Design and Implementation of a QoS-Supportive System for Reliable Multicast

Thesis by

Antonio Di Ferdinando

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

University of Newcastle upon Tyne

Newcastle upon Tyne, UK

2007

(Defended May 11, 2007)

To Elena.

Acknowledgements

I would like to express my sincere gratitude to the following people who have helped me in the completion of this thesis.

First and foremost, I would like to thank my supervisor Dr. Paul D. Ezhilchelvan for his availability, patience and guidance throughout the four years necessary for this research. My thanks also go to Prof. Isi Mitrani for the precious help and supervision in the building of the mathematical model.

The research in this thesis was funded through the TAPAS EU-IST-2001-34069 Project, the EPSR (Engineering and Physical Sciences Research Council) PACE Project GR/S02082/01 and the EPSRC Platform Grant EP/D037743. I also thank Prof. Santosh Shrivastava for hiring me as a Research Associate in the TAPAS project which led me to do this thesis work.

Thanks are also due to many of the staff in the School of Computing Science at the University of Newcastle. The environment within the school has consistently been one of encouragement and support. Particular mention goes to Miss Giovanna Ferrari for her support.

Abstract

As the Internet is increasingly being used by business companies to offer and procure services, providers of networked system services are expected to assure customers of specific Quality of Service (QoS) they could offer. This leads to scenarios where users prefer to negotiate required QoS guarantees prior to accepting a service, and service providers assess their ability to provide the customer with the requested QoS on the basis of existing resource availability. A system to be deployed in such scenarios should, in addition to providing the services, (i) monitor resource availability, (ii) be able to assess whether or not requested QoS can be met, and (iii) adapt to QoS perturbations (e.g., node failures) which undermine any assumptions made on continued resource availability. This thesis focuses on building such a QoS-Supportive system for reliably multicasting messages within a group of crash-prone nodes connected by loss-prone networks.

System design involves developing a Reliable Multicast protocol and analytically estimating the multicast performance in terms of protocol parameters. It considers two cases regarding message size: small messages that fit into a single packet and large ones that need to be fragmented into multiple packets.

Analytical estimations are obtained through stochastic modelling and approximation, and their accuracy is demonstrated using simulations. They allow the affordability of the requested QoS to be numerically assessed for a given set of performance metrics of the underlying network, and also indicate the values to be used for the protocol parameters if the affordable QoS is to be achieved. System implementation takes a modular approach and the major sub-systems built include: the QoS negotiation component, the network monitoring component and the reliable multicast protocol component. Two prototypes have been built. The first one is built as a middleware system in itself to the extent of testing our ideas over a group of geographically distant nodes using PlanetLab. The second prototype is developed as a part of the *JGroups Reliable Communication Toolkit* and provides, besides an example of scenario directly benefitting of such technology, an example integration of our subsystem into an already-existing system.

Declaration

All work contained within this thesis represents the original contribution of the author. Most of the material in this thesis has been, or will shortly be, published in conference proceedings and a journal, as listed below. The material in chapter 3 has been published in 2 and 3; the material in chapter 4 has been published in 2, 3 and 4; the material in chapter 5 has been published in 5; the material presented in chapter 6 has been published in 5; the material in chapter 7 has been published in 1 and 2. In addition, an extract of each of the chapters composing this thesis has been submitted for publication as 6.

1. Integration of the Group Communication Protocol into *JGroups*, A. Di Ferdinando and P.D. Ezhilchelvan. In *TAPAS EU-IST Project Deliverable D13: Second Year Evaluation and Assessment Report.* S. Shrivastava et al. (Eds). http://tapas.sourceforge.net/deliverables/D13.pdf, March 2004.

2. Performance Evaluation of a QoS-Adaptive Reliable Multicast Protocol, A. Di Ferdinando, P.D. Ezhilchelvan and I. Mitrani. Technical Report, University of Newcastle, April 2004.

3. QoS-Adaptive Group Communication, A. Di Ferdinando, P.D. Ezhilchel-

van, I. Mitrani and G. Morgan *TAPAS EU-IST Project Deliverable D8.* http://tapas.sourceforge.net/deliverables/D8.pdf, May 2004.

4. Design and Evaluation of a QoS-Adaptive System for Reliable Multicasting, A. Di Ferdinando, P.D. Ezhilchelvan and I. Mitrani. In *Proc of the 23rd IEEE International Symposium on Reliable Distributed Systems* (SRDS04), Brazil, pp. 31-40, IEEE Computer Society, October 2004.

5. A QoS-Negotiable Middleware System for Reliably Multicasting Messages of Arbitrary Size, A. Di Ferdinando, P.D. Ezhilchelvan, M. Dales and J. Crowcroft. In *Proc of the ninth EEE International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC 2006), Korea, pp. 253-260, IEEE Computer Society, April 2006.

6. Design and Evaluation of a QoS-Adaptive System for Reliable Multicasting, A. Di Ferdinando, P.D. Ezhilchelvan and I. Mitrani. Submitted to *IEEE Transactions on Dependable and Secure Computing.*

Contents

ii

iv

Acknowledgements **v**

vi

Abstract **vii**

Declaration **ix**

xviii

xxiv

1 Introduction **1**

1.1 Background and motivation 1

1.2 Dissertation Focus 3

1.3 Primary Contributions 5

1.4 Dissertation Overview 7

2 Group Communication and QoS **9**

2.1 Introduction 9

2.2 Reliability and Timeliness 10

2.2.1 Reliability 11

2.2.1.1 Deterministic reliability 11

2.2.1.2 Probabilistic reliability 15

2.2.2 Timeliness 17

2.2.2.1 Resource Reservation 17

2.2.2.2 Dynamic Routing 19

2.3 Discussion 21

2.3.1 Considering both Reliability *and* Timeliness 21

2.3.2 Negotiation 26

2.3.3 Our approach: reliable and timely multicast 28

2.4 Concluding Remarks 28

3 System Model and Architecture **31**

3.1 Introduction 31

3.2 System model 33

3.3 QoS-Adaptive Middleware Architecture 36

3.4 QoS-Supportive Reliable Multicast System Architecture 39

3.4.1 Network Measurement Component 43

3.4.2 Negotiation Component 44

3.4.3 RMcast Component 46

3.4.4 Sequential Interaction Model 48

3.5 Concluding Remarks 50

52

4 Single Packet Protocol **53**

4.1 Introduction 53

4.2 System context 54

4.3 Design of the Reliable Multicast Protocol 55

4.3.1 Specification of Protocol Guarantees. 56

4.3.2 Design features 57

4.3.2.1 Redundancy 58

4.3.2.2 Responsiveness 59

4.3.2.3 Selection 60

4.4 Details of the protocol 61

4.5 Reliability and Latency Estimations 64

4.5.1 Reliability 64

4.5.2 Latency Estimations 67

4.5.3 Relative Latency 69

4.6 Simulation Results 71

4.7 Concluding Remarks 80

82

5 Multicasting Messages of Arbitrary Size **83**

5.1 Introduction . . . 83

5.2 From single-packet to multi-packet messages . . . 84

5.2.1 Per-Message approach. . . . 85

5.2.2 Per-Packet approach. . . . 91

5.2.2.1 Remarks . . . 92

5.3 Analytical estimations . . . 95

5.3.1 Reliability. . . . 95

5.3.2 Latencies for a single-packet message . . . 95

5.3.3 Estimations for the Per-Message approach. . . . 98

5.3.3.1 Reliability estimations . . . 98

5.3.3.2 Latency estimations . . . 99

5.3.3.3 Reliability in the Per-Packet approach. . . . 101

5.3.3.4 Latency in the Per-Packet approach. . . . 102

5.4 Simulation experiments . . . 102

5.4.1 Negotiability . . . 106

5.4.1.1 Negotiability in the Per-Message approach . . . 106

5.4.1.2 Negotiability in the Per-Packet approach . . . 113

5.4.2 Cost of the system . . . 119

5.4.2.1 Cost of the Per-Message approach . . . 119

5.4.2.2 Cost of the Per-Packet approach . . . 121

5.5 Limitations and applicability domains . . . 124

5.5.1 Per-Message extension . . . 124

5.5.2 Per-Packet extension . . . 129

5.6 Concluding remarks 130

132

6 Network Performance Measurement 133

6.1 Introduction 133

6.2 Monitoring the Communication Subsystem 134

6.3 Sampling and Probing Technique 138

6.4 Metrics of Interest 141

6.5 Delay Probabilities Calculation 144

6.5.1 Statistical Evaluation 145

6.5.2 Experimental Calculation 147

6.5.3 Criticisms 147

6.6 Concluding Remarks 148

150

7 Prototype Implementations 151

7.1 Introduction 151

7.2 Design and implementation tools 153

7.2.1 UML Notation 154

7.2.1.1 Package Diagrams 155

7.2.1.2 Component Diagrams 156

7.2.1.3 Sequence Diagrams 159

7.3 *alipes* 161

7.3.1 Core Components 162

7.3.1.1 Container 164

7.3.1.2 Negotiation Component 167

7.3.1.3 RMcast Component 173

7.3.1.4 Network Measurement Component 178

7.3.2 Non-core components 182

7.3.2.1 XML parser 182

7.3.2.2 Group Membership Protocol 183

7.3.2.3 Time and Network Update Metrics Update subsystems 185

7.3.3 Testing in a Real Scenario 187

7.3.3.1 Environment and methodology 187

7.3.3.2 Latency 193

7.4 *QoS-JGroups* 195

7.4.1 The *JBoss* application server 197

7.4.2 The *JGroups* Reliable Communication Toolkit 199

7.4.2.1 *Channel API* 199

7.4.2.2 *Building Blocks* 200

7.4.2.3 *protocol stack* 201

7.4.3 Integration: the `RMCAST` protocol object 204

7.4.4 A word on testing 209

7.5 Criticisms 210

7.6 Concluding Remarks 211

8 Conclusions **213**

8.1 Discussion 213

8.2 Take-home Message 219

8.3 Directions of Further Research 220

8.4 Concluding Remarks 224

A A Gossip-based Group Management Protocol **227**

A.1 Introduction 227

A.2 Design features 228

A.2.1 Gossip 229

A.3 Analytical approximations 230

A.4 Simulation Experiments 234

238

Bibliography **239**

List of Figures

3.1 Comparison of models 36
3.2 QoS-Adaptive middleware architecture. 37
3.3 Position in the ISO OSI hierarchy. 40
3.4 System architecture. 41
3.5 Network Measurement on a best effort CS. 44
3.6 Network Measurement Component on an ISP-managed CS. 44
3.7 Negotiation Component in presence of a best effort CS. 46
3.8 RMcast Component in presence of a best effort network. 48
3.9 Sequential system logic base on main tasks: network monitoring (phase 1), negotiation (phase 2) and service provision (phase 3) . 50

4.1 Pseudo-code for $RMCast(m)$ 61
4.2 Pseudo-code for $RMdeliver()$ 62
4.3 Pseudo-code for $Broadcaster(m)$ 63
4.4 Protocol's failure probability, for $\beta = 0.02$, $\gamma = 100$ hours 67
4.5 Results of group 1 experiments 72
4.6 Results of group 2 experiments 73
4.7 Results on group 3 experiments 74

4.8 Results on group 4 experiments 75

4.9 Relative error on Latency bounds 77

4.10 Relative error on Relative Latency 79

5.1 Per-Message when $\rho = 1$ and $\pi = 3$ 86

5.2 Per-Message multicast process. 90

5.3 Usage of the broadcast primitive in the Per-Message approach. . . . 91

5.4 Per-Packet multicast process. 92

5.5 Single-threaded Per-Packet approach 94

5.6 Performance of the Per-Message approach when $\pi = 3$ and $\omega = 1$ 103

5.7 Performance of the Per-Message approach when $\pi = 7$ and $\omega = 1$ 104

5.8 Performance of the Per-Message approach when $\pi = 10$ and $\omega = 1$ 105

5.9 Performance of the Per-Message approach when $\pi = 3$ and $\omega = 0$ 106

5.10 Performance of the Per-Message approach when $\pi = 7$ and $\omega = 0$ 107

5.11 Performance of the Per-Message approach when $\pi = 10$ and $\omega = 0$ 108

5.12 Performance of the Per-Message approach in the direct receivers scenario when $\pi = 3$ and $\omega = 1$ 109

5.13 Performance of the Per-Message approach in the direct receivers scenario when $\pi = 7$ and $\omega = 1$ 110

5.14 Performance of the Per-Message approach in the direct receivers scenario when $\pi = 10$ and $\omega = 1$ 111

5.15 Performance of the Per-Packet approach when $\pi = 3$ and $\omega = 1$. 112

5.16 Performance of the Per-Packet approach when $\pi = 7$ and $\omega = 1$. 113

5.17 Performance of the Per-Packet approach when $\pi = 10$ and $\omega = 1$ 114

5.18 Performance of the Per-Packet approach when $\pi = 3$ and $\omega = 2$. 115

5.19 Performance of the Per-Packet approach when $\pi = 7$ and $\omega = 2$. 116

5.20 Performance of the Per-Packet approach when $\pi = 10$ and $\omega = 2$ 117

5.21 Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 3$ and $\omega = 1$ 118

5.22 Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 7$ and $\omega = 1$ 119

5.23 Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 10$ and $\omega = 1$ 120

5.24 PlanetLab slice 124

5.25 Growth of Q^{PM} compared to q. 126

5.26 Loss rate in the PlanetLab slice 127

5.27 Average jitter in the PlanetLab slice 128

6.1 Monitoring of a best effort CS. 137

6.2 Network monitoring over an ISP-managed CS. 138

6.3 Three-way ping probing technique. 139

7.1 Example of UML Package Diagram 155

7.2 Example of UML Component Diagram 158

7.3 Example of UML Sequence Diagram 160

7.4 Package structure for the *alipes* prototype 161

7.5 System components interaction scheme 163

7.6 Structure of the `Container` class. 164

7.7 Initialization of the `Container` object. 168

7.8 Structure of the Negotiation `Component` interface. 169

7.9 Component diagram for the Negotiation Component 171

7.10 Sequence diagram for the negotiation process 173

7.11 Structure of the RMcast `Component` interface. 174

7.12 Component diagram for the RMcast Component 176

7.13 Sequence diagram for the RMcaster 178

7.14 component diagram for the Network Measurement Component. . 179

7.15 Group Management protocol 183

7.16 Nodes affiliations and numbering. 188

7.17 Nodes location. 190

7.18 Results from tests on the Planet Lab slice 196

7.19 Architecture of *JGroups* 199

7.20 Definition string for the default protocol stack 202

7.21 *JGroups* protocol stack with integration of the `RMCAST` protocol object 205

7.22 New default definition string, with `RMCAST` integrated 208

8.1 Architectural extension of the QoS-Adaptive middleware architecture. 221

A.1 Coverage time for the gossip protocol when $\xi = 2.5$ ms and τ is variable with the group size 234

A.2 Value of τ based on the group size 235

A.3 Performance with various values of ξ, when $\tau = 8$ and gossip targets=2. 236

Chapter 1

Introduction

1.1 Background and motivation

The Internet is increasingly being used by organizations for offering and procuring online services. Examples of this trend are business outsourcing and applications service provisioning[88]. In this context, business organizations outsource Internet-based services to Application Service Providers (or ASPs). These build sophisticated services (e.g. e-auctions) which are then made available to e-business companies as *e-services* under payment of a certain amount of money.

ASPs are required to guarantee and maintain *Quality of Service* (QoS) to fulfill a range of needs which vary from client to client. Meeting this requirement is a challenging task because the promised QoS guarantees have to be maintained for the entire duration of service provision despite the occurrence of potentially performance-degrading events. In other words, the service provisioning process needs to account for variations such as, for instance, increased packet delay due to network congestion. Moreover, it is becoming increasingly common for

e-services to be dynamically composed out of other e-services. This means that the users will negotiate the required QoS guarantees as the need for a service arises. This in turn means that the system should be able to decide instantly whether or not a given QoS request can be met and, if so, offer the service with the requested QoS. That is, the service provisioning is preceded by a *QoS negotiation* process that involves estimation of appropriate system performance metrics and making feasibility assessments on the requested QoS. From the perspective of QoS negotiation, the system can be regarded to be composed of several, hierarchically-structured subsystems. Therefore, the systemwide QoS provided can be seen as an aggregation of the QoS provided by all subsystems involved in providing a service. Putting it differently, the QoS that can be offered while providing a service is an aggregation of the QoS that each subsystem involved can be expected to offer while contributing to the composite service. Therefore, each subsystem that makes up a service should desirably have a *predictable* QoS behavior. At the lowest level of the system hierarchy are subsystems directly handling resources. Among these, the *Communication Subsystem* (CS), that supports message exchange between processes implementing a distributed application, holds particular importance as it provides facilities to communicate with the external.

Making behavior of the CS predictable is a challenging task when communicating nodes are geographically distant and when data is sent on a best effort basis. Moreover, in such scenarios communication data travels across the Internet infrastructure, with its heterogeneous domain-wise traffic policies and

bottlenecks, and therefore are subject to QoS variations.

There are two known approaches to obtaining a CS with predictable QoS behavior. The first is to purchase the CS services from an Internet Service Provider (ISP) such as AT&T. This approach however restricts the service construction/composition to be ISP-centric: while each ISP offers attractive QoS guarantees within its own domain, ISPs do not normally cooperate between themselves to offer such guarantees across their domains; consequently, the CS of 'networked system' is merely the domain of a single ISP. The second approach is to build a *Group Communication* (GC) *middleware* which, in addition to exporting sophisticated communication primitives for application development, monitors and extrapolates the CS behavior for QoS assessment and negotiation purposes.

1.2 Dissertation Focus

The work described in this thesis is the first step in the direction of building a QoS-Negotiable Group Communication Middleware, and builds a basic Group Communication primitive of *Reliable Multicast* (*RMcast* for short). By means of these, a crash-prone process can multicast messages to named destination sets. *Reliability* and *timeliness* of multicast delivery will be the two QoS attributes considered for service level guarantees; on the latter, we consider two forms of latency: the message delivery time for an operative destination that received the message (*absolute latency*) and the differential delay within which

two operative processes can receive the message (*relative latency*).

The core of the system is a fault-tolerant Reliable Multicast[58] (RMcast) protocol capable of supporting QoS-sensitive communication. The protocol performs runtime adaptations in order to fulfill specific (QoS) guarantees, which are negotiated with the user application anticipately to service provision.

The system does not rely on any particular network support. For example, it does not assume the use of any particular delivery model such as IP multicast[32] for fulfillment of promised guarantees. Moreover, it has been designed for the use on a network providing unreliable connectionless (i.e. UDP-based[95]) communication.

Negotiation of QoS requirements is facilitated through a negotiation interface, that allows the user application to request a customized QoS level prior to service provision. The negotiation interface is based on a stochastic model of the RMcast protocol. Its estimation aims to predict behavior of the system in the near future. Prediction is done on a statistical basis, and as such is not expected to change drastically soon thereafter.

The stochastic model estimates behavior of the RMcast protocol based on current network conditions. This latter type of information is provided by a network monitoring facility in the form of statistical data. Given the statistical nature this information, data provided by the network monitoring facility is not expected to be subject to drastic variations.

The protocol subject of this thesis is designed for the use among a group of hosts communicating through the Internet regardless of the CS providing

best-effort facilities or being managed by an ISP-centric environment.

1.3 Primary Contributions

The system presented in this thesis is novel in a way that it considers both reliability and timeliness as equally important parameters for the overall success of the multicast communication process. In addition, the protocol allows reliability and timeliness attributes to be negotiated with the user-application prior to service provision, In doing so, it makes a number of contributions.

The first contribution lies in designing a system which considers reliability and latency as negotiable QoS attributes for reliable multicasting of messages across the Internet. The communication process is performed in respect of reliability and timeliness bounds through an anticipate negotiation between the system and the user application. The system then provides guarantees on fulfillment of the operation within performance levels complying with *both* attributes.

The second contribution lies in the extension of the basic protocol to provide likewise guarantees on messages of arbitrary size. Two extension approaches are presented and studied. Each of them impacts the network differently and, therefore suit a specific network environments. Together, these two approaches cover a wide range of application environments.

The third contribution lies in providing an architecture and analytical expressions for negotiating QoS metrics. The negotiation architecture provides facil-

ities allowing the system to negotiate QoS metrics with the user application. Negotiation takes place anticipately to start of the multicast operation, and is performed by estimating a stochastic model of the core RMcast protocol. This latter model is fully described, and its accuracy is studied by means of simulations.

The fourth contribution lies in evaluating a protocol of adaptive design and configurable parameters. The core RMcast protocol is designed so as to be configurable around a set of parameters which allow to optimize its behavior based on runtime network performances.

The fifth contribution lies in the description of a network measurement engine capable of providing statistical QoS metrics on a regular basis. The engine provides the basis upon which to estimate the stochastic model for negotiation by describing average network packet loss, delay and jitter. Moreover, it provides two ways of estimating conditional probabilities into numerically tractable equations.

The sixth contribution lies in describing two prototype implementations of the system. A first implementation suits middleware environments and is included in a GC system including also a basic group management protocol. This middleware suite can be used as basis for provision of more sophisticated QoS-supportive services. A second prototype provides an example of integration of our system into a workable application, that can benefit from such integration. It consists in a version of the *JGroups Toolkit for Reliable Multicast Communication*[13] tool, modified so as to utilize our RMcast system

in conjunction with a basic transport layer so as to bring support for QoS in multicast communication.

1.4 Dissertation Overview

This thesis is structured as follows: in chapter 2 we relate our work with other works in the same area. We introduce well known GC protocols and systems found in literature, and describe the techniques there employed to guarantee QoS in a multicast process, along with the correspondent type of QoS they are concerned with.

In chapter 3 we describe the system model and architecture adopted in our work. The former is defined formally and also compared to the traditional system models to derive its applicability domain. The system architecture is described in all fundamental components and interaction models.

Chapter 4 describes the design of the basic RMcast protocol for provision of guarantees on messages of standard size, i.e. messages that fit into a single packet. The stochastic model driving the negotiation process for the multicast of such messages is also described, and its accuracy is discussed through simulations.

Chapter 5 describes two proposed approaches to the extension of the basic RMcast protocol described in chapter 4 to contexts where the same QoS guarantees must be provided on the multicast of messages of arbitrary size, i.e. messages that need to be fragmented into multiple packets. The original stochastic

model is also extended according to characteristics of each of the extension approaches. The two stochastic models so obtained are then analyzed, through simulations, in terms of *negotiability* and *cost of the system*. The former will assess accuracy of the stochastic model, while the latter will give a measure of the additional cost of usage of the system in terms of message overhead.

Chapter 6 describes in detail the network monitoring engine. Techniques used to gather network information and measure QoS metrics are described. The role and usefulness of this subsystem is described in contexts where the CS is assumed to provide best-effort communication facilities and when this latter is managed by an ISP.

Chapter 7 describes structure and paradigms used to develop the two prototype implementations, while chapter 8 draws some conclusion and outlines future developments research. Finally, in appendix A we give an example of group management protocol that might be used in conjunction with the RMcast protocol to form a GC system.

Chapter 2

Group Communication and QoS

2.1 Introduction

Group communication (GC) is a well known and very important paradigm by means of which several, possibly geographically distant, processes can engage in a multiparty coordination to the extent of achieving a common goal. At the base of this paradigm, a *multicast* communication protocol handles transmission of the desired data to all destinations.

The term *multicast* refers to a communication paradigm that allows transmission of a message to a selected group of destinations. It is long considered to be a fundamental communication feature, due to importance of its topic in distributed contexts. Moreover, protocols and GC systems are typically enriched with functionalities that couple the multicast service with QoS guarantees. Among the possible QoS aspects, *reliability* and *timeliness* attributes are retained to be fundamental building blocks for the provision of more complete and sophisticated services. As such, they hold particular importance.

Reliability is intended as the ability to guarantee eventual reception of data

transmitted by *all* destinations in a multicast operation. Timeliness, on the other hand, refers to the ability to guarantee a time bound on message delivery.

In this chapter we investigate provision of reliability and timeliness with concern to the multicast operation, relating relevant approaches found in literature with our own and putting emphasis on the differences.

2.2 Reliability and Timeliness

Reliability is the most basic form of QoS, and its provision is a first step towards the building of more complete services. Its utilization is mandatory for provision of sophisticated guarantees such as ordered delivery[16, 17]. Moreover, many other QoS attributes, such as security, can benefit from the provision of multicast reliability.

In recent years, the Internet has gained popularity as a mean for doing business. One of the firsts consequences of this growth in consideration is that the sole guarantee of *eventual* delivery of multicast information became insufficient to satisfy needs of business applications. A nice example of this trend is the use of networked auctions in the financial market. There, auctions are handled by trading agents which exploit the real-time nature of the Internet by reducing duration of auctions[54]. Bidders are required to be able to take actions in a timely way, and this has inevitably repercussions on the need for timely, other than reliable, communication.

Literature is rich of communication systems and protocols which can suit a wide range of application contexts. However, to the best of our knowledge, *none* of the approaches analyzed addresses reliability *and* timeliness issues in the multicast process. That is, none of the approaches found in literature considers reliability and timeliness as *functional* requirements for a multicast system or protocol. For this reason, in the remainder of this section we describe a selection of the most interesting and successful techniques used to provide separate reliability and timeliness.

2.2.1 Reliability

Reliability of a protocol/system refers to the capability of guaranteeing delivery of data transmitted in a communication operation to all intended destinations. In particular, this guarantee does not have to involve any time constraint, i.e. delivery is eventual.

Reliability is the most basic form of QoS a system can provide, and approaches studied to achieve it can be divided into two broad categories. The first contains protocols aiming to provide *deterministic* guarantees, while the second contains approaches aiming to provide *probabilistic* guarantees.

2.2.1.1 Deterministic reliability

Protocols in this category provide guarantees on the eventual delivery of data on a deterministic basis. Reliability is typically achieved by employing techniques based on *ACKnowledgements* (ACKs) and/or *Negative AcKnowledge-*

*ment*s (NAKs) to notify the originator and/or the rest of the group of reception and/or loss respectively, of messages. The use of ACK/NAK-based techniques requires efforts to avoid implosion of packets at the source and exposure of receivers to redundant packets.

An effective solution to these problems has been found in grouping receivers hierarchically in tree-like structures. This solution has been exploited by different protocols in several variations. The *Log-Based Receiver Reliable Multicast*[68] (LBRRM), originally designed to support Distributed Interactive Simulations (DIS), arranges its hierarchy in primary and secondary logging servers, to handle retransmissions within a subgroup of the multicast group; receivers request retransmissions from the secondary logging servers, which in in turn request retransmission from the primary logging server. LBRRM uses a variable heartbeat scheme to provide detection of lost packets. The scheme implies transmission of heartbeat messages containing information such as the sequence number of the sequence number of the most recently transmitted message. Messages are sent at a higher frequency rate immediately after a data transmission, while the frequency rate decreases as time from data transmission elapses. The variable heartbeat scheme is shown to allow sooner detection of packet losses in scenarios where the transmission rate is expected to be low. However, in contexts characterized by higher transmission rate, such as scenarios where multiple senders broadcast at the same time, the use of the variable heartbeat scheme by each of them results in a dramatic increase of the message overhead.

A somewhat similar approach, yet structured in a simpler hierarchy, is used by the *Reliable Multicast Transport Protocol* [93] (RMTP). In this, receivers are arranged into local regions with a *Designated Router* responsible for maintaining group membership information and aggregating ACKs and NAKs. RMTP solves the problem of exacerbating the message overhead in scenarios subject to high transmission mentioned for LBRRM, but raises the new problem of drastic control overhead. In fact, RMTP limits exposure of receivers to redundant transmissions and feedback implosion at the source by forcing receivers to be grouped into *local regions* based on their proximity in the network. The grouping into local regions, whose management is expected to be handled by end hosts, trades accuracy, and therefore effectiveness, with control overhead, with the consequence of an accurate and efficient grouping requiring a dramatic increase of the control overhead. This problem is not solved in *Reliable Multicast Transport Protocol II*[113] (RMTP-II), which represents the evolution of RMTP. RMTP-II introduces graceful support for TCP-based congestion control algorithms[8] by interacting with a trusted *Top Node* from which it accepts configuration information. Also suffering from the same weakness are the *Tree-based Multicast Transport Protocol*[117] (TMTP) and the *Local Group Concept*[67] (LGC).

The well known *Pragmatic General Multicast* (PGM) approaches solution of this problem by relying on network support, which also prevents feedback implosion and receiver exposure. Active routers suppress redundant NAKs by forwarding only the first NAK for a given packet sequence number towards

the originator. Subsequent NAKs for the same sequence number are dropped, but the downstream interfaces on which they arrive are marked. The scope of retransmission from the originator is then limited to only those marked interfaces. Receivers observe an exponential back-off prior to sending a NAK, which allows upstream active routers to suppress redundant NAKs by multicasting a confirmation message on the interface of an arriving NAK.

The *Scalable Reliable Multicast*[48] (SRM), on the other hand, solves the problem of preventing implosion of control packets by allowing receivers to wait for a certain time, calculated as a function of the distance of the requesting node from the originator, before sending repair requests, i.e. NAKs, and retransmissions[1]. Implosion is reduced by allowing multicast of NAKs and retransmissions to the entire group. By doing so, in fact, hosts on the point of failing on the same packet the NAK refers to will realize that another host has already requested retransmission and will therefore refrain from sending their own NAK. This has the effect of limiting to one the number of NAKs needed to ask retransmission of a packet regardless of the number of receiver requesting retransmission. Exposure of receivers to redundant packets is limited in SRM by allowing any host having received the packet the NAK refers to to retransmit the packet after expiration of a timeout similar to the one described for the repair request.

[1]This technique is borrowed from the *Xpress Transport Protocol*[103] (XTP), with the only difference that here the time to wait was randomly generated.

2.2.1.2 Probabilistic reliability

Protocols offering probabilistic guarantees on the eventual delivery of data traditionally achieve so by making use of a technique based on *gossiping*, originally invented at Xerox[34]. The basic idea is to *gossip* messages to a randomly chosen subset of the group in order to achieve probabilistic reliability guarantees. Protocols employing this technique are referred to as *gossip-based* or *epidemic* protocols, this latter denomination coming from the protocol dissemination patterns being similar to the one typical of epidemics. Reliability in this category of protocols is achieved through a number of redundant transmissions, to all or a faction of the group, carried out at regular times of specified length. Reliability is guaranteed on a probabilistic basis, and therefore small chances remain that guarantees are not fulfilled.

In the *Ensemble*[62] system, Birman et al. proposed a two-phase approach named *Bimodal Multicast*[14]. In the first phase the message is sent to a group by means of a dissemination protocol using unreliable communication primitives (either via IP multicast[32] or a randomized dissemination protocol[63, 43]). Each destination is assumed to have a set of pseudorandomly generated spanning trees, and the broadcast of a message to the entire group is carried out through the use of one of these. The second phase uses an *anti-entrophy* protocol which ensures reliability by allowing members to exchange summaries of message histories and compensating for inconsistencies.

