The Performance of the *x*AMp Protocol on Token-Bus and FDDI NACs

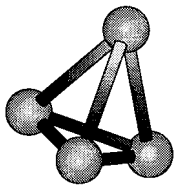
INESC Technical Report RT 101-91

P. Veríssimo, J. Rufino, H. Fonseca, L. Rodrigues

November 1991

ESPRIT II P2252

DELTA4 - PHASE 3



Performance Testing Report IS3

Partner: INESC

Authors: Paulo Veríssimo, José Rufino, Henrique Fonseca, Luís Rodrigues

Workpackage: 9

Date: November 1991 **Reference:** 09-R-11/P

The Performance of the xAMp Protocol on Token-Bus and FDDI NACs

Paulo Veríssimo, José Rufino, Henrique Fonseca, Luís Rodrigues

November 1991

Abstract

There is an increasing number of distributed applications, some of them fault-tolerant, and it has been recognized that its construction may benefit from the existence of reliable broadcast protocols. Some systems are clock-driven, exhibiting tight synchrony: they rely on clock synchronization and space redundancy. Others, like the *xAMp*, an atomic multicast protocol for local area networks, are clock-less.

This work deals with the performance implications of supporting *soft real-time* distributed applications, with clock-less reliable broadcast protocols. In particular it analyses the performance of the lower layers of the Delta-4 communication system, i.e. it studies the time domain behaviour of both Abstract Network and *x*AMp components. Throughout this report we will develop a generic model that allow us to predict protocol execution times in any architecture. Case studies concerning protocol execution on two target LANs — the 10Mbps Token-Bus and the 100 Mbps FDDI — will be presented, showing how protocol performance scales with the utilization of high-speed networks.

Additionally a set of guidelines concerning the dimensioning of *x*AMp timers and performance optimizations, useful for both system configuration implementation, are presented.

Chapter 1

Introduction

There is an increasing number of distributed applications, some of them fault-tolerant, and it has been recognized that its construction may benefit from the existence of reliable broadcast (RB) protocols. These protocols provide a service which has a set of order, agreement and synchronism properties, in the presence of disturbing factors (load and faults). In consequence, the user is alleviated from the task of ensuring them for each application.

Reliable broadcasting has deserved considerable attention recently. Some works, exhibiting tight synchrony, are clock-driven: they rely on clock synchronization and space redundancy [Babaoglu 85,Cristian 85]. The *AMp*, an atomic multicast protocol for local area networks, is a loosely-synchronous protocol which does not use clocks. It relies on network properties, to enforce timeliness. It compares with other clock-less approaches [Birman 87,Cart 87,Chang 84], although we have worked in bounding termination times to suitable limits, which make it possible to define the situations for which the communication system is capable of representing real-time interactions, both from the point of view of meeting communication delay bounds and respecting temporal precedence of events.

This work deals with the performance implications of supporting distributed applications, with reliable broadcast protocols. In application-independent systems, most of the time domain requirements are not of the *hard real-time* kind – which clock-driven protocols are most suited for – but more of the *on-line*, or *soft real-time* kind. That is, rather than requiring an exact meeting of deadlines and a high degree of simultaneity, for actions or messages, applications require responsivity, fastest possible reaction and a probabilistic treatment of worst-case response times. To encourage utilization of reliable broadcast protocols in such applications, it is mandatory that the above-mentioned benefits in quality of service are not considered too costly in performance, by the user(s).

The service properties and the operation of *AMp* are detailed in [Verissimo 89]: the protocol offers a service featuring strong unanimous agreement and consistent and causal order properties. From the point of view of fault-tolerance and distributed computing, they are useful to implement distributed synchronization and replication control algorithms.

The original AMp[Verissimo 89] has recently evolve to an extended multi-fold primitive [Rodrigues 91b], featuring weaker properties (eg. FIFO order, datagrams), with the corresponding improvement in performance. In this work we only address the performance of the

stronger quality of service (atomic), through the development of an architecture independent performance model. Given that the quality of service we are analyzing is the highest possible in terms of agreement and order, we believe the results we present give some insight into the expectable performance of the multi-fold service.

Performance-wise, there are three questions in the design of reliable broadcast protocols, which influence the final result: (i) which fault model; (ii) what level in communications stack; (iii) which network.

The fail-silent assumption

Given the cost associated to distributed agreement approaches [Lamport 82], that consider arbitrary behaviour of components, we assume the communication system fails in a controlled manner, exhibiting what is called the *fail-silent* property [Powell 88]: the network adapters where AMp runs are thus confined to always deliver correct messages, tolerate a bounded number of temporary errors – such as transmission errors – and halt after their first failure. This already represents a performance head-start, because it simplifies protocol design. However, the main improvement of the protocol, performance-wise, is to take advantage of LANs.

The low-level approach

The protocol was designed both to run on top of LANs and not to depend on a particular LAN. The architecture is built on standard LANs, in view of taking advantage of the availability of communications hardware and of the possibility of coexistence with standard stations, in the same network. Additionally, LANs have architecture and technology attributes which can be used for improved performance and dependability.

The data link layer seemed to be the adequate level to engineer the AMp. This low-level approach is justified by several reasons: for one, tradeoffs regarding performance are best made, using LAN facilities; for another, it is low enough to allow several options for upper layers: OSI-like multipoint stacks [Powell 88], transfer layers (eg. XTP) [Chesson 88] or high-performance distributed application support environments [Barrett 90].

So the AMp offers a data link level interface to the user. In order to make the protocol design LAN independent, and thus very portable, it interfaces an "abstract local area network", whose properties we will discuss in the next chapter.

High-speed LANs?

Now that we have a LAN independent interface, we start by analyzing the suitability of the AMp for LANs in general, by making some predictions about its execution time, on a target LAN: a 10Mbps ISO 8802/4 Token-Bus. After, we analyze whether AMp performance will benefit from migrating to a higher throughput LAN, such as the 100 Mbps ISO 9314 FDDI.

We show that not only AMp throughput increases, a natural consequence, but also *speed*, measured in duration of single AMp executions, for small messages. This fact is of outstanding importance, since it has been recognized that communication speed is the dominating requirement for distributed computing [Birrell 84,Chesson 88,Hutchinson 88]. On the other hand, it shows a way of using technology to improve performance without compromising portability. While keeping a neat, independent interface in the LAN world, something can be done to increase performance, by merely changing LAN.

*
* *

In this work we will start to present a brief description of the components that will be subject to our time domain analysis: an overview of the xAMp assumptions and operation is given in chapter 2. This chapter also contains an enumeration of Abstract Network properties, in which protocol design relies, as well a brief description of their support.

The analysis of the time domain behaviour of the protocol begins in chapter 3, where we develop a performance model describing the execution of the xAMp and Abstract Network components. A wide set of non-faulty and faulty scenarios will be analysed and finally integrated into a consolidated performance model. This model will be used in chapter 4 to derive performance figures, while chapter 5 presents the results of an experimental evaluation of protocol performance.

In chapter 6 we will return to model analysis for extracting additional results on performance optimization. Namely, rules for the correct dimensioning of protocol timers will be established and guidelines for enhanced performance implementations will be presented.

Chapter 2

Reliable Multicasting with the AMp

The AMp provides highly parallel reliable group communication, which takes place inside groups of participant entities. The AMp offers a service of reliable group communication which ensures that when a message is delivered, it is delivered to all correct participants (*unanimity*), in the same precedence order to all of them (*order*) and within a known bounded time (*termination*). The message delivered is the one transmitted by the sender and it is always delivered, unless some participant(s) is(are) inaccessible (*validity*). Inaccessibility is a temporary state whereby a recipient may refuse, in a given execution, to accept a message, causing the protocol to terminate timely with a negative confirmation [Verissimo 89]. Since we are essentially concerned with the cases where a message is delivered, inaccessibility is not discussed here. The protocol is resilient to omission errors and stopping faults during its execution. A bounded omission degree is tolerated, but any number of nodes may fail, in a single phase of the protocol[1]. In case of sender failure, a termination protocol is ran by a monitor function, to ensure completion of the transmission in course. Node failures and group membership changes are indicated to all participants, consistently ordered relatively to the information messages. Any group member may initiate a transmission and the sender also receives the message transmitted. Multicasting is *transparent*, in the sense that only one message is sent and there is no need for a previous knowledge by the user, of the whereabouts or number of destinations. In fact, the destination address is location independent and related to a unique designation each group possesses in the system.

Groups

Each **gate**, the entity used by a participant to communicate, uses at each station a local instantiation of the AMp machinery, comprising the *Emitter Machine* and the *Receiver Machine*, a local *GroupMonitor* agent, which participates in error recovery and fault treatment procedures, and two context structures, the *GroupView* and the *ReceiveQueue*, containing, respectively, the group composition and the frames received for that group. Error detection is done on a transfer-by-transfer basis, and relies on consistency of the group views of all

[1] A **phase** is a well delimited portion of a protocol execution, which is a containment domain for error detection. A protocol may have several phases. The organization of AMp in phases is important for the enforcement of timeliness properties.

members. We proceed by describing some assumptions that support protocol operation; then we present the description of the protocol operation itself.

Assumptions

A set of assumptions establish the foundation of the correctness of operation of our two-phase-accept protocol implementation. These assumptions are:

Pa.1 There may be several competitive transmissions in course, in the same group.

Pa.2 There may be several concurrent transmissions in course in the network, from different groups.

Pa.3 At any time, there may be at most one transmission in course, per group, per node.

Pa.4 The sender positively confirms that all correct participants receive a decision, if it is *reject*.

An EmitterMachine, once started, executes atomically, i.e. it is not preempted by other sending actions in the group, for example, from the ActiveMonitor.

The decision frame is not acknowledged, for *accept*, in the interest of improved performance. Safety of this method is based in a simple algorithm:

- assumption Pa.4 above;
- all participants log the sequence number of the last message sent ($lastDeliv$ for sender) or accepted ($lastAccept$ for recipients);
- omission errors in decision can then be recovered very simply: a recipient requests a missing decision, and the sender may respond *accept* if $lastDeliv$ has a higher sequence number; else, it is not yet finished with processing it.

Protocol Operation

The atomic multicasting service relies on a **two-phase accept** protocol. Its operation resembles that of a commit protocol, only in that the *sender* coordinates the operation: it sends a message, implicitly *querying* about the possibility of its acceptance, to which recipients reply (dissemination phase). In the second phase (decision phase), the sender checks whether *responses* are all affirmative, in which case it issues an *accept* – or *reject*, if otherwise. Protocol execution is carried on, in the event of sender failure, by a termination protocol. However, in this case, delivery is no longer ensured. Should the message be delivered, unanimity is nevertheless fulfilled. The phases are implemented with *transmission-with-response rounds*, and end after reception of all expected responses.

An atomic multicast transmission is initiated by the protocol coordinator, the sender (E), by sending a multicast frame containing the message. The *Dissemination* phase (Fig. 2.1) then proceeds as follows:

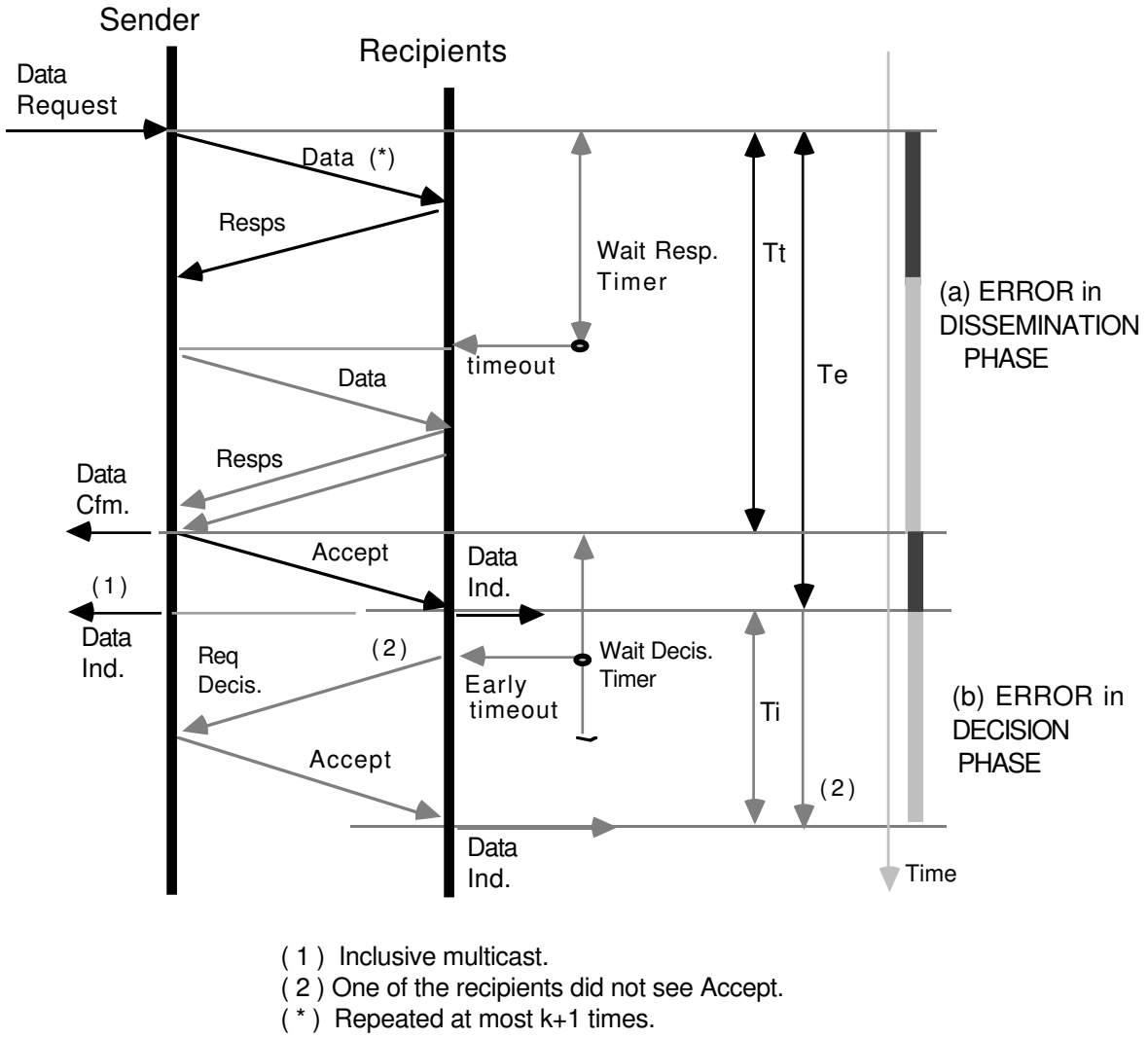


Figure 2.1: Execution diagram of AMp: In grey, in both phases, a transient omission error: a) Dissemination; b) Decision.

— After transmission, E will expect a number of responses indicated by its group view, within a predefined response time (TwaitResponse). When all responses arrive or TwaitResponse has elapsed, they are analyzed and if some recipient cannot accept the frame, decision= *reject*.

