. 49(5), pp. 1165-1174, 2002.

[28] Averill M. Law and W. David Kelton, *Simulation modeling and analysis*, 3rd ed. New York: McGraw-Hill, 2000.

[29] Richard A. Meyer and Rajive Bagrodia, "PARSEC User Manual, Release 1.1", UCLA Parallel Computing Laboratory, User Manual August 1998. Available online at http://pcl.cs.ucla.edu/projects/parsec/manual.pdf.

[30] Pawel Gburzynski, "An Overview of SMURPH: an Object-oriented Configurable Simulator for Low-Level Communication Protocols", Dept. of Computing Science of University of Alberta, Edmonton, Alberta, Canada, Technical Report, 1995. Available online at http://www.cs.ualberta.ca/~pawel/SMURPH/report.pdf.

[31] S.V. Rice, A. Marjanski, H.M. Markowitz, and S.M. Bailey, "Object-oriented SIMSCRIPT", in proceedings of the 37th Annual Simulation Symposium (ANSS'04), Hyatt Regency Crystal City, Arlington, VA, pp. 178-186, 2004.

[32] J.R. Holmevik, "Compiling SIMULA: a historical study of technological genesis", *IEEE Annals of the History of Computing*, vol. 16(4), pp. 25-37, 1994.

[33] H. D. Schwetman, "Using CSIM to model complex systems", in proceedings of the Winter Simulation Conference, San Diego, California, pp. 246 - 253, 1988.

[34] Xinjie Chang, "Network simulations with OPNET", in proceedings of the Winter Simulation Conference, Pheoenix Az, pp. 307-314, vol.1, 1999.

[35] B. Lewis Barnett, "An Ethernet performance simulator for undergraduate networking", in proceedings of the 24th SIGCSE technical symposium on Computer science education, Indianapolis, Indiana, United States, pp. 145 - 150, 1993.

[36] C.S. McDonald, "A Network Specification Language and Execution Environment for Undergraduate Teaching", in proceedings of the ACM Computer Science Education Technical Symposium, San Antonio, Texas, pp. 25-34, 1991.

[37] S. McCanne and S. Floyd, "ns-Network Simulator", University of California, Berkley, Online Resources, 1995. Available online at http://www-mash.cs.berkeley.edu/ns/.

[38] S. Keshav, "REAL 5.0 User Manual", Cornell University, User Manual, August 13th 1997. Available online at http://www.cs.cornell.edu/skeshav/real/user.html.

[39] A. Varga, "OMNeT++", *IEEE Network, Software Tools for Networking Column*, vol. 16(4), 2002.

[40] A. Varga, "OMNeT++ Discrete Event Simulation System", v2.3, 2004. Web Site: http://www.omnetpp.org/.

[41] John Stankovic, "Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems", *IEEE Computer*, vol. 21(10), pp. 10-19, 1988.

[42] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System", *The Computer Journal, British Computer Society*, vol. 29(5), pp. 390-395, 1986.

[43] A. Burns, "Scheduling Hard Real-Time Systems: A Review", *IEEE Software Engineering Journal, Special Issue on Real-Time Systems*, vol. 6(3), pp. 116-128, 1991.

[44] C. M. Krishna and Kang G. Shin, *Real-time systems*. New York: McGraw-Hill, 1997.

[45] C. L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM (JACM)*, vol. 20(1), pp. 46-61, 1973.

[46] J. Leung and J. Whitehead, "On the Complexity of fixed-priority Scheduling of Periodic Real-Time Tasks", *Performance Evaluation, Elsevier Science*, vol. 22(4), pp. 237-250, 1982.

[47] E. Lawler and C. Martel, "Scheduling Periodically Occurring Tasks on Multiple Processors", *Information Processing Letters, Elsevier Science*, vol. 12(1), pp. 9-12, 1981.

[48] K. Tindell, "An extendible approach for analysing fixed priority hard real-time tasks", University of York Dept. of Computer Science, Heslington, York, England, Technical report YCS-92-189, 1992. Available online at ftp://ftp.cs.york.ac.uk/reports/YCS-92-189.ps.Z.

[49] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritence protocols: An approach to Real-Time Synchronization", *IEEE Transactions on Computers*, vol. 39(9), pp. 1175-1185, 1990.

[50] N. Audsley, A. Burns, M. Richardson, K Tindell, and A Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *IEEE Software Engineering Journal*, vol. 8(5), pp. 285-292, 1993.

[51] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: controller area network (CAN)", in proceedings of the 15th Real-Time Systems Symposium (RTSS'04), pp. 259-263, 1994.

[52] E. Tovar and F. Vasques, "Pre-run-time schedulability analysis of P-NET fieldbus networks", in proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (IECON'98), Aachen, Germany, pp. 236-241, 1998.

[53] E. Tovar, F. Vasques, and A. Burns, "Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation", *Real-Time Systems Journal, Kluwer Academic Publishers*, vol. 22(3), pp. 229-249, 2002.

[54] E. Tovar and F. Vasques, "Real-time fieldbus communications using Profibus networks", *IEEE Transactions on Industrial Electronics*, vol. 46(6), pp. 1241-1251, 1999.

[55] H. Hansson, M. Sjodin, and K. Tindell, "Guaranteeing real-time traffic through an ATM network", in proceedings of the 30th Hawaii International Conference on System Sciences (HICSS'97), Maui, Hawaii, pp. 44-53, vol. 5, 1997.

[56] K. Pawlikowski, H. D. J. Jeong, and J. S. R. Lee, "On credibility of simulation studies of telecommunication networks", *IEEE Communications Magazine*, vol. 40(1), pp. 132-139, 2002.

[57] David S. Moore and George McCabe, "From probability to Inference", in *Introduction to the practice of statistics*, 3rd ed: W.H. Freeman and Company, 1999, pp. 373-431.

[58] Jerry Banks, John S. II Carson, Barry L. Nelson, and David M. Nicol, *Discrete-Event System Simulation*. Upper Saddle River: Prentice Hall, 2001.

[59] P. Heidelberger and P. A. W. Lewis, "Quantile Estimation in Dependent Sequences", *Operations Research*, vol. 31(1), pp. 185-209, 1984.

[60] J.-S. R. Lee, D. McNicle, and K. Pawlikowski, "Quantile Estimations in Sequential Steady-State Simulation", in proceedings of the European Simulation Multiconference (ESM'99), International Society for Computer Simulation, Warsaw, pp. 168-174, 1999.

[61] Yequiong Song, Anis Koubaa, and Francois Simonot, "Switched Ethernet for Real-Time Industrial Communication: Modelling and Message Buffering Delay Evaluation", in proceedings of the 4th IEEE Int. Workshop on Factory Communication Systems (WFCS02), Vasteräs, Sweden, pp. 27-35, 2002.

[62] Jean-Philippe Georges, Eric Rondeau, and Thierry Divoux, "Evaluation of switched Ethernet in an industrial context by using the Network Calculus", in proceedings of the 4th IEEE Int. Workshop on Factory Communication Systems (WFCS02), Vasteräs, Sweden, pp. 19-26, 2002.

[63] S. Deering, "Host Extensions for IP Multicasting", *RFC 1112*: Stanford University, 1989.

[64] Giorgio C. Buttazzo and L. Abeni, "QoS Guarantee Using Probabilistic Deadlines", in proceedings of the 11th IEEE Euromicro Conference on Real-Time Systems (ECRTS'99), York, UK, pp. 242-249, 1999.

[65] J. Díaz, D. García, K. Kim, C. Lee, L. Lo Bello, J. López, S. L. Min, and O. Mirabella, "Stochastic Analysis of Periodic Real-Time Systems", in proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02), Austin, Texas, pp. 289-300, 2002.

[66] M. K. Gardner, "Analysis and Scheduling of Critical Soft Real-Time Systems", PhD thesis, University of Illinois, Urbana Champaign, 1999. Available online at http://www.cs.uiuc.edu/Dienst/UI/2.0/Describe/ncstrl.uiuc_cs/UIUCDCS-R-99-2114.

[67] J. Lehoczky, "Real-Time Queuing Network Theory", in proceedings of the 18th The IEEE Real-Time Systems Symposium (RTSS'97), San Francisco, California, pp. 58-67, 1997.

[68] J. P. Lehoczky, "Real-Time Queuing Theory", in proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96), Washington, DC, pp. 186-195, 1996.

[69] T. S. Tia, Z. Dens, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. S. Liu, "Probabilistic Performance Guarantees for Real-Time Tasks with Varying Computation Times", in proceedings of the Real-Time Technology and Applications Symposium (RTAS'95), Chicago, Illinois, pp. 164-173, 1995.

[70] S. Edgar and A. Burns, "Statistical Analysis of WCET for Scheduling", in proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01), London, UK, pp. 215-224, 2001.

[71] S. Coles, *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag, 2001.

[72] D. Starobinski and M. Sidi, "Modeling and Analysis of Heavy-Tailed Distributions via Classical Telegraffic Methods", *Queueing Systems (QUESTA)*, vol. 36(1-3), pp. 243-267, 2000.

[73] S. Asmussen, D. P. Kroese, and R. Rubinstein, "Heavy Tails, Importance Sampling and Cross-Entropy", University of Aarhus August 2003. Available online at http://mefast.sta.unipi.gr/iwap2004/Abstracts/FinalAbstracts/IWAP2004_Rubinstein.pdf.