The *Lightweight Probabilistic Broadcast*[44] merges the two phases of the afore-

mentioned Bimodal Multicast, also integrating notions of membership management by means of which the gossip is shown to be made scalable. The protocol assumes each host to know only a random subset of its neighbors. The actual communication is carried out by gossiping to randomly selected subsets of neighbors. Gossip messages contain several information, some of which aimed to amend group view of the host receiving it.

The approach of including group membership information in the gossiped messages is further exploited by the *Directional Gossip*[82], which takes into account network topology when gossiping. The connectivity of each node is labeled with a value named the *weight*. The larger the weight of a node, the higher the possibility for it to receive a given packet from other nodes. Nodes with a higher weight are then chosen with a smaller probability when gossiping, reducing redundant transmissions. In particular, LANs are represented as single nodes to distant LANs, and "long" routes between two such representatives are seldom chosen.

The *Randomized Reliable Multicast Protocol*[116] (RRMP) combines techniques described for deterministic protocols with probabilistic techniques. As for deterministic protocols, the RRMP arranges receivers in a tree-based structure based on their geographical position. However, unlike deterministic protocols, responsibility for error recovery lies on all members rather than on a single repair server. In RRMP an error can be *local*, i.e. recoverable by asking retransmission to some other member in the local region, or *regional*, i.e. recoverable only by asking retransmission to a member from another region. The former

error is fixed in a local recovery phase, while the latter can be fixed by a remote recovery phase. Both phases are executed concurrently as part of the protocol.

2.2.2 Timeliness

Timeliness refers to the capability of terminating a communication process within a predictable interval of time.

Resource reservation and *dynamic routing* are the best known techniques to provide timely multicast communication, and their study led to development of several protocols.

2.2.2.1 Resource Reservation

Resource reservation is a very well known technique to support timeliness. As the name suggests, the idea behind this technique is to reserve the amount of network resources needed anticipately to start of service provision.

The *Internet Engineering Task Force* (IETF)'s *Integrated Services*[115] (IntServ) architecture represents the result of an effort aimed to define a standard model for support of *fine-grained* QoS. In this architecture, an application that needs a specific QoS asks for a reservation based on the type of traffic it will be generating and the QoS guarantees it needs. In IntServ terminology, such description is called a *Flow Spec*.

The actual reservation if performed by the *ReSource reserVation Protocol*[21], and works as follows: all machines on the network capable of supporting some

QoS send a `PATH` message every 30 seconds, which spreads out through the network. Those who want to listen to them send a corresponding `RESV` (short for "Reserve") message which then traces the path backwards to the sender. The `RESV` message contains the Flow Specs. The routers between parties engaged in the reservation process have to decide whether the reservation being requested can be supported or not, and in this latter case they send a reject message to let the listener know about it. If the reservation can be supported, it is accepted and routers are due to carry the traffic. The routers then store the nature of the flow, and also apply a certain policy over it.

This reservation process is done in soft-state, so if nothing is heard for a certain length of time, the reader will time out and the reservation will be cancelled. The individual routers may, at their option, police the traffic to check that it conforms to the flow specs.

An approach similar to RSVP is adopted by Wang and Schulzrinne in [110]; the *Resource Negotiaton And Pricing* protocol[109, 110] (RNAP) is used to reserve resources anticipately to service provision. Reservation is based on exchange of messages at regular intervals. This technique allows to maintain soft-state state information, and when the message chain is broken the reservation is intended to expire. RNAP architecture is designed to work in a distributed (RNAP-D) as well as a centralized (RNAP-C) fashion.

The *Differentiated Services*[18, 77] (DiffServ) architecture is the *Internet Engineering Task Force*'s solution to provision of *coarse-grained* QoS. In it, multiple flows with similar traffic characteristics and performance requirements

are aggregated into a finite set of *classes*. This approach requires either end-user applications, first hop routers or *Ingress* routers, i.e. interfaces where packets enter an administrative domain, to mark individual packets so as to indicate the service class they belong to. The backbone routers provide per-hop differential treatments to different service classes as defined by the *Per Hop Behavior*s[25] (PHBs) specification.

This architecture features two service models, namely *assured service*[64] and *premium service*[71]. The former is intended for customers that need reliable services from service providers. Customers are themselves responsible for deciding how their applications share the amount of bandwidth allocated. The latter, on the other hand, provides low-delay and low-jitter service and is suitable for applications such as Internet telephony, video conferencing and creating virtual lease lines for Virtual Private Networks.

Although flow aggregation improves scalability, the level of statistical guarantees provided by DiffServ, and the question whether such guarantees do exist at all, is unclear. May et al. propose an analysis of DiffServ performance service models in [85], and several studies [24, 91] examine the loss and delay behaviors of the DiffServ architecture using a variety of models.

2.2.2.2 Dynamic Routing

Dynamic routing refers to the capability of determining the source-destination path dynamically based on specific factors, which can be specified by the user.

As an example, the routing protocol might be required to find a path which does not exceed a specific number of hops between source and destination, or a path where the average packet delay does not exceed a given threshold. Different protocols use different techniques to find the routing path.

Deterministic approaches[10] find the path from the source to destination by means of deterministic criteria. For example, an algorithm might choose the shortest path towards a destination as best way to reach it[83, 111, 84], or might choose the shortest-widest path, where width is given by the amount of bandwidth in the path[57].

Randomized approaches[53, 55, 72, 73] allow discovery of the path from a source to a destination in a probabilistic fashion. Once on a hop, the choice of the next segment is taken based on a conditional probability. This latter, in turn, is influenced by the QoS specification to respect. As an example, Gelenbe proposes in [53] an approach where QoS metrics are defined mathematically with respect to a corresponding unit of data (which usually is a packet). Then, a *sensible routing policy* is defined as a probability distribution which selects a path based on such QoS metrics. The routing policy influences determination of the path from source to destination. This policy is in fact applied on each hop, and the path is determined by selecting incrementally segments satisfying the conditional probability (inherently) specified by the routing policy.

2.3 Discussion

2.3.1 Considering both Reliability *and* Timeliness

Approaches described in section 2.2 are concerned with provision of either reliability *or* timeliness guarantees on the communication operation, and none of them considers *both* QoS attributes as equally important guarantees to be provided in the multicast operation. The ones there described represent a selection of approaches that is possible to find in literature. However, as mentioned at the beginning of this chapter, to the best of our knowledge *none* of the approaches found in literature address provision of reliability *and* timeliness.

In some cases, provision of basic QoS guarantees is extended with functionalities which make the service more sophisticated. As an example, reliability can be extended with different ordering schemes to form the so called *Service Composition Frameworks*[108, 90, 99, 9]. These frameworks foresee the presence of *micro-protocol*s [66], which can be stacked so as to offer a more sophisticated service on the whole. The outcome of the use of a service composition framework is typically QoS-sensitive transport layer[114] which is executed on top of the basic TCP[96] or UDP[95] layer. However, applicability of this technique can be considered only when limited to addition of *functionalities* to extend the completeness of guarantees on the actual QoS provided, such as extending reliability with ordering schemes, rather than extending the service to offer guarantees in terms of other QoS attributes. In fact, a widespread

opinion in the international scientific community, also confirmed by experience in provision of QoS, states that provision of QoS, as in our case reliability and timeliness, must be directly addressed as requirement since the design phase of a system. This comes from the consideration that QoS is a *non-functional* system aspect, and as such it cannot be integrated at a second time. In other words, when designing a communication system with QoS features in terms of certain attributes, provision of QoS guarantees needs to be considered as a *functional* requirement in the design phase. Otherwise, the non-functional nature of QoS makes its integration impossible in systems not natively designed for its support.

In protocols offering deterministic reliability guarantees, described in section 2.2.1.1, the ACK/NAK scheme employed is based on reception of ACK/NAK packets by means of which receivers can inform the originator (or any other process designed to handle recovery) about reception/loss of a packet respectively. However, in a network prone to unpredictable failures and delays such as the Internet, application of this scheme might result in ACK/NAK packets to be lost or unduly delayed, with the effect of unduly prolonging the time needed for error recovery and consequently prolonging execution of the protocol for an unpredictable time. Thus, utilization of this error recovery scheme naturally exacerbates the concept of *eventual* delivery guarantees, which contrasts with the concept of *timely* delivery advocated in chapter 1 and a major goal of this thesis. As a side remark, a further limitation for this category of protocols lies in that the vast majority of them assumes network support, such as the IP

multicast[32] datagram delivery model in the case of SRM and RMTP, or enabling special routers, as for PGM, to support provision of reliability. In small and medium environments, the assumption of existence of this type of support can hold. However, the same support cannot be assumed in communications carried out on environments composed by highly heterogeneous means as the Internet.

The problem of recovery scheme naturally excluding provision of timeliness guarantees can be found also in protocols offering probabilistic reliability guarantees, described in section 2.2.1.2. In these protocols, in fact, recovery situations are handled through variations of receiver-side schemes on which loss of a packet is realized after a non predictable amount of time. In detail, protocols and systems in this category of protocols, imply processes to realize loss of packets by unduly *waiting*. Consider for instance the Bimodal Multicast[14]. Receivers come acquainted of loss of a packet only when the anti-enthropy protocol allows exchange of message summaries with processes having received the message eventually lost. Reception of such a summary cannot be predicted to be bounded in time, and therefore usage of such scheme does not allow to provide systematic guarantees on timely message delivery.

Introduction of the Internet as communication medium between nodes employing probabilistic reliability protocols implies introduction of potentially high latency links between nodes. This has repercussions on timeliness issues on this category of protocols regardless of utilization of topology information: in protocols that do not use network topology information in the recovery mech-

anism, such as Bimodal Multicast and Lightweight Probabilistic Broadcast, introduction of the Internet as a communication mean might result in suffering unpredictably high delays in error recovery. On the other hand, in protocols whose error recovery mechanism takes into account topology information, might result in construction of static hierarchies with the consequence of introducing potential single point if failure weaknesses (as in the case of Directional Gossip and RRMP) or, in the best case, considerable message overhead due to adaptation to network instability.

The problem with all approaches mentioned seems to be in "recoverability". In fact, they all rely on recovery mechanisms whose activation cannot be bounded in time. For this reason, we advocate the use of *proactive recoverability* as a fundamental concept to support timeliness in multicast communication: receiving processes should be proactive in triggering a recovery mechanism when delays or losses are detected to affect reception of information. Doing so, in fact, will ensure that receiving processes maintain multicast time bounds.

Trying to add reliability to protocols natively offering timeliness guarantees triggers different problems and mainly tied to application to a multicast context. Consider for instance RSVP[21] described in section 2.2.2.1. The reservation process requires each router on the path from a source to a destination to store a set of states defining characteristics of the resources to reserve. The number of states to be stored grows when the communication operation is a multicast rather than a unicast. As a result, RSVP tend to suffer the multiplicity of states that must be stored in each router. This limits scalability, and

for this reason its usage in multicast contexts is today very limited.

This is a well known limitation of the IntServ architecture and of RSVP in particular, and has been considered heterogeneously by the scientific community. Among the solutions proposed to solve this problem, Baker et al. propose in [12] a solution based on a multi-level approach: resource reservation is proposed to take place on a per-microflow basis (i.e. on the basis of individual users) at the edge network, while in the core network resources are reserved for aggregate flows only. Moreover, routers in between these different levels are proposed to adjust the amount of aggregate bandwidth reserved from the core network. This will allow reservation requests for invididual flows from the edge network to be better satisfied. However, although feasible, the proposed solution requires a considerable amount of work in terms of restructuring of RSVP and, at present, no mention of development or effectiveness of this approach can be found in literature.

On the other side of the scientific community, scientists question about usefulness of multicast support [51] against lightness in NSIS signaling protocols[75] extensions, and open scenarios where multicast is no longer supported in RSVP.

In addition to the problem described above, resource reservation protocols (and in particular RSVP) often depend on routers enabled to deal with specifically tagged traffic. In particular, the RNAP approach assumes each hop (in the distributed architecture, RNAP-D) or each router (in the centralized architecture, RNAP-C), along the path towards destinations, to execute the protocol.

When applied to cross-Internet multicast communication, hosts in the source-destination path belong to separate domains. Each of these will have diverse security policy, and the assumption that all of them support such a proprietary traffic format cannot hold.

Timeliness supported through dynamic routing might also suffer weaknesses. Deterministic routing approaches are shown to perform well in networks providing accurate link state information[92]. However, this approach is inherently insensitive to dynamic network changes, and therefore its performance degrades in scenarios where link state information becomes inaccurate[100, 92, 98, 101]. As a consequence, hypothetical application on the context of geographically distant hosts communicating through the Internet would probably result in communication processes offering unpredictable QoS service levels.

Randomized routing algorithms do not suffer insensibility problems described for the deterministic ones, and are shown to perform well in presence of inaccurate link-state information[72, 73]. However, as for others approaches, routing approaches imply its architecture to be extensively installed on routers and/or hops on the source-destination path. Therefore it might not suit best-effort Internet-scale environments which suffer of frequent topological changes.

2.3.2 Negotiation

The lack of possibility to negotiate the QoS level with the user is another major limitation of the vast majority of the approaches discussed in section 2.2. In chapter 1, in fact, we advocate the need of negotiation services in the context

of provision of e-services, where services to be provided are dynamic in nature and provision consequently involves runtime negotiation of newly aggregated services. Majority of systems and protocols described in section 2.2 provide either of the QoS attributes under consideration with the inherent assumption that the service level obtained is always the maximum possible. Hypothetical application of such protocols in the context of e-services under consideration would result in heavy scalability limitations in terms of possible concurrent operations, as the service provided would soon start decreasing in quality.

Wherever handled, as in RNAP[110], negotiation is intended as a network price billing followed by resource reservation. As such, negotiation takes place between the entity providing the network service and the single customer who chooses the service based on the price he is willing to pay. This type of negotiation, typical of unicast communication schemes, is followed by a resource reservation in the path provider-customer, and suffers the same (scalability) problems already mentioned for the approaches based on resource reservation. In addition, application of the approach proposed by RNAP on a network of hosts communicating through the internet through the decentralized architecture approach (RNAP-D) assumes network support by requiring the protocol to be installed on all routers on the path and this assumption, as already mentioned, cannot hold on an Internet context.

2.3.3 Our approach: reliable and timely multicast

The QoS-Supportive Reliable Multicast System subject of this thesis is radically different from all systems/protocols described in previous sections. Its key features, in term of provision of QoS guarantees, can be enumerated in two points:

(i) Provision of reliability *and* timeliness QoS attributes at the design level, and

(ii) Provision for QoS negotiation.

As a consequence of (i), the protocol provides combined service guarantees on *both* attributes. As a consequence of (ii), the protocol allows such guarantees to be negotiated with the user anticipately to service provision. This latter possibility, in particular, gives the application user the freedom to consider the service level as guaranteed with respect of the communication. The two features above make, to the best of our knowledge, our system unique in the panorama of multicast protocols, as *all* systems/protocols previously described are designed so as to provide either reliability *or* timeliness in the multicast service.

2.4 Concluding Remarks

We have studied provision of QoS in point-to-point multicast communication, putting emphasis on the choice of reliability and timeliness as fundamental

attributes.

Reliability can be traditionally achieved by means of deterministic or probabilistic approaches. Former approaches are driven by ACK/NAK-based techniques on hierarchically structured receivers, while protocols using the latter approach use gossip-based techniques on all or factions of the group.

Resource reservation and dynamic routing are the techniques traditionally employed to achieve timeliness. The former aims to reserve resources anticipately to service provision, while the latter implies (deterministic or probabilistic) routing techniques to find the optimal path towards destinations based on specific QoS parameters.

We have discussed how the *non-functional* nature of QoS does not allow it to be integrated at a second time, and identified the problems of adding support for timeliness in systems/protocols offering reliability as caused by the "passiveness" of their respective recovery mechanisms. Therefore, we advocated the use of *proactive recoverability* as a fundamental concept to support timeliness in multicast communication.

In protocols providing timeliness, we identified the problem as the support for the multicast scenario. We discussed how approaches based on resource reservation suffer the multiplicity of states that need to be stored. On the other hand, protocols employing dynamic routing to provide timeliness guarantees are shown to be ineffective in presence of inaccurate link state and are, consequently, insensitive to dynamic network changes. Besides, both approaches need to be self-supported by installation of own architecture on the whole net-

work.

We also showed how QoS negotiation, whose need we advocated in chapter 1, is very limitedly supported in systems/protocols discussed and, whenever supported, in protocols offering timeliness, suffers scalability problems due to the lack of support for the multicast context.

Finally, we claimed the system subject of this thesis as unique in the panorama of multicast systems/protocols. The assertion derives from the consideration that, to the best of our knowledge, our system is the only system capable of providing negotiable reliability *and* timeliness guarantees in the multicast process.

Chapter 3

System Model and Architecture

3.1 Introduction

Systems and protocols described in the previous chapter assume either the synchronous or asynchronous system model as a conceptual basis. Both models have proven to be effective in providing guarantees in terms of eventual delivery on cross-Internet communications,. However, timeliness guarantees in cross-Internet communications have probabilistic, rather than deterministic, nature and therefore both models become inappropriate. In fact, QoS-supportive communication on a cross-Internet environment implies acommunication delays to be essentially *finite* and *known*.

The synchronous model allows delays to be known, but it does not assume they are finite. On the other hand, the asynchronous model assumes delays to be finite but no assumption on the bound can be made, i.e. they are assumed not to be known.

Today's Internet communication are characterized by fluctuations deriving from the heterogeneity of infrastructures and traffic between source and des-

tinations. In order to capture this nature, a different conceptual model is needed.

The system subject of this thesis is based on a model, called *Probabilistic Asynchronous*[45] (PA) model, that enhances QoS support by offering a probabilistic approach. Components are assumed to meet their performance requirements most of the time, adhering to the classical asynchronous model only when such requirements are not met.

The PA model combines probabilistic design techniques with asynchronous ones, and characterizes the context in which many practical and Internet-based applications are built by allowing QoS guarantees which are probabilistic in nature.

Use of the PA model is coupled with the use of a *QoS-adaptive middleware architecture*[40] that enhances QoS-sensitive communications by providing a QoS management interface to support the traditional service interface. The QoS management interface exports performance information in terms of metrics such as packet delay, loss and jitter. The idea is that by providing runtime performance information on a determinate subsystem, other system components can retain behavior of that subsystem predictable in the long term.

The RMcast system subject of this dissertation is designed in a modular way based on this architecture. The QoS management interface is implemented by the *Negotiation Component* (NC), which provides QoS negotiation facilities through performance evaluation algorithms. The service interface, on the other hand, is implemented by the *RMcast Component* (RMC) through the

use of a fault-tolerant protocol providing a configurable reliable multicast[58] service. Furthermore, the middleware system is designed for the use on top of a *Communication Subsystem* (CS) offering basic communication primitives. As a basic resourceful subsystem, this latter does not usually export a QoS management interface, which contradicts with requirements of the QoS-adaptive middleware architecture to be able to assess resources of low-level subsystems. To this extent, the RMcast system features a *Network Measurement Component* (NMC) capable of assessing network metrics, on a statistical basis, in order to describe behavior of the CS.

3.2 System model

As mentioned in the previous section, all systems and protocols described in chapter 2 are designed based on the synchronous or asynchronous model, or variations of them.

The synchronous model assumes processing and communication delays to be known but not necessarily finite, while the asynchronous model assumes delays to be finite but no assumption on the ability to deduce delay bounds and distribution, regardless of their accuracy, can be made. Systems designed on the synchronous model require an accurate provisioning of system resources, together with a complete knowledge of the user environment, to provide QoS guarantees. Therefore, synchronous systems suit only a restricted set of applications. On the other hand, systems basing their design on the asynchronous

model can provide eventual correctness only, and QoS issues are left as second thoughts. However, experience has shown that QoS provisioning can only be achieved by considering it a core objective in the design phase.

Today's Internet is composed by infrastructures which are heterogeneous in bandwidth availability and data-management policies. As such, performance is subject to fluctuations which do not allow to make predictions in the long term, with the obvious consequence that QoS provision becomes a complex task.

The RMcast system subject of this thesis uses a different conceptual system model, namely the *Probabilistic Asynchronous* (PA) model[45]. The model characterizes the behaviour of *Communication Subsystem* (CS), which manages the capacity to communicate information. This subsystem is denoted as S_R in Figure 3.2. This model regards that system components do meet their performance requirements most of the time, and occasionally they may not. Objective of the PA model is to allow systems to adaptively meet QoS obligations to the end users when system components meet their QoS guarantees or violate them only marginally; eventual correctness is never compromised when components fail in their QoS obligations.

The system is made up of nodes that communicate using the CS. A node or any process hosted within it functions correctly until and unless it crashes. A node (or a process) that does not crash is said to be correct. To present the probabilistic model, we will assume a global clock which is not accessible to processes.

- *Transmission Delays*: If a correct process i sends a message m to another correct process j at time t, then
  - m is delivered to j (i.e., m arrives at the buffer of j) with some probability $1 - q$ (m may be lost in transmission with probability q).
  - if m is not lost, it is delivered at $t + \delta$ where δ is a random variable with some known distribution.

If the distribution of δ is uniform with some known mean and $q = 0$, then the probabilistic model refers to the well-known synchronous model which permits upper bounds on δ to be determined with certainty. The asynchronous model considers the bounds on the delay δ to be finite; neither the bounds nor the delay distributions can be known with certainty. For example, any bound on delays, however judiciously deduced, is vulnerable to being violated with unknown probabilities. The probabilistic model, on the other hand, assigns probabilities or coverage to quantification of delay bounds. Figure 3.1 shows a table comparing the PA model with the classical synchronous and asynchronous models.

Note that the synchronous model is subsumed in, or is a special case of, the PA model. This means that any PA protocol should run correctly (not necessarily efficiently) in a synchronous system. Conversely, if a problem is unsolvable in a synchronous system, then it cannot be solved in the PA model.

Models Parameters	Synchronous	Asynchronous	Probabilistic Asynchronous
Bound on successive transmission losses, k	known	finite and known	random variable on $[0, \infty]$
End-to-end delay for a "sent" message, δ	has a known bound	has a finite and unknown bound	random variable on $[0, \infty]$ with known distribution, if message not lost

Figure 3.1: Comparison of models

3.3 QoS-Adaptive Middleware Architecture

For a QoS adaptive system to be feasible, resourceful subsystems, indicated as S_R in figure 3.2 must export a QoS management interface in addition to the traditional service interface. Using this interface, the middleware system S_M can request S_R whether a specified distribution for the delay variable δ and a specified loss probability (q) can be supported; this in turn would help determine whether a given set of requirements on bandwidth capacities additionally needed to support an end user requirement can be met. If the request for a specified distribution for the delay variable cannot be supported, S_R may respond with the delay distributions which it can currently support.

The middleware system S_M will have two components: a service component ($service_M$) and QoS management component (qos_M):

- $service_M$ implements a specified service tolerating at most φ node crashes;
- qos_M evaluates the delay distributions of $service_M$ as a function of such distributions offered by $service_R$. qos_M will also take into account the

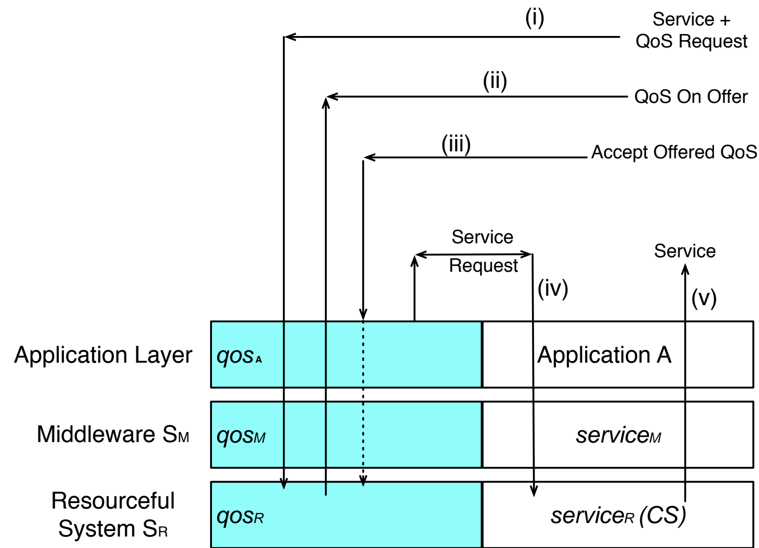

Figure 3.2: QoS-Adaptive middleware architecture.

overhead that $service_M$ would incur given the size of input from the higher level.

At the top of the stack are the application (A) and its QoS manager (qos_A). When a user submits a request with the required (probabilistic) delay and throughput guarantees (interaction (i) in Figure 3.2), the application QoS manager qos_A computes and passes down the QoS guarantees expected of S_M to qos_M. The QoS feasibility evaluation thus travels down to S_R which computes if it can maintain the necessary mean and the variance of delay distributions for the overall resource requirement. If it is possible, then the user request will be accepted; else, S_R returns the mean and variance it can sustain and the reverse computations are made by qos_M upwards (interaction (ii) in the figure). The user is then informed of the QoS guarantees the system

can offer.

Suppose that a user request is accepted for a set of given QoS metrics. qos_M records the QoS requirements that $service_M$ needs to meet (interaction (iii) in Figure 3.2). The service request is submitted to the application whose execution invokes both $service_M$ and $service_R$ (interaction (iv) in Figure 3.2). The (fault-tolerant) protocol that implements $service_M$ is designed with configurable parameters; the choice of these parameter values will influence the protocol behaviour and thus the QoS offered by $service_M$. These parameters can be regarded as *QoS control knobs*, and, in what follows, we will term $service_M$ as *(QoS) controllable protocol based* $service_M$, or simply as CPS_M. The responsibility for setting appropriate parameters is upon the QoS management component, qos_M, so that CPS_M (or $service_M$ in Figure 3.2) can meet its QoS obligations in providing its services to application A. This parameter setting and the feasibility analyses carried out prior to accepting the user request will require that qos_M be equipped with algorithms to evaluate the performance of CPS_M in terms of these parameters. Specifically, qos_M should be able to evaluate the QoS metrics offered by CPS_M for a given set of parameter values (e.g., latency for a given level of redundancy) and vice versa, and also derive the parameter values from the QoS guarantees from $service_R$ below (e.g., the level of redundancy for a given loss probability) and vice versa. The module which contains these evaluation algorithms is called the (QoS) *Negotiation* module and offers a set of services called the (QoS) *Negotiation Services.* Developing algorithms for (QoS) Negotiation module involves stochastic mod-

elling and performance evaluation. Tractable performance analyses generally warrant approximations to be taken and in the system subject of this document such approximations tend to underestimate the actual performance. This means that $service_M$ tends to perform better than predicted by the Negotiation module, offering a better QoS to application A than promised. The overall system thus has an inherent tendency not to fail on the end-to-end QoS promised to the application.

When the CS is managed by an ISP, it is possible that the QoS guarantees agreed by the ISP are violated for a prolonged period. These violations can lead to the middleware system being unable to meet QoS obligations at run time. So, a requirement for qos_M is to monitor the QoS offered by $service_R$ to CPS_M, and attempt to re-adapt the protocol of CPS_M so that CPS_M still maintains its QoS guarantees to application A. The monitoring and reporting activities are carried out by the QoS Monitoring module within qos_M, and its services are collectively called the *Monitoring Service*.

3.4 QoS-Supportive Reliable Multicast System Architecture

The RMcast system lies on top of the network and the kernel is below the applications to be hosted. As per the ISO OSI hierarchy, it is at level 5 (*session*) with network (layer 4) providing a basic (unicast) communication support as

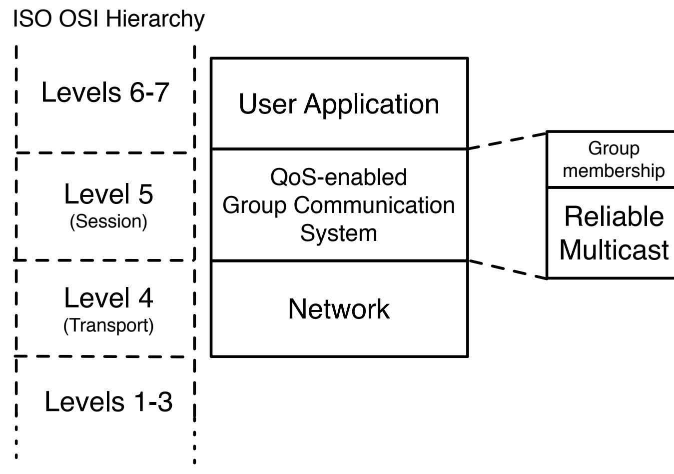

Figure 3.3: Position in the ISO OSI hierarchy.

can be seen in Figure 3.3. Referring to the figure, group membership service provides a realistic view to the application as to which processes are deemed to have crashed. For this view to be consistent, it needs to reach agreement with processes regarded to be operative. Usually this service is provided by a group management protocol. Appendix A describes and studies an example group management protocol, which makes use of gossip-based techniques, that might be used in conjunction with the RMcast system to form a GC system.

Architecture of the QoS-Supportive RMcast system is shown in figure 3.4. The *RMcast Component* (RMC) contains a fault tolerant protocol which offers the reliable multicast service. Reliability guarantees are “tailored” to specific QoS requirements by means of configuration parameters. These influence behavior of the protocol by allowing the protocol to adapt to fulfillment of those QoS requirements.

Configuration parameters are generated by the *Negotiation Component* (NC)

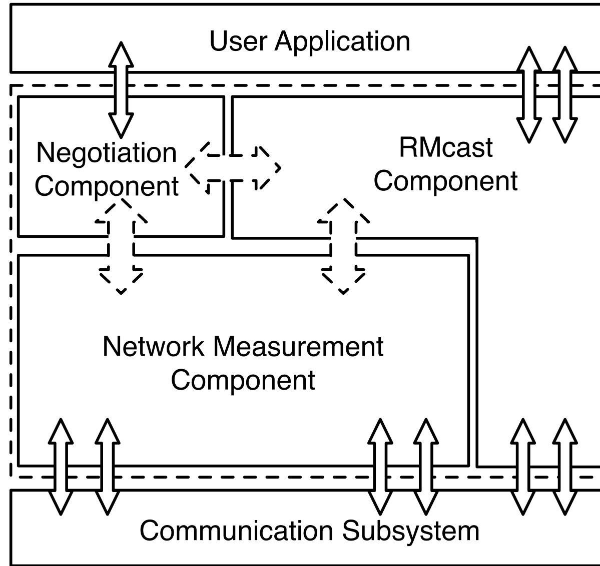

Figure 3.4: System architecture.

in the process of evaluating feasibility of QoS requirements by the user, as part of the QoS negotiation. In addition, the *Network Measurement Component* (NMC) provides an interface to monitor behavior of the CS. Monitoring data, which has statistical nature and is calculated based on a configurable slot of time, allows components of the system to retain behavior of the CS predictable in that interval. This, in turn, allows the NC to base evaluation of the stochastic model on such data, and the RMC to adapt behavior of the RMcast protocol accordingly

Figure 3.4 shows composition of the system. The Network Measurement Component gather network performance figures in terms of metrics which are made statistical over an interval time. This latter is initially specified in advance and configurable based on stability of the network.