— Normally, responses are of "can accept" type, meaning recipients are accessible; then, if all recipients responded, according to the sender GroupView, decision=*accept*. If there are responses missing, the data frame is retransmitted.

— If some station does not answer within the retry mechanism, it is considered failed. However, the execution proceeds, allowing timely termination: an *accept* decision may be sent if all the remaining stations can accept the message. Stations considered as having failed are removed from the group view, by the Group Monitor.

The *decision* phase is implemented in the following way:

— The *reject* frames always require response (assumption Pa.4). A station that does not answer within the retry limit, is considered failed.

— The *accept* frame, on the other hand, does not require response. A timeout mechanism, at the recipients, covers omission errors in the transmission of a decision: after receiving an information frame and responding, a timer is started with a predefined TwaitDecision time. If no decision is received within this time, a recipient requests the decision to the sender[2].

Note that in case of reject, a sender only starts a new transmission after assuring that all group members received the reject (assumption Pa.4). So, when a sender receives such a request decision it can answer with an accept without any knowledge of the past, or proceed, if it was still processing that frame. The recipients will retransmit the request decision, until the retry limit is exceeded. When that happens, the sender is considered failed and the GroupMonitor is called upon, to reestablish group coherence.

The *accept* decision being the most frequent completion of the protocol, was chosen to be negatively acknowledged, which optimizes transmission rate, due to the pipelining effect, in absence of faults. It allows to decrease the transmission cycle time, once a new transfer may start right after issuing the previous one's accept. However, the detection of omission errors in a negatively acknowledged transmission is slower than its positively acknowledged counterpart, since a recipient must wait a worst case transmission time, before issuing a decision request frame. A performance improving consequence of assumptions Pa.3 and Pa.4 is that a recipient may accept a pending frame if it receives a new frame from the same sender. This is expected to avoid the expiration of the waitDecisionTimer, in situations of fair to high traffic, maintaining the pipelining effect.

Group Monitor

There is a *Group Monitor function*, which executes, under a privileged state, critical activities relevant to correct operation of the protocol. Namely, it maintains consistency of the GroupView, recovering from station failures. Additionally, it runs the termination protocol in case of sender failure. It also controls joins and leaves from a Group, so that all GroupViews change consistently.

The distributed Group Monitor function relies on information provided by the several local GroupMonitors of a group. It may be invoked by several groups simultaneously, executing with total independence from the monitors of other groups. The local GroupMonitors are normally inactive. At most one monitor is active at a time, in each group, and if it fails, it is replaced by another GroupMonitor who detects the failure. The procedure is recursive.

The monitor executes in two phases (StepOne and StepTwo). The first phase (StepOne) includes the identification of failed stations and search for the presence of pending messages from failed senders. It is an investigation process, responded by the local monitor entities. After this, a second phase (StepTwo) is performed, including the decision to *accept* or *reject* those messages and finally the dissemination of the new group view.

[2]The reader will notice, in Fig.2.1, that the first decision request is triggered by an *early timeout*. In fact, the waitDecision timer is a two-shot timer. Please refer to section 3.2, for details.

Abstract Network

The design of **AMp** relies and takes advantage of LAN properties. Through the definition of a general set, the network interface although LAN oriented is, in essence, LAN independent. Although this concept can be extended to a broad range of networks, we have limited ourselves to guarantee that they can be fulfilled by a set of standardized token based LANs, namely ISO 8802/4 Token-Bus, ISO 8802/5 Token-Ring and ISO 9314 FDDI. The properties are defined in terms of a network delivering each frame, transmitted at a source, to all destinations. The expected network behaviour — in terms of correctness, order and timeliness — is thus described by the list of properties referred in Table 2.1. As a general rule, Abstract Network properties are guaranteed either by each particular VLSI controller or by a harmonizing driver built on the top of the exposed MAC[3] interface.

Abstract Network Properties
An1 - Broadcast: Destinations receiving an uncorrupted frame transmission, receive the same frame.
An2 - Authentication : Destinations detect any corruption by the network in a locally received frame.
An3 - Network Order : Any two frames indicated in two different destination access points, are indicated in the same order.
An4 - Full Duplex : Indication, at a destination access point, of reception of the frames transmitted by the local source access point, may be provided, on request.
An5 - Simultaneity: Those destinations receiving a broadcast message, receive the message at real time values differing, at most, by δT_b.
An6 - Bounded Transmission Delay : Every frame queued at a source access point, is transmitted by the network within a bounded delay $T_{td} + T_{ina}$.
An7 - Bounded Omission Degree : A network, in a known interval T_{rd}, may do at most k consecutive omission errors.
An8 - Bounded Inaccessibility : A network, in a known interval T_{rd}, may be inaccessible at most i times.

Table 2.1: List of A.N. Properties

The first two properties *Broadcast* and *Authentication* – by a frame check sequence – ensure that although frames may be lost, the destinations that receive a frame, receive the one that was transmitted.

[3]Medium Access Control sub-layer.

Property An3 — *Network Order* — guarantees that any two frames received at any two sites, are received in the same order, while property An4 — *Full Duplex* — implies that the sender itself is also included in this ordering property as a recipient.

Behaviour in the time domain is described by the remaining properties. The property given by **An5** — *Simultaneity* — is satisfied directly by LANs. In single LAN segments, δT_b is essentially given by the end-to-end propagation delay. This property is particular important for the implementation of clock synchronization services [Rodrigues 91a].

Property An6 — *Bounded Transmission Delay* — specifies T_{td} as the maximum transmission delay, in the absence of faults. It depends on the particular network, its sizing, parameterizing and loading conditions [Gorur 88,Jain 90]. Conversely, T_{ina}, the maximum duration of inaccessibility depends on the network alone, and can be predicted for a set of local area networks [Rufino 92,Tusch 88]. Network inaccessibility is a period of time when the network, although remaining operational, refrains from providing service[4]. Essentially it is due to glitches in network operation (e.g. token regeneration, upon its loss). Property An8 ensures that the occurrence of these glitches are bounded.

As a general rule, network inaccessibility have a very disturbing effect on protocol operation. This problem is thoroughly analyzed in [Verissimo 91], that also presents two distinct solutions to cope with it.

Bounded Omission Degree[5] specifies the number of successive transmission errors of a correct network to be lower than a known value k. Enforcement of the bounded omission degree with acceptable coverage is obtained through redundancy in the physical and "medium" layers. In the standard FDDI, a dual-reconfiguring ring, is able of survive to one interruption of the ring[FDDI 86]. In Token-Bus, dual-media could be custom-implemented, as an extension to the standard [Verissimo 88a].

[4]A formal definition of inaccessibility is presented in [Verissimo 90].

[5]Omission degree (Od) is the number of successive omission errors that a correct component does. A component exceeding its specified Od is considered failed and must shut-down.

Chapter 3

Performance Model

This chapter will be entirely dedicated to the time domain analysis of protocol operation. The analysis of AMp performance is based in its phases of *transmission-with-response* ($TxwResp$) rounds. Absolute duration of an execution depends very much on the LAN used, and the particular protocol implementation. Let us begin by discuss some time-related properties of AMp. We define:

⇒ *Execution Time* (T_e): The time between the send request primitive and the issuing of the last receive indication for that message.

The protocol *execution time* can be evaluated as a sum of individual contributions, some of them strictly due to protocol execution, others related with its operation over a LAN. Bounded execution times are only possible if all the contributions in T_e are bounded. Althought the protocol can be ported to a wide set of networks, our performance model is oriented for token based LANs, like the ISO 8802/4 Token-Bus, ISO 8802/5 Token-Ring or ISO 9314 FDDI.

Two critical contributions here appear at the network level interface: *raw access delay* and *queue delay*. Essentially they are network and user related, depending on the global and individual offered load, respectively. In LANs, they can only be effectively bounded if all nodes cooperate in some form to control the offered load. In cyclic hard real-time systems, where the delay of urgent messages must be bounded with accuracy, we believe it is preferable to tune operation in order that queue delay is zero — at least for high priority/urgency services — when running on single LANs. We are looking at systems with mixed cyclic and bursty traffic, so we consider average values, but we will use this design principle, which requires, very simply, that the user cycle is lengthier than the network cycle. The latter is given by the network access delay, which is influenced by user load, indirectly, through a measure of the *average channel utilization* – ρ.

For token based LANs this dependency can be expressed by the average time elapsed between two consecutive token visitations, i.e. by the *average token rotation time* – $R_{av}(\rho)$. For those networks, the token rotation time formulates therefore the influence of LAN operation, on protocol performance.

In this chapter we will establish a set of expressions for the protocol execution time – T_e – corresponding to its operation under different disturbing factors. The effect of network

load will always be considered. Protocol operation, in the presence of faults, will be analyzed through a set of distinct scenarios. Nevertheless, all these definitions will be integrated into a consolidated performance model, at the end of the chapter. Additionally, a set of parameters that are used by the model are defined, in appendix A, for a protocol implementation interfacing the Abstract Network layer.

3.1 General Definitions

Let us assume we have the token-less LAN-independent protocol, thourougly described in [Verissimo 89]. A set of time-related protocol parameters will be extensively used in the definition of our performance model.

Some of these parameters account the duration of the different frames used in protocol operation. Effective frame duration depend on both network data rate and frame length. Their values for several token-based networks are presented in Table 3.2, assuming a MAC address length of 48 bits. For those cases where frame duration depends of operational parameters, a generic expression instead of a numeric value, is given. Table 3.1 characterizes the MAC sub-layer for the considered networks.

	Symbol	Token-Bus			FDDI	
		Length	Duration (μs)		Length	Duration (μs)
		(octects)	$5Mbps$	$10Mbps$	(octects)	$100Mbps$
Bit Time	t_{bit}		0.2	0.1		0.01
Octect Time	t_{oct}		1.6	0.8		0.08
MAC head/tail	t_{HrTr}	28	44.8	22.4	28	2.24
Token	t_{TK}	28	44.8	22.4	11	0.88

Table 3.1: Characterization of MAC sublayer (MAC addressing=48bits)

Let us also define a set of *consolidated processing times* accounting the CPU processing times required to execute specific steps of the protocol, as seen from a network point of view, i.e. as they can be measured by an omnipresent observer located at the LAN-MAC interface of all stations. An illustration of some of these times is provided in Figure 3.1, while the definition of their values is presented in appendix A. Nevertheless, let us now generically define:

- t_{prUsr} - Processing time for an user request.
- t_{prRx} - Processing time of generic protocol frames reception.
- t_{prRxRp} - Processing time for the reception of information frames and generation of the corresponding response.
- t_{prRxRq} - Processing time for the reception of a request decision frame.
- $t_{prRxBg}(n)$ - Time required to receipt and process a bag of n responses.

xAMp Frame	Symbol	Length xAMp Field (octets)	Duration (μs)		
			Token-Bus		FDDI
			$5Mbps$	$10Mbps$	$100Mbps$
protocol response	t_{resp}	28	89.6	44.8	4.48
decision frame	t_{Dec}	28	89.6	44.8	4.48
request decision	t_{rqDec}	28	89.6	44.8	4.48
step one	t_{Step1}	32	96.0	48.0	4.80
step two	t_{Step2}	$32 + 24.N_{fail}$	$t_{oct}.(60 + 24.N_{fail})$		
step one response	t_{sResp}	$36 + 24.N_{fail}$	$t_{oct}.(64 + 24.N_{fail})$		
information frame	t_{Inf}	$28 + l_{Inf}$	$t_{oct}.(56 + l_{Inf})$		
Note: N_{fail} is the number of failed stations detected during StepOne; l_{Inf} is the length of the user data.					

Table 3.2: Characteristics of xAMp frames ($N_{st} = 32$; MAC addressing=48bits)

- $t_{prBg}(n)$ - Time required to receipt and process a bag of n responses, and to prepare a decision.
- $t_{prBgTim}(n)$ - Processing time of a bag of responses, upon timeout.
- $t_{prDecTim}$ - Processing time upon wait decision timeout.
- t_{prTxCf} - Processing time of frame transmission confirmation.
- $t_{prS1Gen}$ - Processing time required for the generation of a step one request.
- t_{prS1Rp} - Processing time for the reception of step one requests and generation of the corresponding response.
- $t_{prBgStep1}(n)$ - Time required to receipt and process a bag of n step one responses, and to prepare a decision.

3.2 Protocol Execution Scenarios

In order to obtain a generic expression for the AMp execution time – T_e – we will analyze protocol operation under different scenarios. We start our analysis with the most simple case of a fault free scenario. Gradually we introduce more severe faulty conditions, until taking into account all the possible errors.

No Faults

The most favorable scenario for AMp execution is the one where there are no frame omissions or station failures. For this case the *average execution time* of AMp can be expressed by equation (3.1), where the two main terms represent the duration of the protocol dissemination and decision phases. After a request has been made, it is processed by the protocol within t_{prUsr}, waiting afterwards a *consolidated access delay* – t_{acc} – for the message transmission

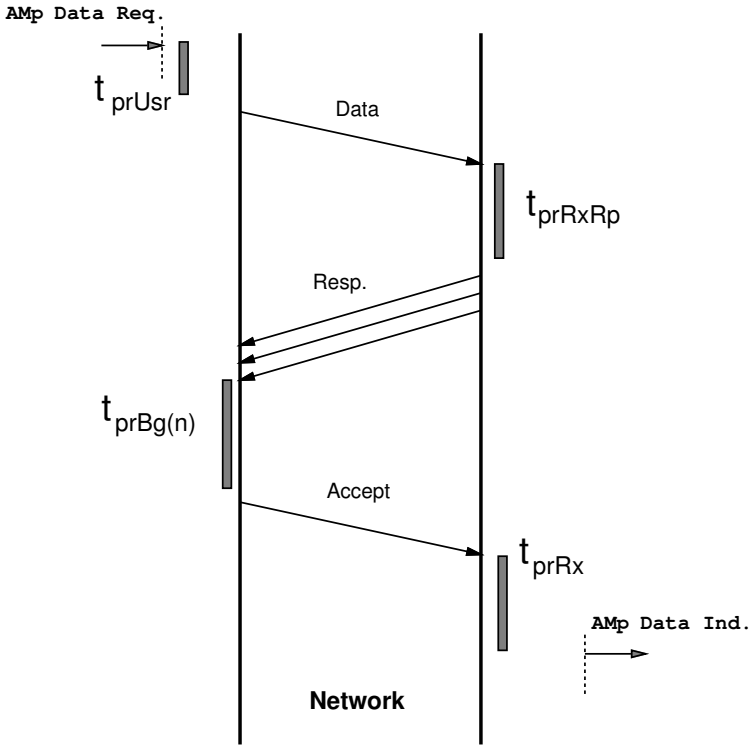


Figure 3.1: Illustration of some consolidated processing times

to the network. This time is both protocol and LAN related and an expression for it will be established in appendix A. An access to the network is also performed, in the beginning of the decision phase, for issuing a decision to *accept* or *reject* that particular message. The time required to perform this access is represented by t_{acDec} and its value will be established later, in this section.

$$T_{e \leftarrow nofault} = [t_{prUsr} + t_{acc} + t_{xwr}(Inf) + t_{prBg}(n)] + [t_{acDec} + t_{Dec} + t_{prRx}] \qquad (3.1)$$

The $t_{xwr}(Inf)$ stands for the time required by the transmission of an information frame – Inf – with response. Since we expect to take advantage of the token rotation, to cycle our transmissions-with-response, we redefine *token rotation time subsequent to a transmission*, $t_{RT}(r, \rho)$, as the time needed for the token to rotate, carrying r responses, over a background load of ρ:

$$t_{RT}(r, \rho) = R_{av}(\rho) + r \,.\, (t_{resp} + t_{IFS}) \qquad (3.2)$$

This expression accounts the time, in addition to the average token rotation, required to transmit the r responses, as well as the time wasted by the VLSI controllers between consecutive transmissions from the same station. For high performance LAN controllers, the *interframe spacing* delay – t_{IFS} – can often be made equal to zero.