[74] J.K. Townsend, Z. Haraszti, J.A. Freebersyser, and M. Devetsikiotis, "Simulation of rare events in communications networks", *IEEE Communications Magazine*, vol. 36(8), pp. 36-41, 1998.

GLOSSARY

C++SIM Programming tool for simulation of discrete processes based on the C++ language. *See Section 3.2*.

CAN *Controller Area Network* is a serial bus standard originally developed by Robert Bosch GmbH for connecting electronic control units. It was initially created for the automotive market (as a vehicle bus). But nowadays it is used in many embedded control applications. The CAN data link layer protocol is standardised in ISO 11898-1 (2003). This standard describes mainly the data link layer - comprised by logical link control (LLC) sublayer and the Media Access Control (MAC) sub-layer - and some aspects of the physical layer of the ISO/OSI Reference Model. All the other protocol layers are left to the network designer's choice.

CIP *Control and Information Protocol* is an application layer protocol that implements a distributed object model. It is a very versatile protocol that has been designed with the automation industry in mind. However, due to its very open nature, it can be applied to many more areas. *See Section 2.3*.

CNET Discrete-event network simulator enabling experimentation with various data-link layer, network layer, routing and transport layer networking protocols. *See Section 3.3.*

ControlNet ControlNet technology provides deterministic and repeatable, communications for the most demanding factory-floor automation applications. It delivers high-speed transport of both time-critical I/O and messaging data on single or redundant physical media. *See Section 2.3.*

CORBA *Common Object Request Broker Architecture*. The CORBA standard is created and controlled by the Object Management Group (OMG). It defines APIs, communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA therefore provides platform and location transparency for well-defined objects, which are the fundamental underpinnings of any distributed computing platform.

CoS *Change-of-State*. *See Section 2.3.*

COTS *Commercial-Of-The-Shelf*. Term for systems which its components are manufactured commercially. COTS systems are in contrast to systems that are produced entirely and uniquely for the specific application.

CSIM Programming tool for simulation of discrete processes based on the C language. *See Section 3.2*.

CSMA/CD *Carrier Sense Multiple Access with Collision Detection* is a network control protocol in which a carrier sensing scheme is used and a transmitting data station that detects another signal while transmitting a frame, stops transmitting that frame, transmits a jam signal, and then waits for a random time interval before trying to send that frame again.

CTDMA *Concurrent Time Domain Multiple Access*. A time slice medium access algorithm used by ControlNet. *See Section 2.3*.

DCOM *Distributed Component Object Model* is a Microsoft proprietary technology for software components distributed across several networked computers. It is descended from COM, later part of COM+. It has been deprecated in favour of Microsoft .NET.

DEC *Digital Equipment Corporation* is a pioneering company in the American computer industry. This acronym was once officially used by DEC itself, but discarded in favour of "Digital" in order to avoid a trademark dispute. They were later acquired by Compaq, who subsequently merged with Hewlett-Packard. As of 2004 their product lines are still produced under the HP name.

DeviceNet The DeviceNet network is an open low-level network that provides connections between simple industrial devices (such as sensors and actuators) and higher-level devices (such as PLC controllers and computers). *See Section 2.3*.

EDF *Earliest Deadline First* is a dynamic priority real-time scheduling policy. With EDF, the task with the earliest deadline is always executed first. *See Section 4.2*.

EIP *Ethernet/IP*.

Ethernet/IP Ethernet/IP (EIP), where IP stands for Industrial protocol is a high-level industrial protocol for industrial automation applications. Built on the standard TCP/IP protocol suite, it uses all the traditional Ethernet hardware and software to define an application layer protocol for configuring, accessing and controlling industrial automation devices. Ethernet/IP classifies Ethernet nodes as predefined device types with specific behaviours. The set of device types and the EIP application layer protocol is based on the Control and Information Protocol (CIP). *See Section 2.3*

Fieldbus A generic term used to describe a common communications protocol for control systems and/or field instruments.

FPS *Fixed Priority Scheduling*. *See Section 4.2.*

IEEE The *Institute of Electrical and Electronics Engineers* is a non-profit, professional organisation based in the United States. The IEEE was formed in 1963 by the merger of the Institute of Radio Engineers (IRE) and the 'American Institute of Electrical Engineers' (AIEE). The IEEE has branches in many parts of the world. Its members are electrical engineers, computer scientists, telecommunications workers, etc. Its goal is to promote knowledge of electrical engineering. One of its most important roles is in establishing standards for computers formats and devices.

IGMP The *Internet Group Management Protocol* is a communications protocol used to manage the membership of Internet Protocol multicast groups. IGMP is used by IP hosts and adjacent multicast routers to establish multicast group memberships. It is an integral part of the IP multicast specification, like ICMP for unicast connections.

Java/RMI The *Java Remote Method Invocation* API, or RMI, is an application programming interface for performing remote procedural calls.

MAC *Medium Access Control*. The lower sub-layer of the OSI data link layer, the interface between a node's Logical Link Control and the network's physical layer. The MAC differs for various physical media. The MAC sub-layer is primarily concerned with breaking data up into data frames, transmitting the frames sequentially, processing the acknowledgment frames sent back by the receiver, handling address recognition, and controlling access to the medium.

NED *NEtwork Description* is an OMNeT++ specific language usage to define a network's properties in a modular way. *See Section 3.4.*

NetSim Simulation package intended to offer a very detailed simulation of Ethernet.3.3

Ns-2 *Network Simulator 2* is a discrete event simulator targeted at networking research. Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. *See Section 3.3*.

OMNeT++ *Objective Modular Network Testbed in C++*, a public-source, object-oriented modular discrete event simulation package that can be used for modelling: communication protocols, computer networks and traffic modelling, multiprocessors and distributed systems, etc. OMNeT++ also supports animation and interactive execution. *See Section 3.4*.

OPC *OLE for Process Control* is the original name for a standard developed in 1996 by an industrial automation industry task force. The standard specified the communication of real-time plant data between control devices from different manufacturers. The original standard was based on the OLE COM and DCOM technologies developed by Microsoft Corporation for the MS Windows operating system family. The standard is now maintained by the OPC Foundation and has been renamed the OPC Data Access standard.

OPNET OPNET is widely held has the state-of-art in network simulation, its suite of products that combines predictive modelling and a comprehensive understanding of networking technologies to enable design, deployment, and management of network infrastructures, network equipments, and networked applications. *See Section 3.3*.

OSI The *Open Systems Interconnection* Reference Model (OSI Model or OSI Reference Model for short) is a layered abstract description for communications and computer network protocol design, developed as part of the Open Systems Interconnect initiative. It is also called the OSI seven layer model. The model divides the functions of a protocol into a series of layers. Each layer has the property that it only uses the functions of the layer below, and only exports functionality to the layer above. A system that implements protocol behaviour consisting of a series of these layers is known as a 'protocol stack' or 'stack'. Protocol stacks can be implemented either in hardware or software, or a mixture of both. Typically, only the lower layers are implemented in hardware, with the higher layers being implemented in software.

PARSEC *PARallel Simulation Environment for Complex systems*, a C-based simulation language, developed by the UCLA Parallel Computing Laboratory, for sequential and parallel execution of discrete-event simulation models. *See Section 3.2.*

PRNG A *Pseudo-Random Number Generator* is an algorithm which generates a sequence of numbers, the elements of which are approximately independent of each other.

RM *Rate Monotonic* real-time scheduling policy. The term “rate monotonic” originated as a name for the optimal task priority assignment in which higher priorities are accorded to tasks that execute at higher rates (that is, as a monotonic function of rate). Rate monotonic scheduling is a term used in reference to fixed priority task scheduling that uses a rate monotonic prioritisation. *See Section 4.2*.

RPI *Requested Packet Interval*. Parameter that defines the periodicity of CIP connections. *See Section 2.3*.

SIMSCRIPT SIMSCRIPT is a simulation language with both declarative and procedural features, designed for discrete-event and hybrid discrete/continuous modelling. *See Section 3.2.*

SIMULA The SIMULA programming language was designed and built at the Norwegian Computing Center (NCC) in Oslo between 1962 and 1967. It was originally designed and implemented as a language for discrete event simulation, but was later expended and re-implemented as a full scale general purpose programming language. *See Section 3.2.*

SMURPH A *System for Modelling Unslotted Real-time PHenomena* is intended to the simulating communication protocols at the medium access control level. *See Section 3.3.*

TCP/IP The Internet protocol suite is the set of protocols that implement the protocol stack on which the Internet runs. It is sometimes called the TCP/IP protocol suite, after the two most important protocols in it: the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which were also the first two defined.

TCP *Transmission Control Protocol* is a connection-oriented, reliable delivery byte-stream transport layer protocol currently documented in IETF RFC 793.

UDP The *User Datagram Protocol* is a minimal message-oriented transport layer protocol that is currently documented in IETF RFC 768.

TCP/UDP/IP In this dissertation, used to denominate the Internet protocol suite, emphasising the use of the UDP protocol layer.

TDMA *Time Division Multiple Access* is a medium access control scheme for shared medium networks. It allows several users to share the same frequency by dividing it into different time slots. The users transmit in rapid succession, one after the other, each using their own timeslot. *See Section 5.4.*

WCET *Worst-case Execution Time*. See *Section 4.2.*

WWW The *World Wide Web* (the "Web" or "WWW" for short) is a distributed (not centralised) hypertext system that operates over the Internet. Hypertext is browsed using a program called a web browser which retrieves pieces of information (called "documents" or "web pages") from web servers (or "web sites") and displays them on your screen. You can then follow hyperlinks on each page to other documents or even send information back to the server to interact with it.