Data so obtained is then exchanged among other system components (i.e.

Negotiation Component and RMCast Component) and used in the QoS negotiation process and in the RMcast protocol.

The RMcast Component is in charge applying the RMcast protocol to the data the user desires to multicast. Communication towards the user application is realized through an interface exporting sophisticated communication primitives. The application user can utilize these primitives to gain access the RMcast service.

The Negotiation Component is in charge of allowing the application user to negotiate QoS requirements with the system. To this end, it provides a negotiation interface containing negotiation primitives. QoS negotiation implies, on the system side, knowledge of the QoS the CS can support. This information is obtained by evaluating a stochastic model of the RMcast protocol the Negotiation Component contains. The stochastic model is intended to be based on current network conditions. This latter information is obtained through statistical network metrics by coordinating with the Network Measurement Component. If a QoS negotiation is retained to be successful, evaluation of the stochastic model allows generation of parameters which allow the RMcast protocol, in the RMcast Component, to adapt its behavior to fulfill the agreed QoS requireements. Therefore, if a negotiation is successful the Negotiation Component shares such parameters with the RMcast Component to the extent of providing initial setup for the RMcast protocol there contained.

3.4.1 Network Measurement Component

The Network Measurement Component performs a monitoring activity of network behavior, aimed to estimate performance by calculating certain network metrics on a statistical basis. Monitoring is performed with a technique based on sampling of the CS at regular intervals to infer information about its behavior, and processing results in a statistical fashion to calculate metrics of interest. Finally, the Network Measurement Component notifies other components with corresponding relevant metrics as figure 3.5 shows.

The Network Measurement Component represents the QoS management interface for the CS. Its presence allows to retain behavior of the CS predictable in the long term by calculating statistical network metrics over specific intervals of time, and is fundamental to the RMcast system regardless the of the way the CS provides communication.

When the CS is managed by an ISP using resource management models for the Internet[47] a *Service Level Agreement* (SLA) specifies commitment of the ISP to provide a compliant service level throughout the entire provision time. Service level is thereby specified in probabilistic terms, and network behavior is guaranteed to comply to certain specifications agreed upon anticipately to service provision. However, figures in the SLA do not reflect network fluctuations and cannot be taken as values describing network performance in a determinate moment. Therefore, even in case of an ISP managing the underlying CS, the need for measurement of network performance on a run-time

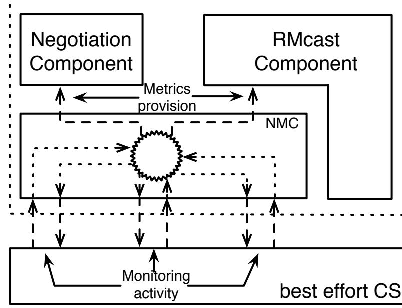

Figure 3.5: Network Measurement on a best effort CS.

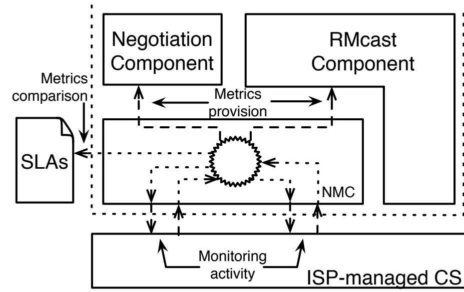

Figure 3.6: Network Measurement Component on an ISP-managed CS.

basis still holds. In addition, the ISP does not usually provide tools to monitor correct provision, and therefore the monitoring activity brings the added value of proofing that the service is provided within the terms agreed, while providing other components with accurate network metrics. To these extents, data is monitored as in the case of the best effort CS described and, besides updating other components, metrics are also matched with performance guarantees specified in the SLA, as shown in figure 3.6. If comparison terminates with figures calculated to fall in the range of admissible service level, than the service provision is taking place correctly. Otherwise, if they fall in an inadmissible range or contrasts with SLAs in any other way, then the ISP is violating the agreement. Violation is therefore reported, whereas a violation detection system is foreseen, or the user is simply notified.

3.4.2 Negotiation Component

The Negotiation Component contains functionalities to allow negotiation of specific QoS levels with the user application. Negotiation is based on algo-

rithms capable of carrying out feasibility analyses for specific service levels. These latter analyses aim to evaluate an analytical model of the (reliable multicast) service offered by the RMcast Component, in order to pursue a decision on whether the service level requested can be supported or not. Evaluation of the analytical model approximates reliability and timeliness properties of the protocol and takes into account current environmental conditions (such as group size and network performance and conditions).

The Negotiation Component obtains an assessment of the QoS currently supported by the CS, in the form of statistical metrics, from the Network Measurement Component. Such an assessment is updated on a regular basis in order to reflect network behavior, and the information is used in the process of evaluation of the analytical model.

When a negotiation is retained successful, i.e. the system decides the QoS level requested is achievable, evaluation of the stochastic model produces parameters to allow configuration of the reliable multicast protocol in the RMcast Component to achieve the QoS level requested. Setting of these constitutes the configuration phase of the RMcast Component.

Interactions just described are depicted in Figure 3.7, where the Negotiation Component sustains a successful negotiation, also by using metrics provided by the Network Measurement Component, and provides the RMcast Component with parameters. The circle in the middle of the component represents the negotiation process, where data provided by the Network Measurement Component is used in order to evaluate the analytical model and compare results

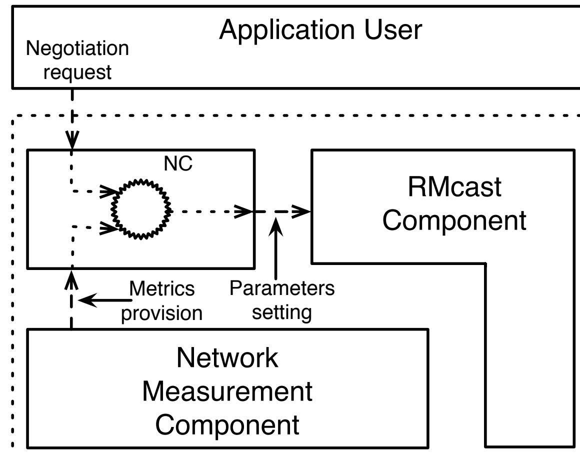

Figure 3.7: Negotiation Component in presence of a best effort CS.

with user requests.

3.4.3 RMcast Component

The RMcast Component contains the service offered by the system, provided through a fault-tolerant protocol. Its design is based on redundant transmissions of data to be carried out at fixed intervals, and its structure is described in detail in th next chapter.

Level of redundancy and interval time between subsequent retransmissions influence the behavior of the protocol, and are specified *a priori* as protocol configuration parameters. Responsibility of the setting of these is upon the Negotiation Component. Figure 3.8 shows the interaction model for the RMcast Component. Referring to this figure, setting of parameters represents the configuration phase, and is the first step of the initialization phase. Other steps in this latter phase involve creation of a communication down-stream (towards

the CS) and a communication up-stream (towards the user). Both streams are bidirectional, and allow the system to handle incoming and outgoing traffic. The first allows the system to access the basic CS communication primitives by means of which incoming data will be received and outgoing data will be transmitted. The second, on the other hand, will provide a communication channel with the user, where outgoing data will be received and incoming data will be delivered. In between these two streams, the RMcast protocol will apply its logic to both types of traffic in such a way to achieve the agreed QoS guarantees.

The protocol is adaptive in a way that its behavior accounts for current network performance. However, adaptation is based on knowledge of current network conditions. In detail, the *proactive recovery* mechanism is based on timeouts whose length accounts for current network conditions. Detection of the network changing its behavior consistently will be reflected in provision of up-to-date network metrics, and the protocol will automatically adapt timeouts so as to account for the new conditions by integrating updated network metrics. To this extent, the RMcast Component coordinates with the Network Measurement Component in order to obtain timely updated on network metrics.

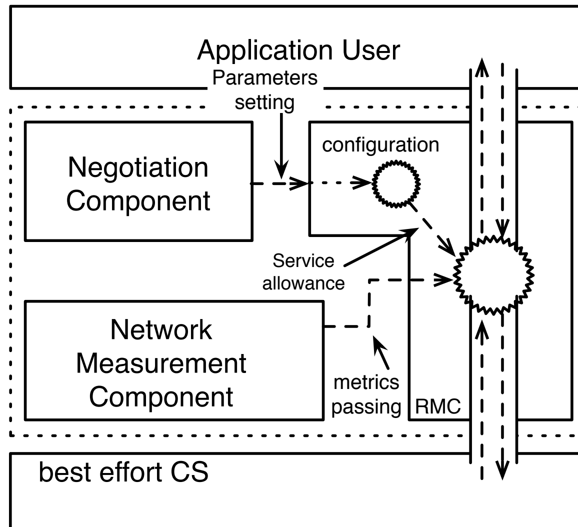

Figure 3.8: RMcast Component in presence of a best effort network.

3.4.4 Sequential Interaction Model

The lifecycle of the system components is dominated by three principal operations, encapsulated into phases in figure 3.9:

- Phase 1 (Initialization). Starts when execution of the system starts, and terminates when the system is ready to accept negotiation requests. Components involved in this phase are essentially the ones performing background operations which do not need to interact with the user application. The Network Measurement Component and the Negotiation Component are instantiated. The former starts measuring network performances, while the latter idles for the user application to request a negotiation while obtaining up-to-date network metrics from the Network Measurement Component. This phase is depicted in the *PHASE1* diagram of figure 3.9.

- Phase 2 (Negotiation). Sometimes after termination of the initialization phase, the user application requests a negotiation process. The system then enters the negotiation phase, as shown in *PHASE2* diagram of figure 3.9.

  Negotiation is requested by asking for a multicast operation to be completed within specified reliability and timeliness performance bounds. Invocation of the negotiation primitive conveys this information to the Negotiation Component.

  The analytical model is then evaluated and a decision on whether to accept the request or not is taken based on the comparison between values generated in the evaluation and user application ones.

  Successful negotiations generate parameters for configuration of the RMcast protocol. Such parameters are therefore used to setup the RMcast Component for the service level to be achieved ((2a) in the *PHASE2* diagram of figure 3.9). In case the negotiation fails, i.e. the system cannot support the requested service level, the user is notified ((2b) in the *PHASE2* diagram of figure 3.9), eventually providing the service level it can actually sustain.

- Phase 3: (Service Provision). Configuration of the RMcast Component for achievement of the agreed service level starts this phase. Configuration also involves setup of the inter-component communication with the Network Measurement Component to the extent of obtaining up-to-date

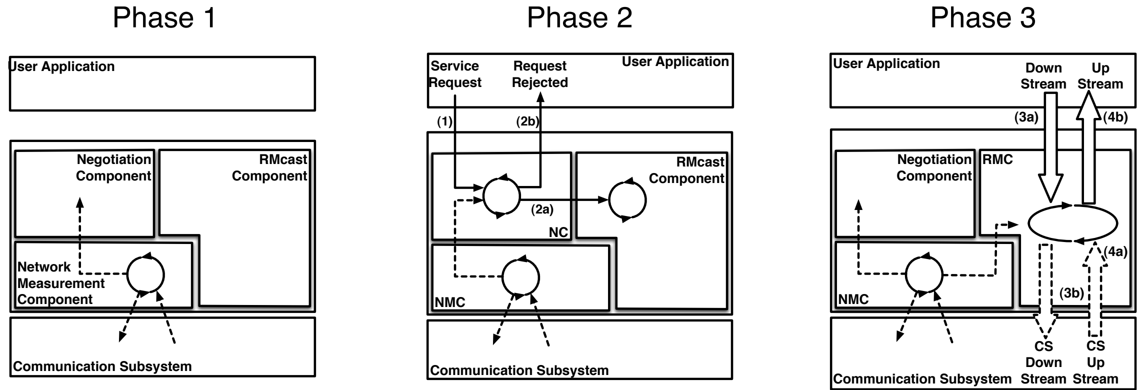

Figure 3.9: Sequential system logic base on main tasks: network monitoring (phase 1), negotiation (phase 2) and service provision (phase 3)

network metrics of interest. The next initialization step is to create appropriate streams for outgoing and incoming data flow ((3a) and (4a) in the *PHASE3* diagram of figure 3.9), which will be handled by the RMcast protocol based on its logic and delivered (incoming data, (3b) in the *PHASE3* diagram of figure 3.9) or transmitted (outgoing data, (4b) in the *PHASE3* diagram of figure 3.9). The basic CS communication primitives are used by the RMcast Component based on the logic contained in its protocol in order to be reliably multicast.

3.5 Concluding Remarks

In this chapter we have described the architecture of the RMcast system. We have shown how the use of a QoS-Adaptive Middleware Architecture provides support for negotiable QoS guarantees by combining the use of the traditional

service interface with a QoS management interface. We have described how in our system the former, which we named RMcast Component, offers a reliable multicast service through a fault-tolerant protocol, and the latter, named Negotiation Component, provides QoS negotiation and evaluation analyses through anticipate evaluation of an analytical model of the service protocol.

We have also showed how both components rely on up-to-date information about current network conditions, putting emphasis how particularly important this is to bring adaptation in the RMcast protocol. To this extent, we described the interaction model of each component with the Network Measurement Component, which monitors the CS on a constant basis to the end of producing statistical metrics which describe the network behavior.

Finally, we have discussed the sequential interaction model that allows the system to initialize, negotiate and provide the reliable multicast service.

Chapter 4

Single Packet Protocol

4.1 Introduction

The system offers a multicast service whose reliability level can be negotiated before start of the service provision. Its core is constituted by an RMcast protocol providing probabilistic guarantees on multicast delivery. Guarantees are provided in terms of reliability, by provision of agreement, and timeliness, by ensuring respect of specified latency delay bounds.

Reliability and timeliness guarantees are agreed through a negotiation which takes place before start of service provision. Negotiation involves evaluation of an analytical model capable of approximating behavior of the protocol based on current network conditions.

Accuracy of approximations and additional overhead, in terms of message traffic, are discussed by comparing the approximated stochastic model with results obtained by simulating the protocol on an environment identical to the one assumed in approximations.

4.2 System context

We consider a system of n, $n > 1$, distributed nodes that communicate using an ISP supported communication subsystem (CS). Each node hosts a distinct process p_i, $0 \leq i \leq n-1$. These processes cooperate with each other as a group G for supporting some distributed application. The group size is known to members. Without loss of generality, the numbering of processes is assumed to imply a 'seniority' ordering: process p_i is said to be 'more senior' than process p_j if $i < j$. A node (or the process hosted on it) functions correctly until and unless it crashes (i.e., ceases to be operative). A process that does not crash until the group G needs to exist, is said to be *correct*. Each process has a primitive *send*(m) using which it can send a message m to another process. The *send*(m) is said to be *successful* if m is deposited in the receive buffer of the destination process. The message m is assumed to have standard size, i.e. we assume that the message does not need to be fragmented before being multicast.

We assume that the processes of G are over-provisioned with computational capacity. That is, queueing delays, processing delays, and scheduling delays can be assumed to be negligibly small compared to network delays. This means that a process can instantaneously receive a message which the CS deposits into its receive buffer, and the inter-process communication delay will be the message transmission delay over the network. The CS ensures that when an operative process invokes *send*(m) to send m to another operative process,

then

- the *send*(m) operation is successful with a known probability $1 - q$, i.e. m is lost with probability q; and,

- if *send*(m) is successful, the network transmission delay of m is an independent random variable with some known distribution.

4.3 Design of the Reliable Multicast Protocol

The protocol exports two primitives: $RMcast(m)$ and $RMdeliver()$. When a process wishes to multicast a message reliably to processes in G, it invokes the operation $RMcast(m)$. This process will be called the *originator* of m. A message m sent by an invocation of $RMcast(m)$ is delivered to a destination process by $RMdeliver()$. The protocol is designed with configurable parameters using which QoS offered can be set to the desired level. The QoS guarantees are probabilistic in nature and fall into two broad categories: *reliability* and *latency*.

Invocation of the $RMcast(m)$ primitive is subject to a prior successful negotiation. In fact, the Negotiation Component is responsible for the generation, in the negotiation process, of configuration parameters to allow the RMcast protocol to adapt so as to achieve the negotiated QoS level. A negotiation will then be necessary each time the $RMcast(m)$ will be invoked. This need will be relaxted in the next chapter, where approaches to provision of likewise guarantees on a set of packets, rather than a single message, will be introduced

and studied.

4.3.1 Specification of Protocol Guarantees.

The protocol offers probabilistic guarantees concerning reliability.

- **Validity**. If the originator of m does not crash until its invocation of $RMcast(m)$ completes, then all correct destinations deliver m with a probability which can be made arbitrarily close to 1.
- **Agreement** or **Unanimity**. The probability that if a multicast message is delivered to a process, it is delivered to all correct destinations, is very high and can be evaluated in advance.

Note that the *agreement* guarantee actually refers to what is known as the *uniform agreement* property [59]: even if a process crashes shortly after delivering m, then all correct destinations are guaranteed to deliver m with a high probability. This means that if a crashed process has invoked $RMcast(m')$ soon after delivering m, then any correct process that delivers m' is guaranteed to deliver m with a high probability.

The protocol offers the following guarantees on latency.

1. The interval between an originator invoking $RMcast(m)$ and the first instant thereafter when all correct destination processes have delivered m, does not exceed a given bound, D, with a probability, r_D, which can be evaluated in advance.

2. If, following an invocation of $RMcast(m)$, the message arrives at a process, then it will be delivered at all correct processes within a further interval of a given length, S, with a probability, u_S, which can be evaluated in advance.

These properties will be referred to as *latency bound* and *relative latency bound*, respectively (the latter is also known as *tightness* in the literature.). They enable an application developer to reason about timeliness: a process that invoked $RMcast(m)$ at time t may assume that, by time $t + D$, m is delivered to all correct destinations (with probability r_D); a destination process that has delivered m (through $RMdeliver(m)$) at time t may assume that, by time $t + S$, all correct destinations have delivered m (with probability u_S).

4.3.2 Design features

The *RMcast* protocol has three features which are designed to assure high probability of success at tolerable cost in message traffic:

(a) The execution of $RMcast(m)$ comprises more than one invocation of a $broadcast(m)$ operation. Each of these invocations sends the message m to each destination in a concurrent fashion.

(b) The responsibility for invoking $broadcast(m)$ initially rests with the originator of the message, but may devolve to another process, and then to another, in consequence of crashes, message losses or excessive delays.

(c) In the event of such a devolution, a decision procedure attempts to select exactly one process to take over the broadcasting responsibilities.

These features can be described as *Redundancy*, *Responsiveness* and *Selection*, respectively.

4.3.2.1 Redundancy

The redundancy of the protocol is controlled by two parameters:

(i) An integer, ρ, specifies the level of redundancy; the originator of a message makes $\rho + 1$ attempts to broadcast it (if operative); these attempts are numbered $0, 1, \ldots, \rho$; typically, $\rho \geq 1$.

(ii) The interval between consecutive broadcasts is of fixed length, η; that length is chosen to be as small as possible, but sufficiently large to make any dependencies between consecutive broadcasts negligible.

One way of choosing η is to require that the transmission delay between a source and a destination is less than η with a given probability, α (reasonably close to 1). In the case of exponentially distributed delays with mean d, η is given by

$$\eta = -d \log(1 - \alpha) \ .$$

More conservatively, η can be chosen so that it exceeds the *largest* of $n - 1$ parallel transmission delays with probability α. In the exponential case, that choice would imply

$$\eta = -d \log(1 - \alpha^{\frac{1}{n-1}}) \ .$$

4.3.2.2 Responsiveness

If the originator of a message crashes before its broadcast attempts are completed, the destination processes respond by taking over the broadcasting responsibility upon themselves. To facilitate this takeover, each copy of a message, m, has fields $m.copy$, $m.originator$ and $m.broadcaster$; these specify the number of the current broadcast attempt $(0, 1, \ldots, \rho)$, the index of the originating process, and the index of the process that actually broadcast the message m, respectively. The values of $m.originator$ and $m.broadcaster$ will be different if a destination process carries out the broadcasting of m. Every process that receives a message, m, such that $m.copy = k < \rho$, must be prepared to become a broadcaster of m if necessary. It does so by setting a timeout interval of length $\eta+\omega$, with some suitable value of ω (η is the interval between consecutive broadcasts, while ω accounts for differences in transmission delays, or 'jitter'). If copy $k+1$ of m arrives from the broadcaster of copy k before the timeout expires, then all is well with that broadcaster; the receiver process sets a new timeout of $\eta + \omega$ for the next copy (if there is one). Otherwise, the receiver pessimistically assumes that the process $m.broadcaster$ has crashed while broadcasting copy k of m, and that it is the only process to have received any copy of m. It therefore prepares to appoint itself as a broadcaster of copies $k, k + 1, \ldots, \rho$. However, the $m.broadcaster$ may not in fact have crashed; copy $k+1$ of m may just be delayed unduly or lost; moreover, even if $m.broadcaster$ has crashed, this receiver may not be the only process that has

observed the crash. In order to avoid multiple receivers becoming broadcasters unnecessarily, a further random wait, ζ, uniformly distributed on $(0, \eta)$, is added to the timeout interval $\eta + \omega$. If a copy number k or higher is not received before the expiration of ζ, this receiver appoints itself as a broadcaster. Otherwise it sets a new timeout of $\eta + \omega$.

4.3.2.3 Selection

The protocol guards against multiple self-appointed broadcasters. It requires that any broadcaster with index i, whose latest broadcast has been of copy k of the message, should relinquish its broadcasting role in any of the following circumstances:

1. Process i receives m with $m.copy = k$ and either $m.broadcaster < i$ or $m.broadcaster = m.originator$. That is, a more senior process has assumed the duties of broadcaster, or the originator has not in fact crashed.
2. Process i receives m with $m.copy > k$, indicating that it has missed one or more copies of m, and another broadcaster is closer to completing the protocol.

Suppose that process i has abandoned its broadcasting role and has set a timeout expecting a copy, say, k, from broadcaster j. It will have to reset that timeout if either copy k is received later from a broadcaster more senior than j or from the originator, or copy $k + 1$ or higher is received from any broadcaster. This is necessary because when process j receives the message

which process i has just received, it would relinquish its broadcasting role. The purpose of these provisions is to minimise unnecessary broadcasts and hence message traffic, while still making the best effort to ensure that $\rho + 1$ copies of the message are broadcast. The idea is that when any broadcaster crashes, all receivers that time out on $\eta + \omega + \zeta$ will briefly become broadcasters, but after that only one of them is likely to continue broadcasting, at intervals of length η. That process will be a receiver process if the originator has crashed or its messages suffer excessive delays.

4.4 Details of the protocol

A more detailed pseudo-code description of the reliable multicast protocol executed by process i is presented in figures 4.1 and 4.2. An execution of $RMcast(m)$ starts by setting the field $m.originator$, and also a unique message identifier called $m.sequenceNo$; then $(\rho + 1)$ invocations of $broadcast(m)$ are performed, with $m.copy = 0, 1, \ldots, \rho$. The primitive $broadcast(m)$ sets the $m.broadcaster$ field and concurrently sends m to processes in G.

RMcast(m)

(1) $m.originator \leftarrow i$; $m.broadcaster \leftarrow i$; $m.SequenceNo \leftarrow seq_number$;
(2) $m.copy \leftarrow 0$;
(3) **repeat**$(\rho + 1)$ times $\rightarrow$
(4) $\{$*broadcast(m)*; $wait(\eta)$; $m.copy \leftarrow m.copy + 1$;$\}$

Figure 4.1: Pseudo-code for $RMCast(m)$

```
RMdeliver()
  begin
   // message-handling part
   cobegin
(5)     receive(m);
(6)      if new(m) →
(7)       begin
(8)        max_recd_i(m) ← −1;
(9)        leader_i(m) ← −1;
(10)        last_own_bcast_i(m) ← −1;
(11)        deliver(m);
(12)       end

(13)     if (m.copy = ρ) → {max_recd_i(m) ← MAXINT;}

(14)     if(m.copy > max_recd_i(m)) ∨
(15)      (m.copy = max_recd_i(m) ∧
        (m.broadcaster = m.originator ∨
         m.broadcaster < leader_i(m))) →
(16)      begin
(17)       max_recd_i(m) ← m.copy;
(18)       leader_i(m) ← m.broadcaster;
(19)       set timeout for η + ω;
(20)       end
   coend
   cobegin
              // timeout-triggered, timer-driven part
    timeout(m) ∧ (max_recd_i(m) < ρ) ⟶
     begin
(21)       leader_i(m) ← MAXINT;
(22)       wait(ζ) ;
(23)       if ((leader_i(m) = MAXINT) ∧ (max_recd_i(m) < ρ)) →
(24)        {leader_i(m) ← i; create thread Broadcaster(m);}
     end
     coend
  end
```

Figure 4.2: Pseudo-code for $RMdeliver()$

The protocol for delivering a reliable multicast message is $RMdeliver()$, and is structured into two concurrently executed parts. The first part handles a received message and the second part the expiry of timeout $(\eta + \omega)$. Three integer variables are maintained for a received message m distinguished by $m.originator$, $m.sequenceNo$:

```
Broadcaster(m)
  begin
(25) while((max_recd_i(m) < ρ) ∧ (leader_i(m) = i)) do
(26)     m.copy ← max{last_own_bcast_i(m) + 1, max_recd_i(m)};
(27)     broadcast(m);
(28)     max_recd_i(m) ← m.copy;
(29)     last_own_bcast_i(m) ← m.copy;
(30)     wait(η)
(31)  od
(32)  die; // the thread dies.
  end
```

Figure 4.3: Pseudo-code for $Broadcaster(m)$

- $max_recd_i(m)$ has the largest copy number received for m.
- $leader_i(m)$ has the index of the process from which m, with copy $max_recd_i(m)+1$, is expected.
- $last_own_bcast_i(m)$ contains the copy number of m which process i broadcast when it last acted as a self-appointed broadcaster.

A received message calls for one or more of the following three actions:

- New m: The three variables defined earlier are initialized to -1, and m is delivered (lines 6-12).
- $m.copy = \rho$: Blocks any future action, by setting $max_recd_i(m)$ to ∞ (MaxInt) (line 13). Note that a new m can have $m.copy = \rho$ if all earlier copies are lost or excessively delayed.
- Change of $leader_i(m)$: The received m indicates one of the circumstances (described earlier) in which the process i needs to either relinquish its broadcasting role or change the broadcaster from which the next copy

is expected. A new timeout $(\eta + \omega)$ is set after $max_recd_i(m)$ and $leader_i(m)$ are updated (lines 14-20).

When the timeout $(\eta + \omega)$ for m expires, an additional timeout ζ is set, during which a message with appropriate copy number from any broadcaster is admissible. So, $leader_i(m)$ is set to MaxInt (line 21). If no such message is received, process i appoints itself as a broadcaster and sets up a thread $Broadcaster(m)$ (lines 22-24). The thread $Broadcaster(m)$ broadcasts m only if the process i remains to be the broadcaster (i.e., $leader_i(m) = i$) as per selection rule and if $max_recd_i(m) < \rho$; otherwise, it dies (lines 25-32).

4.5 Reliability and Latency Estimations

4.5.1 Reliability

If the originator of a message m does not crash, then the only reason why some correct processes may not receive it, is losses in transmission. Since each transmission is lost with probability q, a given correct process will fail to receive all $\rho+1$ copies of m with probability $q^{\rho+1}$. Hence, the probability that all correct processes receive at least one copy of m, i.e. the reliability of the protocol, r, is given as:

$$r = (1 - q^{\rho+1})^{n-1} . \tag{4.1}$$

Clearly, this probability can be made as close to 1 as desired, by increasing ρ. Of course, the price paid for high reliability is higher message traffic.

When crashes are taken into account, the possibility that the uniform agreement property may be violated, is exacerbated. The following scenario may be realized: the originator crashes in the middle of the first broadcast, after executing only a few *send*(m) commands; one or more of the destinations deliver m and act upon it (e.g., become originators of new message(s), m'), but then they all crash before their timeouts expire and therefore fail to propagate m. In those circumstances, correct processes fail to receive m, while crashed ones had delivered it. Such a scenario may be termed a 'uniform disagreement'. Intuitively, the occurrence of a uniform disagreement is very unlikely, because it involves the conjunction of more than one event, each of which is unlikely. Nevertheless, it may be useful to estimate that small probability in terms of the crash characteristics, as discussed below.

There may be two kinds of crashes. Let v be the probability that a process crashes before its timeout expires. If the time-to-failure (TTF) is distributed exponentially with mean $1/\gamma$, then v is given by

$$v = 1 - e^{-\gamma(\omega+2\eta)} \tag{4.2}$$

(pessimistically, ζ is assumed to take its largest possible value, η). v is typically a small number because γ is small (When mean TTF is very pessimistically taken to be, say, 2.5 hours, $1/\gamma$ is 0.9×10^7 milliseconds.).

Another crash mechanism operates while a process is broadcasting. Let β be the probability that the process crashes just after a given *send*(m) operation,

independently of the others. Then we can write a recurrence relation for the probability, w_n, that a disagreement will occur in a group of size n.

$$w_n = \sum_{k=1}^{n-1} \left[(1-\beta)^{k-1}\beta \sum_{j=1}^{k} \begin{pmatrix} k \\ j \end{pmatrix} (1-q)^j q^{k-j} [v^j + j v^{j-1}(1-v) w_{n-j}] \right] . \quad (4.3)$$

This relation quantifies the probabilities in the scenario outlined above: the originator crashes during the first broadcast, having executed k *send*(m) operations; j of those messages are received at their destinations; then, either all j destination processes crash before their timeout expires, or one survives, becomes a broadcaster, but a disagreement occurs within the new group of size $n-j$ (the probability that more than one survive to become broadcasters, and still a disagreement occurs, is considered negligible).

The initial condition for the recurrences (4.3) is $w_2 = 0$, since a disagreement cannot occur with less than 3 nodes. When v and β are both small, the right hand side of (4.3) is on the order of $v\beta(1-q)$.

As an example, Figure 4.4 shows the protocol's probability of failure for $\beta = 0.02$ (probability to fail inside each broadcast to 2%), allowing a failure every 100 hours on average ($\gamma = 100$ hours) and a timeout of $\omega + 2\eta = 5600$ millis. In this case, probability that a process crashes before its timeout expires is $v = 1.55 \times 10^{-5}$ and the overall probability of failure, increasing with the group size, seems to stabilize around a scale of 10^{-7}.