The duration of a transmission of an information frame – Inf – with response, is then:

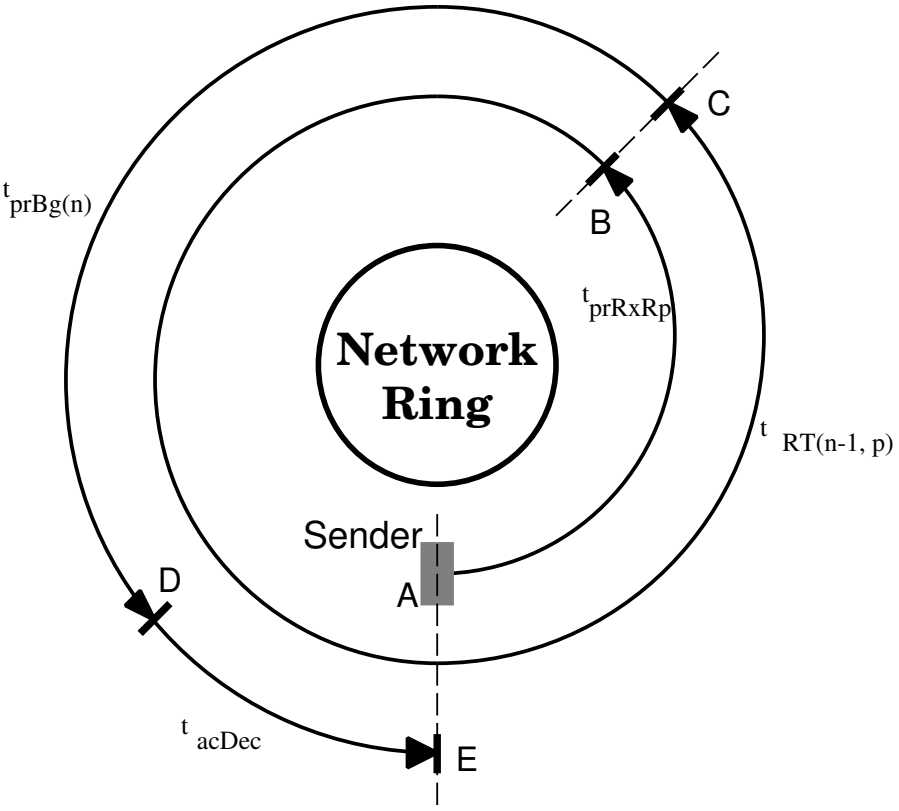


Figure 3.2: Spiral of times for the dissemination-decision phases

$$t_{xwr}(Inf) = t_{Inf} + t_{prRxRp} + t_{RT}(n-1, \rho) \tag{3.3}$$

From all the terms on equation (3.1) only t_{acDec} is still unknown. In order to establish its value we will take a more close look at the network evolution, just after the sender station has transmitted a message. This message will be received at the same time in all the recipients, if we neglect the influence of the network propagation delay. In consequence, message transmission is a notorious event on protocol and network operation. All the additional traffic generated at the recipients sites, during the process of response collection, is a consequence of that message transmission. Since we are looking for a timing relationship between message transmission and the corresponding decision issuing, we will represent protocol evolution as a spiral timing diagram, surrounding the logical or physical network ring (see Figure 3.2).

Sooner or later, after message transmission, the token is passed to the downstream stations. Nevertheless the recipients are unable to respond, until the message is processed and a response is prepared. According with our previous definitions, responses from the recipients will become available for transmission only after a given delay, defined by t_{prRxRp}, has elapsed. The timing of this event is clearly identified in Figure 3.2 (point B). At this point, we can imagine that the token initiates a dedicated rotation for the collection of response frames. Due to this sort of additional traffic the token rotates more slowly than $R_{av}(\rho)$, as expressed by equation (3.2). Once the response collection has been completed (point C), the sender processes the bag of n responses and prepares a decision (point D on Figure 3.2), waiting afterwards – t_{acDec} – for the next token visitation, that allows its transmission to the network.

From the spiral timing diagram of Figure 3.2 we are particularly interested in those sections where the token rotates accordingly to its average value $R_{av}(\rho)$, i.e all but the section given by $t_{RT}(r,\rho)$. The sum of all spiral sections where the token rotates in $R_{av}(\rho)$ can be transformed into an equivalent number of integer token rotations, expressed by j. We can now write the following relations:

$$\begin{cases} (j-1)\,.\,R_{av}(\rho) < t_{prRxRp} + t_{prBg}(n) \leq j\,.\,R_{av}(\rho) \\ \\ \qquad t_{prRxRp} + t_{prBg}(n) + t_{acDec} = j\,.\,R_{av}(\rho) \end{cases} \qquad (j \in N) \tag{3.4}$$

The first relationship establishes a unique value for j, given by equation:

$$j = \left\lceil \frac{t_{prRxRp} + t_{prBg}(n)}{R_{av}(\rho)} \right\rceil \tag{3.5}$$

where $\lceil \ \rceil$ represents the *ceiling* function[1]. The second relationship enables us to express t_{acDec} in terms of the remaining variables:

$$t_{acDec} = R_{av}(\rho)\,.\left\lceil \frac{t_{prRxRp} + t_{prBg}(n)}{R_{av}(\rho)} \right\rceil - t_{prRxRp} - t_{prBg}(n) \tag{3.6}$$

The value of t_{acDec} represents therefore the time that the sender must wait, for the token, before it can transmit the *decision frame* to the network, under a fault free scenario. Mainly it depends on the relative values of CPU consolidated processing times and average token rotation time, assuming values between zero and $R_{av}(\rho)$.

Replacing the above equation in expression (3.1) we obtain, after some simple calculations:

$$\begin{aligned} T_{e \leftarrow nofault} \quad = \quad & t_{prUsr} + t_{acc} + t_{Inf} + t_{RT}(n-1,\rho) + \\ & R_{av}(\rho)\,.\,\left\lceil \tfrac{t_{prRxRp}+t_{prBg}(n)}{R_{av}(\rho)} \right\rceil + t_{Dec} + t_{prRx} \end{aligned} \tag{3.7}$$

The execution time given by (3.7) reveals an important property related with the CPU processing times, that we wish to highlight here: in the absence of faults and from the strict view of a single protocol execution, there are only a few selected points where an improvement on processing times originates a faster protocol termination, as expressed by the aforementioned equation. As a general rule, improvements on the *consolidated processing times* only indirectly affect the protocol execution in the sense that they improve the overall performance. Later we will return to this topic, with a more thorough analysis.

Omissions In Dissemination

In order to mutually control activity, we had seen in the last section that both sender and recipients use timers. In this one and through the remaining scenarios we limit ourselves

[1]The *ceiling* function $\lceil x \rceil$ is defined as the smallest integer not smaller than x.

to proceed at their identification. The discussion of AMp timers dimensioning will only be performed in section 6.1.

At the sender site a timer $TwaitResponse$ defines the maximum waiting period, for response arrival, in each transmission-with-response round. The sender starts timer *TwaitResponse*, after receiving confirmation of transmission by the network and stops them upon the reception of all the expected responses. If no responses arrive, within that time period, a new transmission-with-response round is started, upon $TwaitResponse$ timeout. The lack of responses can be due to omission errors or may arise because network inaccessibility.

An unified characterization of network behaviour, aggregating omission errors and inaccessibility, can be performed and a *consolidated omission degree bound* ($K = k + i$), can be defined [Verissimo 91]. The Abstract Network properties ensure that, in the absence of permanent failures, a successful transmission-with-response will be performed, at the most, after K rounds.

Let be $O_e \leq K$ the number of *omission errors* within the dissemination phase. The protocol execution time is therefore given by equation (3.8) where the first term represents the duration of the O_e rounds affected by omission errors. A new round is started upon $TwaitResponse$ timeout after having processed the available responses and identified the missing ones. The second term represents the duration of the last and successful transmission-with-response round.

$$T_{e \leftarrow DisOm} = O_e \,.\, [\, t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n) \,] + T_{e \leftarrow nofaults} \qquad (3.8)$$

In the definition of expression (3.8) we have considered that the beginning of each transmission-with-response round is not time co-related with the end of the previous one. The utilization of an entity external to the protocol – *timer agency* – eventually affected by different scales in terms of their processing times and variabilities, justifies the rationality of such approach.

Recipients Failure

Upon recipients failure all the transmission-with-response rounds will be consumed in unsuccessful tries to collect their responses. Nevertheless, at the end of the $K + 1$ rounds (being K the *consolidated omission degree bound*), a *decision* is issued, enabling the remaining correct participants to take a decision on that particular message. An additional monitor action is required to withdraw the failed recipients from the *group view* but since this action is postponed until protocol termination, it is not accounted in equation (3.9).

$$\begin{aligned} T_{e \leftarrow rcpfail} \;=\; & t_{prUsr} + \\ & (K+1) \,.\, [\, t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n) \,] + \\ & t_{acc} + t_{Dec} + t_{prRx} \end{aligned} \qquad (3.9)$$

In this expression, no timing relationship is established between the end of the dissemination phase and the issuing of the sender decision. As in the previous case, the timing of

these events depends very much on the *timer agency* behaviour and, in consequence, it seems to be more reasonable to consider them as independent.

Single Omission In Decision

After having analyzed the errors that, in essence, can affect the dissemination phase let us turn our attention to the decision phase. The latter is also coordinated by the sender and starts with the issuing of a *decision*.

The deadline for decision reception is locally controlled, at each recipient, through the utilization of a $TwaitDecision$ timer. Besides, its first purpose, in absence of permanent failures, is to detect and recover from omissions in the (non acknowledged) *decision*; in consequence, it is a two shot timer, with a first timeout given by $t_{earlyDecision}$.

If a decision did not come, within the $t_{earlyDecision}$ period, the recipient sends a request decision – $rqDec$ – datagram. By initiating the recovery process, at the first suspicion that a decision may not come, the *earlier timeout* mechanism introduces a speed-up on protocol termination, when this phase is disturbed by a single network omission.

The recovery process is first initiated by the recipient where the absence of a decision is detected sooner. Nevertheless its precise timing is rather dependent of the station position on the network ring. A best case scenario is obtained when the station dealing with a missing decision immediately follows the point B, as defined in the spiral diagram of Figure 3.2, i.e. that particular station was the first to provide a response to the sender. Conversely, a worst case scenario is obtained when that station has responded in last place, i.e. it is located at point C, of the same figure. The consideration of a single network omission leads therefore to the following expression, for the protocol execution time:

$$\begin{aligned} T_{e\leftarrow DecOm} \;=\; & t_{prUsr} + t_{acc} + t_{Inf} + t_{prRxRp} + F_4 \,.\, t_{RT}(n-1,\,\rho) + t_{prTxCf} + \\ & t_{earlyDecision}(n) + t_{prDecTim} + \\ & t_{acc} + t_{rqDec} + t_{prRxRq} + \\ & t_{acc} + t_{Dec} + t_{prRx} \end{aligned} \tag{3.10}$$

where, depending on the value given to F_4, we obtain:

$$F_4 = \begin{cases} 0 & \text{best case time} \\ \frac{1}{2} & \text{average case time} \\ 1 & \text{worst case time} \end{cases} \tag{3.11}$$

Note that the actions for decision request and the corresponding decision retransmission take place at different stations. Since they represent, indeed, two independent network interactions, no timing relationship can be established for the network access delays concerning these events. In consequence, an average delay, given by t_{acc}, is used in equation (3.10) for both cases.

Multiple Omissions in Decision

The method outlined in the previous scenario provides coverage for the eventual loss of the first decision transmitted by the sender. However, omission errors often occur in bursts and, in consequence, a stronger detection/recovery mechanism is required. Such mechanism consists in a variant of the *transmission-with-response* function, now used for request a decision to the sender. The process is started upon the second signal of the two-shot $TwaitDecision$ timer.

Let be $O_{dec} \leq K$ the number of *omission errors* within the decision phase. The protocol will execute, therefore, at the most, $(O_{dec}+1)$ decision requests[2]. Their execution time, under such conditions is given by equation (3.12).

$$\begin{aligned} T_{e \leftarrow mDecOm} \quad = \quad & t_{prUsr} + t_{acc} + t_{Inf} + t_{prRxRp} + F_4 \,.\, t_{RT}(n-1,\, \rho) + t_{prTxCf} + \\ & t_{waitDecision}(n) + t_{prDecTim} + \\ & O_{dec} \,.\, (\, t_{acc} + t_{rqDec} + t_{prTxCf} + t_{waitDecResponse} + t_{prDecTim}\,) + \\ & t_{acc} + t_{rqDec} + t_{prRxRq} + \\ & t_{acc} + t_{Dec} + t_{prRx} \end{aligned} \tag{3.12}$$

where F_4 is given by equation (3.11) and timer $TwaitDecResponse$ is a variant of $TwaitResponse$ which consider that a single *decision*, instead of multiple *responses*, is currently being awaited for. The definition of the timer value is provided in section 6.1.

Omissions in Dissemination and Decision Phases

In the previous scenarios we have considered separately the existence of omissions errors within the dissemination and decision phases. The case where omission errors affect both phases of the same protocol execution, is introduced in the present scenario.

For the dissemination phase we consider that it is affected by $O_e \leq K$ omission errors. The effect of omissions in the decision phase is not different from the one described in the last two scenarios. The protocol execution time is therefore given by:

$$\begin{aligned} T_{e \leftarrow disDecErrors} \quad = \quad & O_e \,.\, [\, t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n)\,] + \\ & T_{e \leftarrow xDecOm} \end{aligned} \tag{3.13}$$

where T_{xDecOm} represents the time given by one of the expressions (3.10) or (3.12).

Recipients Failure and Omissions in Decision

This scenario is very similar to the previous one. The $t_{earlyDecision}$ may eventually expire sooner then $TwaitResponse$, but its request decision datagram will be ignored by the sender;

[2]Notice that we are not accounting the decision request issued upon the first signal of $TwaitDecision$. Given its *early* timeliness, it does not seem reasonable to perform their inclusion in this fault tolerance technique.

otherwise the timeout obtained from $TwaitDecision$ will always occur after $TwaitResponse$ expiration[3]. Therefore the protocol execution time, in this scenario, is obtained from the previous expression simply making $O_e = K$:

$$T_{e \leftarrow rfailDecOm} = K \,.\, [\, t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n) \,] + T_{e \leftarrow xDecOm} \tag{3.14}$$

Sender Failure

So far we have only considered faulty scenarios which dealt with omission errors and recipients failures. In order to complete our performance model we will now consider the failure of the sender. Taking into account our previous results we assume that before sender failure there may be $O_e \leq K$ omission errors, in the dissemination phase.