INDEX

Appendixes

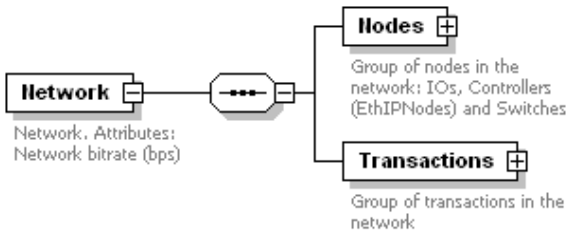
APPENDIX A

A.1 Network Definition Schema Documentation

For calculating the results of the analytical model, a small tool was developed. The most arduous task of this tool is the handling of the network definitions. For this, the tool reads XML files that define the topology of a network along with all the variables needed for the calculations. The next sections document the XML schema for these files.

A.1.1 Element Network

diagram



namespace ethipnet-schema

children **Nodes** **Transactions**

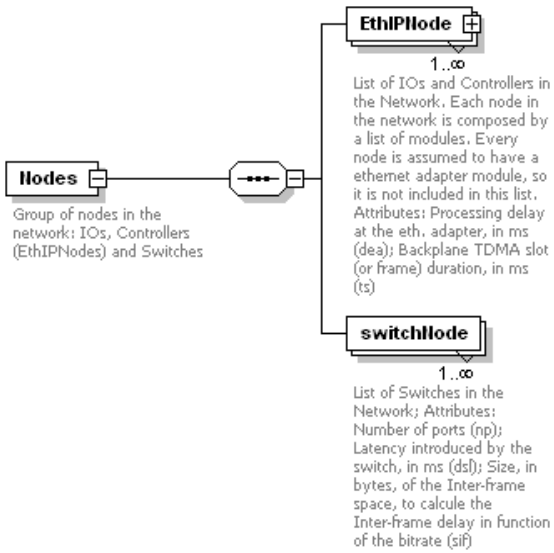
attributes

Name	Type	Use	Default	Fixed	Annotation
name	xs:string	required			
bps	xs:long	required			

annotation documentation Network. Attributes: Network bitrate (bps)

A.1.2 Element Network/Nodes

diagram

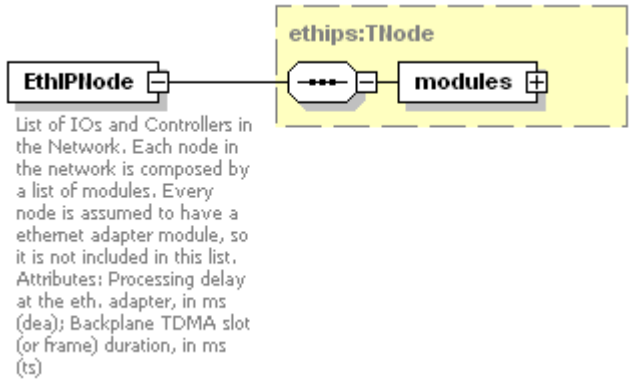


children EthIPNode switchNode

annotation documentation Group of nodes in the network: IOs, Controllers (EthIPNodes) and Switches

A.1.3 Element Network/Nodes/EthIPNode

diagram



type ethips:TNode

children modules

attributes

Name	Type	Use	Default	Fixed	Annotation
nodeType	ethips:TNodeType	required			
id	xs:ID	required			
dea	xs:double	required			
ts	xs:double	required			

annotation

documentation: List of IOs and Controllers in the Network. Each node in the network is composed by a list of modules. Every node is assumed to have a ethernet adapter module, so it is not included in this list. Attributes: Processing delay at the eth. adapter, in ms (dea); Backplane TDMA slot (or frame) duration, in ms (ts)

A.1.4 Element Network/Nodes/switchNode

diagram

type: ethips:TSw

attributes

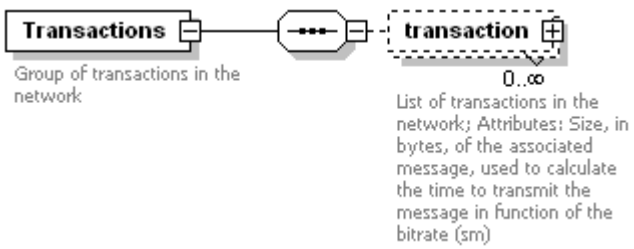
Name	Type	Use	Default	Fixed	Annotation
id	xs:ID	required			
np	xs:int	required			
dsl	xs:double	required			
sif	xs:int	required			

annotation

documentation: List of Switches in the Network; Attributes: Number of ports (np); Latency introduced by the switch, in ms (dsl); Size, in bytes, of the Inter-frame space, to calcule the Inter-frame delay in function of the bitrate (sif)

A.1.5 Element Network/Transactions

diagram

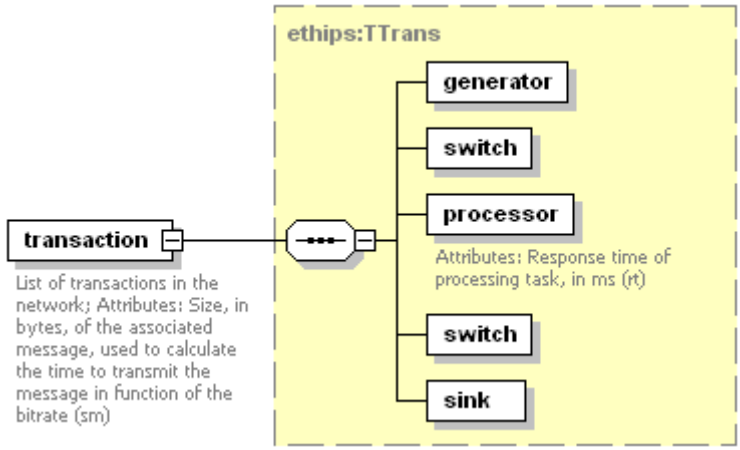


children transaction

annotation documentation Group of transactions in the network

A.1.6 Element Network/Transactions/transaction

diagram



type extension of ethips:TTrans

children generator switch processor switch sink

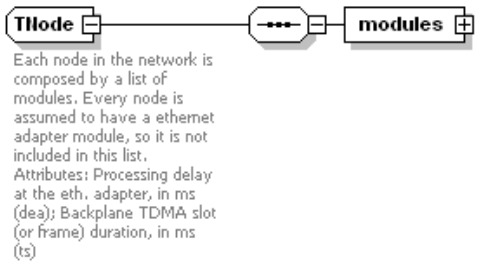
attributes

Name	Type	Use	Default	Fixed	Annotation
tid	xs:ID	required			
sm	xs:int	required			

annotation documentation List of transactions in the network; Attributes: Size, in bytes, of the associated message, used to calculate the time to transmit the message in function of the bitrate (sm)

A.1.7 Complex Type TNode

diagram



namespace ethipnet-schema

children modules

used by element Network/Nodes/EthIPNode

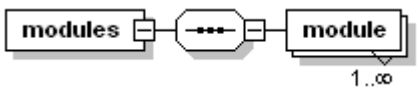
attributes

Name	Type	Use	Default	Fixed	Annotation
nodeType	ethips:TNodeType	required			
id	xs:ID	required			
dea	xs:double	required			
ts	xs:double	required			

annotation documentation Each node in the network is composed by a list of modules. Every node is assumed to have a ethernet adapter module, so it is not included in this list. Attributes: Processing delay at the eth. adapter, in ms (dea); Backplane TDMA slot (or frame) duration, in ms (ts)

A.1.8 Element TNode/modules

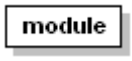
diagram



children **module**

A.1.9 Element TNode/modules/module

diagram

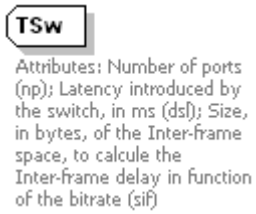


attributes

Name	Type	Use	Default	Fixed	Annotation
mid	xs:ID	required			
type	ethips:TModuleType	required			

A.1.10 Complex Type TSw

diagram



namespace ethipnet-schema

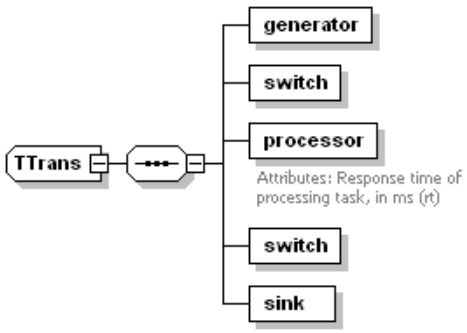
used by element Network/Nodes/switchNode

attributes	Name	Type	Use	Default	Fixed	Annotation
	id	xs:ID	required			
	np	xs:int	required			
	dsl	xs:double	required			
	sif	xs:int	required			

annotation documentation: Attributes: Number of ports (np); Latency introduced by the switch, in ms (dsl); Size, in bytes, of the Inter-frame space, to calcule the Inter-frame delay in function of the bitrate (sif)