The RMcast protocol can be transformed to a uniform RMcast, by forcing a process to deliver m after $(\omega + 2\eta)$ time following the first reception of m. Then

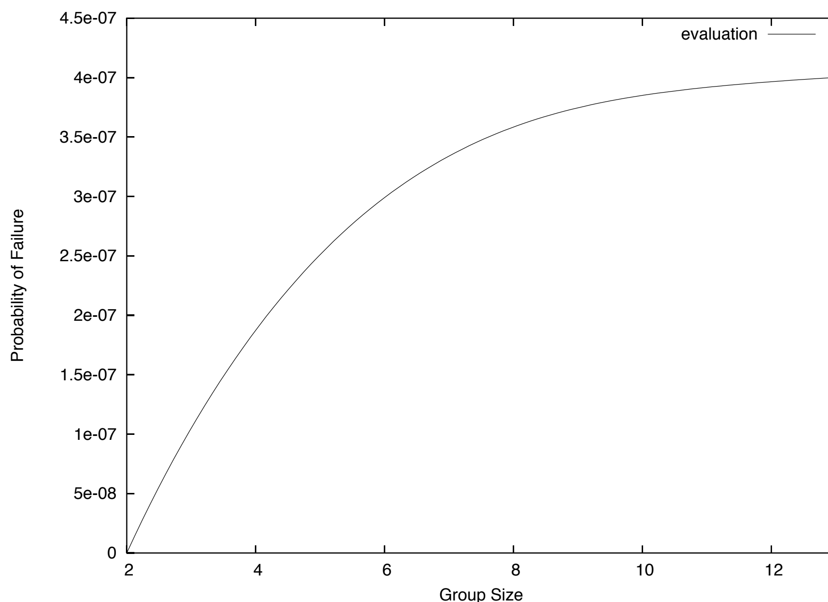

Figure 4.4: Protocol's failure probability, for $\beta = 0.02$, $\gamma = 100$ hours

a uniform disagreement can be caused only by events (such as m being lost to all correct processes even when $broadcast(m)$ is completed) whose probability is negligible.

4.5.2 Latency Estimations

The probability, r_D, that all operative destinations receive at least one copy of a multicast message within a given interval of time, D, can be approximated by *assuming* that the originating process does not crash during the interval of $(\rho\eta)$ time. This is a reasonable approximation for two reasons. First, the probability of a crash during $\rho\eta$ is $(1 - e^{-\gamma\rho\eta})$ which is small as γ is small. Secondly, it

will generally be a pessimistic approximation, since if the originator crashes at some point after broadcast 0 but before broadcast ρ, some of the processes that receive the last broadcast copy will make at least one broadcast themselves. Thus, the number of senders and hence the probability of success will increase. Of course it is possible that the originator crashes during broadcast 0, and no operative process receives any message; we consider the probability of that event to be negligible. Let ξ be the random variable representing the execution time of a *send*(m) operation, i.e., the transmission time of a message from a given source to a given destination. The probability, $h(x)$, that such an operation *does not* succeed within time x, is equal to

$$h(x) = q + (1 - q)\mathcal{P}(\xi > x) \ , \tag{4.4}$$

where q is the probability that the message is lost. By definition, $h(x) = 1$ if $x \leq 0$. In the case of exponentially distributed transmission times (with mean d), the above expression becomes

$$h(x) = q + (1 - q)e^{-x/d} \ , \tag{4.5}$$

and $h(x) = 1$ for $x \leq 0$. Since the originator makes its kth broadcast at time $k\eta$ ($k = 0, 1, \ldots, \rho$), the probability, g_D, that *a given destination* does not

receive any of the $\rho + 1$ copies within time D, is given by

$$g_D = \prod_{k=0}^{\rho} h(D - k\eta) \; . \tag{4.6}$$

Hence, the probability, r_D, that every destination receives at least one copy of the message within an interval of length D is equal to

$$r_D = (1 - g_D)^{n-1} \; . \tag{4.7}$$

If some of the destinations have crashed, then (4.7) is an underestimate of the probability that all *operative* destinations receive at least one copy within time D. This is so because the term $(1 - g_D)$ would then be raised to a lower power, which would make the resulting probability larger. A user requirement, stated in terms of a success probability R and latency D, is achievable if the probability evaluated by (4.7) satisfies $r_D \geq R$; otherwise it is not achievable.

4.5.3 Relative Latency

Suppose now that at a given moment, t, a given process, p_i (different from the originator), receives copy number k of the message. Of interest is the probability, $u_k(S)$, that all other processes will receive at least one copy of the message with relative latency S, i.e., before time $t + S$.

The implication of p_i receiving copy number k is that the originator has started broadcasting no later than at time $t - k\eta$ in the past, and has issued at least

k broadcasts. Consider a given process, p_j, different from the originator and from p_i. The probability, $g_k(S)$, that p_j will not receive any of those $k+1$ copies before time $t+S$ is no greater than

$$g_k(S) = \prod_{m=0}^{k} h(S + m\eta) \ , \tag{4.8}$$

where $h(x)$ is given by (4.4). In addition, if $k < \rho$, p_j may receive copies $k, k+1, \ldots, \rho$ from p_i, in the event of the originator crashing. Those latter broadcasts would be issued at times $t+\eta+\omega+\zeta$, $t+2\eta+\omega+\zeta$, ..., $t+(\rho-k+1)\eta+\omega+\zeta$, assuming that no other process starts broadcasting. Since ζ is uniformly distributed on $(0,\eta)$, we can pessimistically replace ζ by η. The probability, $\tilde{g}_k(S)$, that p_j will not receive any of the messages from p_i before time $t+S$ is thus approximated by

$$\tilde{g}_k(S) = \prod_{m=1}^{\rho-k+1} h(S - (m+1)\eta - \omega) \ , \tag{4.9}$$

where $\tilde{g}_\rho(S) = 1$ by definition; also, $h(x) = 1$ if $x \leq 0$. Thus, a pessimistic estimate for the conditional probability, $u_k(S)$, that all other processes will receive at least one copy of the message with relative latency S, given that a given process has received copy number k, is given by

$$u_k(S) = [1 - g_k(S)\tilde{g}_k(S)]^{n-2} \ . \tag{4.10}$$

A pessimistic estimate for the conditional probability, u_S, that all other

processes will receive at least one copy of the message with relative latency S, given that a given process has received any copy, is obtained by taking the smallest of the above probabilities:

$$u_S = \min[u_0(S), u_1(S), \ldots, u_\rho(S)] \; . \tag{4.11}$$

This quantity may be used in deciding whether a user requirement, stated in terms of a success probability U and relative latency S, is achievable or not: the requirement is achievable if $u_S \geq U$. Intuitively, one would expect the minimum in the right-hand side of (4.11) to occur for $k = 0$, so that $u_S = u_0(S)$. Indeed, this has been the case in all examples evaluated.

4.6 Simulation Results

Performance of the protocol is simulated for a variety of parameter values, and the results are compared with results obtained by evaluating the analytical approximations described in the previous section. Each experiment consists of 100 independent runs, using the same parameter values but different random number streams.

The probability r_D is estimated as the fraction of the 100 runs for which all destinations receive m within time D. Similarly, u_S is estimated as the fraction of the 100 runs for which all remaining *operative* destinations receive m within time S after its arrival at a given *operative* process. Latencies are expressed

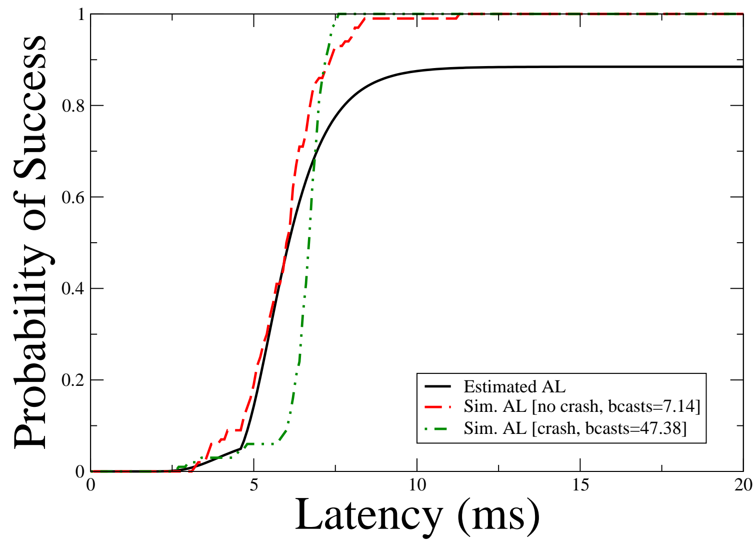

Figure 4.5: Results of group 1 experiments

in units of milliseconds (ms). The following scenarios were considered.

1. *No crashes.* All processes remain operative throughout.

2. *Originator crashes.* The originator crashes after completing the broadcast of copy number 0. Due to message losses, some receivers may not receive m directly from the originator.

3. *Originator crashes with a small set of direct receivers.* The originator crashes while broadcasting copy number 0, such that only a small set of processes directly receive m. This set is called the *direct receivers* and its size is varied.

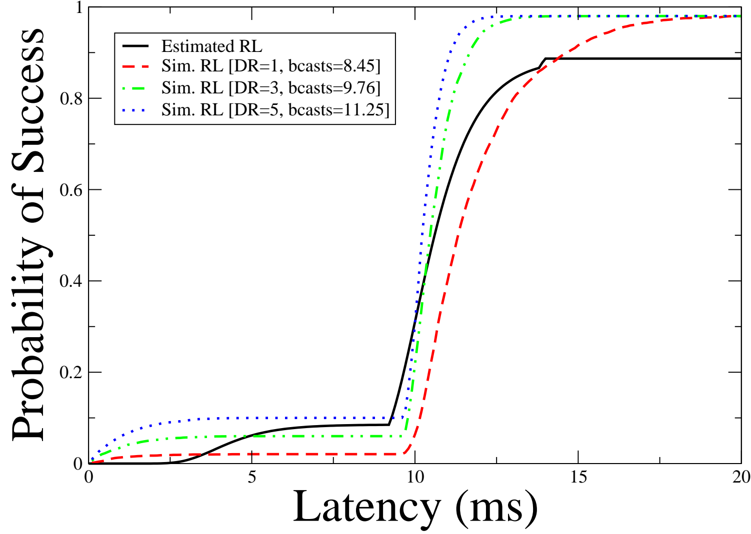

Figure 4.6: Results of group 2 experiments

In all simulations, message transfer times are distributed exponentially with mean $d = 1$ ms; the message loss probability is $q = 5\%$; the group size is $n = 50$; the level of certainty is $\alpha = 99\%$, resulting in $\eta = 4.6$ ms. In addition, the simulations count the total number of broadcasts performed during each run; these counts, averaged over the 100 runs, is denoted as *bcasts* in graphs. Its value should ideally be $(\rho + 1)$.

Four groups of experiments were performed: In group 1, scenarios 1 and 2 were implemented, with $\omega = 1$ ms and $\rho = 1$. In group 2, scenario 3 holds, again with $\omega = 1$ ms and $\rho = 1$; the number of direct receivers was: 1, 2, and 5. Groups 3 and 4 are the same as 1 and 2 respectively, except that $\rho = 2$.

The results obtained are displayed in Figures 4.5, 4.6, 4.7 and 4.8.

Figure 4.5 shows the estimated and observed probability of success, r_D, as a

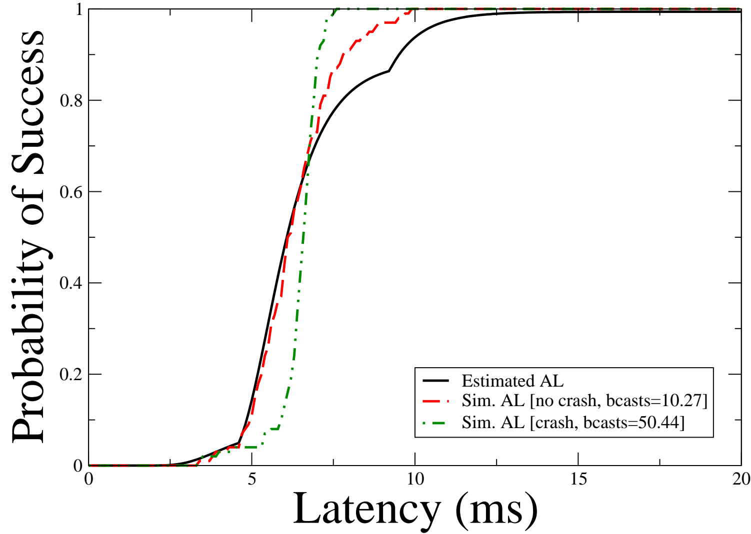

Figure 4.7: Results on group 3 experiments

function of D, for group 1. The estimated probability of success, showed as a straight line in the graph, reflects injection of the ρ redundant transmissions. The first transmission, at time 0, allows an increase in the probability of success, which is further exacerbated by injection of the other message at time 4.6 (η). It is worth noting that in this scenario the value $\rho = 1$ is not sufficient to allow the probability of success to reach values close to 1.

In the no crashes scenario, showed in figure 4.5 with a dashed line, simulations follow estimations over-estimating this latter throughout. However, while the two lines are initially very close, the over-estimation becomes more consistent as latency increases. The reason for this lies in the fact that the approximation ignores the possibility that receivers may time out and become broadcasters; the latter is not unlikely, since $\omega = 1$ ms (in fact, an average of 7.14 broadcasts

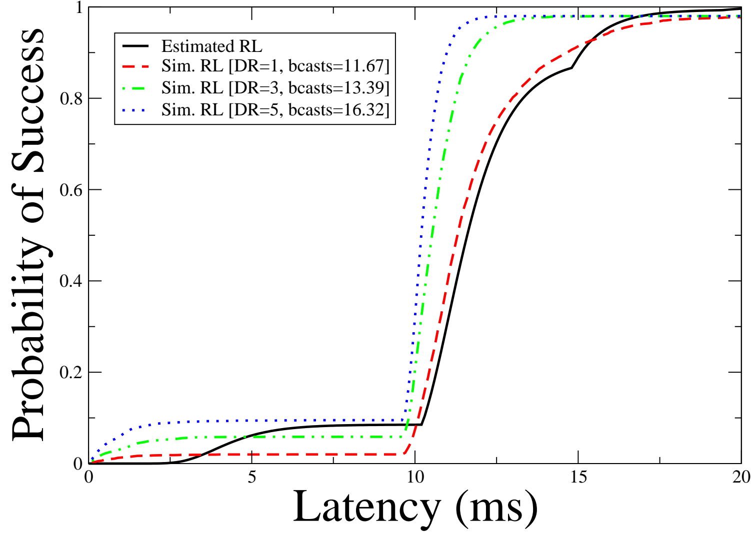

Figure 4.8: Results on group 4 experiments

were observed, instead of the ideal value 2).

Simulations for the originator crash scenario, showed as a line with mixed dash-dot pattern in the graph, can be seen to under-estimate approximations for small latency delays. This is due to the originator crashing after completion of the first broadcast. In fact, this will leave the scenario without a source of messages until those processes having previously received copy 0 from the originator will timeout and start broadcasting. This will happen after $\eta+\omega+\zeta$ from reception from the originator, and in this interval of time the probability of success is prevented from growing. As mentioned above, approximations ignore the possibility that originator crashes (and, consequently, the possibility that receiving processes start to broadcast), and this is the cause of under-estimation in figure 4.5. However, when receiving processes time out, injection

of new messages increases the probability of delivery which, in turn, allows a sudden growth of the probability of success.

Figure 4.6 illustrates the results for group 2, where the originator crashes while attempting to broadcast copy number 0, and the number of direct receivers is quite small. The probability of success, u_S, is plotted against the relative delay, S (relative to the first receiver). As mentioned above, the number of direct receivers is fixed at 1 (showed by the dashed line in the graph), 3 (dashed-dotted line) and 5 (dotted line), and results of such simulations are compared with the straight line representing the approximated relative latency.

As it can be seen from the graph, all lines exploit a similar behavior, which reflects the occurrence of the originator crashing. The approximation line shows the original message being received by a certain number of processes, which increases the probability of success. This number remains constant until those same processes time out and start broadcasting increasing the probability of success. As for the previous case, the graph clearly shows that the level of redundancy $\rho = 1$ is not statistically sufficient to reach the maximum probability of success.

As expected, the larger the number of direct receivers in simulations, the better the performance. When the direct receiver is one sole process, simulations clearly under-estimate approximations for nearly the whole latency interval considered. This is due to the fact that, in simulations, all processes but the direct receiver are prevented to receive the message from the originator, whereas this does not happen in approximations. However, as in simulations

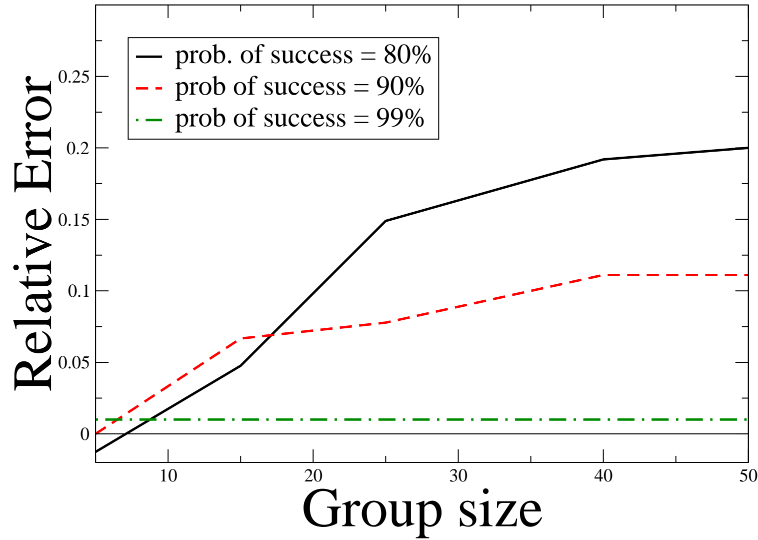

Figure 4.9: Relative error on Latency bounds

the direct receiver starts the recovery phase by broadcasting, more processes deliver the message and this results in the probability of success in simulations to grow faster than in approximations.

The initial under-estimation is reduced when the direct receivers are 3, and disappears when are 5 due to the higher number of processes delivering the message. Moreover, the corresponding simulation lines exploit a faster growth due to the increasing number of direct receivers timing out and broadcasting. Figures 4.7 and 4.8 represent groups 3 and 4 respectively, with $\rho = 2$. In 4.7, the probability of success, r_D, is plotted against the absolute latency, D. The increased value of ρ improves the approximated probability of success considerably.

In figure 4.8, the probability of success, u_S, is plotted against the relative delay,

S (relative to the first receiver), when the originator crashes while attempting to broadcast copy number 0. Because the few direct receivers now make 3 broadcasts, u_S is closer to 1 for large values of S. The approximation is again an over-estimation initially when the number of direct receivers is 1 or 3, for the reasons mentioned above, but becomes an under-estimation as more processes broadcast after timing out.

Consider the observed message traffic. When $\rho = 1$ and the originator remains operative, ideally there would be 2 broadcasts in total, whereas the observed average is 7.14; when the originator crashes after making 1 broadcast, the ideal figure is 3 and the observed one is 47.38 (figure 4.5). The reason for this dramatic increase lies in the originator crash leaving the vast majority of processes (95% on average, given the 5% of loss probability) to time out and start broadcasting. However, the Selection feature allows a quick selection of the new broadcaster. Similar ratios of ideal/observed number of broadcasts hold when $\rho = 2$ (figure 4.7).

Figures 4.9 and 4.10 show the *relative error* of the analytical estimations with respect to the simulations. Three probabilities of success are chosen (80%, 90% and 99%). For each of these, the maximum estimated latency/relative latency and the maximum latency/relative latency observed through simulations were found. The relative error was computed as: *(observed - estimated) / observed*. Positive errors mean that the approximations under-estimate the achievable performance, while negative errors mean that the approximations

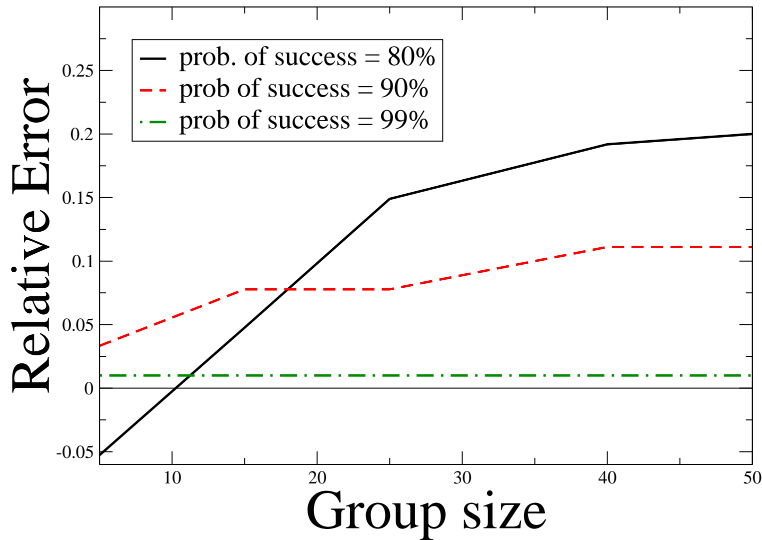

Figure 4.10: Relative error on Relative Latency

over-estimate the real service offered.

Positive errors are desirable because the Negotiation Component vets a QoS request based on the estimated values and the request will not be accepted if the Negotiation Component judges it to be infeasible. Therefore, when the approximations underestimate what the protocol can indeed offer, any admitted QoS request is guaranteed to be met when the protocol is executed with the parameters determined by the Negotiation Component.

Figure 4.9 shows the relative errors on the latency, while figure 4.10 shows the one in relative latencies. The relative error is evaluated for various group sizes. The simulations have been carried out with the same set of parameters as group 3 explained at the beginning of this section.

The maximum negative error observed is -0.05%, while in most cases the er-

ror is positive meaning that simulations over-estimate approximations. Over-estimation is shown to increase as the group size increases. In both cases, this is likely to be the effect of broadcasts carried out by receiving processes timing out.

4.7 Concluding Remarks

In this chapter we have studied the RMcast protocol contained in the RMcast component of the QoS-Supportive Reliable Multicast system subject of this thesis. The protocol guarantees delivery of a message to all or none of the correct destinations, despite sender or receiver crashes and message losses, within a time bound which is negotiated and guarantees anticipately to start of the multicast operation.

We have described the stochastic model used in the QoS negotiations, and showed how it has deliberately been designed to be conservative and act as under-estimate.

It has been shown by experimentation that QoS negotiations do indeed under-estimate the performance of the protocol, except in extreme cases which are very unlikely to occur in practice. In addition, simulations performed confirm that the additional overhead, in terms of message traffic, is not large when the originator does not crash. On the other hand, when the originator crashes the proactive recoverability increases the price to pay for the QoS guarantees.

The logic from the described protocol has been implemented in two distinct

prototypes, described in [40] and [35]. Full description of both prototypes will however be provided in chapter 7 of this same document.

Chapter 5

Multicasting Messages of Arbitrary Size

5.1 Introduction

The protocol described in the previous chapter assumes that the messages transmitted to have standard size, i.e. they do not need to be fragmented. This assumption does not always hold in the real world, as it may well happen that messages to multicast have arbitrary size. As an example, consider the online streaming scenario: information flow is continuous over a prolonged time and needs indeed to be fragmented into several packets in order to be multicast.

When the multicast operation implies provision of guarantees for a specific QoS level on messages of arbitrary size, the protocol described in the previous chapter is not capable of supporting QoS on the whole information flow. This motivates the need to extend the basic protocol presented in the previous chapter in such a way to extend coverage of QoS guarantees to messages

formed by more than one packet.

This chapter describes and studies approaches to allow the QoS-Supportive Reliable Multicast System to multicast messages of arbitrary size maintaining the same reliability and timeliness features.

In particular, two distinct approaches are presented. The first considers the arbitrarily-sized message as a single logical packet, while the second considers the message as a sequence of physical packets. The two approaches are named *Per-Message* and *Per-Packet* respectively. Each of these is described here and also analytical estimations of performance metrics are established. In addition, accuracy of estimations is shown through a simulation study.

The chapter is organized as follows: the Per-Message and Per-Packet approaches are introduced and described in section 5.2. The analytical model for each of them is derived in section 5.3. Accuracy of these latter is evaluated through simulations in section 5.4. From results there obtained, we infer ideal applicability domains for each approach in section 5.5, while in section 5.6 we finally draw some conclusions.

5.2 From single-packet to multi-packet messages

The protocol described in chapter 4 offers a robust service for an originator to multicast messages in a predictably reliable and timely manner. However, messages are assumed to have standard size, and therefore are assumed to fit

into a single transmission packet. That protocol, termed here as the *single-packet* RMcast protocol, is now extended in order to deal with messages m that need to be divided into several packets before transmission and reassembled at the receiving end.

For the sake of exposition, we assume that a message m is fragmented into π packets denoted as $\{pkt_1, \ldots, pkt_\pi\}$ and call the protocol dealing with such packets as the *multiple-packet* RMcast protocol.

The remainder of this section will provide a description of the two studied extensions: the Per-Message approach in subsection 5.2.1, and the Per-Packet approach in subsection 5.2.2.

5.2.1 Per-Message approach.

In this approach, π packets of a given m are considered as a single logical packet (even though each packet is treated independently at low level). The single-packet RMcast protocol is applied on the logical packet.

The originator broadcasts all π packets (ρ+1) times, with successive broadcasts separated by η interval. A destination process receiving *all* π packets of a given k^{th} broadcast, $0 \leq k \leq \rho$, is equivalent to that process receiving copy k in the single-packet RMcast protocol. Figure 5.1 depicts a scenario for $\rho = 1$ and $\pi = 3$.

In this figure, the originator broadcasts $\pi = 3$ packets at two timing instants separated by η interval. If a destination receives *all* packets of the first

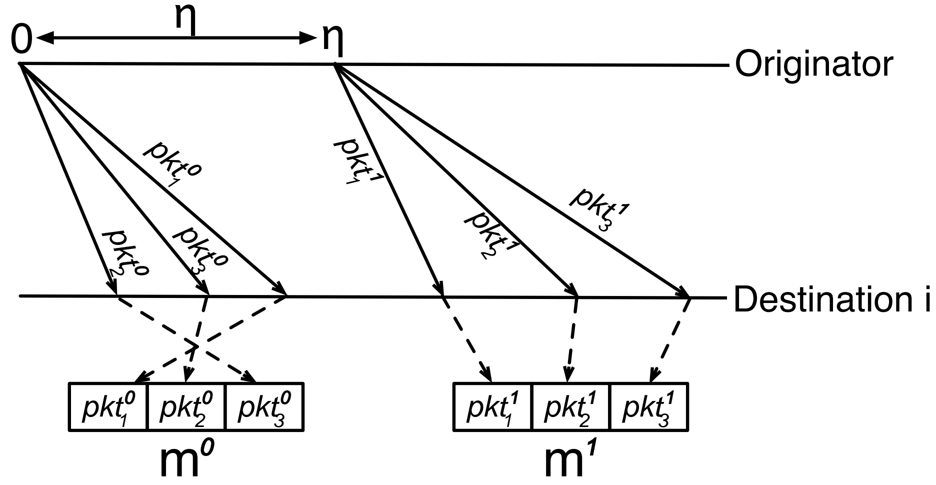

Figure 5.1: Per-Message when $\rho = 1$ and $\pi = 3$

broadcast, it is able to reassemble copy 0 of m (indicated as m^0 in Figure 5.1), and initiates the single-packet RMcast treating m^0 as *the* packet; even if one of the $\pi = 3$ packets is missed, m^0 cannot obviously be formed. This is equivalent to copy 0 not being received in the single-packet protocol. Similarly, receiving or not receiving all $\pi = 3$ packets of the next broadcast will be treated as copy 1 of m (indicated as m^1 in Figure 5.1) having been received or not received respectively. Now, four cases arise:

(i) m^1 is not received and m^0 is also not;

(ii) m^1 is received but m^0 is not;

(iii) m^1 is not received but m^0 has been;

(iv) both m^1 and m^0 are received.

Each case can be treated exactly as in the single-packet protocol:

(i) both message copies are missed, and the protocol does not start;

(ii) the protocol starts execution with the reception of m^1. Missed reception of m^0 is ignored, and $m^2 \ldots m^\rho$ are awaited.

(iii) the protocol starts normally with the reception of m^0 but fails to receive m^1. The $\eta + \omega$ timeout on reception of m^1 expires and the consequent attempt to appoint the local host as new broadcaster for the remainder of the multicast operation takes place;

(iv) both copies are received correctly, execution continues smoothly by waiting for $m^2, \ldots, m^\rho$ (if $\rho > 1$).

This straightforward extension of the single-packet protocol tends to inflate the probability of a message not being received during a given broadcast. If any of the packets of a given m^k is not received, m^k is considered not being received, even though the missing packet may have been received in an earlier broadcast (i.e. $m^0, \ldots, m^{k-1}$). So, two optimizations are identified:

- *Retention*: Each destination process retains the first received copy of each packet, except for the πth one for which the last received copy is retained.

- *Composition*: Once all π packets are available, m is assembled; the assembled m is copy m^k if k is the copy number of the πth packet used.

These two properties together make the Per-Message approach less vulnerable to packet losses. For example, let $\rho = 2$; if any of the $1, \ldots, \pi - 1$ packets

that were received in the first broadcast got lost in the second, m^1 can still be composed so long as the πth packet of the second broadcast is not lost.

The use of Composition and Retention allows to narrow the range of events for which a message copy m^k cannot be rebuild, causing the RMcast operation to fail. In fact, packet j ($1 \leq j < \pi$) of copy k is retained and used in combination with packets from other copies whenever needed to reassemble a message copy m^k, which will be delivered if and only if packet pkt^k_π is received. However, m^k can be delivered under two conditions:

(i) collective reception of packets $pkt_1, \ldots pkt_{\pi-1}$ from broadcasts of $m^0, \ldots, m^{k-1}$, and

(ii) reception of packet pkt^k_π.

Condition (i) can be breached only if packets $pkt_1, \ldots pkt_{\pi-1}$ are not collectively received in broadcasts $m^0, \ldots, m^{k-1}$, while condition (ii) is breached by the missed reception of the relevant packet. As an example, suppose $\pi = 2$ and $\rho = 1$. A process p_i cannot reassemble copy m^k if:

- misses packets pkt^0_1 and pkt^1_1, or

- misses pakets pkt^0_1 and pkt^1_2.