The execution of the $(O_e + 1)^{th}$ round offers an opportunity for decision issuing but two kind of situations may happen, at this point:

i) *The sender fails before sending the decision*, and a recovery procedure, based on the aforementioned transmit-with-respond variant, will be entered at the recipients sites, upon $TwaitDecision$ timeout.

ii) *The sender only fails after decision issuing* and the following may happen:

- **a)** The decision is correctly received by all the recipients — In this case the sender failure will be detected only in a future transmission. The protocol execution ends, within a time given either by (3.7), (3.8), (3.9) or (3.13).
- **b)** At least one recipient does not receive the decision — Recipients with a missing decision cannot distinguish this situation from the one described in **i)** and that recovery procedure is entered.

When the sender fails, as described in **i)** and **ii b)**, all the $(K + 1)$ tries of looking for a decision will fail and, in consequence, a monitor action required to reestablish a consistent *group view* and to generate a decision for the pending message, is performed. In this case the protocol execution time is given by:

$$\begin{aligned} T_{e \leftarrow sfail} = {} & t_{prUsr} + O_e \,.\, [\, t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n) \,] + \\ & t_{acc} + t_{Inf} + t_{prRxRp} + F_4 \,.\, t_{RT}(n-1,\, \rho) + t_{prTxCf} + \\ & t_{waitDecision}(n) + t_{prDecTim} + \\ & (K+1) \,.\, [\, t_{acc} + t_{rqDec} + t_{prTxCf} + t_{waitDecResponse} + t_{prDecTim} \,] + \\ & t_{AM \leftarrow x} \end{aligned} \tag{3.15}$$

where $t_{AM \leftarrow x}$ represents the duration of the monitor actions, under different scenarios.

[3]Please, see section 6.1 for details on timer dimensioning.

In the best-case the **Group Monitor** runs successfully at the first time, within a time given by equation (3.16). That equation highlights the fact the Group Monitor activity is also structured in phases of transmit-with-response rounds. The duration of the first phase is given by the first line in equation (3.16) and represents the time taken in the identification of failed stations and search for the presence of pending messages (StepOne). During the second phase (StepTwo), with a duration given by the second line of the same equation, the active monitor replaces the sender in the generation of a decision for those messages. This phase also includes the dissemination of the new *groupview*. Although a transmission-with-response round is carried out in StepTwo, we only account the time required to transmit and process the step two frame. Notice that we have explicitly identify, by $t_{acStep2}$, the time required to access the network for this latter transmission. The group monitor activity is performed considering a group dimension of $(n-1)$, from where the failed sender was withdraw.

$$\begin{aligned} t_{AM \leftarrow noretries} \quad = \quad & t_{prS1Gen} + t_{acc} + t_{xwr}(Step1) + t_{prBgStep1}(n-1) + \\ & t_{acStep2} + t_{Step2} + t_{prRx} \end{aligned} \tag{3.16}$$

The $t_{xwr}(Step1)$ stands for the time required by the transmission of a step one frame with response. Once again, we expect to take advantage of the token rotation, to cycle our transmissions-with-response. In consequence we define *token rotation time subsequent to a step one*, $t_{RS1}(r, \rho)$, as the time needed for the token to rotate, carrying r step one responses, over a background load of ρ:

$$t_{RS1}(r, \rho) = R_{av}(\rho) + r \; . \; (t_{sResp} + t_{IFS}) \tag{3.17}$$

The duration of a step one frame transmission with response is:

$$t_{xwr}(Step1) = t_{Step1} + t_{prS1Rp} + t_{RS1}(n-2, \rho) \tag{3.18}$$

The aforementioned value of $t_{acStep2}$ is co-related with processing times and average token rotation. It is a situation similar to the one considered in our initial non-fault scenario of AMp operation. The spiral diagram for this situation is presented in Figure 3.3 and the expression of $t_{acStep2}$ can be easily obtained:

$$t_{acStep2} = R_{av}(\rho) \; . \left\lceil \frac{t_{prS1Rp} + t_{prBgStep1}(n-1)}{R_{av}(\rho)} \right\rceil - t_{prS1Rp} - t_{prBgStep1}(n-1) \tag{3.19}$$

Using this value in equation (3.16) we obtain, after some simple calculations:

$$\begin{aligned} t_{AM \leftarrow noretries} \quad = \quad & t_{prS1Gen} + t_{acc} + t_{Step1} + t_{RS1}(n-2, \rho) + \\ & R_{av}(\rho) \; . \left\lceil \frac{t_{prS1Rp} + t_{prBgStep1}(n-1)}{R_{av}(\rho)} \right\rceil + t_{Step2} + t_{prRx} \end{aligned} \tag{3.20}$$

The group monitor activity can also be disturbed by omission errors and station failures, that are detected and/or recovered through successive transmission-with-response rounds.

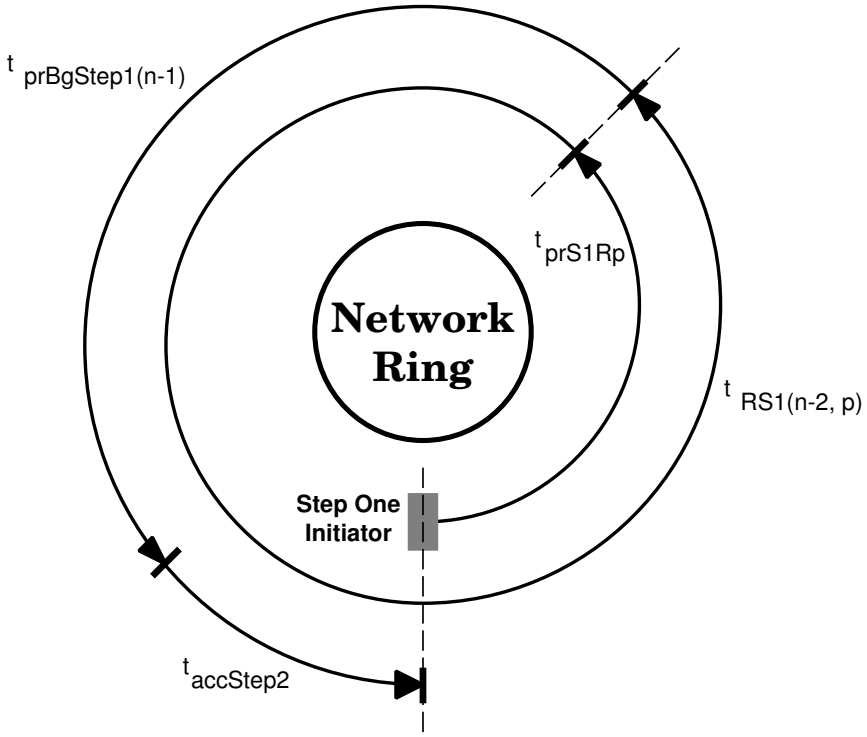


Figure 3.3: Spiral of times for the StepOne-StepTwo monitor actions

Let be $O_{s1} \leq K$ the number of omission errors within StepOne phase. In this case the duration of the monitor actions is given by:

$$t_{AM \leftarrow s1retries} = O_{s1}.[t_{acc} + t_{Step1} + t_{prTxCf} + t_{waitS1Resp}(n-1) + t_{prBgTim}(n-1)] + t_{AM \leftarrow noretries} \tag{3.21}$$

where $t_{waitS1Resp}$ is a variant of $TwaitResponse$, to be used in the StepOne phase.

When the StepTwo phase is also disturbed by $O_{s2} \leq K$ omission errors it does not seem reasonable, or at least worthwhile, to maintain the timing relationship between the end of one phase and the beginning of the other. In this case, the duration of group monitor activity is given by:

$$\begin{aligned} t_{AM \leftarrow retries} = {} & O_{s1}[t_{acc} + t_{Step1} + t_{prTxCf} + t_{waitS1Resp}(n-1) + t_{prBgTim}(n-1)] + \\ & t_{prS1Gen} + t_{acc} + t_{xwr}(Step1) + t_{prBgStep1}(n-1) + \\ & O_{s2}.[t_{acc} + t_{Step2} + t_{prTxCf} + t_{waitResponse}(n-1) + t_{prBgTim}(n-1)] + \\ & t_{acc} + t_{Step2} + t_{prRx} \end{aligned} \tag{3.22}$$

Finally we analyze the situation where, in addition to omission errors, stations fail during the StepOne phase. In this case, no responses will be obtained from these stations, within the $(K+1)$ retry limit. A new investigation must then be started to seek for pending transmissions from these stations. The duration of the active monitor action is then given by:

$$
\begin{aligned}
t_{AM \leftarrow mfails} \;=\; & R_{s1}.[t_{acc} + t_{Step1} + t_{prTxCf} + t_{waitS1Resp}(n-1) + t_{prBgTim}(n-1)] + \\
& t_{prS1Gen} + t_{acc} + t_{xwr}(Step1) + t_{prBgStep1}(n-1) + \\
& O_{s2}.[t_{acc} + t_{Step2} + t_{prTxCf} + t_{waitResponse}(n-1) + t_{prBgTim}(n-1)] + \\
& t_{acc} + t_{Step2} + t_{prRx}
\end{aligned}
\tag{3.23}
$$

where $R_{s1} = O_{s1} + B_7(K+1)$ is the number of StepOne retries, upon failure, and B_7 is a boolean variable which indicates whether or not the *groupview* has changed during the StepOne phase.

3.3 Consolidated Performance Model

The diversity of expressions used in the time domain description of AMp can be integrated into a single expression. As a matter of fact, this integration can be performed not only for the expressions describing protocol execution but also for those given the duration of the monitor actions.

These results are presented in Table 3.3, where each one of the studied scenarios can be selected through a set of boolean variables.

	Consolidated Execution Times
T_e	$O_e.(\ t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n)\ )+$ $\overline{B}_2[\ \overline{B}_1.(T_{e \leftarrow nofaults})+$ $B_1.(\ t_{prUsr} + t_{acc} + t_{Inf} + t_{prTxCf} + t_{waitResponse}(n) + t_{prBgTim}(n)\ )\ ]+$ $B_2[\ t_{prUsr} + t_{acc} + t_{Inf} + t_{prRxRp} + F_4\ .\ t_{RT}(n-1,\rho) + t_{prTxCf} +$ $\overline{B}_3.t_{earlyDecision}(n) + B_3.t_{waitDecision}(n) + t_{prDecTim} +$ $B_3.O_{dec}.(t_{acc} + t_{rqDec} + t_{prTxCf} + t_{waitDecResponse} + t_{prDecTim}\ ) +$ $\overline{B}_5.(t_{acc} + t_{rqDec} + t_{prRxRq}) +$ $B_5.(t_{acc} + t_{rqDec} + t_{prTxCf} + t_{waitDecResponse} + t_{prDecTim} + t_{AM})\ ]+$ $(\overline{B}_2.B_1 + B_2.\overline{B}_5).(t_{acc} + t_{Dec} + t_{prRx})$
	Active Monitor action
t_{AM}	$(O_{s1} + B_7(K+1)).[t_{acc} + t_{Step1} + t_{waitS1Resp}(n-1) + t_{prBgTim}(n-1)] +$ $\overline{B}_6.t_{AM \leftarrow noretries} +$ $B_6.[t_{prS1Gen} + t_{acc} + t_{Step1} + t_{prStep1} + t_{RS1}(n-2,\rho) + t_{prBgStep1}(n-1) +$ $O_{s2}.(t_{acc} + t_{Step2} + t_{waitResponse}(n-1) + t_{prBgTim}(n-1)) +$ $t_{acc} + t_{Step2} + t_{prRx}]$
	Omission Errors
O_e	Number of omission errors in dissemination.
O_{dec}	Number of omissions in decision request.
O_{s1}	Number of omissions in Step1.
O_{s2}	Number of omissions in Step2.
	Variables for error situations
B_1	Recipient(s) failure: true - $B_1 = 1 \Rightarrow O_e = K$; false - $B_1 = 0$.
B_2	Decision errors: true - B_2=1; false - B_2=0.
B_3	Omissions in first dec. req.: true - B_3=1; false - B_3=0.
F_4	Request decision scenario: Worst case - $F_4 = 1$; Average case - $F_4 = \frac{1}{2}$; Best case - $F_4 = 0$
B_5	Sender failure: true - $B_5 = 1 \Rightarrow B_2 = 1$ and $B_3 = 1$ and $O_{dec} = K$; false - B_5=0.
B_6	Errors in Step2: true - B_6=1; false - B_6=0.
B_7	Group has changed: true - B_7=1; false - B_7=0.

Table 3.3: Consolidated temporal expressions for AMp execution time, T_e, with several error scenarios.

Chapter 4

Analytic Results

In this section we will made a prediction of AMp performance in a set of fault-free and error scenarios, for both Token-Bus and FDDI networks. This evaluation uses for model parameters the values, presented in the next section, that were obtained by experimental evaluation, made on a target test-bed network. Notice that the processing times only depend of each execution platform, being therefore LAN independent.

For our predictions we consider a general scenario, in an industrial environment, for example a small cell network for real-time manufacturing control. The network cable, with a length $C_l = 500m$, presents a typical propagation delay of $5\mu s/Km$. The total number of stations $N_{st} = 32$.

Token-Bus Environment:

Let us assume that our manufacture cell network is an ISO 8802/4 Token-Bus network:

- channel data rate is $C_r = 10Mbps$.
- token duration: $t_{TK} = 22.4\mu s$.
- station delay $t_{SD} = 21\mu s$.
- propagation delay: $t_{PD} = 2.5\mu s$.
- minimum token rotation time in token-bus:

$$\begin{aligned} R_{mn} &= N_{st}.(t_{PD} + t_{SD} + t_{TK}) \\ &= 1469\mu s \end{aligned} \tag{4.1}$$

- consolidated average access delay, given by equations (A.18) and (A.19), presented in appendix A. The relation of $R_{av}(\rho)$ with network parameters and load is a result well known in literature [Janetzky 86,Jayasumana 89] and is given by:

$$R_{av}(\rho) = \frac{R_{mn}}{1-\rho} \tag{4.2}$$

For a background offered load of $\rho_{BK} = 10\%$, we have:

- average token rotation time: $R_{av}(0.1) = 1632\mu s$.
- consolidated average access delay ($\rho_{BK} = 10\%$): $t_{acc} = 5316\mu s$.

– protocol frame characterization is given by the data presented in Table 3.2.