A.1.11 Complex Type TTrans

diagram



namespace: ethipnet-schema

children: generator switch processor switch sink

used by: element Network/Transactions/transaction

attributes	Name	Type	Use	Default	Fixed	Annotation
	tid	xs:ID	required			

A.1.12 Element TTrans/generator

diagram

type: ethips:TTransModule

attributes	Name	Type	Use	Default	Fixed	Annotation
	mid	xs:IDREF	required			

A.1.13 Element TTrans/switch

diagram

type: ethips:TTransSw

attributes

Name	Type	Use	Default	Fixed	Annotation
id	xs:IDREF	required			

A.1.14 Element TTrans/processor

diagram

type extension of ethips:TTransModule

attributes

Name	Type	Use	Default	Fixed	Annotation
mid	xs:IDREF	required			
rt	xs:double	required			

annotation documentation Attributes: Response time of processing task, in ms (rt)

A.1.15 Element TTrans/switch

diagram

type ethips:TTransSw

attributes

Name	Type	Use	Default	Fixed	Annotation
id	xs:IDREF	required			

A.1.16 Element TTrans/sink

diagram

type ethips:TTransModule

attributes

Name	Type	Use	Default	Fixed	Annotation
mid	xs:IDREF	required			

A.1.17 Complex Type TTransModule

diagram

namespace ethipnet-schema

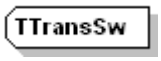
used by elements TTrans/generator TTrans/processor TTrans/sink

attributes

Name	Type	Use	Default	Fixed	Annotation
mid	xs:IDREF	required			

A.1.18 Complex Type TTransSw

Diagram



namespace ethipnet-schema

used by elements TTrans/switch TTrans/switch

Attributes

Name	Type	Use	Default	Fixed	Annotation
id	xs:IDREF	required			

A.1.19 Simple Type TModuleType

namespace ethipnet-schema

Type restriction of xs:string

used by attribute TNode/modules/module/@type

Facets
- enumeration Input
- enumeration Output
- enumeration Controller

A.1.20 Simple Type TNodeType

namespace ethipnet-schema

Type restriction of xs:string

used by attribute TNode/@nodeType

facets
- enumeration Ctrl
- enumeration IO

APPENDIX B

B.1 Simulation Model Documentation

The documentation for the simulation model developed is presented next. The first three sections document the NED modules developed: Simple Modules, Compound Modules and Channels. Then the Messages defined are documented.

Finally, the subsequent sections document the several C++ classes implemented.

B.1.1 Simple Modules

B.1.1.1 Module Backplane

File: Backplane.ned

Module Name: Backplane

Abstract: Models an Ethernet/IP device backplane. in[i], out[i] gate pairs represent the backplane interface of each module. The backplane is encharged of serving these queues, simulating a time divison protocol. Insted of doing time divison, the backplane can just insert a delay, in function of the number of connections to the backplane. For this, the parameter "timeDivison" must be false. This is usefull for simulation performance reasons.

It inspects the connection id of each packet in order to make the the apropriate decisons about where it should deliver the packets.

IMPORTANT: To improve simulation performance, it is possible to disable the backplane time divison. To do this, set the parameter "timeDivison" to false.

To be able to make decisions where to deliver de received packets, it depends on the connected modules having a CIPLayer (or CIPBridgeLayer), with one or more connected IOConnection modules. It analyses the connected modules in order to know the produced and consumed connections in each module.

In the case of a CIPBridgeLayer, the parameters "connectionIDProducedList" and "connectionIDConsumedList" of the of the connected CIPBridgeLayer. These parameters define which connections are bridged throught the CIPBridgeLayer. They are defined by a string, with the several connection IDs separated by ":".

NOTE: No fragmentation of the received CIP messages is made.

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.2 Used in compound modules:

EthIPController Module Name: EthIPController

EthIPIO Module Name: EthIPIO

B.1.1.3 Parameters:

Name	Type	Description
tTableTime	numeric	transmit table time
frameTime	numeric	frame transmit time
timeDivison	bool	defines if the backplane will really do time divison;

B.1.1.4 Gates:

Name	Direction	Description
in [ ]	input	each in[k], out[k] pair represents a backplane interface of a module
out [ ]	output	each in[k], out[k] pair represents a backplane interface of a module

B.1.1.5 Module CIPBridgeLayer

File: **CIPBridgeLayer.ned**

Module Name: CIPBridgedLayer

Abstract: A CIPLayer that bridges between the backplane and the TCP/IP stack

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.6 Used in compound modules:

EthIPAdapter Module Name: EthIPAdapter

B.1.1.7 Parameters:

Name	Type	Description
nodename	string	parameters
multicastGroupPrefix	string	

B.1.1.8 Gates:

Name	Direction	Description
from_bp [ ]	input	
from_ntw	input	
to_bp [ ]	output	
to_ntw	output	

B.1.1.9 Module CIPLayer

File: **CIPLayer.ned**

Module Name: CIPLayer

Abstract: A CIPLayer that encapsulates/decapsulates CIP data items into CIP transport packets. Layers between IOProcessing and the backplane.

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.10 Used in compound modules:

EthIPControllerModule	Module Name: EthIPControllerModule
EthIPIOModule	Module Name: EthIPIOModule

B.1.1.11 Parameters:

Name	Type	Description
nodename	string	parameters

B.1.1.12 Gates:

Name	Direction	Description
from_cipio [ ]	input	
from_bp	input	
to_cipio [ ]	output	
to_bp	output	

B.1.1.13 Module ControllerTask

File: **ControllerTask.ned**

Module Name: ControllerTask

Abstract: Models a controller Task. Receives a messages in the input gate and delivers them in the corresponding output gate (output gate with the same index), after a defined task response time

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.14 Used in compound modules:

EthIPControllerModule Module Name: EthIPControllerModule

B.1.1.15 Parameters:

Name	Type	Description
nodename	string	
responseTime	numeric	

B.1.1.16 Gates:

Name	Direction	Description
in [ ]	input	
out [ ]	output	

B.1.1.17 Module IOConnection

File: **IOConnection.ned**

Module Name: IOConnection

Abstract: Layers between Input, Output or Controller modules and IOConnection. Either acts like an input connection or output connection. But not both at the same time. Its behaviour is defined by the in gate connected (from_cip = output; from_input = input) When acting as an input connection it will generate a message at each RPI with the last data item value received. As an output it will simply forward messages to the out gate.

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.18 Used in compound modules:

EthIPControllerModule Module Name: EthIPControllerModule

EthIPIOModule Module Name: EthIPIOModule

B.1.1.19 Parameters:

Name	Type	Description
nodename	string	
connectionID	numeric	Used to construct the multicast group addr.; should be unique, between 1-255
ASICDelay	numeric	Configurable ASIC delay
rpi	numeric	Requested Packet Interval for the input

B.1.1.20 Gates:

Name	Direction	Description
out	output	in an output this is used to send messages to a controller
from_input	input	input gate for an input module (cannot be connected at the same time as from_cip gate)
from_cip	input	output gate for an output module (cannot be connected at the same time as from_input gate)

B.1.1.21 Module Input

File: **Input.ned**

C++ definition: **click here**

Module Name: Input

Abstract: Models an input data generator. Creates cip data Items to be send to lower processing. Sets the timestamp of the messages, so the receiving output can calculate the end-to-end response time.

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.22 Used in compound modules:

EthIPIOModule Module Name: EthIPIOModule

B.1.1.23 Parameters:

Name	Type	Description
nodename	string	
period	numeric	Configurable period of data generation
filterDelay	numeric	Configurable filter delay (IMPORTANT: Must be less than the data generation period)
hwDelay	numeric	Configurable hardware delay
dataLength	numeric	CIP class 0 or class 1 packet length

B.1.1.24 Gates:

Name	Direction	Description
out	output	

B.1.1.25 Module NetworkLayers

File: **NetworkLayers.ned**

Module Name: NetworkLayers

Abstract: NetworkLayers implement the TCP/IP Layer and Physical Layer functionalities.

Responsible for encapsulating/decapsulating CIP UDP transport packets to be sent to the TCP/IP ethernet network.

Assumes that a “intelligent” switching device is interconnecting the nodes

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.26 Used in compound modules:

EthIPAdapter Module Name: EthIPAdapter

B.1.1.27 Parameters:

Name	Type	Description
nodename	string	
ipAddress	string	
ipProcessingDelay	numeric	

macProcessingDelay	numeric

B.1.1.28 Gates:

Name	Direction	Description
from_phy	input	
from_application	input	
to_phy	output	
to_application	output	

B.1.1.29 Module Output

File: **Output.ned**

Module Name: IOProcessing.ned

Abstract: Receives output data. Records statistics about the received data. Namely the end-to-end response time.

Author: Nuno Pereira

Environment: OMNet++ simple module

Revision History: Created Feb 2004.

B.1.1.30 Used in compound modules:

EthIPIOModule Module Name: EthIPIOModule

B.1.1.31 Parameters:

Name	Type	Description
Nodename	string	
connectionID	numeric	

B.1.1.32 Gates:

Name	Direction	Description
In	input	

B.1.1.33 Module Switch

File: **Switch.ned**

Module Name: Switch

Abstract: This is a model of a switch for an Ehternet/IP network. It delivers a copy of the received packets to all corresponding ports, where consumers of the connection are present. To know where to deliver the packets, it uses parameters “connectionIDProducedList” and “connectionIDConsumedList” of the connected modules.