In both cases, the RMcast operation terminates unsuccessfully. The probability of each of these occurrencies to happen depends mainly on the network failure rate and, on a lesser extent, on the size of π. Besides this probability must be taken into account, current network tecnologies allow the failure rate

to be very small even on very wide area networks, as we shall show in section 5.5.1. Therefore, without loss of generality we claim that the probability of one of the two aforementioned conditions to happen can be considered negligible. Figure 5.2 shows the process of multicasting and receiving the message with the Per-Message approach. Invocation of the $RMcast(m)$ primitive starts with the original message m fragmented into π packets. These are then passed to a $RMcast_pkt(pkt_1, \ldots, pkt_\pi)$. This primitive applies the RMcast protocol exactly as in the single-packet case, by considering all π packets as a single logical packet, and delegates the actual broadcast operation to a $broadcast_pkt(pkt_1, \ldots, pkt_\pi)$ primitive, that is invoked $\rho + 1$ times with an η interval time between subsequent invokations. Implementation of the $broadcast_pkt(pkt_1, \ldots, pkt_\pi)$ primitive involves concurrent invocations of the $broadcast(pkt_i)$ primitive, as shown in figure 5.3. The $broadcast(pkt_i)$ primitive is also used in the single-packet RMcast protocol, and is in charge of carrying out the actual broadcast of a single packet. The $broadcast_pkt(pkt_1, \ldots, pkt_\pi)$ primitive uses thus the $broadcast(pkt_i)$ primitive to broadcast all π packets independently. Packets so broadcast are forwarded to the CS which sends them with standard lower level communication facilities.

At destination, reception of packets is realized through a $receive_pkt(pkt_i)$ primitive, shown in figure 5.3. This primitive is also used in the single-packet RMcast protocol, and receives packets by means of independent instances. Packets received are then forwarded to a reassembler which composes the orig-

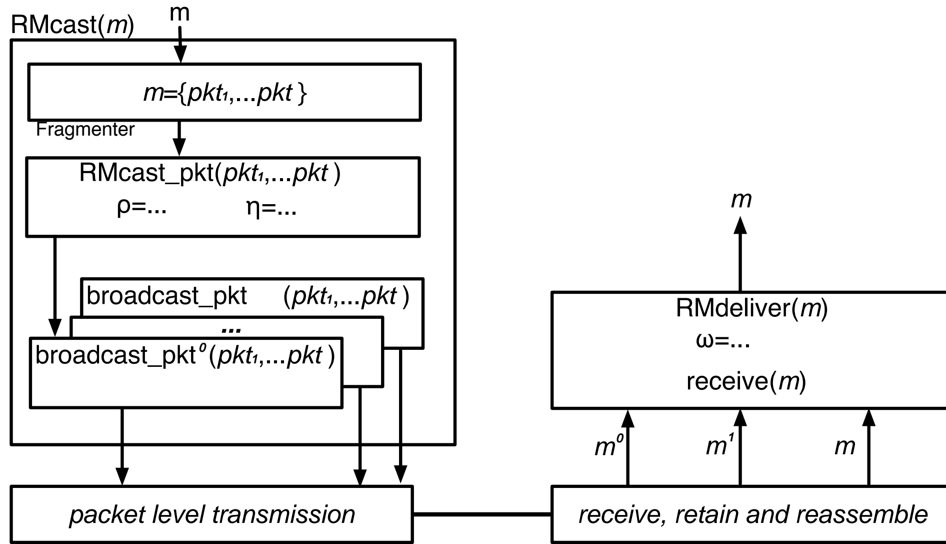

Figure 5.2: Per-Message multicast process.

inal message by exploiting the use of Retention and Composition.

The newly composed message is then passed to a $receive(m)$ primitive, shown in figure 5.2, which forwards it to the $RMdeliver(m)$ primitive for delivery to the user. The $RMdeliver(m)$ primitive also applies the reception side of the single-packet RMcast protocol, and therefore sets up the timeout within which to expect the next message copy. Expiration of such timeout triggers the receiving process to attempt to appoint itself as new broadcaster, and the first step towards appointment is the broadcast of the π packets with the latest copy number received. It is worth noting that in the context of the multiple-packets RMcast protocol this is possible only if the process has received all π packets al least once.

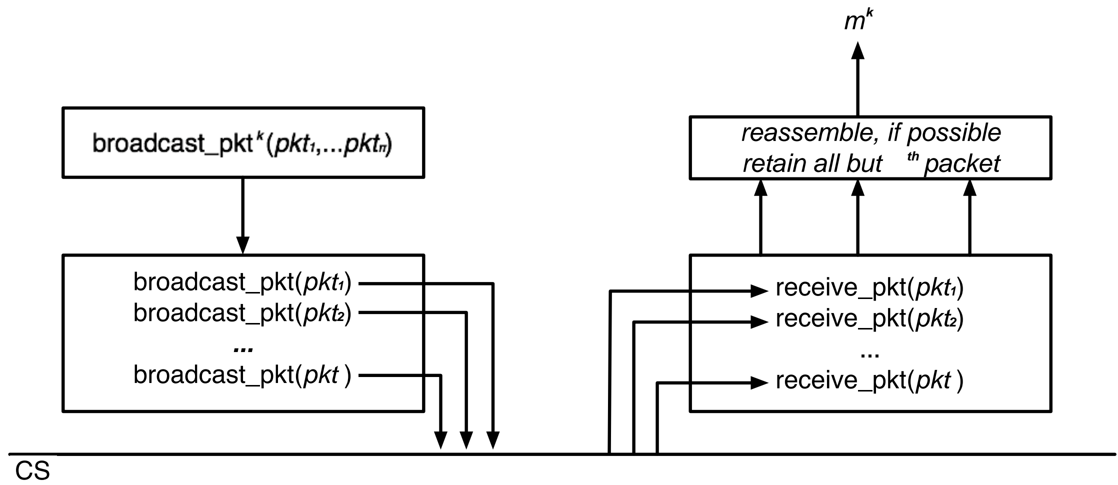

Figure 5.3: Usage of the broadcast primitive in the Per-Message approach.

5.2.2 Per-Packet approach.

In the Per-Packet approach, packets of a given m are treated independently and each one is sent using a dedicated instance of the single-packet RMcast protocol.

Figure 5.4 shows the Per-Packet multicast process. Referring to this figure, after fragmentation of the message, the single-packet RMcast protocol is applied concurrently on each of the π packets obtained. The primitive used to apply the RMcast protocol on each packet is the the $RMcast_pkt(pkt_i)$ primitive. This is at all effects the primitive used to RMcast messages in the single-packet protocol, described in chapter 4 under the name of $RMcast(m)$. As such, its implementaion involves $\rho+1$ invocations of the the $broadcast_pkt(pkt_i)$ (named $broadcast(m)$ in the single-packet RMcast protocol) to send packet copies.

Similarities with the single-packet RMcast protocol are also at destination,

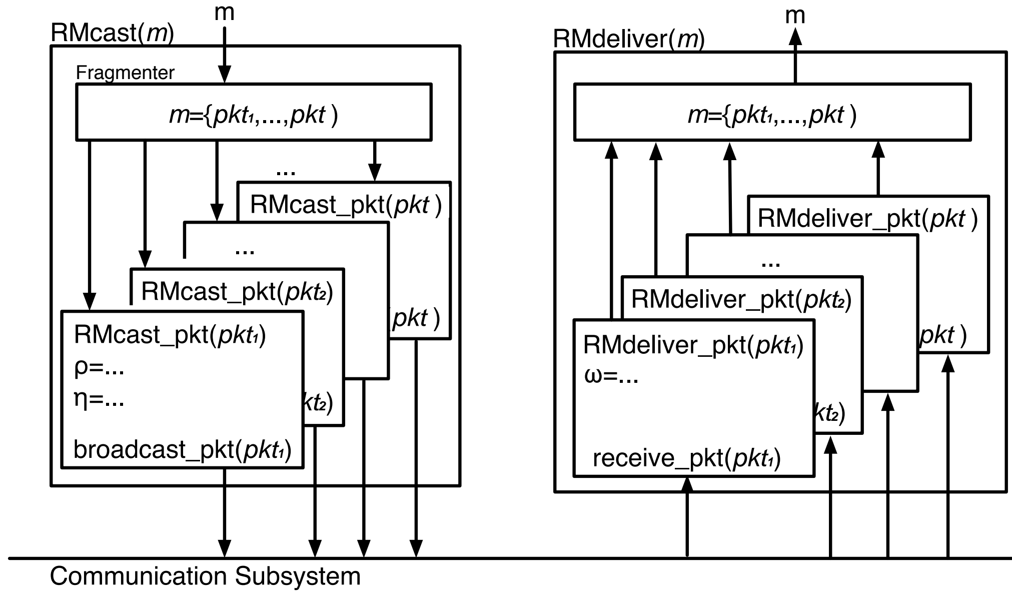

Figure 5.4: Per-Packet multicast process.

where packets are first received by concurrent $receive_pkt(pkt_i)$ primitives, which are governed by as many instances of $RMdeliver_pkt(pkt_i)$ that reliably deliver packets. These two latter primitives are also used in the single-packet RMcast protocol, under the name $receive(m)$ and $RMdeliver(m)$ respectively. Instances of the $RMdeliver_pkt(pkt_i)$ primitive deposit the RMdelivered packets in a buffer from which they are reassembled in the original message m, once all π are delivered.

The $RMdeliver_pkt(pkt_i)$ primitive is also in charge of setting the timeouts within which to expect subsequent copies of a packet upon reception of a copy.

5.2.2.1 Remarks

Packets delivery and recovery mechanism: expiration of the aforementioned timeouts is handled in the Per-Packet approach locally by the instance

if the single-packet protocol which is RMcasting the packet. As a consequence, a process using the Per-Packet approach is not required to have delivered at least a full set of π packets in order to try to appoint itself as new broadcaster.

Load distribution towards termination: the Per-Packet approach inherently admits scenarios where RMcast operations are terminated by more than one group member concurrently. Consider for example an RMcast operation where the originator process p_i needs to RMcast a message m fragmented into $\pi = 5$ packets $\{pkt_1 \dots pkt_5\}$. Suppose p_i transmits correctly the $\rho + 1$ copies for pkt_1 ($pkt_1^0 \dots pkt_1^\rho$) which are received by p_j and p_k among others. Suppose that some problems arise in the RMcast of other packets in the following way. Copy 0 of pkt_2 (pkt_2^0) and pkt_4 (pkt_4^0) are received by p_j and p_k; pkt_2^1 is received by p_j but not p_k, while pkt_4^1 is received by p_k but not p_j. In this case, p_j would timeout on pkt_4^1, while p_k would timeout on pkt_2^1. p_j and p_k would then attempt to become new broadcasters for the remainder of the RMcast of pkt_4^1 and pkt_2^1 respectively, eventually maintaining leadership of the respective single-packet RMcast instance.

Thread considerations: the presentation of the Per-Packet approach made in this section inherently assumes the use of multiple threads working concurrently, each of which handles a separate instance of the single-packet RMcast protocol. This, of course, implies that the hosting machine has availability of enough resources to allow all the needed threads to operate correctly within the expected times. In other words, the Per-Packet approach inherently assumes

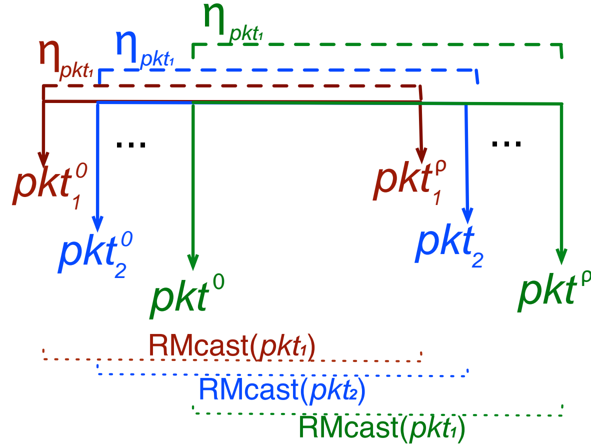

Figure 5.5: Single-threaded Per-Packet approach

that the hosting node is overprovisioned of hardware and software resources. In reality, in contexts where hardware and/or software overprovision is not possible, a single thread can handle all instances. This management model is shown in Figure 5.5.

In this figure, an originator already committed in the RMcast of pkt_1 starts to RMcast remaining packets sequentially while idle for the RMcast operation of pkt_1.

Limitations in the applicability of this technique are obviously proportional to the size of π and the the duration of η. In fact, albeit the time needed to broadcast a message can be considered negligible, it must be taken into account and for a fixed η time, there will exist a value for π for which this approach is not feasible.

The management model just described, that we can name *single-threaded*, has been assumed as the reference management model throughout simulations

whose results will be presented in section 5.4.

5.3 Analytical estimations

5.3.1 Reliability.

Let q denote the probability that a packet is lost during transmission, n the group size, m the message to be multicast, and π the number of packets m gets fragmented into. To start with, let us assume that m to be multicast is a single-packet message ($\pi = 1$), thus eliminating the distinction between the Per-Message and Per-Packet approaches.

A destination process will fail to receive all $\rho + 1$ copies of m with probability $q^{\rho+1}$. Thus, if the packets are lost independent of each other, the reliability r of the protocols is equal to

$$r = (1 - q^{\rho+1})^{n-1}. \tag{5.1}$$

5.3.2 Latencies for a single-packet message

For estimating the absolute latency only, we pessimistically assume that the originator does not crash during protocol execution. In fact, originator's crash can only speed up message delivery as there will be many receiver processes attempting to complete the multicast. Let ξ be the random variable representing the transmission time of a packet from a given source to a given

destination, then the probability that such operation *does not* succeed within time x, denoted as $h(x)$, is:

$$h(x) = q + (1-q)\mathcal{P}(\xi > x), \tag{5.2}$$

with $h(x) = 1$ if $x \leq 0$ by definition. Since the originator makes its kth transmission at time $k\eta$ ($k = 0, 1 \ldots \rho$) the probability g_D that a given destination does not receive any of the $\rho + 1$ transmissions within time D is:

$$g_D = \prod_{k=0}^{\rho} h(D - k\eta). \tag{5.3}$$

From this, the probability r_D that every destination will receive at least one copy of the packet within time D is:

$$r_D = (1 - g_D)^{n-1}. \tag{5.4}$$

Relative Latency estimation is concerned with the following metric: once a process p_i receives copy k of m at time t, then all other correct destinations will receive at least one copy of the message before time $t + S$ with probability u_S. The worst case scenario would occur when the originator crashes before it can transmit m to all destinations and consequently some destinations receive m while others do not[38]. Given that p_i receives copy k of m from the originator at time t, the probability that another p_j will not receive any of the $k + 1$ copies before time $t + S$ is bounded by:

$$g_k(S) = \prod_{l=0}^{k} h(S + l\eta), \tag{5.5}$$

where $h(x)$ is given by (5.2). In addition, if originator crashes and $k < \rho$, p_j may receive copies $k, k+1, \ldots, \rho$ from p_i, assuming the worst case that no other process starts broadcasting. The probability that p_j will not receive any of these copies broadcast from p_i within time $(t + s)$ is then:

$$\tilde{g}_k(S) = \prod_{l=1}^{\rho-k+1} h(S - (l+1)\eta - \omega), \tag{5.6}$$

where $\tilde{g}_\rho(S) = 1$ by definition, and $h(x) = 1$ if $x \leq 0$. A pessimistic estimate for the probability that all other processes will receive at least one copy of the message within S, $u_k(S)$, given that a given process has received copy number k, is given by:

$$u_k(S) = [1 - g_k(S)\tilde{g}_k(S)]^{n-2}. \tag{5.7}$$

Since k can be $0 \ldots \rho$, a pessimistic estimate for u_S will be the minimum of the above probabilities:

$$u_S = min[u_0(S), u_1(S), \ldots, u_\rho(S)]. \tag{5.8}$$

In the subsequent sub sections, we remove the simplifying assumption that $\pi = 1$

5.3.3 Estimations for the Per-Message approach.

5.3.3.1 Reliability estimations

When the copy 0 of m is first transmitted, a destination must receive all π packets to compose copy 0. However, the destination's subsequent attempt at composing any copy k, $k > 0$, can take advantage of having retained the packets received in the earlier transmissions (as stated earlier, a destination retains any received packet - other than the πth one - until the end of the protocol execution). Therefore, it may not require *all* π packets of copy k of m for composition. More precisely, copy k can be composed if the destination receives (i) the πth packet of copy k, and (ii) only those amongst the first $(\pi-1)$ packets which did not reach the destination in any of the earlier transmissions of $0, \ldots, (k-1)$. Since the latter are lost in all k transmissions, they will be $(\pi - 1)q^k$ .Thus, if c_k denotes the expected number of packets in category (ii) that need to be received during transmission k so that copy k (m^k) can be completed, then for all k, $0 \leq k \leq \rho$:

$$c_k = (\pi - 1)q^k + 1 \tag{5.9}$$

where 1 accounts for the πth packet of m^k (category (i) above).

Let Q_k denote the probability that an operative destination is unable to compose copy k, i.e., the probability that one or more of the c_k expected packets would be lost. When packets transmitted are lost independent of each other,

$$Q_k = 1 - (1-q)^{c_k}. \tag{5.10}$$

The reliability r_{PM} achievable in Per-Message approach is:

$$r_{PM} = [1 - Q^{PM}]^{n-1} \tag{5.11}$$

where,

$$Q^{PM} = \prod_{k=0}^{\rho} Q_k. \tag{5.12}$$

Observe that the retaining of received packets (except the πth one) results in $Q_0 \geq Q_1 \geq Q_2 \ldots \geq Q_\rho$. As a consequence, when ρ increases Q^{PM} decreases and r_{PM} increases.

5.3.3.2 Latency estimations

In estimating absolute and relative latencies, we take the approximation that whenever a copy of m is to be transmitted, all packets of copy of m are transmitted in parallel. (In reality, packets are transmitted sequentially, with the πth packet being the last). Let Ξ^k be the random variable representing the interval between the moment a given source transmits copy k of m and the moment a destination is able to compose that copy. Given that a destination needs to receive c_k packets for composing copy k and assuming that only those c_k packets get transmitted to any destination,

$$\Xi^k = max[\Xi_1^k, \Xi_2^k, \dots, \Xi_{c_k}^k]. \tag{5.13}$$

where Ξ_i^k represents the transmission time experienced by i^{th}, $1 \leq i \leq c_k$, arriving packet. Since packets of a given copy are assumed to be transmitted in parallel,

$$\begin{aligned} \mathcal{P}(\Xi^k \leq x) = \mathcal{P}(\Xi_1^k \leq x) \times \ldots \times \mathcal{P}(\Xi_{c_k}^k \leq x) = \\ = [\mathcal{P}(\xi \leq x)]^{c_k} \end{aligned} \tag{5.14}$$

Therefore,

$$\mathcal{P}(\Xi^k > x) = 1 - [\mathcal{P}(\xi \leq x)]^{c_k} \tag{5.15}$$

The probability $h_k(x)$ that copy k of m is not composed at a destination within time x after being transmitted will be:

$$h_k(x) = Q_k + (1 - Q_k)\mathcal{P}(\Xi^k > x), \tag{5.16}$$

and consequently the probability g_D^{PM} that a given destination does not compose m in any of the $\rho + 1$ transmissions within time D is:

$$g_D^{PM} = \prod_{k=0}^{\rho} h_k(D - k\eta). \tag{5.17}$$

The probability u_S^{PM} that two operative destinations will receive the message within S time of each other can be estimated using (5.7) as:

$$u_S^{PM} = min[u_0^{PM}(S), u_1^{PM}(S), \ldots, u_\rho^{PM}(S)] \qquad (5.18)$$

with

$$u_k^{PM}(S) = [1 - g_k^{PM}(S)\tilde{g}_k^{PM}(S)]^{n-2}. \qquad (5.19)$$

$g_k^{PM}(S)$ and $\tilde{g}_k^{PM}(S)$ can be obtained from equations (5.5) and (5.6) respectively, accounting for the fact that $h(x)$ is different for different k (copy number):

$$g_k^{PM}(S) = \prod_{l=0}^{k} h_l(S + l\eta). \qquad (5.20)$$

$$\tilde{g}_k^{PM}(S) = \prod_{l=1}^{\rho-k+1} h_{k+l}(S - (l+1)\eta - \omega). \qquad (5.21)$$

5.3.3.3 Reliability in the Per-Packet approach.

A single-packet RMcast is invoked for each of the π packets of a given m and these invocations operate independent of each other. Consequently, (5.1) leads to:

$$r_{PP} = r^\pi = (1 - q^{\rho+1})^{\pi(n-1)}. \qquad (5.22)$$

It is worth noting that both r_{PM} and r_{PP} can be made arbitrarily close to 1 by choosing large values for ρ.

5.3.3.4 Latency in the Per-Packet approach.

Individual packets are multicast through distinct invocations of the reliable multicast primitive itself. When these invocations are assumed to operate independent of each other, both latency and relative latency probabilities turn out to be an aggregation of the probabilities for the single-packet message. So,

$$r_D^{PP} = (r_D)^\pi = (1 - g_D)^{\pi(n-1)}. \tag{5.23}$$

$$u_S^{PP} = (u_S)^\pi = [min[u_0(S), u_1(S), \ldots, u_\rho(S)]]^\pi. \tag{5.24}$$

5.4 Simulation experiments

Experiments have been conducted by means of simulations to assess the effectiveness of analytical approximations. In analyzing results of such experiments, we shall focus on two main aspects:

(i) negotiability;

(ii) cost of the system;

Negotiability aims to prove the accuracy (or lack of it) of analytical approximations against simulated real scenarios. In the ideal case, approximation results

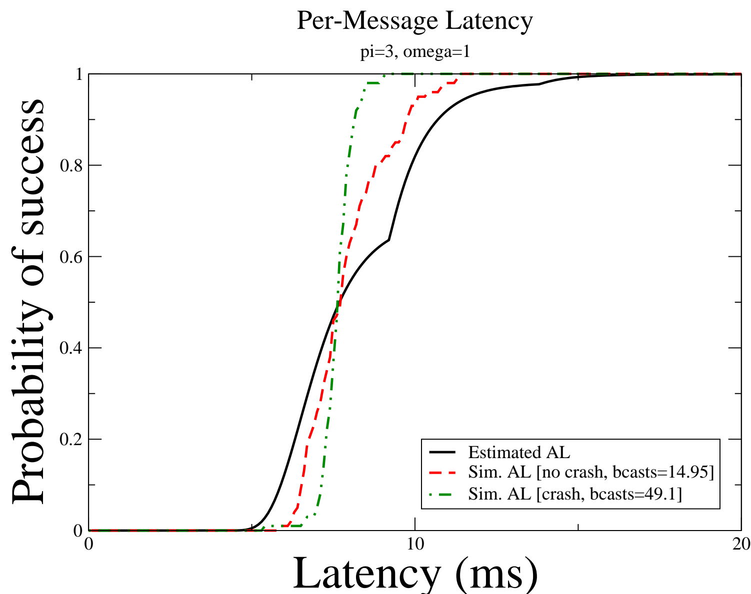

Figure 5.6: Performance of the Per-Message approach when $\pi = 3$ and $\omega = 1$

should underestimate simulation ones as this would mean that the system performs better than what negotiated. By evaluating the cost of the system, on the other hand, we want to evaluate the additional cost, in terms of message overhead, of usage of the system.

Simulation environment is identical to the one described for simulations in section 4.6; the group size is $n = 50$ processes, network packet delays are exponentially distributed with mean $d = 1$ ms, network packet loss is 5%, and the transmission average jitter $\omega = 1$ ms. In such an environment, we fixed the level of redundancy as $\rho = 3$ and the interval time between subsequent redundat transmissions is calculated as $\eta = 4.6$ ms. Finally, in all simulations we fixed the level of certainty to $\alpha = 99\%$.

Simulations are grouped in sessions composed by 100 runs each. For each

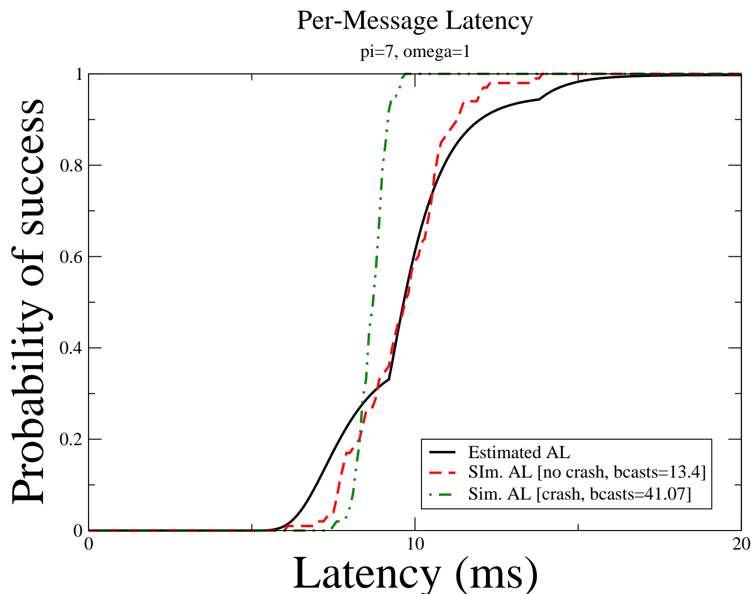

Figure 5.7: Performance of the Per-Message approach when $\pi = 7$ and $\omega = 1$

session we count, as in section 4.6, the number of total broadcasts carried out, which we average to calculate the (average) overhead, indicated as *bcasts* in graphs. Ideally, this should be $(\rho + 1)$.

Simulations whose results are shown in the graphs to follow consider the same range of scenarios as for section 4.6:

(i) *No crashes.* All processes remain operative throughout the protocol execution time.

(ii) *Originator crashes.* The originator crashes just after completing the first broadcast (copy no. 0). Due to message losses, some of the receivers might not receive some or all of the packets from the originator.

(iii) *Originator crashes with a small set of direct receivers.* The originator

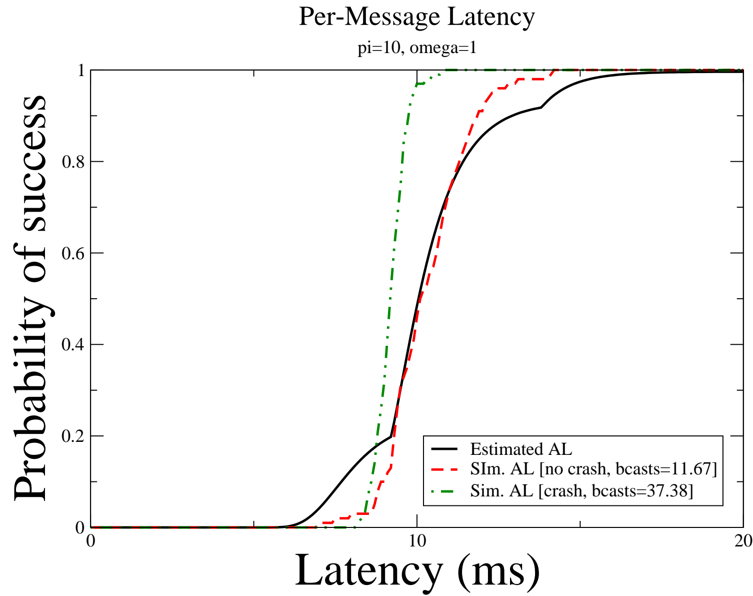

Figure 5.8: Performance of the Per-Message approach when $\pi = 10$ and $\omega = 1$

crashes while broadcasting copy no. 0 of the message. Then, a small set of receivers, called the *direct receivers*, is foreseen to receive the message broadcast. Size of this set is let vary as 1, 3 and 5.

In the Per-Message approach, the originator broadcasts the first copy of the complete set of π packets to the direct receivers, as this is the minimum requirement for these latter processes to start the protocol. On the other hand, in the Per-Packet approach the originator starts π independent instances of the single-packet protocol, and each of them is interrupted in the middle of the first broadcast after having transmitted the the number of random direct receivers. As a consequence, there is no guarantee that the identity of the processes in the set of direct receivers

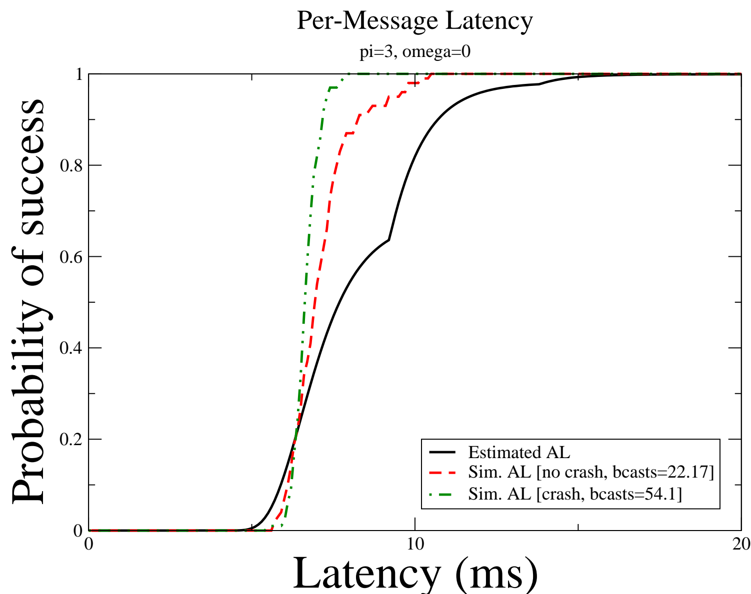

Figure 5.9: Performance of the Per-Message approach when $\pi = 3$ and $\omega = 0$

is the same for each of the π packets.

5.4.1 Negotiability

5.4.1.1 Negotiability in the Per-Message approach

Negotiability of the Per-Message approach is assessed through analysis of graphs in figures 5.6-5.8. Observing these figures we can note that the increase of the value of π causes the probability of success in approximations to require higher latency delays to grow. This is normal, if we consider that delivery of a message implies now collecting π ($\pi > 1$) packets and those packets are subject to network losses and delays. However, this effect decreases as new redundant copies of packets are injected in the network, due to the use of

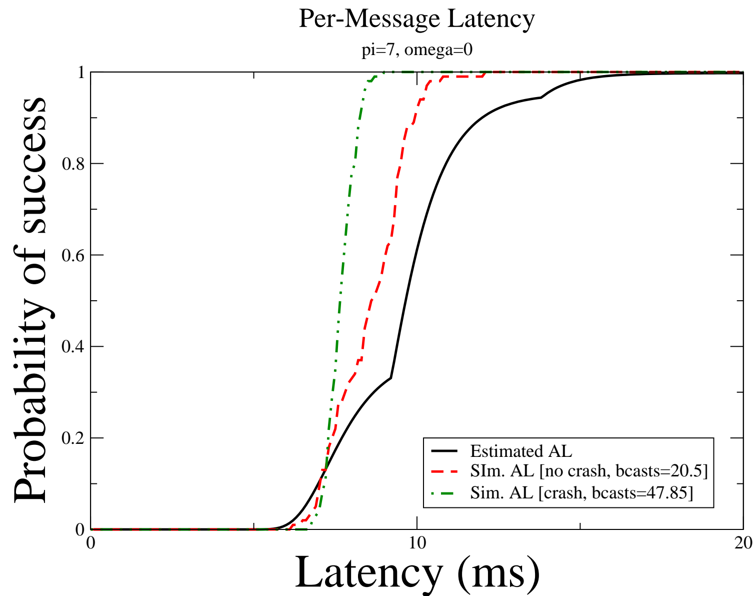

Figure 5.10: Performance of the Per-Message approach when $\pi = 7$ and $\omega = 0$

Retention and Composition.