Scenario	T_e (ms)								
	80 octets			640 octets			1150 octets		
	n=2	n=4	n=6	n=2	n=4	n=6	n=2	n=4	n=6
no faults	12.2	17.2	20.6	12.7	17.7	21.0	13.1	18.1	21.4
1 om. f. diss.	47.3	52.2	55.6	48.2	53.1	56.5	49.0	54.0	57.3
1 om. f. dec.	41.2	41.3	41.3	41.6	41.7	41.8	42.1	42.1	42.2
recip. failure	109.0	109.0	109.0	110.4	110.4	110.4	111.6	111.6	111.6
sender failure	141.0	144.4	149.5	141.4	144.9	149.9	141.8	145.3	150.4

Table 4.1: AMp on Token-Bus: execution time for different message lengths and group participants, in the presence of several errors($K = 2$)

We calculate the AMp timer values, for this environment, considering the CPU and LAN load variabilities of $V_P = 30\%$ and $\Delta\rho = 30\%$. This seems to be a reasonable assumption: given $\rho_{BK} = 10\%$, we are accommodating average LAN loads between 10%–40%; for the CPU, we consider a variation of 30% on the execution of software components.

The values associated with the AMp timers are given by the expressions defined in the section 6.1, leading to the following results, for a group of $n = 4$ participants:

$$\begin{aligned}
t_{waitResponse}(4) &= 12585\mu s \\
t_{earlyDecision}(4) &= 12475\mu s \\
t_{waitDecision}(4) &= 15200\mu s \\
t_{waitDecResponse} &= 11105\mu s \\
t_{waitS1Resp}(3) &= 12705\mu s
\end{aligned}$$

Finally, we extract some example situations from the generic expression of the consolidated performance model of table 3.3, and present a set of relevant values, in Table 4.1, as a function of message length and number of group participants.

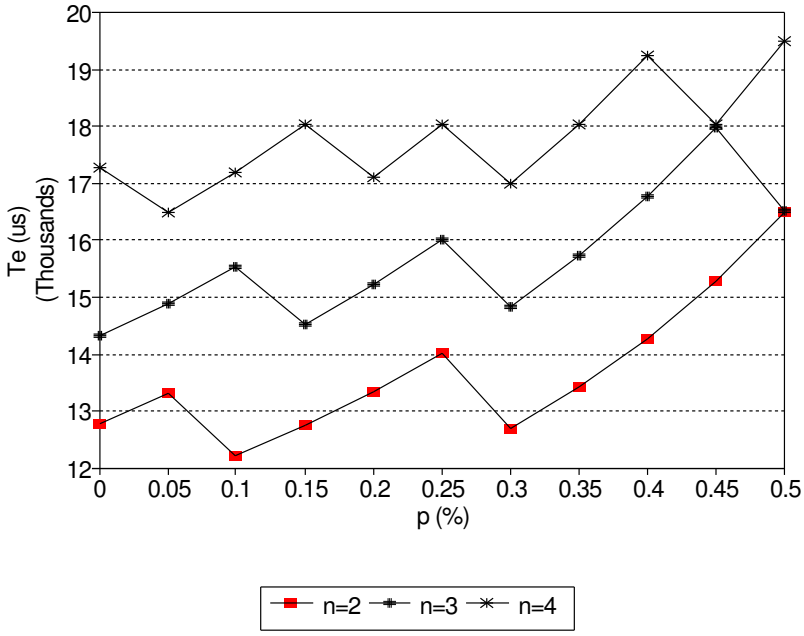


Figure 4.1: Execution time on a 10 Mbps Token-Bus versus load, for several group dimensions

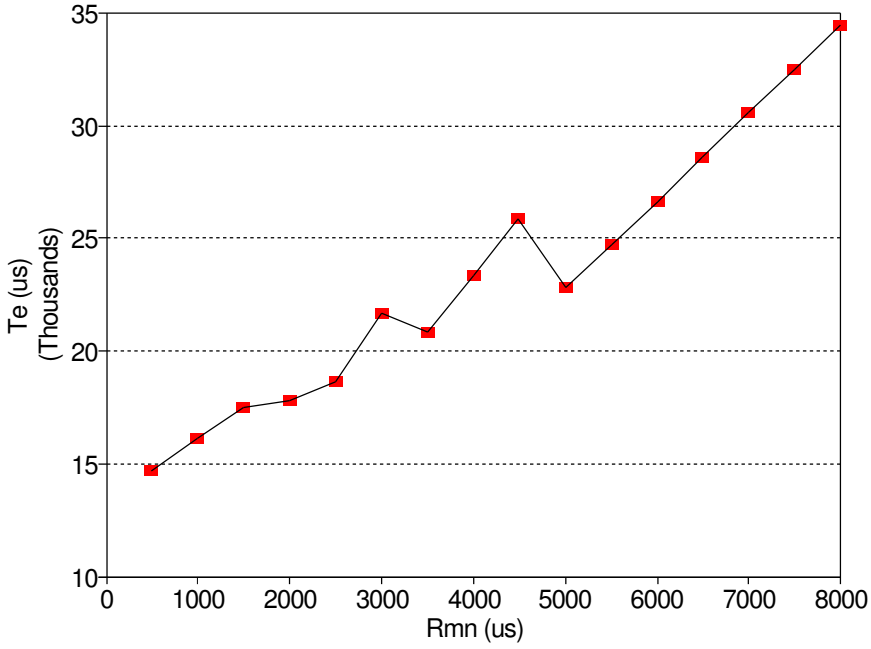


Figure 4.2: Execution time on a 10 Mbps Token-Bus versus LAN size, for $n = 4$

The variation of execution time with the average channel utilization ρ is presented in the graphic of Figure 4.1, for several group dimensions. We can observe that in a global basis execution times grow with an increasing background load, although in a local basis they may exhibit an opposite variation, as described by the "spiral of times". The same effect can be observed when we vary network size, measured by R_{mn}, a function of number of stations and cable length (Figure 4.2).

The variation of protocol execution times with other parameters like gropu dimension, message length and variation of processing times will be performed in conjunction with FDDI, at the end of this chapter.

FDDI Environment:

So far we have studied protocol execution in a particular Token-Bus environment. Nevertheless we are not observing the performance of a particular LAN, but that of a protocol using LANs. Because of that, we now predict the execution times of AMp on FDDI and compare them with those obtained for Token-Bus, afterwards. When performing this comparison we are concerned with the viewpoint — proper of real-time — of the service provided to the individual user, despite some background load. So, we scale down the relative background load, ρ_{BK}, by a ratio of 10:1, to maintain the same absolute load (in octets). Messages maintain their lengths, although decreasing in duration.

The configuration of the network is identical to that used for Token-Bus. Channel length $C_l = 500m$ and number of stations $N_{st} = 32$ remain constant. The channel rate becomes $C_r = 100Mbps$ and the background load becomes, as explained, $\rho_{BK} = 1\%$. Therefore, we redefine the following network operation variables:

- token duration: $t_{TK} = 0.88\mu s$.
- station delay, $t_{SD} = 0.6\mu s$.
- minimum token rotation time for FDDI:

$$\begin{aligned} R_{mn} &= t_{PD} + N_{st}.t_{SD} + t_{TK} \\ &= 22.6\mu s \end{aligned} \tag{4.3}$$

- For a background offered load of $\rho_{BK} = 1\%$, we have:
  - average token rotation time $R_{av}(0.01) = 22.8\mu s$.
  - consolidated average access time ($\rho_{BK} = 1\%$): $t_{acc} = 4511\mu s$
- message and protocol frame durations are those furnished in Table 3.2, for FDDI.

Processing times depend only of the NAC; we assume that they remain unchanged. Variabilities remain at $v_P = 30\%$ and $\Delta\rho_{BK} = 30\%$.

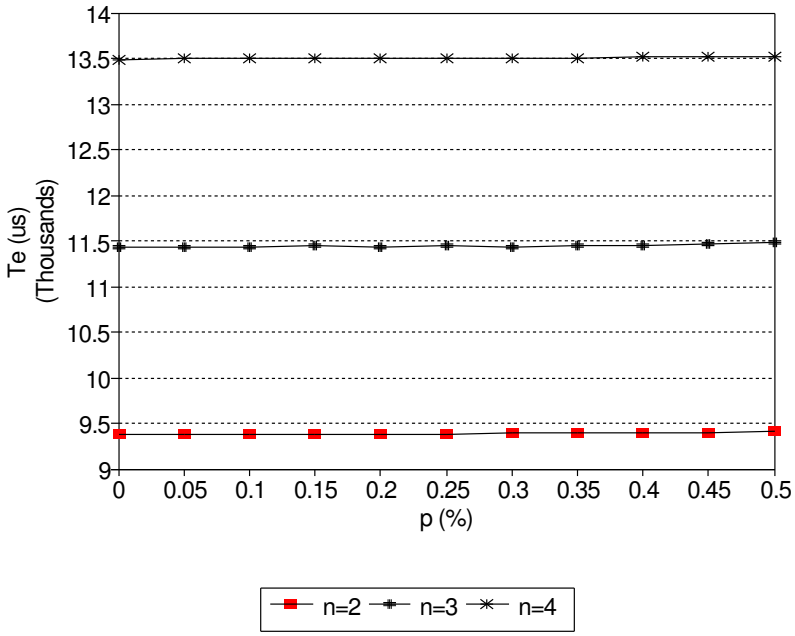


Figure 4.3: Execution time on FDDI versus load, for several group dimensions

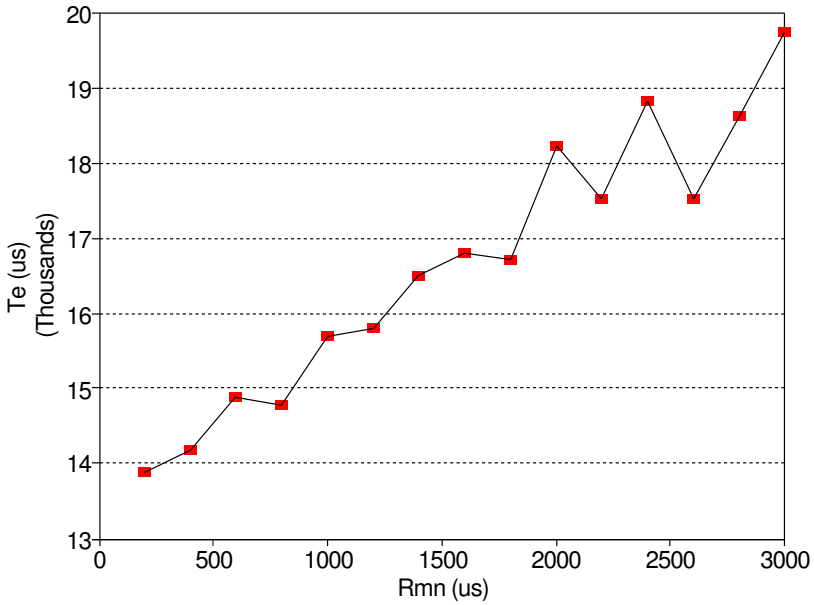


Figure 4.4: Execution time on FDDI versus LAN size, for $n = 4$

Scenario	T_e (ms)								
	80 octets			640 octets			1150 octets		
	n=2	n=4	n=6	n=2	n=4	n=6	n=2	n=4	n=6
no faults	9.4	13.5	17.6	9.4	13.5	17.7	9.5	13.6	17.7
1 om. f. diss.	43.5	47.6	51.8	43.6	47.7	51.8	43.7	47.8	51.9
1 om. f. dec.	37.8	37.8	37.8	37.8	37.8	37.8	37.8	37.8	37.9
recip. failure	105.5	105.5	105.5	105.6	105.6	105.6	105.7	105.7	105.7
sender failure	132.6	137.4	142.3	132.6	137.5	142.4	132.7	137.5	142.4

Table 4.2: AMp on FDDI: execution time for different message lengths and group participants, in the presence of several errors ($K = 2$)

Timers are based in the same expressions, but they do change:

$$
\begin{aligned}
t_{waitResponse} &= 10855\mu s \\
t_{earlyDecision} &= 9135\mu s \\
t_{waitDecision} &= 11860\mu s \\
t_{waitDecResponse} &= 9455\mu s \\
t_{waitS1Resp}(3) &= 10965\mu s
\end{aligned}
$$

We repeat, in Table 4.2, the predictions for T_e in the same situations as done for Token-Bus, in Table 4.1. We also present the corresponding graphical representations of execution times average channel utilization and network size. We can observe that execution times are not heavily affected by network load and that the "timing spiral" effect only arises for large FDDI networks, assuming unchanged processing times.

Comparison of FDDI with Token-Bus

The results presented for the last execution environment show that the utilization of a high speed network, like $100Mbps$ FDDI, seems to be advantageous for complex protocols:

- given the increased available bandwidth, which reduces the impact of protocol frames in channel utilization;
- given the increased speed, because of the shorter rotation and transmission times, for the same load condition.