Author: Nuno Pereira

Environment: OMNet++ compound module

Revision History: Created Feb 2004.

B.1.1.34 Used in compound modules:

EthIPNetwork1 Module Name: EthIPNetwork1

EthIPNetwork2 Module Name: EthIPNetwork1

B.1.1.35 Parameters:

Name	Type	Description
nodename	string	
switchDelay	numeric	

B.1.1.36 Gates:

Name	Direction	Description
in [ ]	input	
Out [ ]	output	

B.1.2 Compound Modules

B.1.2.1 Module EthIPAdapter

File: **EthIPAdapter.ned**

Module Name: EthIPAdapter

Abstract: Ethernet/IP Adapter. Bridges commnunications from the backplane to the ethernet network. Has two communication interfaces: the backplane and full-duplex ethernet.

The parameters “connectionIDProducedList” and connectionIDConsumedList define which connections are bridged throught the adapter:

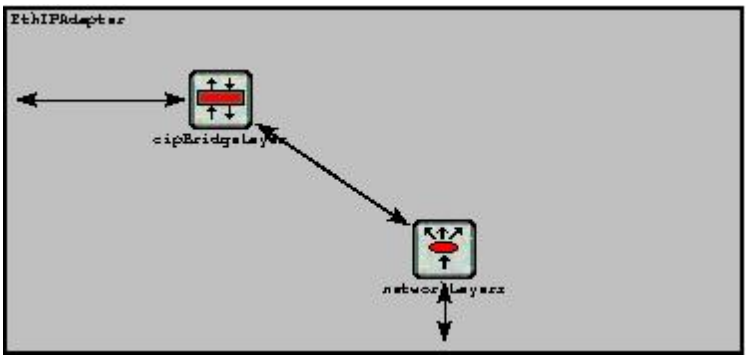
- connectionIDProducedList: Connections comming from ethernet, passing throught this adapter to the backplane - connectionIDConsumedList: Connections comming from the backplane, passing throught this adapter to ethernet

They are defined by a string, with the several connection IDs separated by “:”.

Author: Nuno Pereira

Environment: OMNet++ compound module

Revision History: Created Feb 2004.



B.1.2.2 Contains the following modules:

CIPBridgeLayer Module Name: CIPBridgedLayer

NetworkLayers Module Name: NetworkLayers

B.1.2.3 Used in compound modules:

EthIPController Module Name: EthIPController

EthIPIO Module Name: EthIPIO

B.1.2.4 Parameters:

Name Type Description

nodename	string	
connectionIDProducedList	string	Bridged connections
connectionIDConsumedList	string	Connections comming from the backplane, passing throught this CIPBridgeLayer

B.1.2.5 Gates:

Name	Direction	Description
from_backplane	input	
to_backplane	output	
from_eth	input	
to_eth	output	

B.1.2.6 Module EthIPController

File: **EthIPController.ned**

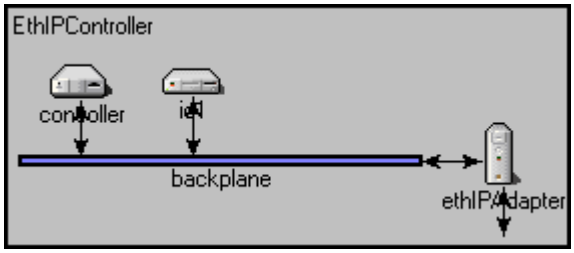
Module Name: EthIPController

Abstract: Model of an Ethernet/IP Controller Composed of one or more controller modules (defined by parameter numControllerModules), one or more IO Modules (defined by parameter numIOModules), a backplane and an EThernet/IP Adapter.

Author: Nuno Pereira

Environment: OMNet++ compound module

Revision History: Created Feb 2004.



B.1.2.7 Contains the following modules:

Backplane	Module Name: Backplane
EthIPAdapter	Module Name: EthIPAdapter
EthIPControllerModule	Module Name: EthIPControllerModule
EthIPIOModule	Module Name: EthIPIOModule

B.1.2.8 Used in compound modules:

EthIPNetwork1 Module Name: EthIPNetwork1

EthIPNetwork2 Module Name: EthIPNetwork1

B.1.2.9 Parameters:

Name	Type	Description
nodename	string	
numIOModules	numeric	modules configuration
numControllerModules	numeric	connections
connectionIDProducedList	string	
connectionIDConsumedList	string	

B.1.2.10 Gates:

Name	Direction	Description
in	input	
out	output	

B.1.2.11 Unassigned submodule parameters:

Name	Type	Description
ioModule[*].numInputs	numeric	i/o configuration
ioModule[*].numOutputs	numeric	
controllerModule[*].numInputs	numeric	i/o configuration
controllerModule[*].numOutputs	numeric	
controllerModule[*].controllerTask.responseTime	numeric	

B.1.2.12 Module EthIPControllerModule

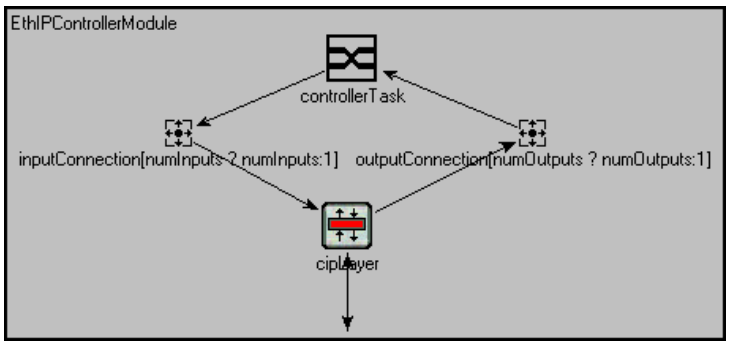
File: **EthIPControllerModule.ned**

Module Name: EthIPControllerModule

Abstract: Model of an Ethernet/IP Controller This controller has one controller task that consumes/produces all IOs of the controller module.

Author: Nuno Pereira

Environment: OMNet++ compound module
Revision History: Created Fev 2004.



B.1.2.13 Contains the following modules:

CIPLayer Module Name: CIPLayer

ControllerTask Module Name: ControllerTask

IOConnection Module Name: IOConnection

B.1.2.14 Used in compound modules:

EthIPController Module Name: EthIPController

B.1.2.15 Parameters:

Name	Type	Description
nodename	string	
numInputs	numeric	i/o configuration
numOutputs	numeric	

B.1.2.16 Gates:

Name	Direction	Description
in	input	
out	output	

B.1.2.17 Unassigned submodule parameters:

Name	Type	Description

controllerTask.responseTime numeric

B.1.2.18 Module EthIPIO

File: **EthIPIO.ned**

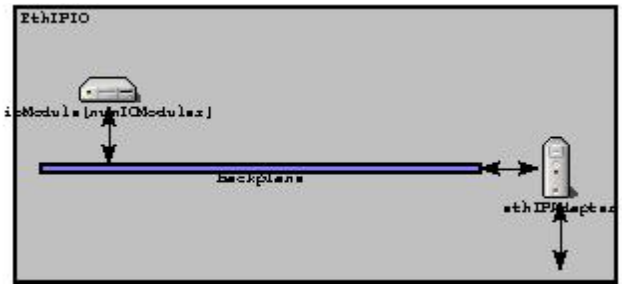
Module Name: EthIPIO

Abstract: Model of an Ethernet/IP IO Composed of one or more IO Modules (defined by parameter numIOModules), a backplane and an EThernet/IP Adapter

Author: Nuno Pereira

Environment: OMNet++ compound module

Revision History: Created Feb 2004.



B.1.2.19 Contains the following modules:

Backplane Module Name: Backplane

EthIPAdapter Module Name: EthIPAdapter

EthIPIOModule Module Name: EthIPIOModule

B.1.2.20 Used in compound modules:

EthIPNetwork1 Module Name: EthIPNetwork1

EthIPNetwork2 Module Name: EthIPNetwork1

B.1.2.21 Parameters:

Name	Type	Description
nodename	string	
numIOModules	numeric	IO modules configuration
connectionIDProducedList	string	
connectionIDConsumedList	string	

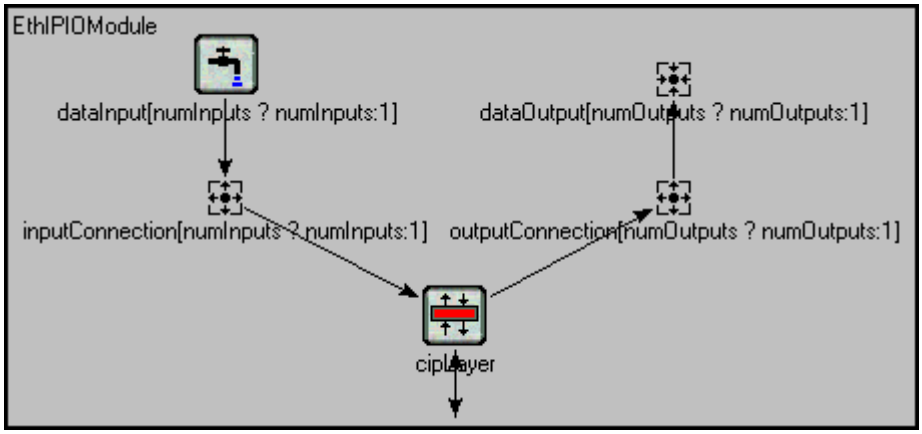
B.1.2.22 Gates:

Name	Direction	Description
in	input	
out	output	

B.1.2.23 Unassigned submodule parameters:

Name	Type	Description
ioModule[*].numInputs	numeric	i/o configuration
ioModule[*].numOutputs	numeric	

B.1.2.24 Module EthIPIOModule
File: **EthIPIOModule.ned**
Module Name: EthIPIOModule
Abstract: Model of an Ethernet/IP IO module
Author: Nuno Pereira
Environment: OMNet++ compound module
Revision History: Created Fev 2003.