In the no crash scenario of figure 5.6, simulations initially underestimate approximations, until the first process times out. This triggers broadcast of a message and, in turn, suddenly increases the delivery probability. The probability of success thus increases until over-estimating approximations. This trend seems to be confirmed for bigger values of π grows, as figures 5.7 and 5.8 witness.

Under-estimation is limited to small probabilities of success (which are unlikely to be requested in a real-world scenario), but this occurrence is nonetheless undesirable and can be avoided by considering what follows. Under-estimation of the simulations over the approximations can be seen as the

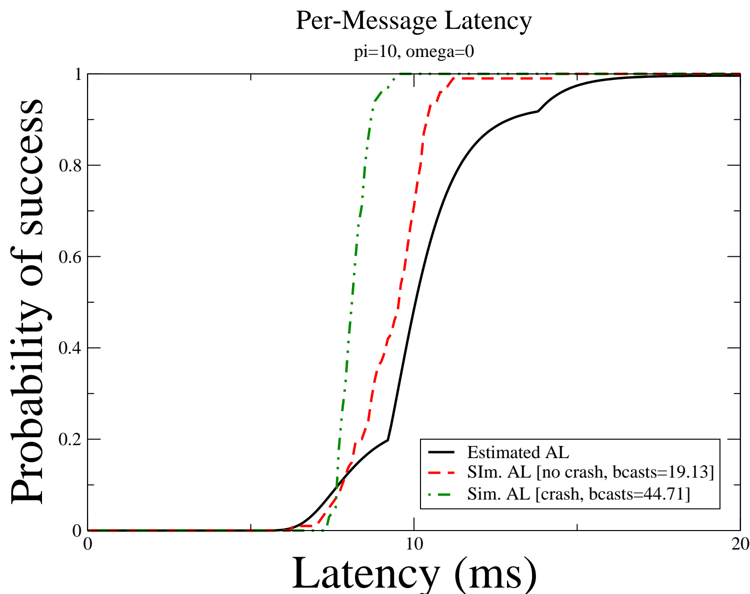

Figure 5.11: Performance of the Per-Message approach when $\pi = 10$ and $\omega = 0$

consequence of the timeliness (or lack of it) in the activation of the recovery mechanism. This parameter, in turn, is based on the length of the timeout set by receiving processes, which account for the interval of time between subsequent broadcasts (η), and the differences in transmission times (i.e. the jitter, ω). The reason for the apparent lack of timeliness is that in approximations this timeout is ignored, as the originator is pessimistically assumed not to crash. Therefore, longer timeouts in simulations will delay activation of the recovery mechanism when packets are lost or delayed unduly (whose occurrence is more likely to happen when π grows), while shorter timeouts will allow timely recovery from packet losses and delays.

Given that the the value of η is fixed throughout an RMcast operation, what

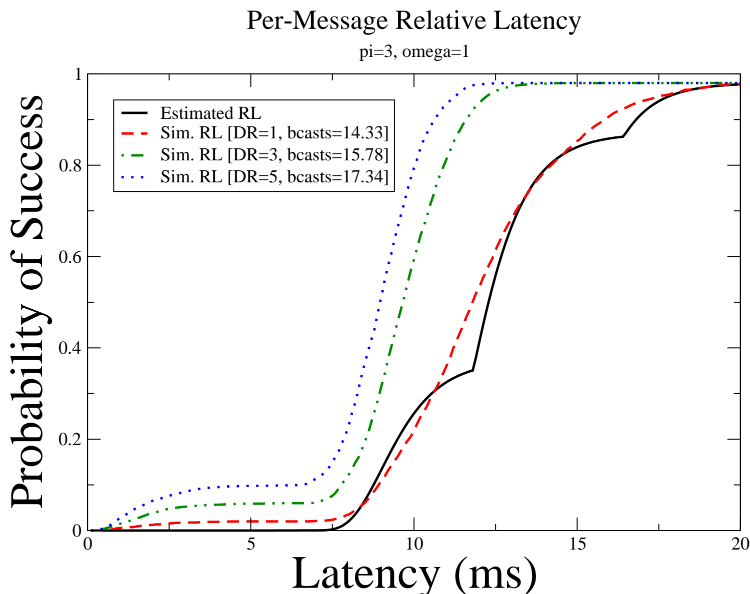

Figure 5.12: Performance of the Per-Message approach in the direct receivers scenario when $\pi = 3$ and $\omega = 1$

causes simulation to under-estimate approximations in figures 5.6-5.8 is the chosen value of $\omega = 1$. The undesirable under-estimation can therefore be avoided by enhancing timeliness in the recovery mechanism. This, in turn, can ve achieved by choosing smaller values for ω in simulations.

In order to confirm our assertion above, we executed a set of experiments on an environment where ll parameters are set to be identical to the one in figures 5.6-5.8, with the only difference of the jitter set to $\omega = 0$ ms (i.e. no jitter present).

Results of these experiments are shown in figures 5.9-5.11, and confirm our thoughts. The initial under-estimation is highly reduced regardless of the value of π. Simulations over-estimate approximations throughout, with a tendency

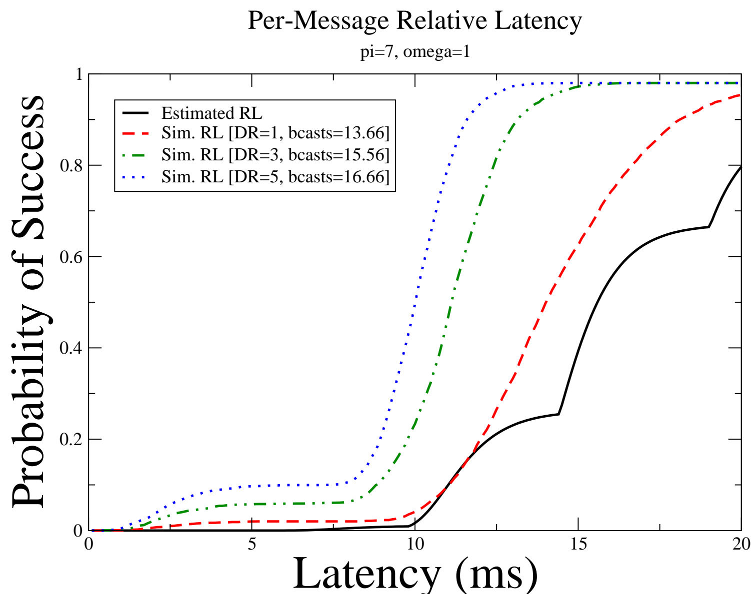

Figure 5.13: Performance of the Per-Message approach in the direct receivers scenario when $\pi = 7$ and $\omega = 1$

to increase for higher probabilities of success.

A similar initial period of under-estimation can be noted in simulations for the originator crash scenario of figures 5.6-5.8, where simulations appear to under-estimate approximations to a more consistent extent then the no crash scenario. At the basis of this phenomenon, as also mentioned in the case of the single-packet protocol in section 4.6, lies the originator crash. In fact, when this latter crashes, receiving processes having started the protocol with reception of the first copy are left waiting for the subsequent copy. This eventually causes the timeout to expire, allowing them to broadcast. In the corresponding simulations, this process results in a low probability of success until the instant when the first receiving process times out and broadcasts the message, which causes a sudden increase of the probability of success. As a side remark, the

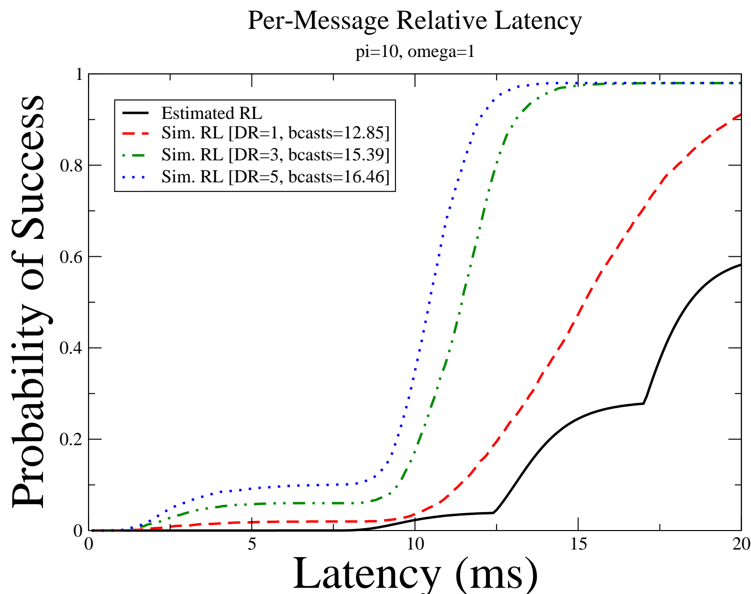

Figure 5.14: Performance of the Per-Message approach in the direct receivers scenario when $\pi = 10$ and $\omega = 1$

initial under-estimation observed in this scenario is observed to reduce in the set of experiments carried out for $\omega = 0$.

Graphs 5.12-5.14 assess negotiability of the Per-Message approach in terms of relative latency in the direct receivers scenario. The effect of increasing the value of π is still clearly visible, as approximations appear to be shifted in time as π grows. The same factor influences the probability of success at the time of the injection of redundant copies of the message.

As for the case of the single-packet protocol presented in chapter 4, the effect of the originator crash is clearly dominant in this scenario: the probability of success starts and grows proportionally to the number of direct receivers receiving the message from the originator. This is followed by a period of apparent inac-

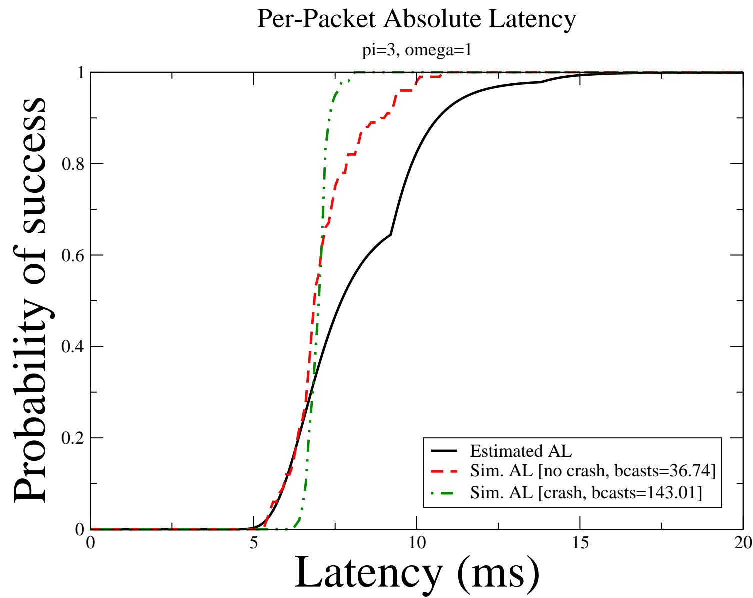

Figure 5.15: Performance of the Per-Packet approach when $\pi = 3$ and $\omega = 1$

tivity, that terminates when those same processes timeout and broadcast. This has the effect of allowing all other processes to start the protocol and progress normally. The effect of varying the number of direct receivers is clearly visible in graphs 5.12-5.14, and the probability of success can be seen growing faster as more processes are allowed to receive the message from the originator. Simulations over-estimate approximations throughout in figures 5.12-5.14. When $\pi = 3$, and the direct receiver is one process, there is an imperceptible under-estimation. However, this is minimal and is quickly compensated. When the number of direct receivers grows, over-estimation of simulations on approximations becomes more consistent. This is normal, considering that approximations are obtained under pessimistic assumptions.

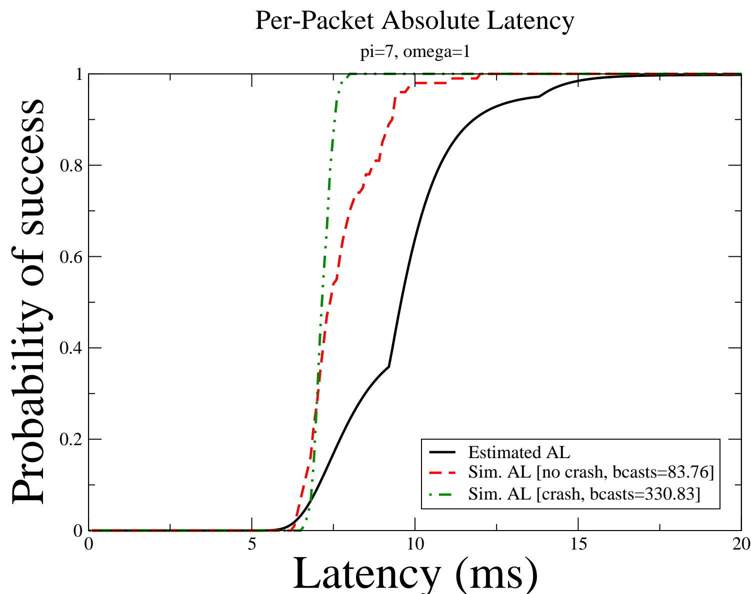

Figure 5.16: Performance of the Per-Packet approach when $\pi = 7$ and $\omega = 1$

5.4.1.2 Negotiability in the Per-Packet approach

Results of experiments on the Per-Packet approach are shown in figures 5.15-5.17. Both approximations and simulations exploit a behavior similar to the Per-Message approach. However, small differences can be noted.

In the no crash scenario of the Per-Packet simulations, the initial period of under-estimation of simulations over approximations is less consistent than the likewise simulations for the Per-Message approach. The reason for this difference can be found recalling that the Per-Packet approach realizes the RMcast operation by means of π concurrent instantiations of the single-packet protocol, while the Per-Message approach employs a single instance where packets are sent sequentially.

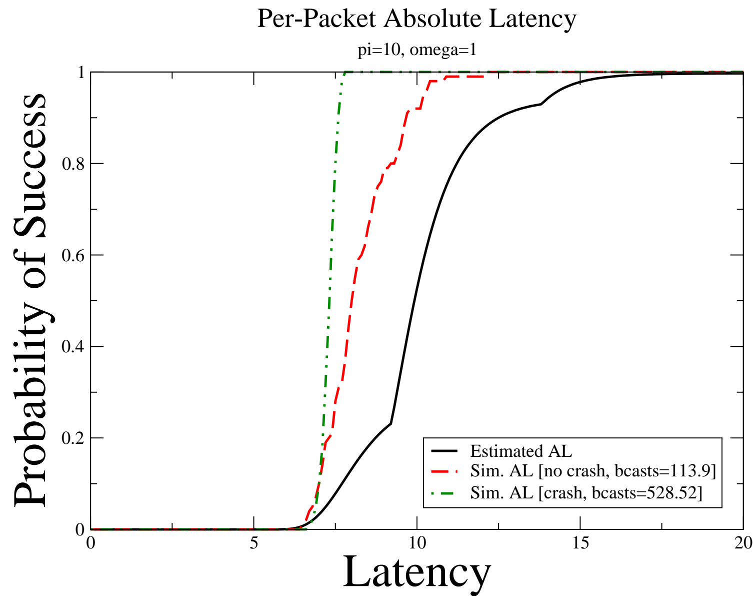

Figure 5.17: Performance of the Per-Packet approach when $\pi = 10$ and $\omega = 1$

This means that the protocol is applied on a per-packet basis in the Per-Packet approach, rather than on a per-message basis in the Per-Message approach. This this has repercussions on timeouts. In fact, application on a single-packet basis implies timeouts to be effective in reacting to losses of packets, whereas application on a per-message basis implies timeouts to be effective in detecting loss of messages. Losses detected on a per-message basis result then to be more shifted in time, as π grows, than losses detected on a per-packet basis. This phenomenon can also be noted by comparing simulations for the Per-Message approach, in figures 5.6-5.8, with simulations for the Per-Packet approach, in figures 5.15-5.17.

Reduction of the initial period of under-estimation can also be noted in

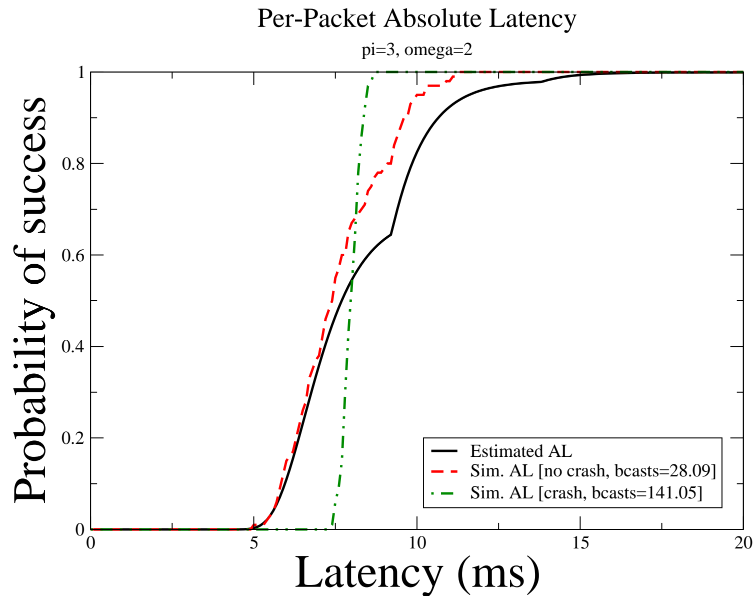

Figure 5.18: Performance of the Per-Packet approach when $\pi = 3$ and $\omega = 2$

the originator crash scenario, with the result that when $\pi = 10$ the under-estimation disappears.

Observing Per-Packet graphs we can note that as π grows, simulations increasingly overestimate approximations. This is still an effect of the length of the timeouts set by receiving processes and, bearing in mind what we said when describing the Per-Message approach, to the value of ω. When used in an environment suffering a jitter delay of $\omega = 1$ ms, simulations show a behavior that tend to over-estimate approximations to an extent that increases as π increases. Consequently, for higher values of π the Per-Packet approach might loose accuracy, undermining negotiability features.

This problem is similar (but opposite) to the one noticed and described above for the Per-Message approach (which was suffering of underestimation

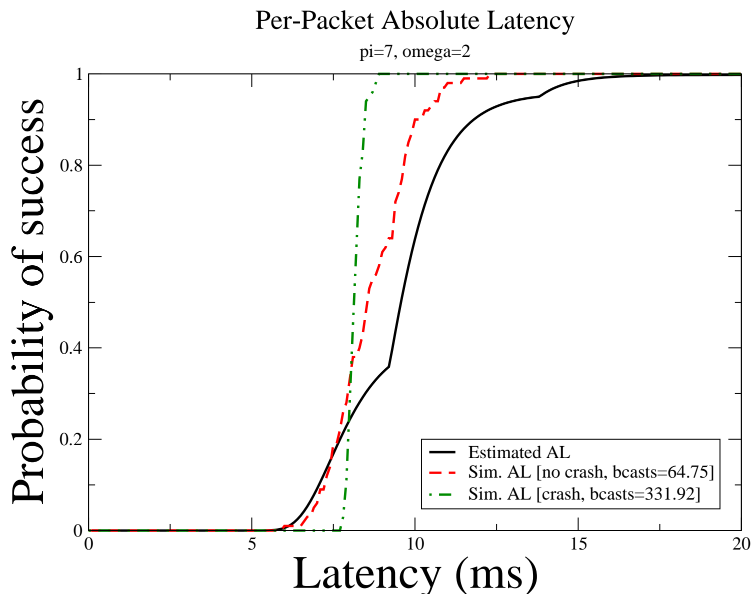

Figure 5.19: Performance of the Per-Packet approach when $\pi = 7$ and $\omega = 2$

problems) and, as well as in that case, it gives us a good indication of the environment where the approach would be more effective.

In the Per-Message approach, under-estimation led us to the consideration that the approach would naturally suit environments where the transmission jitter is reduced or even absent (as figures 5.9-5.11 show). The Per-Packet case, on the other hand, tends to loose accuracy, in negotiations, by over-estimating approximations as π grows. This characteristic seems to suggest that Per-Packet negotiability would be enhanced in environments with higher transmission jitter.

To this extent, we have simulated the Per-Packet approach on an environment that assumes the jitter to be $\omega = 2$ ms while leaving all other parameters similar to experiments in figures 5.15-5.17. Results of these simulations are

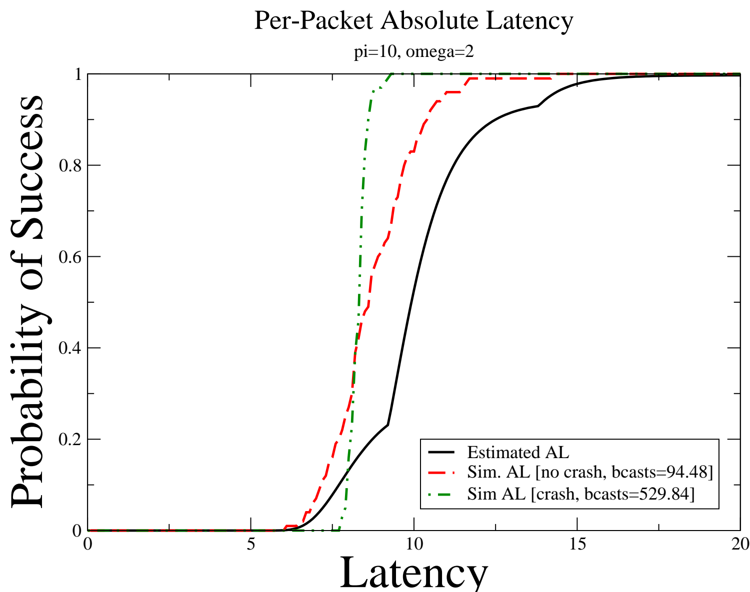

Figure 5.20: Performance of the Per-Packet approach when $\pi = 10$ and $\omega = 2$

shown in figures 5.18-5.20.

These graphs confirm our assertion above; simulations for the non-crash scenario enhance negotiability by reducing the over-estimation excess on approximations. Compared to the likewise graphs in figures 5.15-5.17, we note that simulations over-estimate approximations to a lesser extent, showing thus better negotiability. When $\pi = 10$, in figure 5.20, we still note a tendency to excess in over-estimation. However, results of this set of experiments let us argue that this tendency would decrease as the value of ω increases.

In the direct receivers scenario when the direct receiver is one sole process, the simulation line follows the approximation line closely, over-estimating this latter throughout. A negligible under-estimation can be seen for high probabil-

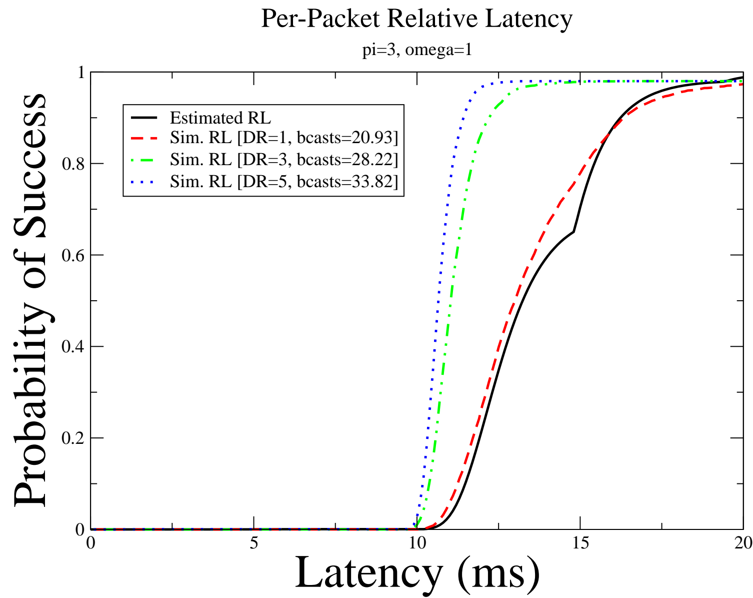

Figure 5.21: Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 3$ and $\omega = 1$

ities of success and, the fact that this seems to be less negligible when $\pi = 10$ in figure 5.23 let us argue that for higher values of π this might become a problem. However, the scenario refers to the one where the originator crashes in the middle of broadcast operation and its message is received by one process only, and we claim that occurrence of such event can be considered unlikely.

As in all other similar cases, the overestimation becomes predictably more consistent as the number of direct receivers increases to 3 and 5 processes, due to the increased number of broadcasts carried out by the direct receivers upon expiration of the timeout.

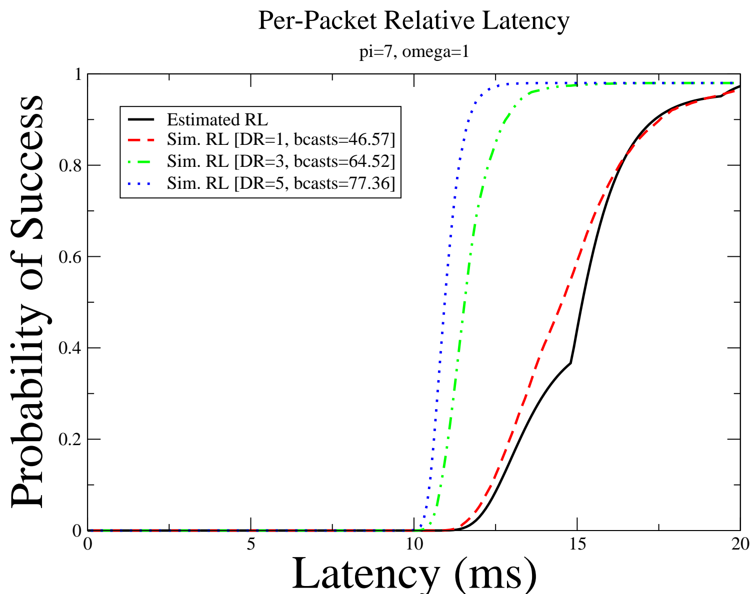

Figure 5.22: Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 7$ and $\omega = 1$

5.4.2 Cost of the system

5.4.2.1 Cost of the Per-Message approach

Referring to results of the Per-Message approach in figures 5.6-5.8, it is interesting to note how the average number of broadcasts changes when π increases.

When $\pi = 3$ and the originator does not crash, 14.95 broadcasts are carried out on average. Besides the expected $\rho + 1$, this number is influenced by the Responsiveness property, needed to compensate the network packet loss and delay to provide multicast delivery of π packets. However, when π becomes bigger this number is seen to reduce to 13.4, when $\pi = 7$, and 11.67, when $\pi = 10$. This finds explanation in the fact that when π increases, fewer processes find themselves in the position of reassembling and delivering a message,

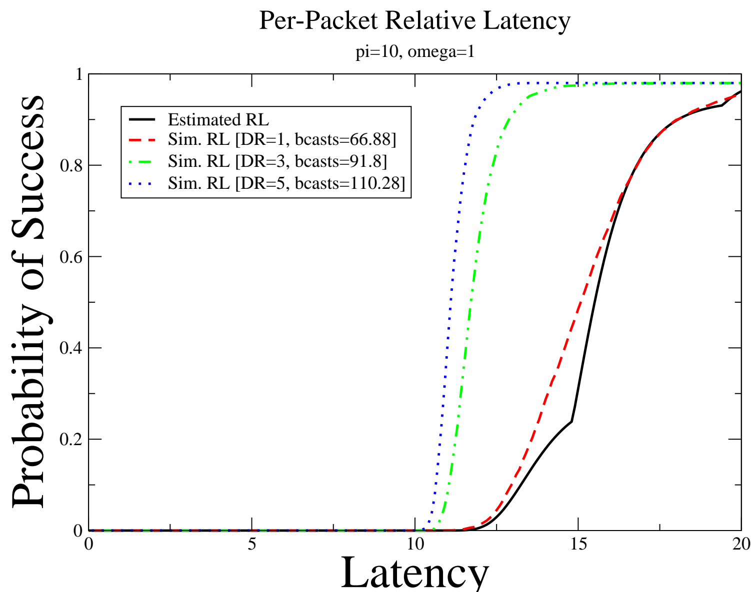

Figure 5.23: Performance of the Per-Packet approach in the direct receiver scenario when $\pi = 10$ and $\omega = 1$

due to packets lost or unduly delayed. Therefore, fewer processes will be in the condition of timing out and broadcasting. This behavior also reflects the influence of employing Composition and, especially, Retention. This latter causes, in fact, last broadcasts to be virtually not affected by packet loss, since all needed packets (but the πth) are expected to be received in the firsts broadcasts. This eventuality is not unlikely, since the packet loss rate is 5%.

When originator crashes, the number of broadcasts carried out increases consistently. This is predictable, as originator crash causes those processes having received the first copy of the message to time out and broadcast. However, this number is seen to decrease as π increases, for reasons explained for the no crash scenario.

The cost of the system is seen to increase when the recovery mechanism is

made more timely by decreasing the value of ω, as figures 5.9-5.11 witness. This additional cost can be explained by considering that smaller timeouts reduce the tolerance to losses and delays. Therefore, the probability of processes suspecting of having lost packets which are in reality unduly delayed increases. Concluding, the use of Retention and Composition reduces the cost of the Per-Message approach. Given the initial redundancy level $\rho = 3$, the price to pay for reliability in the contexts under examination is a 3 to 4-fold increase in message traffic when no crashes are assumed. This ratio grows when crashes are taken into account, as a price to pay for proactive recoverability. However, the cost of the system can be traded with negotiability features.

5.4.2.2 Cost of the Per-Packet approach

In the Per-Packet approach, a broadcast transmits one packet to $(n-1)$ destinations. Therefore, the average number of broadcasts calculated by the *bcasts* parameter refers to the actual average number of *packets* broadcast. Since these are carried out by means of independent instances of the single-packet protocol, the average number is expected to grow when π grows. In the Per-Message approach, on the other hand, a broadcast transmits π packets to $(n-1)$ destinations. The *bcasts* parameter refers thus to the average number of *messages* broadcast. This explains the difference in the value of the *bcasts* parameter for the two approaches.

The *bcasts* parameter can be made homogeneous in the two approaches by dividing the value for *bcasts* in the Per-Packet approach by the value of the

corresponding π. The rationale for this conversion is that packets are transmitted by means of independent operations and therefore they are lost (and retransmitted) with an equal probability.