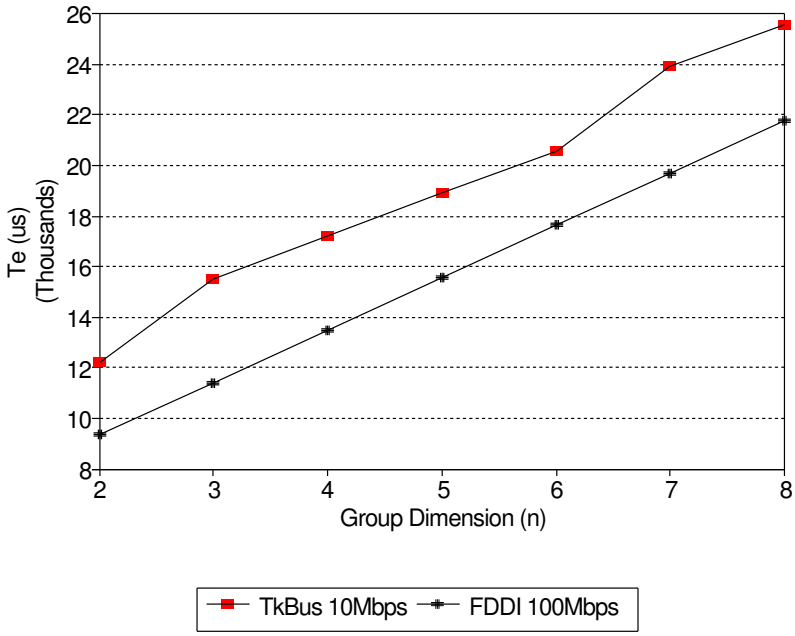


Figure 4.5: Execution Time versus Group Dimension n

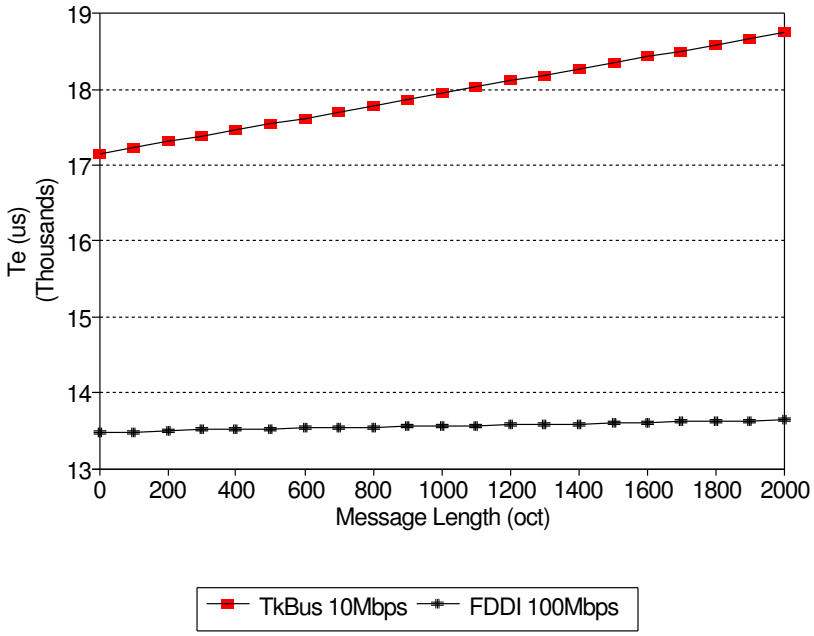


Figure 4.6: Execution time versus Message Length

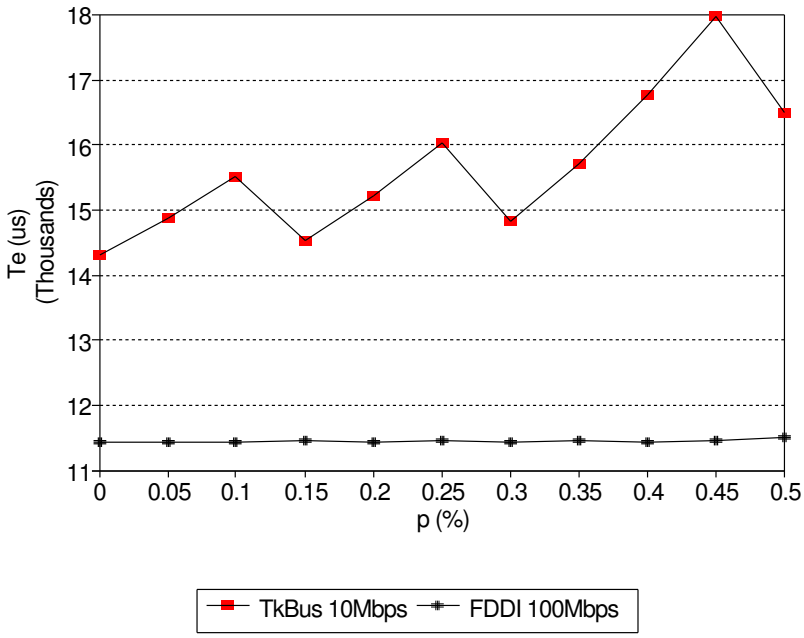


Figure 4.7: Execution time versus LAN load, for $n = 3$

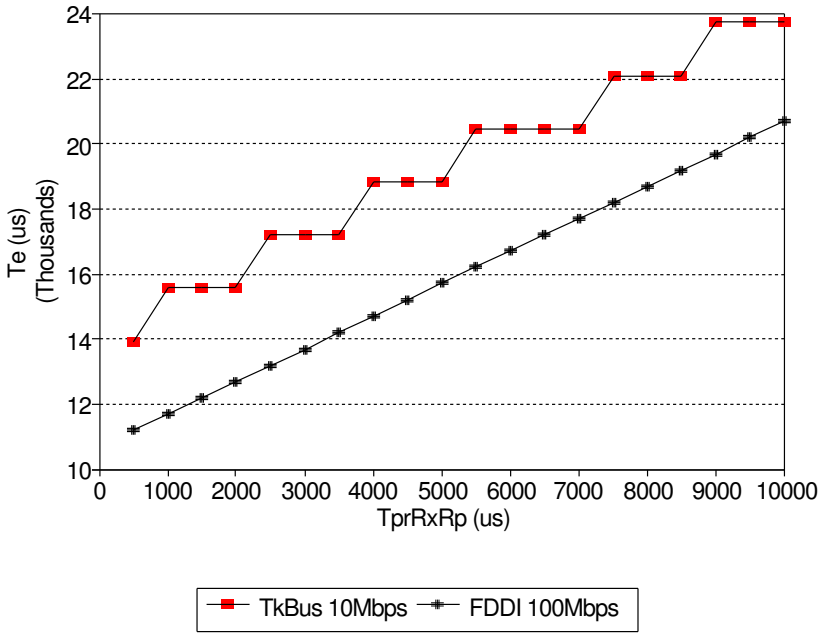


Figure 4.8: Effect of processing times on protocol execution ($n = 4$)

In this section we present a set of graphics for the two studied networks,i.e. ISO 8802/4 Token Bus and ISO 9314 FDDI, that relates the protocol execution time with the following parameters:

- group dimension, n;
- message length, l_{Inf};
- LAN load, ρ;

The final graphic aims to analyze the effect of *consolidated processing times* on protocol execution time. As expected we can see that improvements in the processing times do not always lead to a corresponding reduction in the protocol execution time. Once again this is an effect of the "spiral of times". This fact can be clearly observed in Token-Bus. The FDDI network, due to its fast token rotation, seems to be less susceptible to this effect. Variation of network channel utilization produces a similar effect, as shown in Figure 4.7.

Chapter 5

Experimental Results

In this chapter we report the results of our experimental evaluation to model parameters and AMp execution times.

Our testbed was an experimental ISO 8802/4 Token-Bus Network working at 5 Mbps. Each site attaches to the network through a *Network Attachement Controller* (NAC), where the protocol is executed. The protocol is exercised through a set of test tools that, among other functions, allow network monitoring and parameterization and also traffic generation, with different load patterns and frame sizes. These tools also run on the NACs.

The Token-Bus *Network Attachement Controller* [Verissimo 88b] is a dedicated microprocessor infra-structure built around the Motorola MC68020 @ 16.67MHz [MOT 84]. It incorporates all the functionality needed for the operation of MAC VLSI MC68024 Token Bus Controller [TBC 87]. Our experimental testbed network is made from four of these nodes.

The NACs were instrumented for performance evaluation. A specific device, interfacing an external time measurement instrument[1], has been added to the NAC. This instrumentation allows the definition, by software commancds, of the moments for trigger/stop time counting. Hardware triggered measurements were carried out, where required through a *Tektronix 308 Data Analyzer.*

Evaluation of Model Parameters

We have experimentally evaluated the parameters that define the *consolidated processing times* used in our performance model for the mentioned execution platform. These parameters do not depend on the network, but rather on the processing capabilities of the NAC. The results obtained are summarized in Table 5.1 for the group dimension independent parameters. Parameters which depend on group dimension are listed in Table 5.2, for a small set of group participants.

[1] A PHILIPS-PM6612 Time Counter was used.

AMp Domain		A.N. Domain				Consolidated Parameters	
		Driver		VLSI			
Symbol	Value (μs)	Symbol	Value (μs)	Symbol	Value (μs)	Symbol	Value (μs)
t_{ampUsr}	840	t_{drvTx}	694	t_{macTx}	0	t_{prUsr}	840
t_{ampRx}	457	t_{drvRx}	1052	t_{macRx}	0	t_{prRx}	1509
$t_{ampRxRp}$	1055					t_{prRxRp}	2801
$t_{ampRxRq}$	679					t_{prRxRq}	1731
$t_{ampDecTim}$	687					$t_{prDecTim}$	910
t_{ampCf}	0	t_{drvCf}	1052	t_{macCf}	0	t_{prTxCf}	1052
$t_{ampS1Rp}$	1140					t_{prS1Rp}	2886
$t_{ampS1Gen}$	1152					$t_{prS1Gen}$	1152
$t_{ampS2Gen}$	1004						
$t_{ampDecGen}$	1352						

Table 5.1: Evaluated Performance Parameters

Group Dimension n	AMp Parameters		Consolidated Parameters			
	$t_{ampBg}(n)$ (μs)	$t_{ampBgTim}(n)$ (μs)	$t_{prBg}(n)$ (μs)	$t_{prRxBg}(n)$ (μs)	$t_{prBgTim}(n)$ (μs)	$t_{BgStep1-1}(n)$ (μs)
2	357	511	3455	1409	734	–
3	1361	511	5511	3465	734	4255
4	2365	511	7567	5521	734	6460
5	3369	511	9623	7577	734	8665
6	4373	511	11679	9633	734	10870

Table 5.2: Group Dimension Dependent Performance Parameters

Protocol Execution Times

In order to measure the quality of our performance model we will also evaluate the protocol execution times, for a given set of non-faulty and faulty scenarios, making afterwards the comparison between theoretical and experimental results. Protocol execution times are dependent of each particular LAN characteristics. So, we recalculate LAN parameters that do not remain constant, for the particular case of our testbed environment. Assuming we have our cell manufacture network, with the same length $C_l = 500m$, but with data rate $C_r = 5Mbps$ and $N_{st} = 4$, we have:

- station delay: $t_{SD} = 11\mu s$.
- token duration: $t_{TK} = 44.8\mu s$.
- minimum token rotation time in token-bus: $R_{mn} = 233\mu s$.

Scenario	T_e (ms) – n=2					
	80 octets		640 octets		1150 octets	
	teor.	exp.	teor.	exp.	teor.	exp.
no faults	10.33	11.50	11.23	12.80	12.04	14.00
1 om. f. diss.	1540	1540	1541	1550	1543	1550
1 om. f. dec.	5039	5000	5040	5050	5041	5060
recip. failure	3028	3060	3030	3060	3032	3080
sender failure	8973		8974		8975	

Table 5.3: AMp on a 5Mbps Token-Bus: theoretical and experimental execution times, in the presence of several errors, with n=2 participants and $K = 1$.

Scenario	T_e (ms) – n=4					
	80 octets		640 octets		1150 octets	
	teor.	exp.	teor.	exp.	teor.	exp.
no faults	14.41	14.30	15.31	15.60	16.12	16.80
1 om. f. diss.	1548	1550	1550	1560	1552	1560
1 om. f. dec.	5039	5000	5040	5000	5041	5000
recip. failure	3028	3006	3030	3080	3032	3080
sender failure	8983		8984		8985	

Table 5.4: AMp on a 5Mbps Token-Bus: theoretical and experimental execution times, in the presence of several errors, with n=4 participants and $K = 1$.

– For a background offered load of $\rho_{BK} = 10\%$, we have:

  - average token rotation time: $R_{av}(0.1) = 259\mu s$.
  - consolidated average access delay ($\rho_{BK} = 10\%$): $t_{acc} = 4629\mu s$.

– message and protocol frame durations also change, as indicated in Table 3.2.

For the particular protocol implementation used in this evaluation, protocol timers were not tightly tuned with LAN operation. First, timers do not wait by network confirmations. They are started immediately after the request issuing. We model this effect by letting $t_{prTxCf} = 0$. Secondly, they are assigned as multiples of a given *time-base* constant, defining the following values:

$$
\begin{aligned}
t_{waitResponse} &= 1.5s \\
t_{earlyDecision} &= 5.0s \\
t_{waitDecision} &= 5.0s \\
t_{waitDecResponse} &= 1.5s \\
t_{waitS1Resp} &= 1.5s
\end{aligned}
$$

Some relevant values, considering theoretical and experimental sets, are presented in Tables 5.3 and 5.4, for two different group dimensions. Experimentally, the error scenarios were obtained through fault injection, activated by software components.

Chapter 6

Performance Model Revisited

Through all the chapter 3 we have developed a performance model that allows us to predict xAMp performance for a wide set of token based networks. Provided that model parameters are obtained from experimental evaluation, it will be able to represent close enough xAMp behaviour, in terms of protocol execution times.

In this chapter we continue to explore the results that can be extracted from the performance model. We begin by analysing how the performance model can be used to define a realistic dimensioning of protocol timers, which can largely improve protocol performance in the presence of faults.

A second representation of protocol execution — expressed in a form of a *Gantt Diagram* — will also be analysed since it is extremely useful to obtain guidelines for optimize execution times.

6.1 Dimensioning of AMp Timers

We have already mentioned, more than once, that in order to mutually control activity all the communicating participants use timers. In chapter 3 we have merely identified those timers, without defining the values that should be used in their programming. Now we will establish a set of expressions that allows the definition of their values.

In high performance real-time systems, where timeliness requirements are very stringent, we could obtain effective performance advantages if timer programming is tightly tuned with each particular execution environment. For such systems, timers should be computed by the communications management entity and are dependent both on the *consolidated processing times* and on the network operating conditions.

At the sender site we have seen that timer $TwaitResponse$ defines the maximum waiting period, for response arrival, in each transmission-with-response round. Its purpose is to recover from network omissions and/or to detect recipients failures. The sender starts timer $TwaitResponse$, after receiving confirmation of transmission by the network, with a value which can be be perceived from the timing spiral of Figure 3.2 and is given by equation (6.1). This expression consideres the worst-case values of processing times, as denoted by the

superscript ^{wc}. It begins accounting the time required by the recipient to receive an incoming message and produce the corresponding response. The number of responses expected to come from the network is $r = n - 1$ (n: number of group members), whereas $\Delta\rho$ account for LAN overload, in addition to the expected channel utilization ρ_{BK}. The last term of equation (6.1) accounts the worst-case time required for deliver to the protocol the received responses. Since the timer value depends on group dimension, each group has a timer.

$$t_{waitResponse}(n) = t^{wc}_{prRxRp} + t_{RT}(n-1,\ \rho_{BK} + \Delta\rho) + t^{wc}_{prRxBg}(n) \tag{6.1}$$

After having sent their response to the network, each recipient also starts a time – $TwaitDecision$ – that establishes a deadline for decision reception. This timer also depends on group dimension, n, so, it also belongs to each group. Besides, its first purpose, in absence of permanent failures, is to detect and recover from omissions in the (non acknowledged) *decision*; in consequence, it is a two shot timer, with a first timeout given by $t_{earlyDecision}$:

$$t_{earlyDecision}(n) = t_{RT}(n-2,\ \rho_{BK}) + t_{prBg}(n) + R_{av}(\rho_{BK}) + t_{Dec} + t_{prRx} \tag{6.2}$$

The rationale for dimensioning of this timer is that, after confirmation, from the network, of transmission of the response, the recipient starts the timer ($t_{earlyDecision}$), based on the assumption that it is the first station to reply in the *response collection process*. After given enough time for the collection of all responses, under the average ρ_{BK} channel utilization, each recipient allows an optimistic processing time[1] for the bag of n responses, followed by transmission of the decision, after waiting for the access to the network. All these times are represented in the spiral diagram of Figure 3.2. Strictly speaking, we should have used the value of t_{acDec} for the network access delay, but we have preferred, instead, to use its upper bound $R_{av}(\rho_{BK})$, for two reasons:

- It provides a realistic safety margin. This is particularly important when we are dealing with very small t_{acDec} values. In this scenario the sender may marginally miss the foreseen opportunity, to issue a decision, and one more token rotation is required.
- As aforementioned, the values of the timers are computed in real time by the communications management entity, being the latter more easily obtained than the former.

The last term on equation (6.2) accounts an also optimistic time for the delivery, to the protocol, of frames arriving from the network.