B.1.2.25 Contains the following modules:

CIPLayer	Module Name: CIPLayer
IOConnection	Module Name: IOConnection
Input	Module Name: Input
Output	Module Name: IOProcessing.ned

B.1.2.26 Used in compound modules:

EthIPController Module Name: EthIPController

EthIPIO Module Name: EthIPIO

B.1.2.27 Parameters:

Name	Type	Description
nodename	string	
numInputs	numeric	i/o configuration
numOutputs	numeric	

B.1.2.28 Gates:

Name	Direction	Description
In	input	
Out	output	

B.1.3 Channels

B.1.3.1 Channel ethernet

File: **EthIPNetwork.ned**
(no description)

B.1.3.2 Attributes:

Name	Value	Description
Delay	normal(0.00015,0.00005)	
Datarate	100*10^6	

B.1.4 Messages

B.1.4.1 Message CIPDataItem

File: **CIPUDPTransportPacket.msg**

CIP Data Item definition

According to Ethernet/IP Specification: Connected Data Item: Type ID + Length + Data

B.1.4.2 Fields:

Name	Type	Description
headerSize	int	indicates the fixed header size = Type ID + Length (bytes)
sequenceNumber	unsigned long	Sequence number in data item, in order for the outputs to control the data received
dataLength	unsigned int	Data Item - Connected Data Item: Type ID + Length + Data DataTypeID - Connected Data Item
dataItem	string	Class 0 or class 1 packet; (Here, just a string)

B.1.4.3 Message CIPUDPTransportPacket

File: **CIPUDPTransportPacket.msg**

CIP UDP Transport Packet definition

According to Ethernet/IP Specification: Common Packet Format = Item Count + Address Item + Data Item

B.1.4.4 Fields:

Name	Type	Description
headerSize	int	indicates the fixed header size = Item Count + Address Item (bytes)
connectionID	unsigned int	Common Packet Format: Item Count + Address Item + Data Item ItemCount - Number of items to follow AddressItem - Sequenced Address Item: Type ID + Length + Data AddTypeID - Sequenced Address Item AddLength Data - Sequenced Address Item: Connection ID + Sequence Number

B.1.4.5 Message EthernetFrame

File: **EthernetFrame.msg**

Ethernet v.2.0 frame definition
Header=DADDR, SADDR,Len/Type (14 Bytes) + Data (46-1500 Bytes) + FCS (4 Bytes)

B.1.4.6 Fields:

Name	Type	Description
headerSize	int	indicates the fixed header size = Ethernet Header
dAddr	string	Header (14 Bytes)
sAddr	string	(6 Bytes)
Type	int	(2 Bytes) Len/Type: 0x0800 (IP Datagram); 0x0806 (ARP request/reply); 0x0835 (ARP request/reply); Data will be encapsulated in packet; (46-1500 Bytes)
Fcs	int	Frame Check Sequence (CRC)

B.1.4.7 Message NetworkTransportPacket

File: **NetworkTransportPacket.msg**
Network Transport Packet definition

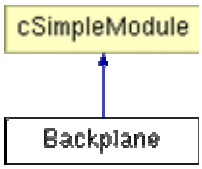
B.1.4.8 Fields:

Name	Type	Description
headerSize	int	indicates the fixed header size = UDP Header + IP Header (bytes)
sourcePort	unsigned int	
destinationPort	unsigned int	
sourceAddress	string	
destinationAddress	string	

B.1.5 Class Documentation

B.1.5.1 Backplane Class Reference

Inheritance diagram for Backplane:



B.1.5.2 Public Member Functions

	Module_Class_Members (Backplane, cSimpleModule, 0) virtual void initialize()
	Do initialization of class data members and any other initialization procedure necessary.
virtual void	handleMessage (cMessage *msg)
	Process messages received.
virtual void	finish ()
	Saves summary results (if any).

B.1.5.3 Private Member Functions

void	tokenize (const char *str, std::vector< double > &array)
	Tokenize string containing numbers separated by ":" into the array.

B.1.5.4 Private Attributes

cPar *	tTableTime
	retrieved module parameter; Transmit table time, defines the "time slot" length for serving connections
cPar *	frameTime
	retrieved module parameter; Frame Time, is the delay introduced to transmit a frame
bool	timeDivison
	flag to define if is doing time divison
int	servingInterfaceIndex
	index of the interface being served
int	servingConnectionIndex
	index of the connection being served
int	numQueuedMessages
	count for the number of queued messages

int	numInterfaces
	number of interfaces connected to the backplane
int	numProducedConnections
	total number of producing connections connected
BackplaneInterface *	bpInterface
	array of interfaces of the backplane (one for each input/output gates pair)

B.1.5.5 Detailed Description

Backplane simple module class definition.

Models an Ethernet/IP device backplane.

The backplane has a BackplaneInterface with several queues to store messages from each connection of the modules connected to the backplane.

It inspects the connection id of each received packet in order to make the the apropriate decisons about where it should queue the received packets .

At a defined transmit table interval, the various connection queues are served, emulating a time divison protocol.

Insted of doing time divison, the backplane can just insert a delay, in function of the number of connections to the backplane. For this, the parameter “timeDivison” must be false. This is usefull for simulation performance reasons.

Warning:

To improve simulation performance, it is possible to disable the backplane time divison. To do this, set the parameter “timeDivison” to false.

To be able to make decisions where to deliver de received packets, it depends on the connected modules having a CIPLayer (or CIPBridgeLayer), with one or more connected IOConnection modules. It analyses the connected modules in order to know the produced and consumed connections in each module.

In the case of a CIPBridgeLayer, the parameters “connectionIDProducedList” and “connectionIDConsumedList” of the of the connected CIPBridgeLayer. These parameters define which connections are bridged throught the CIPBridgeLayer. They are defined by a string, with the several connection IDs separated by “:”.

Attention:

No fragmentation of the received CIP messages is made.

Author:

Nuno Pereira

Date:

Fev 2004.

B.1.5.6 Member Function Documentation

void Backplane::finish( ) `[virtual]`

Saves summary results (if any).

void Backplane::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

The messages shall be processed according to the receiving gate. At reception of a message from upper modules, the message is queue in an appropriate queue, related to the originating connection id. At a defined transmit table interval, the various connection queues are served, emulating a time divison protocol.

If time division is disabled (transmit table interval is zero), the messages received will be imediately processed and sent to the apropriate output gates, after a defined delay.

Parameters:

msg - cMessage* : Message received.

Backplane::Module_Class_Members(Backplane ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void Backplane::tokenize(const char * *str*,
std::vector< double > & *array*
) `[private]`

Tokenize string containing numbers separated by ":" into the array.

Parameters:

str - const char* : String to be parsed

array - std::vector<double>& : Object array to hold the resulting vector of numbers

Returns:

The object array with the resulting vector of numbers

B.1.5.7 Member Data Documentation

BackplaneInterface* Backplane::bpInterface `[private]`

array of interfaces of the backplane (one for each input/output gates pair)

cPar* Backplane::frameTime `[private]`

retrieved module parameter; Frame Time, is the delay introduced to transmit a frame

int Backplane::numInterfaces `[private]`

number of interfaces connected to the backplane

int Backplane::numProducedConnections `[private]`

total number of producing connections connected

int Backplane::numQueuedMessages `[private]`

count for the number of queued messages

int Backplane::servingConnectionIndex `[private]`

index of the connection being served

int Backplane::servingInterfaceIndex `[private]`

index of the interface being served

bool Backplane::timeDivison `[private]`

flag to define if is doing time divison

cPar* Backplane::tTableTime `[private]`

retrieved module parameter; Transmit table time, defines the "time slot" length for serving connections

B.1.5.8 BackplaneInterface Class Reference

B.1.5.9 Public Member Functions

BackplaneInterface ()
~BackplaneInterface ()

B.1.5.10 Public Attributes

InterfaceConnection *	intConnection	
		array of Interface Connections (one per producing connection)
int	numProducedConnections	
		number of connections delivered from this interface
int	numConsumedConnections	
		number of connections to be delivered to this interface
int	numDeliverIndexes	
		count of indexes of the out gates for packetes received in this

	interface
int *	producedConnections
	array if connectionIDs delivered from this interface
int *	consumedConnections
	array if connectionIDs to be delivered to this interface

B.1.5.11 Detailed Description

BackplaneInterface class definition.