When $\pi = 3$ and the originator does not crash in the Per-Packet approach, figure 5.15 shows that 36.74 broadcasts are needed on average to fulfill QoS guarantees. Considering that a message is composed by $\pi = 3$ packets, we can calculate an average of 12.24 messages needed. On the other hand, in the originator crash scenario of the same simulation set, the Per-Packet approach needs 143.01 packets to be broadcast. This leads to an average of 47.67 messages broadcast. These results are in line with the results for the Per-Message approach, although showing a slightly lower cost.

When $\pi = 7$ and $\pi = 10$, in figures 5.16 and 5.17 respectively, the average broadcasts increase to 83.76 and 113.9 respectively in the no crash scenario (330.83 and 528.52 respectively in the originator crash scenario). Considering the corresponding value of π, these equal to an average 11.96 messages (47.26 in the originator crash scenario), when $\pi = 7$, and 11.39 (52.85 in the originator crash scenario) when $\pi = 10$. As it can be seen, in the no crash scenario the number of broadcasts decreases as the size of the message, i.e. π increases in a pattern similar to the Per-Message approach

In the Per-Message approach, this was due to the use of Retention and Composition. In the Per-Packet approach, who cannot make use of these properties, the decrease is due to the fact that the π packets are RMcast by means of independent instances of the single-packet protocol. Thus, when a packet is not

received within the timeout, the Per-Packet approach foresees retransmission of that packet whereas the Per-Message approach foresees retransmission of the entire π packets, when the message copy cannot be reassembled within the timeout.

This allows the Per-Packet approach to exploit a more sensitive recovery mechanism, compared to the one in the Per-Message approach, and the decrease in the average number of *messages* broadcast is a consequence of such sensitivity. The same phenomenon is not observed, on the other hand, in the originator crash scenario. In fact, unlike the likewise scenario in the Per-Message approach where the use of Retention and Composition allowed a reduction in the average number of broadcasts carried out, in the Per-Packet approach the average number of broadcasts is observed to grow as the value of π grows. This is due to the fact that the aforementioned sensitivity is not effective in the originator scenario. In fact, originator's crash inevitably causes all destination processes having previously received copy 0 to time out and broadcast.

Summarizing what said above, and given the originally planned $(\rho + 1)$ transmissions, the cost of the Per-Packet approach is a 3 to 4-fold increase in message traffic when the originator does not crash. On the other hand, when the originator crashes the the price to pay for reliability and timeliness with the Per-Packet approach increases consistently.

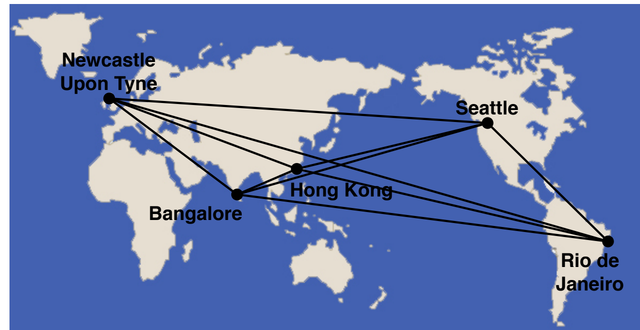

Figure 5.24: PlanetLab slice

5.5 Limitations and applicability domains

Per-Message and Per-Packet are conceptually different approaches, each making use of different network characteristics. The obvious observation is that in particular circumstances or environments the use of either of the approaches might experience limitations. This, in turn, leads to considering specific domains where application of an approach would be particularly effective. These *applicability domains* are derived by binding limitations to network properties, and considering environments where such (limiting) characteristics have minor impact, whereas do not impact at all.

5.5.1 Per-Message extension

Principal limitations of the Per-Message approach relate to the message loss rate. In fact, unreliably multicasting π packets, each one equally important to rebuild an original message m, causes an increase in the loss rate experienced to receive an entire message. We shall refer to this latter as the *message* loss

rate, while we shall refer to the *packet* loss rate intending the rate with at which packets can be lost, fixed in both simulations and approximations to 5%.

Table 5.25 shows how the message loss rate in the Per-Message approach (Q^{PM} in the table) changes based on the packet loss rate. When this latter is small, the Per-Message approach seems to experience small message loss rates due to the use of Retention and Composition. However, as the former increases the two optimizations seem to loose effectiveness and the message loss rate becomes subject to higher increases. This is obvious if we consider that an increased packet loss rate allows reception of less packets in a single broadcast. The time needed to reassemble a message copy will be longer, thus increasing the probability that a packet is suspected to be lost (and consequently the number of broadcasts) reducing so the effectiveness of the Composition property. Assuming high values for the jitter would limit the increase in broadcasts carried out, but the risk would be to jeopardize negotiability according to what we said in section 5.4.1.1.

The ideal applicability domain for application of the Per-Message approach can therefore be bound to environments subject to small loss rates and where the jitter is low. Today's technologies allow construction of such networks. The PlanetLab testbed[2] is an example of such a technology.

PlanetLab is a wide-scale infrastructure, formed by several hundreds of machines that member institutions distributed geographically place at users' avail-

Packet loss rate (q)	PM loss rate (Q^{PM})
0.01	0.001
0.03	0.009
0.05	0.029
0.07	0.057
0.1	0.118
0.13	0.196
0.15	0.255
0.17	0.318
0.2	0.415
0.22	0.479
0.25	0.573
0.27	0.632
0.3	0.821
0.4	0.899

Figure 5.25: Growth of Q^{PM} compared to q.

ability. Its purpose is to provide a distributed testbed for execution of widely distributed applications.

In PlanetLab, users create a so called *slice*, which provides the abstraction of virtual accessible domain. Slices are then populated with *nodes*, i.e. machines. These can be manually selected from the list of member institution, and users can therefore easily exploit geographical distribution. PlanetLab provides classical TCP[96] and UDP[95] communication facilities which work on a best effort basis, and the infrastructure connecting distant nodes is the Internet.

To the extent of our example, we built a slice and populated it with five nodes chosen from member institutions geographically disperse as shown in Figure 5.24. On each node of the so-formed group we ran a mini-application, consisting of the Network Measurement Component only, with the purpose of

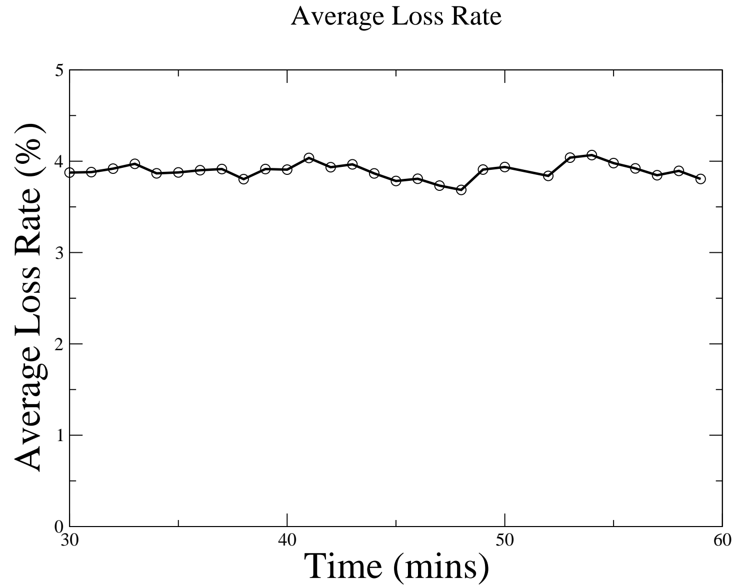

Figure 5.26: Loss rate in the PlanetLab slice

measuring the worst group-wide network loss rate and jitter. Results from these experiments are shown in figures 5.26 and 5.27 respectively.

The graph in figure 5.26 shows the loss percentage, in the vertical axis, as a function of time, on the horizontal axis. Probing sessions were 60 minutes long. The loss rate has been calculated as $(lost_pkts * 100)/total_pkts$, where $lost_pkts$ represents the number of packets lost and $total_pkts$ is the total number of packets sent.

The graph reports the loss rate experienced in the second half of the probing session only, i.e. the last 30 minutes. The reason for omitting the first half of the probing session lies in the fact that the behavior is initially transient, and therefore not meaningful for our purposes.

The graph shows that over time, the group-wide loss rate tends to stabilize

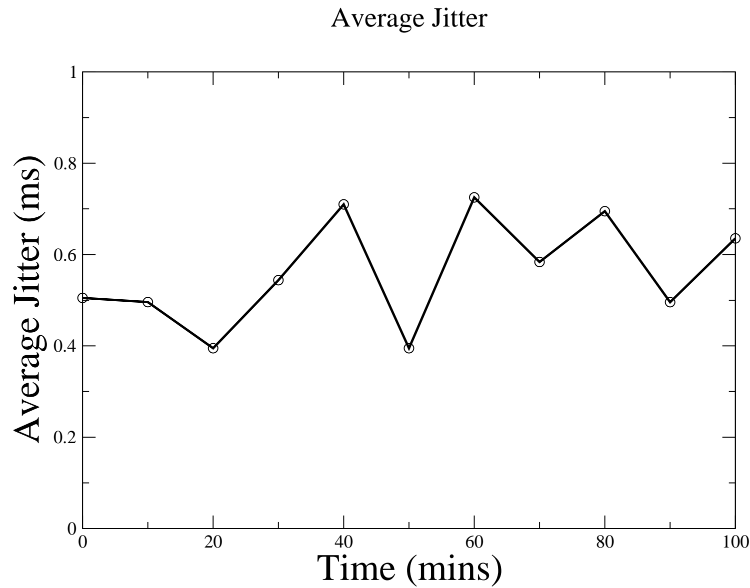

Figure 5.27: Average jitter in the PlanetLab slice

around 4%, which confirms our thoughts about loss rate in best-effort networks also validating our choice in simulations and approximations.

The graph in figure 5.27 shows results of another probing session, where the group-wide worst average jitter is measured. The average jitter in terms of milliseconds, in the vertical axis, is shown as a function of time, in the horizontal axis, expressed in 10 minutes-long sessions.

The graph clearly shows that variations for the average jitter are very small, with peaks contained within 1 millisecond. This confirms, as well, our assertion on existence of best-effort networks subject to small changes in jitter. In addition, results of this graph also validate our choice of jitter $\omega = 1$ ms in simulations and approximations.

Together, graphs in figures 5.26 and 5.27 show that best-effort networks subject to small packet loss rates and reduced average jitter do exist, and provide an example of network ideally suitable for application of the Per-Message approach.

5.5.2 Per-Packet extension

The Per-Packet approach consists in all π packets to be RMcast concurrently. Its execution has, therefore, calls for message overhead. However, it has the benefit of offering good negotiability in environments where the average jitter is high as figures 5.18-5.20 show. Besides improving negotiability, higher values for the jitter allow also a reduction of the message overhead (when originator does not crash).

This seems to suggest the ideal domain for the Per-Packet approach as to be bounded to environments where communications are subject to high differences in transmission times. However, another factor needs to be considered.

In standard environments, i.e. in environments where software/hardware over-provision cannot be assumed, a single thread will handle the π instances of RMcast. Being subject to a Round Robin CPU management policy, it will have to coordinate among operations and can take advantage of the gap (η) between successive redundant transmissions as described in figure 5.5. In this situation, the time needed to multicast a packet copy, albeit considered negligible, need to be taken into consideration and for some high values of π this approach would not be effective, generating another limitation on the use of the Per-

Packet extension approach. The value for such a threshold is proportional to factors such as host workload, network availability and obviously on the value of η which, in turn, increases with the packet loss rate.

Considering what we said so far, we can therefore derive the applicability domain for the Per-Packet extension approach as to environments with a good balance between workload and resources. In practical terms, its execution suits a range of environments varying from high capacity networks[1] (regardless of the workload) to lightly loaded environments with standard resources.

5.6 Concluding remarks

We have developed core protocols for building a QoS negotiable middleware system for reliably multicasting messages of arbitrary size. Their performance was analytically estimated using approximations whose effectiveness are shown to underestimate the performance most of the time - a very desirable scenario for QoS negotiation.

The protocol development involved extending the existing single-packet protocol described in chapter 4 using two approaches, namely Per-Message and Per-Packet. Performance of the resulting protocol, named *multi-packet* protocol, is also compared. In the Per-Message approach, all π packets of a given message are treated as a single logical packet. Its optimal usage is on environments with low packet loss rate and low jitter, as shown by simulation results. Experiments conducted on the PlanetLab testbed showed that such networks

do exist and the assumption of optimality on low-lossy and low-jitter networks is not unreasonable.

The Per-Packet approach, on the other hand, treats each of the π packets independently, with π concurrent instances of the single-packet multicast protocol. Simulations show that this approach finds its optimality in environments where difference in transmission times can be high, although a threshold for the value of π needs to be accounted when independent instances of the single-packet protocol are handled by a single thread in a pseudo-concurrent way. For this reason, the per-packet protocol suits high capacity networks or, alternatively, lightly-loaded applications.

Together the Per-Message and Per-Packet approaches provide a solution applicable to a wide range of environments, and make the QoS-Supportive Reliable Multicast System suitable for real-time contexts.

Chapter 6

Network Performance Measurement

6.1 Introduction

The Network Measurement Component (figure 3.3) is the authoritative source of information on behavior of the CS. Its task is to monitor the network to the extent of producing statistical performance metrics. These are then shared among other components of the system, which interpret that information as an estimation of the behavior of the CS in the near future. This operation is carred out on aregular basis, so that metrics can reflect any change in network behavior.

This chapter provides a detailed description of the Network Measurement Component. The type of information and the techniques used to gather such information will be described. The description will also focus on how information so obtained is processed in order to build the relevant statistical metrics, besides describing in detail the metrics retained to be relevant.

The Network Measurement Component realizes the QoS management interface to comply with specifications of the QoS-Adaptive middleware architecture described in chapter 3. Its role and activity will be discussed in contexts where the CS provides communication facilities on a best effort basis as well as in contexts where it is managed by an *Internet Service Provider* (ISP) using resource management models for the Internet.

6.2 Monitoring the Communication Subsystem

By monitoring the CS, the Network Measurement Component can measure certain metrics concerning performance characteristics of the network. This information is calculated on a regular basis to account for changes likely to occur in the behavior of the CS. The extent of this activity is to provide reasonably accurate information concerning behavior of the CS in such a way to make QoS predictable. Other components in the system will then use this information to account for behavior of the CS in the next future, to be traded against user QoS requirements in negotiation phase.

In certain application contexts, the CS might be managed by an ISP. In these scenarios, the ISP provides guarantees about availability of resoruces in the CS by managing the network based on specific resource management models[47]. Behavior of the CS is therefore guaranteed for the entire period of provision by granting access to a dedicated network environment which makes resources available anticipately to service provision, as shown in figure 6.2. The types and

amount of resources to be provided is specified into one or more agreements, usually encapsulated in what the industry calls *Service Level Agreements*[65] (SLAs).

Agreements establish terms and conditions for the provision, along with penalties to pay in case of violation. Service levels guarantees in SLAs are usually expressed as *average* figures calculated over a period of arbitrary size, or the entire provision period. As an example, the AT&T Managed Services offers 99.99% *average* network availability, a network monthly *average* latency of 60 milliseconds within the US, and an *average* packet loss rate of less than 0.7%. Albeit having statistical nature, performance figures in SLAs typically refer to a prolonged period of time, as for instance the latency in the example above, which is guaranteed on a monthly basis. The time considered for provision of guarantees allows tolerance to fluctuations that might eventually decrease the service level in the short period. The agreement might eventually specify a tolerance threshold, and inherently requires the service provider to compensate to such slowdown in such a way to let the service fall in the range of agreed service with respect to the period considered in the agreement. For instance, consider a situation in the AT& T example above where the (ISP-managed) network experiences an average latency of 90 milliseconds for the first three days of the week. Suppose then that the latency decreases to 30 milliseconds for the rest of the week. Calculated on a monthly basis this would still make the monthly latency fall within the guaranteed 60 milliseconds. However, eventual negotiations performed throughout all the week would account for an average latency

which is not the actual latency experienced. For this reason, such guarantees cannot be considered reliable in the short term. As a consequence, in order to reflect and account for such fluctuations the networked system should consider the network latency performance in the moment the QoS negotiation is being performed.

SLAs also inherently establish terms and conditions for which the agreement can be considered violated with the corresponding penalty to pay. In the above example relative to the AT&T Managed Services, for instance, a monthly average latency of 100 milliseconds (within the US) would be enough to retain the terms soecified on the SLA violated by the service provider. ISPs do not usually provide tools to control that service provision effectively falls in the range of agreed levels. Consequently, where the presence of a *Trusted Third Party* (and/or relative *Violation Detection Service*[40]) is not foreseen, the customer who wants to monitor effective service provision is required to employ external tools.

From what said so far, the monitoring activity is fundamental to sustaining QoS negotiability regardless of the way CS services are provided. For this reason, the Network Measurement component is positioned on top of the CS to so as to abstract the way the CS is managed with respect to the monitoring activity. This can be seen in figures 6.1 and 6.2. These figures depict the monitoring in contexts where the CS works on a best effort basis and is managed by an ISP respectively. In both scenarios, the monitoring activity allows the

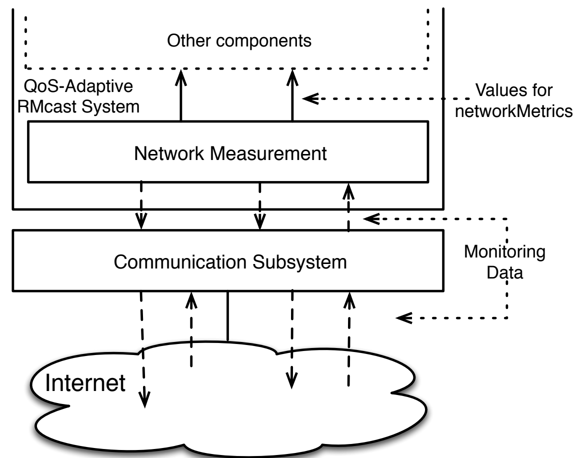

Figure 6.1: Monitoring of a best effort CS.

gathering of up-to-date information about network performance in the form of relevant statistical metrics.

When an ISP manages the CS, the monitoring activity has a twofold purpose. While measuring network metrics and updating other components as in the case of the CS working on a best effort basis, it also ensures that service provision is taking place in a way compliant to the SLA.

Service violations are detected by processing gathered information in a way compliant to terms specified in the SLA, and comparing the so calculated averages with the likewise levels specified in the SLA. In case a violation is detected, the Network Measurement Component might take some action, from simply report the violation to the user application to reporting the violation to a *Violation Detection Service*, if present.

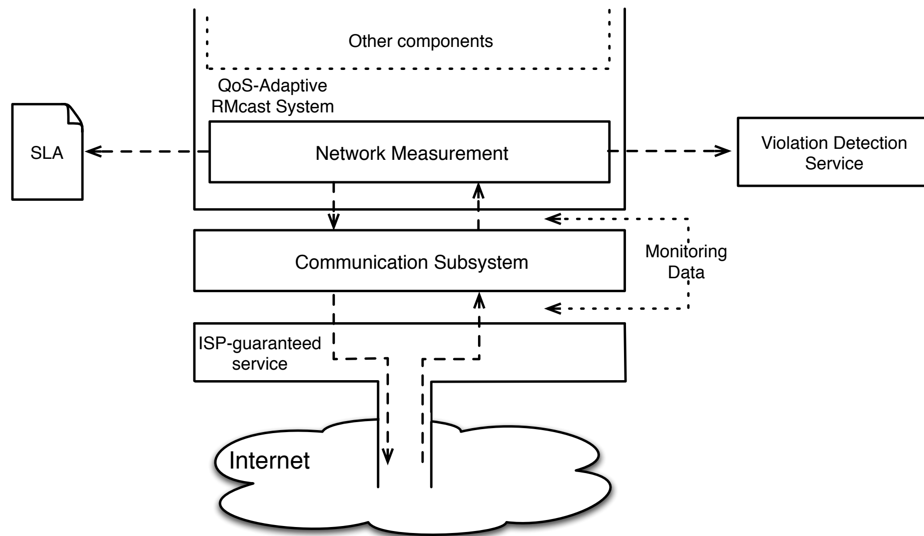

Figure 6.2: Network monitoring over an ISP-managed CS.

6.3 Sampling and Probing Technique

The monitoring strategy employs a technique based on concurrent sampling of each destination. At regular intervals, the channel towards one destination is uniformly chosen and sampled. Sampling is carried out by probing channel connectivity, and involves calculation of the *Round Trip Time* (RTT) towards that destination. This latter is then halved to stimate single way delay time. The RTT is calculated by means of a *three-way ping* algorithm, as shown in figure 6.3. The figure shows a probe session between two processes p_i and p_j. The former is the probe originator, and starts its probing session by sending p_j a $PROBE$ packet. The time at which the packet is sent is recorded as t_0. The packet is received by p_j at time t_1. Upon reception, p_j sends p_i back a $PROBE_ACK$ packet as to acknowledge reception. In addition, this operation is taken as start of a probing session towards p_i, and therefore p_j records time

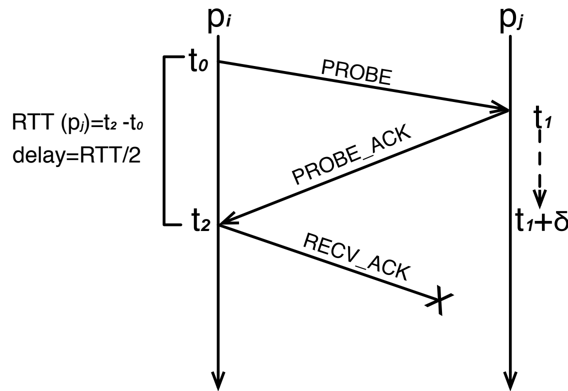

Figure 6.3: Three-way ping probing technique.

t_1. The $PROBE_ACK$ packet is received by p_i at time t_2. At this point, its probing session is complete and p_i can calculate the RTT towards p_j as $t_2 - t_0$. This time is then halved so as to have the delay time towards p_j. In addition, p_i allows for completion of p_j's probing session by sending a $RECV_ACK$ packet. Reception of this latter by p_j would complete its probing session by allowing p_j to perform delay time calculations in the same way as described for p_i. However, the network is prone to failures and a packet might be lost. This situation happens in figure 6.3 when the last packet from p_i to p_j is lost. The possibility of loosing a message is accounted by setting a timeout upon start of own probing session. Expiration is interpreted as the corresponding packet to be lost, and the counter for lost packets is increased. Referring to figure 6.3, missed reception of the $RECV_ACK$ packet triggers the timeout set by p_j to expire, indicated as $t_1 + \delta$, and the packet to be lost. The length of the timeout, i.e. the value of δ, is proportional to the delay time as by previous samples on the same destination, plus the jitter to allow slowdowns.

The Network Measurement Component needs to handle situations where a packet is considered to be lost while in reality it is only unduly delayed. For this purpose, probing packets contain a unique id number and each time a packet is suspected to be lost, its id number is stored in a list along with the sending time. If a packet arrives after expiration of its loss timeout, its id is retrieved in the list, the loss counter is decreased and the probing session continues by calculating the RTT. The list is sorted in descending order based on the sending time so as to improve retrieval by having more recent lost packets at the head, and is periodically pruned to delete entries relative to less recent packets.

The three-way ping technique allows reduction of probing times towards members in a group by combining probe of both members in a single session. In addition, it reduces message overhead by requiring exchange of three packets in a combined session, in opposition to the four packets needed in two distinct traditional ping sessions.

The outcome of the sampling activity is finally stored and, upon expiration of a slot of time, samples are statistically processed in order to calculate the reliability, timeliness and stability performance metrics. The length of the slot of time used to process data influences accuracy of the statistical evaluation and its value is a customizable parameter.

6.4 Metrics of Interest

Behavior of the CS is expressed by reporting reliability, timeliness and stability features. Reliability is expressed through calculation of the packet loss. Timeliness is expressed by the packet delay and the (approximated) statistical pattern such delay follows. Stability, on the other hand, is expressed by the difference in transmission times, i.e. the *jitter*.

Calculation of each of the aforementioned characteristics allows to infer a network performance figure, and is useful to at least one of the other system components. In detail:

- *Average packet loss*: packet loss provides information about robustness of the network environment. On an Internet scale, packet loss can be enhanced by factors such as network workload and cross-boundaries traffic management policies.

  In our system the average packet loss is needed by the Negotiation Component in negotiation phase. Its value accounts for the percentage of packets lost in the considered slot of time, and is calculated by expressing as percentage the ratio between the number of probe packets considered lost, i.e. whose loss timeout is expired, and the total number of probe packets sent.

  The average packet loss value is used by the system to calculate the level of redundancy needed by the RMcast component to achieve negotiated service level, i.e. ρ. In addition, its estimation is also needed when ne-

gotiating latencies.

- *Average packet delay*: packet delay provides information about timeliness of the network. On an Internet context, its value is influenced by the quality of the infrastructure information travels on and service times due to workload of this latter.

  Calculation of the average packet delay allows to estimate the average time needed to reach destination of a probe packet. Together with the *jitter* next to be presented, is the metrics expected to change more frequently. Samples are stored in a table from which the average value is periodically calculated. Once the average value is calculated, the table is then emptied.

  In our system, the value for the average packet delay is required when calculating the amount of time between subsequent broadcasts, i.e. η, in negotiation phase. In addition, its value is needed when estimating conditional delay probabilities whose process is described in next section.

- *Statistical packet delay pattern*: this metric allows to associate the packet delay samples gatehred in the last relevant slot of time to a well known statistical pattern and is a further indicator of the timeliness properties of the network.

  Approximation of the packet delay pattern to a statistical pattern allows to resolve the analytical model contained in the Negotiation Component

into a well known arithmetically tractable set of equations when the system is configured to use this as a way to convert conditional probabilities. A detailed description of how such metric is calculated is provided in the next section.

- *Average Packet jitter*: accounts for the average difference in transmission delays towards a destination over time. The value of the jitter describes stability characteristics of the network. Its value is calculated as the average difference of packet delays within a slot of time. If we consider two delay samples S_i and S_j, take on the same destination at time T_i and T_j respectively, with $T_i < T_j$, then the value $j = S_i - S_j$ is the jitter. Then:

  - If $j < 0$, then S_j was slower than S_i. The network has suffered a *negative* instability, i.e. an instability tending to degrade network performance;
  - If $j = 0$, both samples took the same time to reach the destination, which denotes network stability;
  - if $j > 0$, then S_j was faster than S_i. The network has suffered a *positive* instability, i.e. an instability tending to increase network performance.

The value for the jitter is accounted in the timeout for reception of subsequent message copies. Inclusion of this factor in the timeout for reception

of subsequent message copies allows destination processes to account for network fluctuations, thus offering a primitive form of QoS adaptation.

6.5 Delay Probabilities Calculation

The negotiation process involves evaluation of a stochastic model which contains equations depending on conditional probabilities. In order for the model to be evaluated, the system needs to have a way to translate these occurrences into deterministic equations. As an example, consider equation 4.4 in chapter 4. There, the probability that an operation does not succeed within a certain time x is modeled as $h(x) = q + (1 - q)\mathcal{P}(\xi > x)$. In order for this equation to be numerically tractable, the system must be able to convert the conditional probability of the equation. To this extent, the Network Measurement Component offers two ways of evaluating delay probabilities:

- Statistical evaluation: the set of samples is approximated to a statistical pattern, and the corresponding density function is evaluated in order to calculate the actual conditional probability;

- Experimental calculation: the conditional probability is calculated directly from the table containing the set of samples as the ratio between the number of samples actually satisfying the condition and the total number of samples.

The choice of the right method to use can be made based on limits and guarantees of each one, with respect of the network environment samples are taken

from. Following subsections provide a detailed description of both methods, with their limits and applicability domains.

6.5.1 Statistical Evaluation

Statistical calculation of delay probabilities aims to associate samples in the table to a well known statistical pattern. This information is interpreted by the Negotiation Component as the authoritative source for evaluating conditional probabilities according to the *Probability Distribution Function* (PDF) of the corresponding statistical distribution. In the example of equation (4.4), supposing that the Network Measurement Component detects packet delays to follow an exponential distribution, the equation would be numerically evaluated as $h(x) = q = (1 - q)e^{-x/d}$, where d is the distribution mean which is provided by the average packet delay. If at a later time the component detects packet delays to follow another, different, distribution, then equation (4.4) would be numerically evaluated according to the PDF of that specific distribution.

The actual calculation of the right statistical distribution requires a *goodness of fit* (gof) test to determine how good a set of data fits a specific distribution. Among the possible ones, we chose to use the χ^2-test, given its simplicity and wide use. A full description of this test can be found in [61], and a brief description is produced here for reasons of exposition.

A specific statistical distribution is hypothesized to fit the set of samples in the table, and the χ^2-test is applied for estimating the degrees of confidence in

accepting this hypothesis. To this end, the table is divided into k *buckets* each one referring to a certain delay time. Each of these expects a certain number E_i of samples, calculated as $E_i = N(F(U_i)-F(L_i))$, where F is the cumulative distribution function for the distribution being tested, U_i and L_i are the upper and lower limits respectively. Then, the observed frequency is calculated, by counting the number of samples actually falling into each bucket, as O_i, and the following test statistic is executed:

$$\chi^2 = \sum_{i=0}^{k} (O_i - E_i)^2 / E_i \quad (6.1)$$

This calculation will generate a χ^2 *value* for the set of samples, which will need to be matched with a *critical value of* χ^2 to be found in a sampling distribution table. The right position to look at, in the sampling distribution table, is given by the *degrees of freedom* and the *error threshold*. The former is a measure of how precise estimation of variability is, and is calculated based on the number of buckets the original table has been divided into. The former, on the other hand, states the maximum tolerable error. In the Network Measurement Component is a system parameter that can be dynamically configured.

The gof test is successful if and only if the χ^2 value for the set of samples being considered is larger than the critical value found, which means that the samples table presents a statistically significant relationship between the variables in it.

In this case, the set of samples in the table is assumed to follow the hypoth-

esized distribution, and conditional probabilities in the stochastic model are evaluated according the referring distribution's PDF.

6.5.2 Experimental Calculation

Experimental calculation of delay probabilities is performed by evaluating conditional probabilities experimentally from the set of samples to the extent of satisfying the probability condition. Referring to the example of equation (4.4), experimental evaluation of $h(x) = q+(1-q)\mathcal{P}(\xi > x)$ implies calculation of the percentage of samples in the table that actually satisfies the condition $\xi > x$, where x is fixed and ξ is the sample delay currently being considered.

This evaluation is required during the negotiation phase, and is therefore provided on-demand. The Negotiation Component asks, during negotiation, calculation of a certain conditional probability; the Network Measurement Component calculates such a value from the table and returns the value, which is used by the Negotiation Component to terminate estimation of the stochastic model.

6.5.3 Criticisms

Both ways of calculating conditional delay probabilities are meant to provide reasonably accurate information about timeliness properties of the network. However, they can suffer accuracy weaknesses.