As aforementioned the early timeout mechanism is intended to detect and recover from an eventual loss of the first decision issued by the sender. Under more severe faulty conditions a more powerful mechanism is required, being started upon the second signal of $TwaitDecision$ timer. The dimensioning of their value does not share the optimism placed in the definition of its first signal. Indeed, more restrictive operating conditions are considered, by accounting the following factors:

[1] Optimistic, because the average values are used, i.e. no variabilities are taken into account.

- The possible existence of a LAN load variability – $\Delta\rho$ – over the background traffic – ρ_{BK}.
- The eventual occurrence of some *latency* in the protocol processing machinery, dictating the consideration of worst-case processing times.

The value of the second and final $t_{waitDecision}$ timeout is therefore given by the following equation:

$$t_{waitDecision}(n) = t_{RT}(n-2,\ \rho_{BK}+\Delta\rho) + t^{wc}_{prBg}(n) + R_{av}(\rho_{BK}+\Delta\rho) + t_{Dec} + t^{wc}_{prRx} \qquad (6.3)$$

Another timer to be defined is $TwaitDecResponse$. This timer is also started at the recipient sites, within each try of the request decision transmission-with-response. This timer is a variant of $TwaitResponse$ timer where a single *decision*, instead of multiple responses, is awaited for. Due to this reason, in the definition of $t_{waitDecResponse}$ we have neglected the influence of the *interframe spacing delay* expressed in equation (3.2). Therefore the value of this timer, that does not depend on group dimension, is given by:

$$t_{waitDecResponse} = t^{wc}_{prRxRq} + R_{av}(\rho_{BK}+\Delta\rho) + t_{Dec} + t^{wc}_{prRxBg}(1) \qquad (6.4)$$

Our last expression of timer definition represents another variant of the $TwaitResponse$ timer, used by the active group monitor during the StepOne phase. Since this processing presents a larger duration, relatively to the normal response generation, a different timer value is also defined:

$$t_{waitS1Resp}(n) = t^{wc}_{prS1Rp} + t_{RS1}(n-1,\ \rho_{BK}+\Delta\rho) + t^{wc}_{prRxBg}(n) \qquad (6.5)$$

Using equations 6.1 through 6.5, network operators are able of accurately parameterize protocol timers, with the aim of obtaining the best achievable performance.

6.2 Gantt Diagrams

In order to provide generic guidelines for the correct implementation of the protocol and Abstract Network layers we will analyse protocol execution represented in a form known as *Gantt Diagram*. This diagram provides to the designer helpful information concerning how the execution time is split among each individual action and, in consequence, it is able to identify those actions which are more time consuming. Clearly, if those actions can be performed faster, significant performance enhancements can be obtained. For simplicity of exposition, we will split our analysis in two parts:

- *Abstract Network Gantt* – which includes the harmonization driver and the its interaction with the user and the LAN VLSI.
- *Protocol Gantt* – concerning the execution of the "normal" transmit/receive xAMp data path.

Abstract Network Gantt

The actions performed in the transmit/receive data path within the Abstract Network are represent in the Gantt diagram of Figure 6.1.

Let us start by analysing the actions taking place at the **Emitter Site**, upon a xAMp request. Clearly, we can identify the following main sections:

- *Service of User Requests* – In our analysis we consider an Abstract Network boundary defined in such a way that this component also includes the transaction of the request and its processing before invocation of the remaining "frame" treatment procedures. Therefore we account, inside the Abstract Network domain, the time required to perform and service any request made by the xAMp protocol.

- *Format Conversion* – Protocol layers should use a LAN independent message representation, in order to be easily portable. Therefore, before inserting each frame in the VLSI queue, a format translation must take place. Since this operation represents a functional overhead in the data path processing it should be kept simple, to be quickly executed.

- *Frame Queueing* – In this action frames to be transmitted are delivered to the LAN controller. The associated operations are dependent of each particular VLSI interface. The development of an efficient, performance wise, LAN VLSI interface is a critical issue. The way how such an interface should be actually built are not always clearly documented in the LAN controller data sheets and an additional investigation, for each case, is often required.

- *Full Duplex Copy* – This action is concerned with the support of Abstract Network property An4 and is only performed in systems where this feature is not directly supported by the LAN controller. To speed up protocol execution, this action is only performed after the frame has been queued to the VLSI for transmission. If this was not the case, i.e. if copying was performed before queueing, a performance penalty will always occur.

The confirmation that a queued frame was actually transmitted to the medium is signaled, by the VLSI, through a processor interrupt request. Performance wise, this method presents real advantages when compared with a polling scheme since it saves processing power. The actions required to process transmitted frames are also identified in the Gantt diagram of Figure 6.1:

- *Interrupt Service* – Interrupt service includes the fetching and clearing of the VLSI interrupt status. If that is the case, it will also include auxiliary frame stamping functions, required as a support to the full duplex Abstract Network property. Since this section also interacts with the VLSI, it is also subject to our previous remarks about efficiency of LAN controller interfacing.

- *Frame Unqueueing* – Transmitted frames must be withdrawed from the transmit queue, being the corresponding data structures either released or delivered to the user.

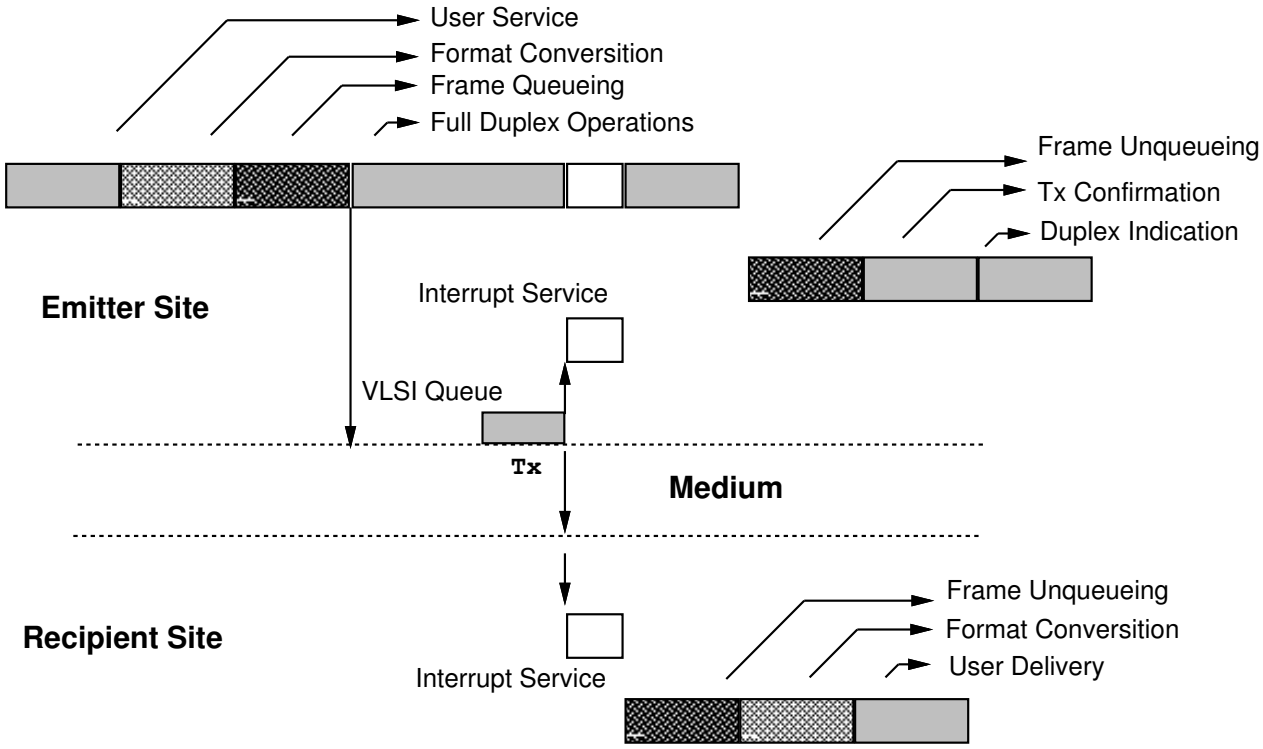


Figure 6.1: Gantt Diagram for the Abstract Network

- *User Delivery* – When requested confirmation of transmissions are delivered to the user, after being processed by the Abstract Network. If that is the case, a "full duplex" copy of the transmitted frame will also be delivered to the user, at this time.

In a given sense the kind of actions performed at the **Recipient Site** are very similar with those concerning confirmation of transmissions. They are also triggered through an interrupt request asserted by the LAN VLSI and include afterwards the following steps:

- *Receive Queue Service* – Which will be responsible by the retrieval of the received frames from the corresponding queue.
- *Format Conversion* – Performing the translation of the VLSI frame format into the LAN independent format used by the upper layer protocols.
- *User Delivery* – Which includes the transaction of the message to the destination end-point.

Given a particular Abstract Network implementation, this time domain analysis is useful in optimizing performance. As a general rule, implementations should follow the guidelines outlined in this section, trying to reduce the processing overheads and supporting an efficient interface with the VLSI. For example, in the implementation of the Token-Bus Abstract Network we have used, the following schemes:

- Merging of LAN independent and VLSI private frame structures, conducting to simpler frame format translations and consequently to an enhanced performance.

- Efficient LAN VLSI interface, using specific resources[2] to prevent the overlapping of requests, with small or even null processing overheads.
- Careful sequencing of actions, avoiding that execution of critical activities will be delayed by the non-critical ones.

Protocol Gantt

The actions performed when concerning the execution of a "normal" transmit/receive xAMp data path are represent in the Gantt diagram of Figure 6.2.

The analysis of the actions takes place when a user issues an *Atomic* request, at the **Emitter Site**. The following main sections can be clearly identified:

- *Scheduler accepting and Initial frame formatting* – Upon the user request of an *Atomic* transmission, some initial frame header formatting is performed. Afterwards, some tests are executed to determine if the frame is allowed to be transmitted. Traffic can be suspended or casual order can prevent the sending of the frame. If this is the case the frame is queued in the Scheduler and awaits the end of these situations (this is not shown in the Gantt diagram). If not the processing continues so that the frame can be transmitted.
- *Final frame formatting* – These operations can only be performed after the knowledge that the frame will be transmitted immediately and not queued in the Scheduler.
- *Frame filling* – The result of formatting is now introduced in the frame to be sent to the Abstract Network for delivery to the medium. This filling should also include the data that the user requests to be sent. Actual implementations of the protocol use the structure sent by the user (already with the data in it) and add the headers determined before. This way no data copying is performed.
- *Send Machine setup* – Now the send machine has to be initialized to this particular type of frame. Here some variables are setup and finally the frame is delivered to the Abstract Network.

Now at the *Recipient Site* the frame is delivered to the xAMp by the Abstract Network.

- *Frame header extraction* – The receiver receives the frame and immediately before any type checking, the frame header is extracted. This header is equal to all types of frames so it can be immediately tested.
- *Type checking and Function branching* – After the header has been extracted, the frame type can be checked and depending on it and on the xAMp status, it can be discarded or accepted. If it is accepted the appropriate function related to this type of frame is then called.

[2]Given fields, on the VLSI shared memory area.

- *Credit and other tests* – Now the protocol is already in the *Atomic* frame specific part and certain values like credit and traffic suspended for example have to tested to see if the frame can be queued, or if it is immediately rejected.
- *Response alloc and filling* – As the frame just received can not be used to resend the response (because it is going to be queued), a new frame has to be allocated. It is then filled with the appropriate values for the response, and sent to the Abstract Network.
- *Frame queueing* – Here is a typical example of optimization in the protocol. The frame is only queued after the response is sent to the Abstract Network. This way some parallel processing can be performed.

As each station sends its response to the frame emitter station, these responses are collected one by one until all have been received and a decision can be sent.

- *Response header extraction* – Like before in the reception of the data frame, the header is extracted.
- *Type checking and Function branching* – Also the procedure is identical to the case above.
- *Send Machine response Test and update* – Now the emitter station knows that the sender of this response has issued a valid acknoledgement to the data frame, and so certain send machine values are updated to reflect this fact. As the emitter station knows how many responses to wait for, it can determine when all responses have arrived.

If all responses have not come yet, this part of the protocol stops here waiting for the rest of the responses from other stations. When all responses have come, then the protocol continues.

- *Frame header extraction*, *Confirmation to user* – As the user has to be notified that the transmission was successful, the frame that was initially sent to the Abstract Network is now sent back to the user. For that the header part has to be extracted before it is delivered up.
- *Decision alloc and filling* – A decision has been reached and it has to be sent to all stations active in this transmission. So a decision frame is allocated, filed with the correct parameters and sent to the Abstract Network.

All *Recipient Sites* now receive the decision frame delivered by the Abstract Network.

- *Decision header extraction*, *Type checking and Function branching* – Actions identical to the previous cases.
- *Queued frame search and accept* – The frame related to the decision just received is searched through the receive queue, and accepted.
- *Indication to the user* – If the frame accepted is at the top of the queue then it can be withdrawn from this queue and an indication is sent to the upper services.