A "helper" class with a queue to store messages from each module connected. Stores information about each module connected, in order to allow the backplane to make decisions on where to deliver each packet

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.12 Constructor & Destructor Documentation

BackplaneInterface::BackplaneInterface( ) `[inline]`

BackplaneInterface::~BackplaneInterface( ) `[inline]`

B.1.5.13 Member Data Documentation

int* BackplaneInterface::consumedConnections

array if connectionIDs to be delivered to this interface

InterfaceConnection* BackplaneInterface::intConnection

array of Interface Connections (one per producing connection)

int BackplaneInterface::numConsumedConnections

number of connections to be delivered to this interface

int BackplaneInterface::numDeliverIndexes

count of indexes of the out gates for packetes received in this interface

int BackplaneInterface::numProducedConnections

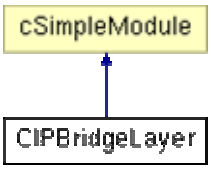
number of connections delivered from this interface

int* BackplaneInterface::producedConnections

array if connectionIDs delivered from this interface

B.1.5.14 CIPBridgeLayer Class Reference

Inheritance diagram for CIPBridgeLayer:



B.1.5.15 Public Member Functions

Module_Class_Members (CIPBridgeLayer, cSimpleModule, 0) virtual void initialize()

Do initialization of class data members and any other initialization procedure necessary.

virtual void handleMessage (cMessage *msg)

Process messages received.

virtual void finish ()

Saves summary results (if any).

B.1.5.16 Private Member Functions

void processMsgFromBP (cMessage *)

Process messages received from the backplane to be delivered to lower UDP.

B.1.5.17 Detailed Description

CIPBridgeLayer simple module class definition.

A CIPLayer that bridges between the backplane and the TCP/IP stack

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.18 Member Function Documentation

void CIPBridgeLayer::finish( ) `[virtual]`

Saves summary results (if any).

void CIPBridgeLayer::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

The messages shall be processed according to the receiving gate.

Parameters:

msg - cMessage* : Message received.

CIPBridgeLayer::Module_Class_Members(CIPBridgeLayer ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void CIPBridgeLayer::processMsgFromBP(cMessage * *msg*) `[private]`

Process messages received from the backplane to be delivered to lower UDP.

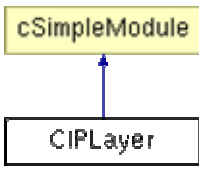
Constructs a new TransportInterfacePacket and encapsulates the received CIPUDPTransportPacket in it

Parameters:

msg - cMessage* : Message of type CIPUDPTransportPacket.

B.1.5.19 CIPLayer Class Reference

Inheritance diagram for CIPLayer:



B.1.5.20 Public Member Functions

Module_Class_Members (CIPLayer, cSimpleModule, 0) virtual void initialize()

Do initialization of class data members and any other initialization procedure necessary.

virtual void handleMessage (cMessage *msg)

Process messages received.

virtual void finish ()

Saves summary results (if any).

B.1.5.21 Private Member Functions

void processMsgFromBP (cMessage *)

Process messages received from the backplane.

B.1.5.22 Private Attributes

CIPConnTable connTable

Connection table, to know where to deliver the received packets.

B.1.5.23 Detailed Description

CIPLayer simple module class definition.

A CIPLayer that encapsulates/decapsulates CIP data items into CIP transport packets. Layers between IOProcessing and the backplane.

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.24 Member Function Documentation

void CIPLayer::finish( ) `[virtual]`

Saves summary results (if any).

void CIPLayer::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

The messages shall be processed according to the receiving gate.

Parameters:

msg - cMessage* : Message received.

CIPLayer::Module_Class_Members(CIPLayer ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void CIPLayer::processMsgFromBP(cMessage * *msg*) `[private]`

Process messages received from the backplane.

Analyses received CIPUDPTransportPacket connection IDs and delivers them to the correct gate

Parameters:

msg - cMessage* : Message of type CIPUDPTransportPacket.

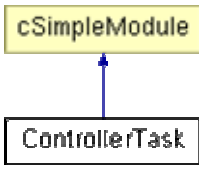
B.1.5.25 Member Data Documentation

CIPConnTable CIPLayer::connTable `[private]`

Connection table, to know where to deliver the received packets.

B.1.5.26 ControllerTask Class Reference

Inheritance diagram for ControllerTask:



B.1.5.27 Public Member Functions

Module_Class_Members (ControllerTask, cSimpleModule, 0) virtual void initialize()

Do initialization of class data members and any other initialization procedure necessary.

virtual void handleMessage (cMessage *msg)

Process messages received.

virtual void finish ()

Saves summary results (if any).

B.1.5.28 Private Attributes

cPar * responseTime

retrieved module parameter; Task response time

B.1.5.29 Detailed Description

ControllerTask.cpp ControllerTask simple module class definition.

Models a controller Task.

Receives a messages in the input gate and delivers them in the corresponding output gate (output gate with the same index), after a defined task response time

Warning:

The number of inputs must be equal to the number of outputs. The correspondean between the input connection and output connection is made by the index of the input and output gates.

Author:

Nuno Pereira
Date:
Feb 2004.

B.1.5.30 Member Function Documentation

void ControllerTask::finish( ) `[virtual]`

Saves summary results (if any).

void ControllerTask::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

Decapsulates the data item from the received message and deliver the message, at the output gate with the same index as the receiving input gate, after a defined task response time.

Parameters:

msg - cMessage* : Message received.

ControllerTask::Module_Class_Members(ControllerTask ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

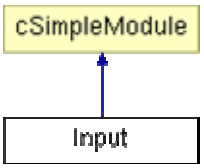
B.1.5.31 Member Data Documentation

cPar* ControllerTask::responseTime `[private]`

retrieved module parameter; Task response time

B.1.5.32 Input Class Reference

Inheritance diagram for Input:



B.1.5.33 Public Member Functions

Module_Class_Members (Input, cSimpleModule, 0) virtual void initialize()
Do initialization of class data members and any other initialization procedure

	necessary.
virtual void	handleMessage (cMessage *msg)
	Process messages received.
virtual void	finish ()
	Saves summary results (if any).

B.1.5.34 Private Member Functions

void	sendInput ()
	Generates and sends a data item.

B.1.5.35 Private Attributes

cPar *	dataLength
	retrieved module parameter; Size of the generated data
cPar *	period
	retrieved module parameter; Period of generation
cPar *	filterDelay
	retrieved module parameter; User defined filter delay
cPar *	hardwareDelay
	retrieved module parameter; Hardware Delay
unsigned long	sequenceNumber
simtime_t	nextGenTime
	Next data generation time.

B.1.5.36 Detailed Description

Input simple module class definition.

B.1.5.37 Member Function Documentation

void Input::finish( ) `[virtual]`

Saves summary results (if any).

void Input::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

This will be called only when receiving a generateNextInput message. Generates a data item message and schedules next generation.

Parameters:

msg - cMessage* : Message received.

Input::Module_Class_Members(Input ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void Input::sendInput( ) `[private]`

Generates and sends a data item.
The data item is sent after the defined filterDelay and hardwareDelay intervals. The timestamp is set with current time, so the receiving output can calculate the end-to-end response time of this message.

B.1.5.38 Member Data Documentation

cPar* Input::dataLength `[private]`

retrieved module parameter; Size of the generated data

cPar* Input::filterDelay `[private]`

retrieved module parameter; User defined filter delay

cPar* Input::hardwareDelay `[private]`

retrieved module parameter; Hardware Delay

simtime_t Input::nextGenTime `[private]`

Next data generation time.

cPar* Input::period `[private]`

retrieved module parameter; Period of generation

unsigned long Input::sequenceNumber `[private]`

B.1.5.39 InterfaceConnection Struct Reference

B.1.5.40 Public Attributes

int	connectionID
cQueue	queue
DeliverIndexList *	deliverIndexList
DeliverIndexList *	deliverIndexList

B.1.5.41 Detailed Description

Interface Connection structure.

holds a queue with information about the respective connection id and information about the gates where to deliver the messages from this connection

B.1.5.42 Member Data Documentation

int InterfaceConnection::connectionID

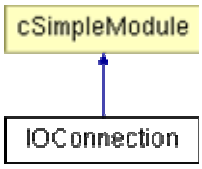
DeliverIndexList* InterfaceConnection::deliverIndexList

DeliverIndexList* InterfaceConnection::deliverIndexList

cQueue InterfaceConnection::queue

B.1.5.43 IOConnection Class Reference

Inheritance diagram for IOConnection:



B.1.5.44 Public Member Functions

	Module_Class_Members (IOConnection, cSimpleModule, 0) virtual void initialize()
	Do initialization of class data members and any other initialization procedure necessary.
virtual void	handleMessage (cMessage *msg)
	Process messages received.
virtual void	finish ()
	Saves summary results (if any).

B.1.5.45 Private Member Functions

void	sendInputData ()
	Constructs a message with the last data item received.

B.1.5.46 Private Attributes

cPar *	rpi
	retrieved module parameter; defined RPI for the connection (only used for input connections)

cPar *	asicDelay
	retrieved module parameter; ASIC delay
bool	inputModule
	internal, calculated variables
CIPDataItem *	dataItem
	defines if io connection processing acts like input or output

B.1.5.47 Detailed Description

IOConnection simple module class definition.

B.1.5.48 Member Function Documentation

void IOConnection::finish( ) `[virtual]`

Saves summary results (if any).

void IOConnection::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

When acting as an input connection, at each RPI sends the last received input data item. When it receives a data item from an input, discards the last and stores it. Acting as an output, it will simply forward messages to the out gate.

Parameters:

msg - cMessage* : Message received.

IOConnection::Module_Class_Members(IOConnection ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void IOConnection::sendInputData( ) `[private]`

Constructs a message with the last data item received.

Sends it after defined asic delay interval.

B.1.5.49 Member Data Documentation

cPar* IOConnection::asicDelay `[private]`

retrieved module parameter; ASIC delay

CIPDataItem* IOConnection::dataItem `[private]`

defines if io connection processing acts like input or output

bool IOConnection::inputModule `[private]`

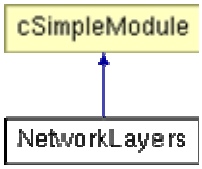
internal, calculated variables

cPar* IOConnection::rpi `[private]`

retrieved module parameter; defined RPI for the connection (only used for input connections)

B.1.5.50 NetworkLayers Class Reference

Inheritance diagram for NetworkLayers:



B.1.5.51 Public Member Functions

Module_Class_Members (NetworkLayers, cSimpleModule, 0)

virtual void initialize ()

Do initialization of class data members and any other initialization procedure necessary.
D:\OMNeT++\models\EthernetIP\doc\sourcedoc\classNetworkLayers.html - a1

virtual void handleMessage (cMessage *msg)

Process messages received.

B.1.5.52 Private Member Functions

void processMessageFromPhy (cMessage *msg)

process messages received physical interface/channel

void processMessageFromApp (cMessage *msg)

process messages received from application layer

B.1.5.53 Private Attributes

cPar * ipProcessing

retrived module parameter; IP Processing Delay

cPar * macProcessing

retrived module parameter; MAC Processing Delay

char	ipAddr [20] IP address.
char	macAddr [20] ethernet MAC address

B.1.5.54 Detailed Description

NetworkLayers.cpp NetworkLayers simple module class definition.

NetworkLayers implement the TCP/IP Layer and Physical Layer functionalities.

Responsible for encapsulating/decapsulating CIP UDP transport packets to be sent to the TCP/IP ethernet network.

Warning:

At reception from the ethernet network, assumes that a "intelligent" switching device is interconnecting the nodes, therefore all messages received are passed up to application layer.

Attention:

All ethernet addresses are constructed based on multicast like MAC address mapping

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.55 Member Function Documentation

void NetworkLayers::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

Verifies arrival gate of the messages, and acts accordinglly

Parameters:

msg - cMessage* : Message received.

void NetworkLayers::initialize( ) `[virtual]`

Do initialization of class data members and any other initialization procedure necessary.

retrive module parameters, by reference

```
NetworkLayers::Module_Class_Members( NetworkLayers ,
                                     cSimpleModule ,
                                     0
                                   )
```

void NetworkLayers::processMessageFromApp(cMessage * *msg*) `[private]`

process messages received from application layer

void NetworkLayers::processMessageFromPhy(cMessage * *msg*) `[private]`

process messages received physical interface/channel

Assumes that a "intelligent" switching device is interconnecting the nodes, therefore all messages received are passed up to application layer.

Parameters:

msg - cMessage* : Message received.

B.1.5.56 Member Data Documentation

char NetworkLayers::ipAddr[20] `[private]`

IP address.

cPar* NetworkLayers::ipProcessing `[private]`

retrived module parameter; IP Processing Delay

char NetworkLayers::macAddr[20] `[private]`

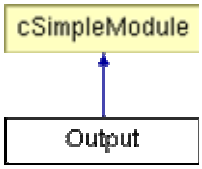
ethernet MAC address

cPar* NetworkLayers::macProcessing `[private]`

retrived module parameter; MAC Processing Delay

B.1.5.57 Output Class Reference

Inheritance diagram for Output:



B.1.5.58 Public Member Functions

Module_Class_Members (Output, cSimpleModule, 0) virtual void initialize()

Do initialization of class data members and any other initialization procedure necessary.

virtual void handleMessage (cMessage *msg)

	Process messages received.
virtual void	finish ()
	Saves summary results (if any).

B.1.5.59 Protected Attributes

unsigned long	lastSequenceNumber
cStdDev	endToEndRTimeStats
	last sequence number received data collected; End to End response time standard deviation
cOutVector	endToEndRTime
	data collected; End to End response vector

B.1.5.60 Detailed Description

Output.cpp Output simple module class definition.

Receives output data. Records statistics about the received data. Namely the end-to-end response time.

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.61 Member Function Documentation

void Output::finish( ) `[virtual]`

Saves summary results (if any).

Statistics of messages received.

void Output::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

At each received message outputs a message and records appropriate statistics.

Parameters:

msg - cMessage* : Message received.

Output::Module_Class_Members(Output ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

B.1.5.62 Member Data Documentation

cOutVector Output::endToEndRTime `[protected]`

data collected; End to End response vector

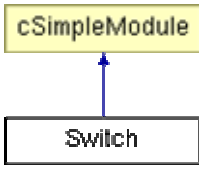
cStdDev Output::endToEndRTimeStats `[protected]`

last sequence number received data collected; End to End response time standard deviation

unsigned long Output::lastSequenceNumber `[protected]`

B.1.5.63 Switch Class Reference

Inheritance diagram for Switch:



B.1.5.64 Public Member Functions

	Module_Class_Members (Switch, cSimpleModule, 0) virtual void initialize() Do initialization of class data members and any other initialization procedure necessary.
virtual void	handleMessage (cMessage *msg) Process messages received.
virtual void	finish () Saves summary results (if any).

B.1.5.65 Private Member Functions

void	tokenize (const char *str, std::vector< double > &array) Tokenize string containing numbers separated by ":" into the array.

B.1.5.66 Private Attributes

cPar *	wcDelay retrieved module parameter; Worst case delay, introduced by the switch

	to each packet D:\OMNeT++\models\EthernetIP\doc\sourcedoc\classSwitch.html - r0
int	numInterfaces number of interfaces connected to the switch
SwitchInterface *	switchInterface array of interfaces of the switch (one for each input/output gates pair)

B.1.5.67 Detailed Description

Switch simple module class definition.

This is a model of a switch for an Ehternet/IP network.

It delivers a copy of the received packets to all corresponding ports, where consumers of the connection are present.

To know where to deliver the packets, it uses parameters “connectionIDProducedList” and “connectionIDConsumedList” of the connected modules.

Warning:

To be able to make decisions where to deliver the received packets, it depends on the connected modules having parameters “connectionIDProducedList” and “connectionIDConsumedList”. These parameters are defined by a string, with the several connection IDs separated by “:”.

> The connection IDs in the parameters are compared with the last byte in the mac address. Therefore, this assumes a direct mapping of the connectionID to the last byte in the ethernet address.

Attention:

Only supports multicast frames

Author:

Nuno Pereira

Date:

Fev 2004.

B.1.5.68 Member Function Documentation

void Switch::finish( ) `[virtual]`

Saves summary results (if any).

void Switch::handleMessage(cMessage * *msg*) `[virtual]`

Process messages received.

The messages shall be processed according to the receiving gate. The messages received will be processed and sent to the apropriate output gates, after a defined delay.

Parameters:

msg - cMessage* : Message received.

Switch::Module_Class_Members(Switch ,
cSimpleModule ,
0
)

Do initialization of class data members and any other initialization procedure necessary.

void Switch::tokenize(const char * *str*,
std::vector< double > & *array*
) [private]

Tokenize string containing numbers separated by ":" into the array.

Parameters:

str - const char* : String to be parsed

array - std::vector<double>& : Object array to hold the resulting vector of numbers

Returns:

The object array with the resulting vector of numbers

B.1.5.69 Member Data Documentation

int Switch::numInterfaces [private]

number of interfaces connected to the switch

SwitchInterface* Switch::switchInterface [private]

array of interfaces of the switch (one for each input/output gates pair)

cPar* Switch::wcDelay [private]

retrieved module parameter; Worst case delay, introduced by the switch to each packet

B.1.5.70 SwitchInterface Class Reference

SwitchInterface class definition. More...

List of all members.

B.1.5.71 Public Member Functions

SwitchInterface ()
~SwitchInterface ()

B.1.5.72 Public Attributes

InterfaceConnection *	intConnection
	array of Deliver Indexes (one per produced connection)
int	numProducedConnections
	number of connections delivered from this interface
int	numConsumedConnections
	number of connections to be delivered to this interface
int	numDeliverIndexes
	count of indexes of the out gates for packetes received in this interface
int *	producedConnections
	array if connectionIDs delivered from this interface
int *	consumedConnections
	array if connectionIDs to be delivered to this interface

B.1.5.73 Detailed Description

SwitchInterface class definition.

A "helper" class. Stores information about each module connected, in order to allow the Switch to make decisions on where to deliver each packet

Author:

Nuno Pereira

Date:

Feb 2004.

B.1.5.74 Constructor & Destructor Documentation

SwitchInterface::SwitchInterface( ) `[inline]`

SwitchInterface::~SwitchInterface( ) `[inline]`

B.1.5.75 Member Data Documentation

int* SwitchInterface::consumedConnections

array if connectionIDs to be delivered to this interface

InterfaceConnection* SwitchInterface::intConnection

array of Deliver Indexes (one per produced connection)

int SwitchInterface::numConsumedConnections

number of connections to be delivered to this interface

int SwitchInterface::numDeliverIndexes

count of indexes of the out gates for packetes received in this interface

int SwitchInterface::numProducedConnections

number of connections delivered from this interface

int* SwitchInterface::producedConnections

array if connectionIDs delivered from this interface