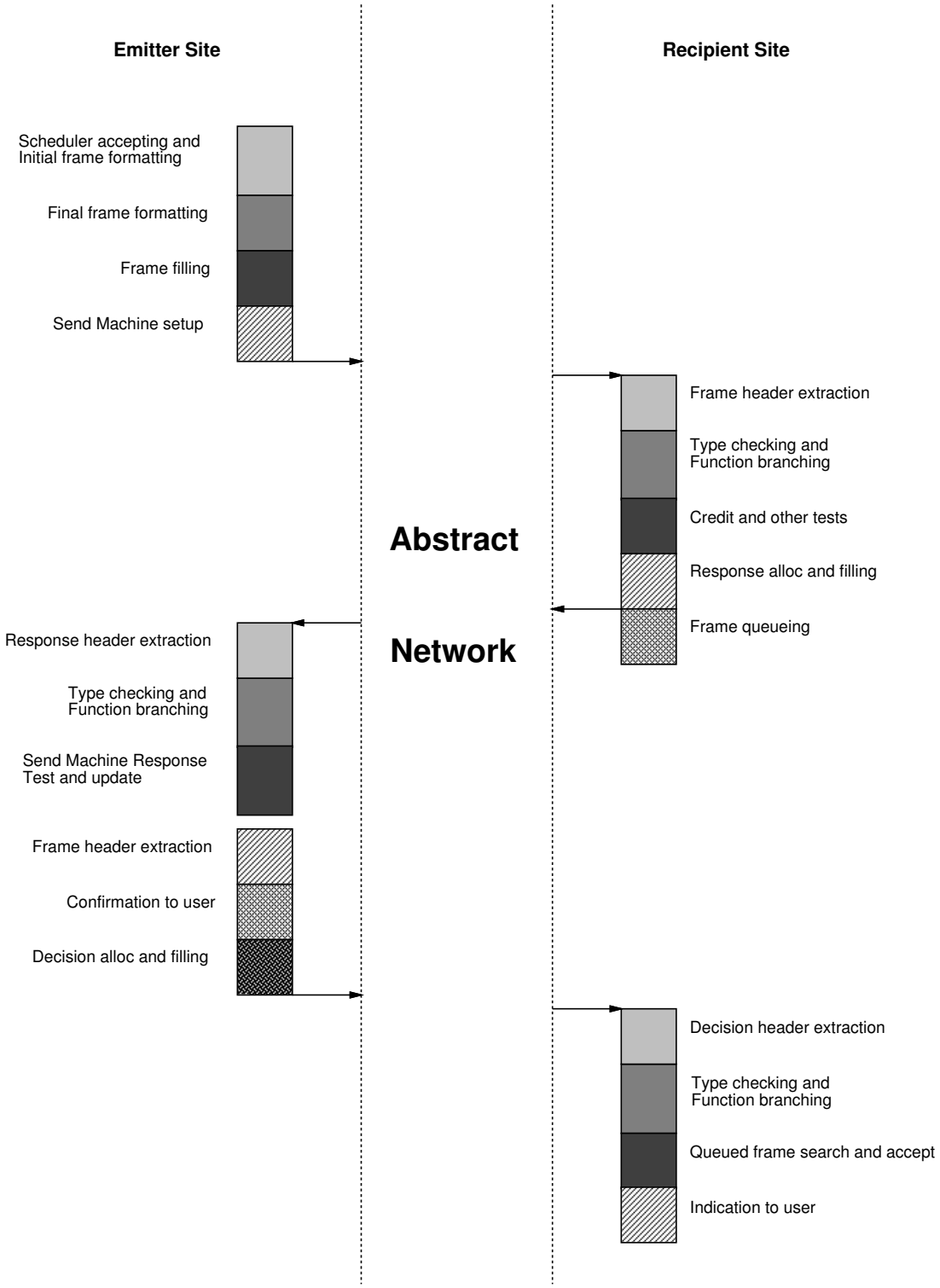


Figure 6.2: Protocol Gantt diagram

Chapter 7

Conclusions

This work has analyzed the performance of xAMp, an *atomic multicast protocol* for local area networks. Our study has focused both theoretical and experimental aspects. A performance model has been developed and an experimental evaluation, performed on an ISO 8802/4 Token-Bus network, has been carried out.

The aim of the experimental evaluation was essentially the validation of the model and the measurement of processing times parameters that characterize the execution of the protocol and therefore the performance model. Nevertheless we have measured the actual execution times of xAMp, in a set of distinct scenarios. The model seems to be quite accurate. Some of the differences found can be explained by the variation of the processing times with some variables like, for instance frame length, that were not considered in the model.

The execution times of xAMp on FDDI were predicted using the performance parameters of our experimental evaluation. As expected the high speed of FDDI benefits the execution of complex protocols, like xAMp. Performance predictions have also shown some interesting dependencies of execution times with network load and processing times. These effects were explained through the "spiral of times", a concept that allows the establishment of a timing relationship between different phases of the protocol, in the absence of faults.

Bibliography

[Babaoglu 85] Ozalp Babaoğlu and Rogério Drummond. Streets of byzantium: network architectures for fast reliable broadcasts. *IEEE Transactions on Software Engineering*, SE-11(6), June 1985.

[Barrett 90] P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, D. Seaton, N. Speirs, and P. Veríssimo. The Delta-4 Extra performance architecture (XPA). In *Digest of Papers, The 20th International Symposium on Fault-Tolerant Computing*, IEEE, Newcastle-UK, June 1990. also as INESC AR/21-90.

[Birman 87] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM, Transactions on Computer Systems*, 5(1), February 1987.

[Birrell 84] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), February 1984.

[Cart 87] Michèle Cart, Jean Ferrie, and Sukrisno Mardyanto. Atomic broadcast protocol, preserving concurrency for an unreliable broadcast network. In J. Cabanel, G. Pujole, and A. Danthine, editors, *Local communication systems: LAN and PBX*, North-Holland, IFIP, 1987.

[Chang 84] J. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM, Transactions on Computer Systems*, 2(3), August 1984.

[Chesson 88] Greg Chesson. XTP/PE overview. In *13th Local Computer Network Conference*, Minneapolis-USA, October 1988.

[Cristian 85] F. Cristian, Aghili. H., R. Strong, and D. Dolev. Atomic Broadcast: From simple message diffusion to Byzantine Agreement. In *Digest of Papers, The 15th International Symposium on Fault-Tolerant Computing*, IEEE, Ann Arbor-USA, June 1985.

[FDDI 86] X3T9.5 FDDI. *FDDI documents: Media Access Layer, Physical and Medium Dependent Layer, Station Mgt.* 1986.

[Gorur 88] R.Mangala Gorur and Alfred C. Weaver. Setting target rotation times in an IEEE Token Bus network. *IEEE Transactions on Industrial Electronics*, 35(3), August 1988.

[Hutchinson 88] Norman C. Hutchinson and Larry P. Peterson. Design of the x-Kernel. In *SIGCOMM'88: Communications Architectures and Protocols*, ACM, Stanford, USA, August 1988.

[ISO 85a] *ISO DIS 8802/2-85, Logical Link Control.* 1985.

[ISO 85b] *ISO DIS 8802/4-85, Token Passing Bus Access Method.* 1985.

[ISO 85c] *ISO DP 8802/5-85, Token Ring Access Method.* 1985.

[Jain 90] Raj Jain. Performance analysis of FDDI token ring networks: effect of parameters and guidelines for setting TTRT. In *SIGCOM'90 Symposium*, ACM, Philadelphia-USA, September 1990.

[Janetzky 86] Dittmar Janetzky and Kym S. Watson. Token bus performance in MAP and Proway. In *IFAC Workshop on Distributed Computer Protocol System*, 1986.

[Jayasumana 89] Anura P. Jayasumana. Throughput analysis of the IEEE 802.4 priority scheme. *IEEE Transactions on Communications*, 37(6), June 1989.

[Lamport 82] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Prog. Lang. and Systems*, 4(3), July 1982.

[MOT 84] *MC68020 32-Bit Microprocessor User's Manual.* Prentice-Hall, USA, 1984.

[Powell 88] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk. The Delta-4 approach to dependability in open distributed computing systems. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, IEEE, Tokyo - Japan, June 1988.

[Rodrigues 91a] L. Rodrigues and P. Veríssimo. *A posteriori agreement for clock synchronization on broadcast networks.* Technical Report, INESC, March 1991.

[Rodrigues 91b] L. Rodrigues and P. Veríssimo. *xAMp: a multi-primitive group communications service.* Technical Report, INESC, March 1991.

[Rufino 92] J. Rufino and P. Veríssimo. A study of the inaccessibility characteristics of ISO 8802/4 Token-Bus LANs. In *IEEE Infocom92 Conference*, IEEE, May 1992. (to appear in).

[TBC 87] *MC68824 Token Bus Controller Data Sheet.* Motorola, January 1987.

[Tusch 88] Jũrgen Tusch, Heinrich Meyr, and Erwin A. Zurfluh. Error handling performance of a Token Ring LAN. In *13th Conference on Local Computer Networks*, IEEE, Minneapolis, USA, October 1988.

[Verissimo 88a] Paulo Veríssimo. Redundant media mechanisms for dependable communication in token-bus LANs. In *13th Local Computer Network Conference*, IEEE, Minneapolis-USA, October 1988.

[Verissimo 88b] Paulo Veríssimo, José Rufino, and José A. Marques. *NAC Implementation Specification: Token-bus Controller.* Technical Report RT/110, Delta-4 Project, INESC, Lisboa, Portugal, April 1988.

[Verissimo 89] P. Veríssimo, L. Rodrigues, and M. Baptista. AMp: a highly parallel atomic multicast protocol. In *SIGCOM'89 Symposium*, ACM, Austin-USA, September 1989.

[Verissimo 90] P. Veríssimo and José A. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *Ninth Symposium on Reliable Distributed Systems*, IEEE, Huntsville, Alabama-USA, October 1990. Also as INESC AR/24-90.

[Verissimo 91] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In *10th IFAC Workshop on Distributed Computer Control Systems*, IFAC, Semmering, Austria, September 1991.

Appendix A

Consolidated Model Parameters

The AMp, as described in [Verissimo 89], is engineered at the data link level. The LAN independence is assured by its interface with an *abstract local area network*, whose properties are guaranteed by a harmonizing driver built on the top of the exposed MAC interface of the particular VLSI LAN controller. This architecture is sketched in Figure A.1, where we have explicitly identified three relevant domains. The processing time parameters of the aforementioned entities are characterized. Together with the underlying network model they are used afterwards to establish the *consolidated parameters* of our performance model.

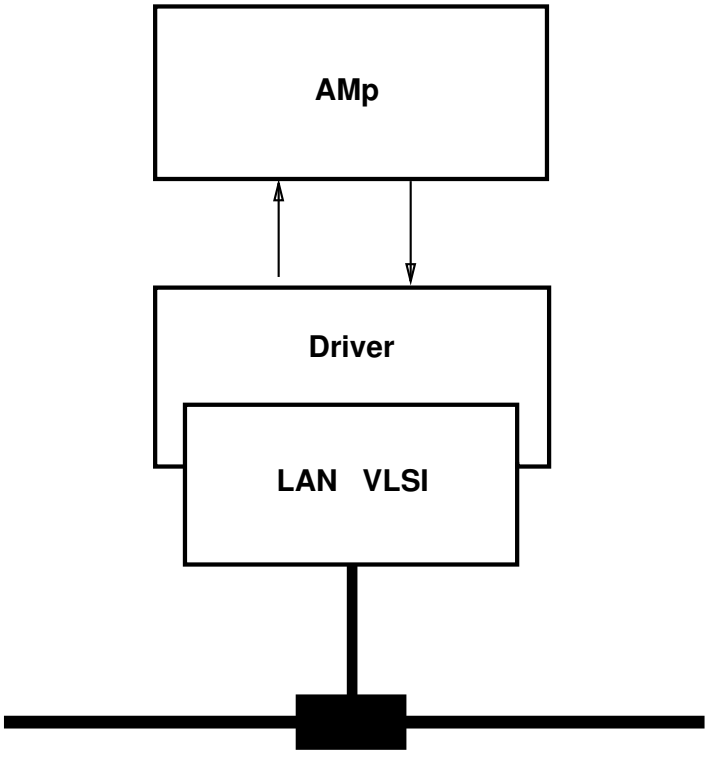


Figure A.1: Communication System Architecture

We start by define the processing times intrinsically associated with **AMp** execution:

- t_{ampUsr} – AMp processing time for an user request.
- t_{ampRx} – AMp processing time for reception of a generic protocol frame.
- $t_{ampRxRp}$ – AMp processing time for reception of an information frame and response generation.
- $t_{ampRxRq}$ – AMp processing time for reception of a request decision and issuing of the corresponding decision.
- $t_{ampBg}(n)$ – AMp processing time for a bag of n responses.
- $t_{ampBgTim}(n)$ – AMp processing time, upon timeout, for a bag of expected n responses and initiating message retransmission.
- $t_{ampDecGen}$ – AMp processing time required to generate a decision, upon the processing of all the responses.
- $t_{ampDecTim}$ – AMp processing time upon $TwaitDecision$ timeout indication
- t_{ampCf} – AMp processing time for the confirmation of a transmission.
- $t_{ampS1Rp}$ – AMp processing time for reception of a step one request and their response generation.
- $t_{ampBgStep1}(n)$ – AMp processing time for a bag of n step one responses.
- $t_{ampS1Gen}$ – AMp processing time for the generation of the step one request.
- $t_{ampS2Gen}$ – AMp processing time for the generation of the step two request.

At the **Abstract Network domain** we define two sets of parameters. One of the sets concerns the processing in the *harmonization driver*:

- t_{drvRx} – Driver processing time for frame receipt.
- t_{drvTx} – Driver processing time for frame transmit.
- t_{drvCf} – Driver processing time required for the confirmation of each transmission.

while the other set accounts some aspects of the bare VLSI performance:

- t_{macCf} – Delay associated with the generation of transmission confirmation, by the VLSI.
- t_{macRx} – Delay associated to the VLSI, for frame receipt.

Additionally we define t_{lseTim} as the time inherently wasted by *local support environment* timer agency, for processing the timeout.

The relationship between the **consolidated processing times** and those previously defined is expressed by equations (A.1) through (A.12).

$$t_{prUsr} = t_{ampUsr} \tag{A.1}$$

$$t_{prRx} = t_{macRx} + t_{drvRx} + t_{ampRx} \tag{A.2}$$

$$t_{prRxRp} = t_{macRx} + t_{drvRx} + t_{ampRxRp} + t_{drvTx} \tag{A.3}$$

$$t_{prRxRq} = t_{macRx} + t_{drvRx} + t_{ampRxRq} \tag{A.4}$$

$$t_{prRxBg}(n) = t_{macRx} + (n-1)\,.\,t_{drvRx} + t_{ampBg}(n) \tag{A.5}$$

$$t_{prBg}(n) = t_{macRx} + (n-1)\,.\,t_{drvRx} + t_{ampBg}(n) + t_{ampDecGen} + t_{drvTx} \tag{A.6}$$

$$t_{prBgTim}(n) = t_{lseTim} + t_{ampBgTim}(n) \tag{A.7}$$

$$t_{prDecTim} = t_{lseTim} + t_{ampDecTim} \tag{A.8}$$

$$t_{prTxCf} = t_{macCf} + t_{drvCf} + t_{ampCf} \tag{A.9}$$

$$t_{prS1Gen} = t_{ampS1Gen} \tag{A.10}$$

$$t_{prS1Rp} = t_{macRx} + t_{drvRx} + t_{ampS1Rp} + t_{drvTx} \tag{A.11}$$

$$t_{prBgStep1}(n) = t_{macRx} + (n-1)\,.\,t_{drvRx} + t_{ampBgStep1}(n) + t_{ampS2Gen} + t_{drvTx} \tag{A.12}$$

The worst-case performance figures, required for timer programming, can be obtained from these ones:

$$t^{wc}_{prRx} = (1 + v_P)\,t_{prRx} \tag{A.13}$$

$$t^{wc}_{prRxRp} = (1 + v_P)\,t_{prRxRp} \tag{A.14}$$

$$t^{wc}_{prRxBg}(n) = (1 + v_P)\,t_{prRxBg}(n) \tag{A.15}$$

$$t^{wc}_{prBg}(n) = (1 + v_P)\,t_{prBg}(n) \tag{A.16}$$

$$t^{wc}_{prS1Rp} = (1 + v_P)\, t_{prS1Rp} \tag{A.17}$$

where v_P accounts the CPU load variability.

The last performance model parameter to be defined is the *consolidated network access delay*. This parameter have a random component, resulting from the time required to access the **network**, and also presents components depending on the AMp and harmonization driver processing times:

$$t_{acc} = t_{drvTx} + t_{racc} \tag{A.18}$$

where t_{racc} is the *average raw access delay*.

$$t_{racc} = \frac{R_{av}(\rho)}{2} \tag{A.19}$$

For token-based networks the value of $R_{av}(\rho)$, besides the explicitly dependency on the network load, also varies with the network length and number of stations, and should be defined for each LAN.