The statistical evaluation of packet delays is based on data contained in the sample table. Samples are hypothesized to fit a statistical pattern, and the gof test evaluates accuracy of such hypothesis. However, the nature of samples in the table might not allow approximation to any known statistical pattern, in which case the error threshold should be relaxed with a consequent loss in accuracy. For this reason, application of this method suits environments where the network is not subject to considerable stability variations.

The same loss of accuracy might happen in the experimental calculation method. In occasions where the network changes its behavior in a way the table does not capture, for instance when a fluctuation is entirely captured into samples of the same slot of time, the table would be partitioned into a first part containing short delays and a second part containing huge delays. As a consequence, conditional probabilities evaluated on such a sample table would not capture the real behavior of the network. In this case, keeping the same sampling rate would cause a loss of accuracy in calculating the delay probabilities, while increasing it to sample the network at shorter times would keep the accuracy stable but increasing the workload of the system.

6.6 Concluding Remarks

We have described the Network Measurement Component together with its techniques to monitor the network and produce average performance metrics in the context of the QoS-Supportive Reliable Multicast System. We have

described its application to a CS that provides best-effort communication facilities, where its task is to act as source of information about network behavior with respect to other components of the system. We have also described how application of the component to a context where the CS offers communication facilities through an ISP brings the added value of verifying that service provision is realized in a way compliant to an agreement specified *a priori*, besides the classical role of source of network-related information.

We have described the three-way ping algorithm at the basis of the monitoring technique, putting emphasis on the how data gathered is processed in order to produce a statistical evaluation of network metrics in terms of reliability, timeliness and stability characteristics by calculation of packet loss, packet delay and jitter respectively. We have also described the two ways, named statistical evaluation and experimental calculation, the Network Measurement Component provides to convert probabilistic equations, present in the analytical model, into deterministic equations to the extent of being automatically evaluated. We have put emphasis on the fact that accuracy of the statistical evaluation method is proportional to the proximity of the jitter to the zero-value, while the experimental calculation method is capable of converting conditional probabilities into deterministic equations with a higher accuracy in environments where the loss rate is slow.

Chapter 7

Prototype Implementations

7.1 Introduction

This section presents and describes two prototype implementations of the single-packet protocol of chapter 4. The first is a middleware suite named *alipes*. The name is taken from the herald of the Olympian gods (also named *Mercury* and *Hermes*), messenger of the gods and known for his cunning and shrewdness. In this, the architecture of the RMcast system is supported by a set of facilities to form a basic GC system. Throughout this chapter, we shall refer to those objects, classes and packages being part of the architecture of the RMcast system as *core* objects, classes and packages. On the other hand, objects, classes and packages being part of the supportive facilities will be referred to as *non-core* objects, classes and packages.

The second prototype implementation has been developed by extracting the core structure from *alipes*, i.e. the single-packet RMcast system, and integrating it into the *JGroups Reliable Communication Toolkit*[13]. Besides providing an example of how to integrate the protocol into a working system, giving birth

to a QoS-supportive version of *JGroups* that we named *QoS-JGroups*, this process brought advantages to the TAPAS (*Trusted and QoS-Aware Provision of Application Services*) EU-IST project[3] as the *QoS-JGroups* system suited perfectly TAPAS needs for a QoS-supportive communication system for clustered application services.

The two prototypes differ essentially in the fact that while *alipes* is a prototype developed *ex novo*, *QoS-JGroups* is a derivation obtained by extracting core functionalities from the former and integrating them as a new object into the already-existing *JGroups* toolkit. Consequently, *alipes* required considerable efforts in both design and implementation phase as compared to *QoS-JGroups*. While describing both prototypes in detail, this chapter will therefore give higher emphasis to the description of the *alipes* prototype.

The structure of *alipes* is described in detail together with interaction schemes driving the inter- and intra-component coordination. Moreover, results of testing focusing on verification of the capability to maintain agreed guarantees are shown. Such testing has been conducted in a real world scenario by means of the Planet Lab[2] testbed.

Results obtained in the testing of *alipes* are assumed to hold for *QoS-JGroups* as well, as they share the same structure and software architecture. Therefore, in describing the latter we shall focus on commenting the phases of integration into the original *JGroups*.

Both prototypes have been implemented using Object Oriented (OO) tools. In particular, the *Unified Modeling Language* (UML)[49] has been used for the

design phase, and its diagrams will be used throughout this chapter to accompany description of concepts.

7.2 Design and implementation tools

Middleware systems are traditionally *Object Oriented* (OO) in nature. OO tools and techniques provide a systematic approach to middleware design which *alipes* takes full advantage of. For example, its structure has been designed by using the *Unified Modeling Language* (UML)[49], and its prototype has been developed by means of the *Java*©[27] programming language.
The choice of Java and UML has been mainly driven by the well known excellent UML-Java interoperability, which made it easy to switch from the design phase to the implementation phase by taking advantage of the possibility to create Java code skeletons from the UML diagrams. Besides this, a variety of other reasons also motivate the choice of UML and Java singularly as modeling and implementation instruments respectively. These reasons are stated below.
UML is becoming a *de-facto* standard tool for designing OO systems. It provides a vast set of diagrams with which it is possible to put emphasis on every functional aspect of the system to be designed. Its diagrams and notations are widely understood and accepted in the Software Engineering community, and its use is regarded to lead to optimal design. However, UML was originally created for design and modeling of large industrial systems, and to this extent

it provides an overwhelming set of diagrams each with customized notation. *alipes*, on the other hand, is a first, basic, prototype and therefore only a subset of UML diagrams, described in the following subsection, has been used in this design.

The choice of Java as the programming language for the implementation has been dictated by the consideration that majority of middleware systems are implemented in Java, and eventual integration would surely take advantage of this. An example of this is offered by the second prototype, which benefits from being implemented in Java since *JGroups* is implemented in Java itself. Moreover, the possibility of eventual integration of subsystems implemented in other programming languages, to extend *alipes* capabilities, is never compromised thanks to the possibility of integrating through the *Java Native Interface*[80] (JNI).

7.2.1 UML Notation

As mentioned above, UML provides an overwhelming variety of diagram types and notations only a subset of which are of interest for our work. In particular, we decided to use *UML 2*[49] notation, rather than the original one, as it is regarded as to have more expressive power than the original version. The remainder of this subsection provides a description of the UML diagram types involved in the design phase of the *alipes* prototype, along with description of the notation used.

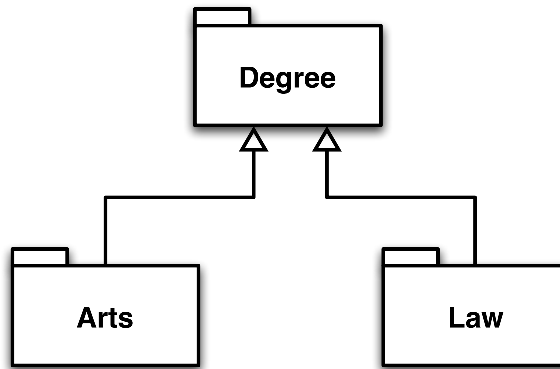

Figure 7.1: Example of UML Package Diagram

For the sake of clarity, throughout this chapter we shall refer to components forming the RMcast system as *system components*, in order to avoid confusion with the UML notion of functional component.

7.2.1.1 Package Diagrams

This type of diagram aims to show structure and dependencies of a hierarchically arranged set of packages. Figure 7.1 shows an example. A Package is indicated as a box with a rectangle on the top left corner, with its name specified in the center of the box. Hierarchy is graphically represented in UML package diagrams by arranging boxes on multiple "levels", as in a in a tree-like structure, with the root package at the top.

Referring to figure 7.1, the `Degree` package is the root package. The `Arts` and `Law` packages are *specializations* of the `Degree` package, to indicate that "arts" and "law" are distinct types of the same "degree" object. This relationship

is graphically represented by an arrow starting from each of the specializing packages and terminating in the super-package.

7.2.1.2 Component Diagrams

UML Component diagrams are used to highlight the technical aspect of the software architecture. In particular, they lay emphasis on the structure of the components and detail the component-to-component and component-to-interface dependencies.

In these diagrams, components are modeled as simple rectangles. A symbol (a rectangle with two smaller rectangles jutting out from the left-hand side) on the top right corner is used as a visual stereotype to indicate that the rectangle represents a component. Alternatively, a "<<*component*>>" inscription is also widely accepted as indication of the nature of the box. This inscription can be replaced with a more precise indication of the nature of the component, whereas this is known in advance. Figure 7.2, which will shortly be described, provides an example of four components whose nature is known and indicated as <<`GUI`>>, <<`Infrastructure`>> and <<`Database`>>.

Components may require and provide interfaces. An interface is a definition of a collection of one or more methods, and zero or more attributes, that define a cohesive set of behaviors. A provided interface is modeled using a *lollipop* notation, i.e. a straight line ending with a circle, while a required interface is modeled using the *socket* notation, i.e. a straight line ending with an open half circle. Interfaces are tied to the component which provides them (for provided

interfaces) or requires them (for required ones) through *ports*.

A port is a feature of a component that specifies a distinct interaction point between the component and its environment. Ports can be named, and can provide unidirectional or bi-directional communication. In a diagram, ports are typically shown whenever necessary, while are omitted when not visually influent to the component. When visualized in a component, a port is shown as a square box from which interfaces start. An example of component diagram with ports can be found in figure 7.9.

Interaction relationships between named interfaces and components are graphically represented as connecting arrows, and presence of an arrow connecting two boxes imply a message exchange across connectors. When the nature of the relationship is known in advance, the arrow relationship can be stereotyped. In the context of this work we have modeled relationships between ports and internal classes, when known in advance, in the following ways:

- delegation relationship: represented as a line with an open arrowhead, indicates that the object the arrow starts from delegates a task to the object the arrow points to;

- realization relationship: represented as a dashed arrow with a closed arrowhead, indicates that the pointed port realizes a service needed by the class the arrow starts from

Figure 7.2 shows an example of UML Component Diagram where four components, namely `Web Browser`, `Credit Check`, `Shopping Cart` and `MySQL` are

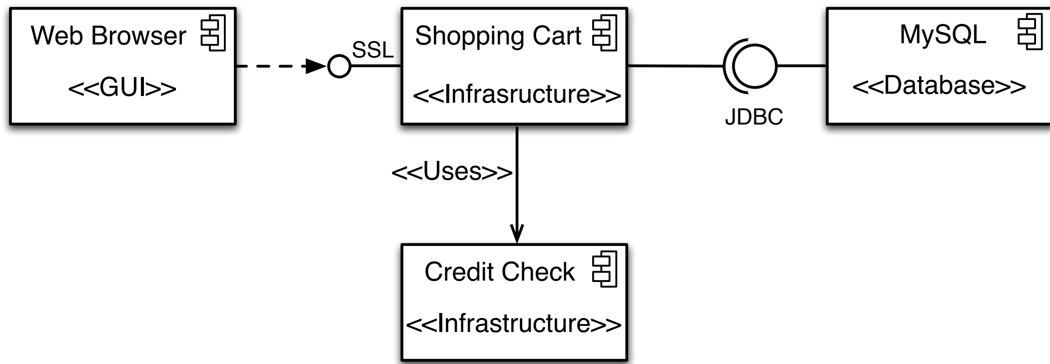

Figure 7.2: Example of UML Component Diagram

engaged in an interaction. The components are self-describing by their names, and allow a user to use an online shopping cart. The web browser is a component, of type GUI, which interacts with the shopping cart, of type Infrastructure. Storing of purchased items into the shopping cart is made secure through the use of the *Secure Socket Layer*[50] (SSL) protocol, and the dashed arrow connecting the web browser to the SSL interface exported by the shopping cart indicates that the former component realizes the latter service (i.e. the shopping cart) through the use of an SSL socket. The shopping cart, in turn, uses a credit check engine to verify the financial credits of the user when this latter uses a credit card as payment. This interaction is emphasized by the stereotyped delegation relationship between the shopping cart and the credit check engine.

Items in the shopping cart are stored into a database. Figure 7.2 shows this by the shopping cart component requiring a JDBC (*Java DataBase Connectivity*[87]) interface, which is provided by the MySQL component of type

Database.

7.2.1.3 Sequence Diagrams

Sequence Diagrams show interactions between objects based on the timing and order they occur. In this type of diagram, each class is represented with a box containing the object name and type in the traditional *name* : *type* UML format. Underneath boxes, a *life line*, denoted by a dashed line, describes the object life cycle in time, and is traditionally interpreted in a top-down way. Interactions, which might be method invocation, RPCs or simple data exchange, are represented as arrows from the requesting object's the life line to the requested object's life line. The shape of the arrow denotes the synchronous/asynchronous nature of the interaction: solid arrowheads denote synchronous interactions, while open arrowheads denote asynchronous interactions. Dashed arrows with open arrowheads denote return messages, while arrows starting and terminating at the same object denote a processing activity which takes place internally to the object. When an interaction request is received, a vertical rectangular box starting just after the request, i.e. the terminating arrow, and overlapping the life line, indicates the object *activation time*, i.e. the time at which an object is activated with respect to the scenario under examination.

Figure 7.3 shows an example, where a sequence diagram is used to describe the interaction of objects involved in a remote system for students enrollment to exams. The example shows a student named J. Smith enrolling for the CSC822

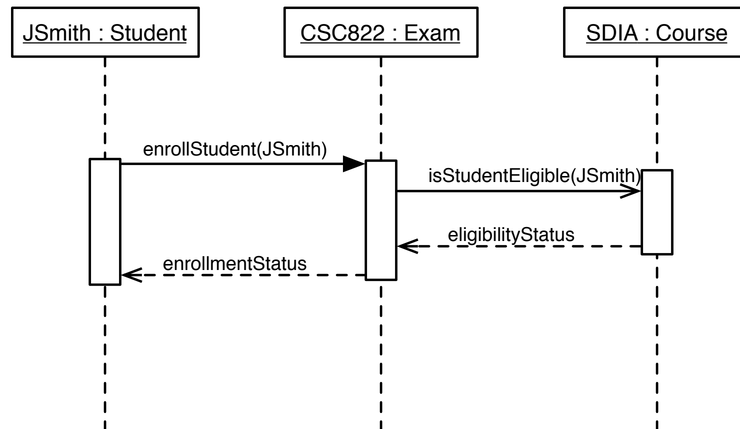

Figure 7.3: Example of UML Sequence Diagram

exam of the SDIA course. The scenario is modeled by the UML sequence diagram in figure 7.3 as follows. Each of the entities involved is represented by a class, whose name and type is specified inside the box representing the class. The class representing the student in the system asks for enrollment via invocation of an $enrollStudent(JSmith)$ method. Note the synchronous nature of the interaction. Invocation implies interaction with the CSC822 object of type Exam. Before granting enrollment, CS822 checks whether the student is admitted to the exam or not. To this extent, CSC822 interacts (asynchronously, as specified by the arrow type) with the SDIA class. This latter, of type Course, checks whether the student belongs to the SDIA course or not. The actual interaction is not showed in the figure for the sake of space. After completion of the check, some time after the request, the SDIA class returns an $eligibilityStatus$ stating whether the student is eligible for the exam or not. This object is received by CSC822 which, based on its value, returns an $enrollmentStatus$ data. Please note that the activation time of CSC822

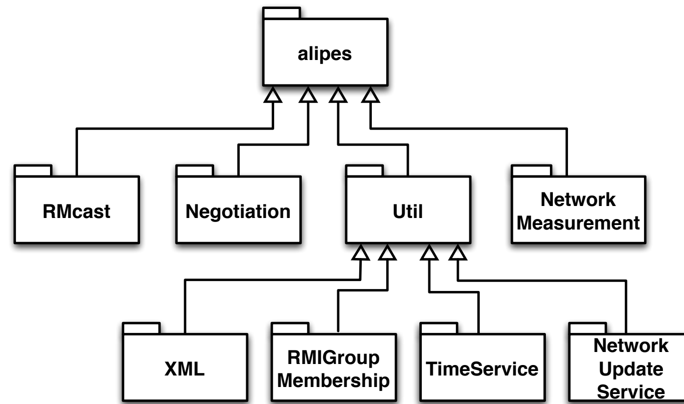

Figure 7.4: Package structure for the *alipes* prototype

and SDIA objects is limited to the time when interactions trigger processing.

7.3 *alipes*

The *alipes* system is designed around the three main system components described in previous chapters. Its logic is contained in a hierarchically arranged set of packages, whose structure is shown in Figure 7.4. The root package `alipes` contains the `Negotiation`, `RMcast`, `NetworkMeasurement`, and `Util` packages. The first three packages contain objects and classes for the corresponding system component, while the last package provides objects which setup the environment and execution of the system. More precisely, the `Util` package contains the following sub-packages:

- `XML`: XML data parsing subsystem, to allow utilization of *alipes* as plugin in systems using XML data format.

- `RMIGroupMembership`: dynamic GM protocol subsystem, based on RMI[94].
- `TimeService`: subsystem to notify processes of timeout expiration.
- `NetworkUpdateService`: subsystem to notify processes of network metrics updates.

All above subsystems have been separated from the main protocol, although essential for the execution, to the extent of decoupling their actual implementation from the core components. These are therefore used, in *alipes*, as plugin systems.

7.3.1 Core Components

The system contains a front object named `Container`. Its purpose is to abstract system's logic away by allowing access to the communication primitives in a transparent and safe way, while optimizing coordination among system components by easing the user from direct handling of components. The container prevents misuse of the communication primitives by checking that sensitive communication primitives are not invoked before termination of a successful negotiation. Figure 7.5 shows the interaction scheme of the container with the main system components.

The container has a reference to each system component through one or more interfaces defining the service provided. Functionalities exported in each interface are as follows: the Negotiation Component exports primitives to access

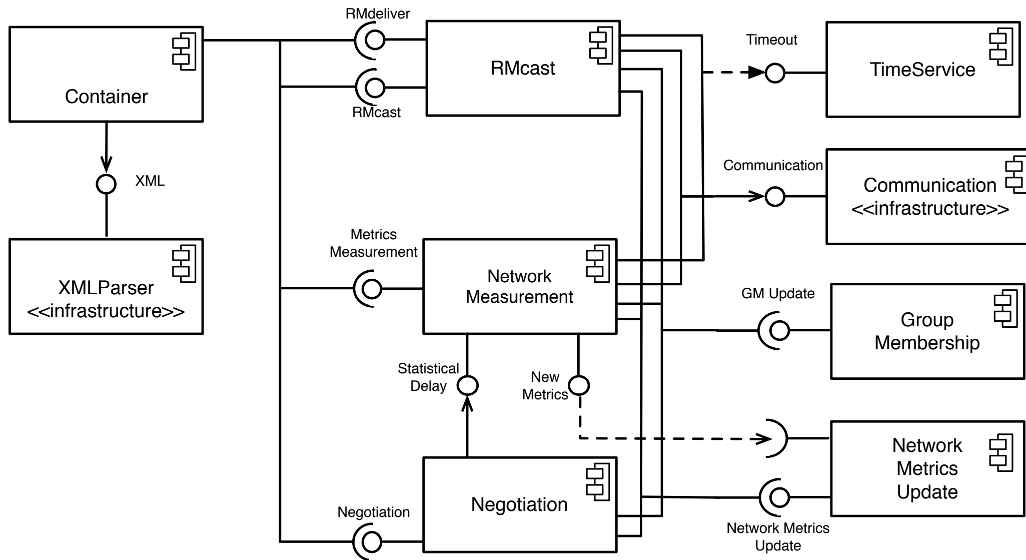

Figure 7.5: System components interaction scheme

negotiation facilities and obtain results; the RMcast component exports primitives concerning the RMcast and RMdeliver operations, while the Network Measurement Component provides primitives for measuring network metrics. System components make use of services provided by subsystems in the `Util` package mentioned above, which add specific capabilities needed for their correct execution. Figure 7.5 shows components providing the services in question; the `TimeService` component brings the logic for handling timeout expiration asynchronously; the `GroupMembership` component handles GM updates, while the `NetworkMetricsUpdate` component handles reception of updates concerning network metrics. These components will be described in detail in subsection 7.3.2.2.

`XMLParser` and `Communication` infrastructures represent components abstracting subsystems providing capabilities totally external to the framework. In par-

```
public class Container{
    public void requestNegotiation(double lat,
                                   double reliab,
                                   String lat_type){...}
    public boolean getNegotiationResult() {...}
    public void RMcast(Object obj) {...}
    public Object RMdeliver(){...}
    public void killMe(){...}
}
```

Figure 7.6: Structure of the `Container` class.

ticular, the `XMLParser` infrastructure abstracts the use of an XML parser to allow the user to submit negotiation requests in XML format. The `Communication` infrastructure, on the other hand, abstracts the use of a lower level Communication Subsystem with which messages are sent to the Internet.

7.3.1.1 Container

The `Container` object is a convenient way to encapsulate main system functionalities. Encapsulation provides a single-object abstraction in *alipes*. This eases the user from having to deal with system components directly, also preventing misuse by monitoring access to sensitive primitives (for example checking whether the user has successfully negotiated the service level every time it accesses the RMcast primitive).

The structure of the Container class is shown in Figure 7.6. Referring to this figure, purpose of each primitive is described as follows:

- `requestNegotiation(double lat, double reliab, String lat_type)`:

is used by the user to request a negotiation. In the invocation, the `lat`, `reliab` and `lat_type` input parameters represent user's QoS level requirements in terms of latency amount, reliability percentage, and latency type respectively. Timeliness-related input parameters, i.e. `lat` and `lat_type`, can remain unspecified in case the user is interested in negotiating reliability only.

- `getNegotiationResult()`: provides the user with the result of a negotiation. This latter primitive is blocking in a way that invocation while the negotiation is still being performed blocks the user until termination.

- `RMcast(Object obj)` and `RMdeliver()`: are invoked by the user to start execution of the RMcast and RMdeliver operation respectively. Invocation of the `RMcast(Object obj)` is subject to a check on a prior successful negotiation. In case of negative answer, the container denies access to the primitive throwing an exception.

  The `RMdeliver()` primitive, on the other hand, allows the user to have delivered messages RMcast from other members of the group, and does not need to undergo through the same check.

- `killMe()`: its invocation causes the system to die by starting a procedure that recursively stops threads and deletes objects in all components. Once all components are stopped, the container (and therefore the system) becomes empty. Invocation of this method has also implications in terms of group management, and equals to user leaving the group.

Note that the `RMdeliver()` primitive does not give direct access to the real message reception primitive, but rather allows the user to have delivered messages eventually arrived from other members' RMcasts in the interval time between instantiation of the container and invocation of the RMdeliver primitive. While, in fact, this time can be retained negligible in the wide majority of cases, under certain conditions such as extremely high workload, it might be prolonged. *alipes* manages these cases by instantiating the lower level reception subsystem, part of the RMcast component, in the container instantiation phase.

The lower level reception primitive, described later in this chapter, handles the receiving-side of the protocol and stores messages from other members' RMcast operations in a storage buffer, which the container object accesses in order to deliver messages to the user.

The rationale behind the choice of separating semantics of the two described reception primitives is that instantiation of the container allows the system to be retained an operative member of the group, and as such it is expected to be able to receive messages. However, once the message has been received user delivery might be delayed for many reasons. Consider an example where the system is used to provide information reliably and timely in an online pay-per-view live streaming service, where the client is charged based on the service level requested. The client, i.e. the user, negotiates the QoS level desired with the *alipes*-based server before starting reception of the broadcast. After successful negotiation, the server might need to wait, before starting the

actual provision, for an online check of client financial credentials. Technically speaking, the client is eligible of receiving information, i.e. would become operative, after successful negotiation but online check of financial credentials might delay the user receiving the actual information.

Instantiation of the container leads to instantiation of the whole system, code in figure 7.7 shows. Objects interested by the instantiation are the ones whose functionalities do not depend on the negotiation process, such as the Network Measurement Component (named `nmc` in figure). In addition, the `Container` object partly instantiates the architecture of the Negotiation and RMcast components (indicated as `nc` and `rmc` respectively in the figure). Objects from the former component are instantiated to allow prompt initialization steps for setting up negotiation facilities, while objects from the latter component are instantiated to the extent of applying the RMcast protocol to incoming traffic, as mentioned earlier. In particular, this step implies invocation of the primitive to access the receiving logic of the protocol. Finally, a *Group Management Update Receiver* is instantiated to allow update of information concerning the group membership.

7.3.1.2 Negotiation Component

The architecture of the Negotiation Component is split into two main parts. The first contains the structures needed to setup the negotiation process, and its instantiation is included in the instantiation of the container. The second

```
public Container(int recvPort) {
    // Instantiation of the Network Measurement Component
    nmc = new NetworkMeasurement.ComponentImpl();
    nmc.measure();

    // Instantiation of the Network Update Service
    static Service nus = new ServiceImpl();
    nus.startMe();

    //Partial instantiation of the Negotiation Component
    nc = new Negotiation.ComponentImpl((ServiceImpl) nus);

    //Partial instantiation of the RMcast Component
    rmc = new ComponentImpl(receive_port, (ServiceImpl) nus);

    // Instantiation of the GM Update Receiver
    GMUpdateReceiver groupUpdater = new GMUpdateReceiver(this);
    groupUpdater.startMe();
}
```

Figure 7.7: Initialization of the `Container` object.

part, on the other hand, contains the objects by means of which the stochastic model can actually be estimated. Instantiation of this part takes place on-demand upon request of a negotiation by the user, which can access the negotiation primitive, along with other primitives, through the negotiation interface exported.

The structure of the negotiation interface is shown in Figure 7.8. More detailedly:

- `negotiate()`: invoked by the container when the user requests a negotiation, allows the user to request a negotiation process. Invocation of this primitive causes instantiation of objects and components, needed for the

```
public interface Component {
    public void negotiate();
    public boolean getNegotiationResult();
    public int getRho();
    public double getEta();
}
```

Figure 7.8: Structure of the Negotiation `Component` interface.

negotiation, instantiated on-demand.

- `getNegotiationResult()`: returns the negotiation result as a boolean value. A `true` return value indicates that the negotiation is successful (and therefore the requested QoS level can be achieved), while a `false` return value indicates that negotiation did not succeed, and consequently the user requested QoS level cannot be achieved. This primitive is blocking in a way that blocks the invoker if the negotiation process is not yet terminated. In case this primitive is invoked *before* a negotiation request, the Negotiation Component replies by throwing an exception. Conditions leading to successful negotiation are discussed later in this section.

- `GetRho()`: returns the value for the level of redundancy (i.e. ρ), generated during negotiation. Its invocation is subject to a prior successful negotiation, and invocation before termination of a negotiation or after a non-successful negotiation causes the Negotiation component to throw an exception.

- `getEta()`: returns the value for the interval time between retransmis-

sions (i.e. η). As for the `GetRho()` primitive above, its invocation is subject to prior successful negotiation, and invocation before termination of a negotiation or after a non-successful negotiation is replied with the Negotiation Component throwing an exception.

The architecture of the Negotiation Component is described in Figure 7.9. The `NCFacade` object has the task of providing a *façade* for the component. Negotiation requests are received by this class and delegated to the `Negotiator` class. This latter is instantiated together with the container and allows coordination among eventual multiple negotiation processes. Negotiations are carried out by instantiating `Negotiation` objects on a per-request basis and delegating negotiation estimations.

`Negotiation` objects perform the actual analytical estimations and provide the `Negotiator` with return values specifying latency delay and reliability values. The `Negotiator` thus compares these values with the user-requested ones in order to carry out feasibility analysis as stated in section 4.5.2. As part of the negotiation process, the component needs to ask conversion of conditional probabilities into numerically tractable equations. To this extent, the Network Measurement Component is contacted through the `Statistical Delay` interface and asked for the conversion.

The negotiation process requires knowledge of the size of the group and current network conditions. This information needs to be provided on a reg-

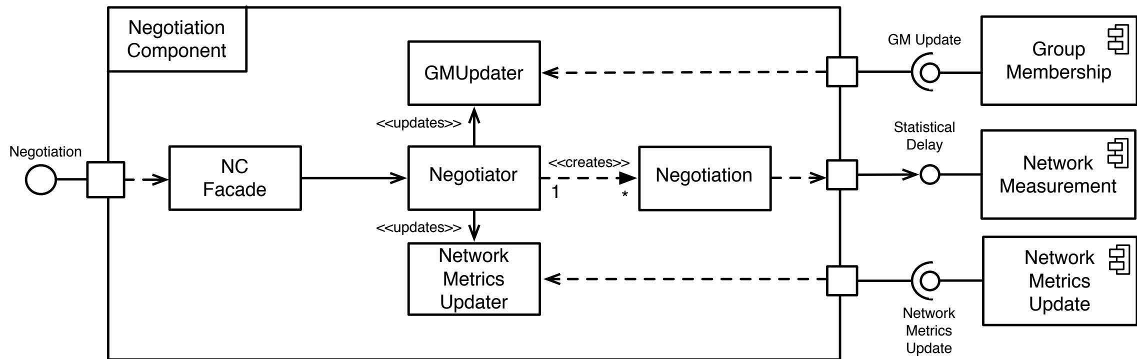

Figure 7.9: Component diagram for the Negotiation Component

ular basis in such a way to be accounted at any time a negotiation request is made. In the Negotiation Component this task is achieved by the `GM Update` and the `Network Metrics Update` interfaces respectively.

The `GM Update` interface allows interaction with the GM subsystem. This latter sends group views whenever an event modifying the group size, such as joins and/or leaves, takes place, and will be described in section 7.3.2. This information is then received and handled by the `GMUpdater` object, to which the `Negotiator` delegates the task of amending the local group view according to updates received.

On the other hand, the `Network Metrics Update` interface allows interaction with the Network Measurement Component to the extent of receiving up-to-date network metrics. This information is received by the `NetworkMetricsUpdater`, to which the `Negotiator` delegates reception and management of updates. Once all the above information becomes available, the negotiation process can start and the analytical model evaluated.

The negotiation fails if at least one of the estimated QoS metrics values is

smaller than the required ones. On the other hand, generation of values which are larger than (or equal to) or equal to values required by the user is taken as a guarantee that the corresponding service level can be achieved. Negotiation is thus considered successful and the system accepts multicast responsibilities. Figure 7.10 shows a UML sequence diagram depicting a negotiation request scenario. Referring to this figure, the container (represented by the `MyContainer` object) invokes the `negotiate()` method to start the negotiation process. This request is then received by the `NCGate` object and forwarded to the `UserNegotiator` class that spans a `Negotiation_i` class specifically for the ith operation. Once negotiation terminates the `Negotiation_i` class returns the result, which travels back to `MyContainer`.

Knowledge of certain network metrics allows the analytical model to take into account present network conditions when estimating behavior of the protocol in the next future. In the context of the Negotiation Component, metrics used are:

- average packet delay;
- average packet loss;
- average jitter;

All these parameters are directly involved in the process of analytically estimating protocol performance. In addition, the Negotiation Component can demand the Network Measurement Component an evaluation of conditional

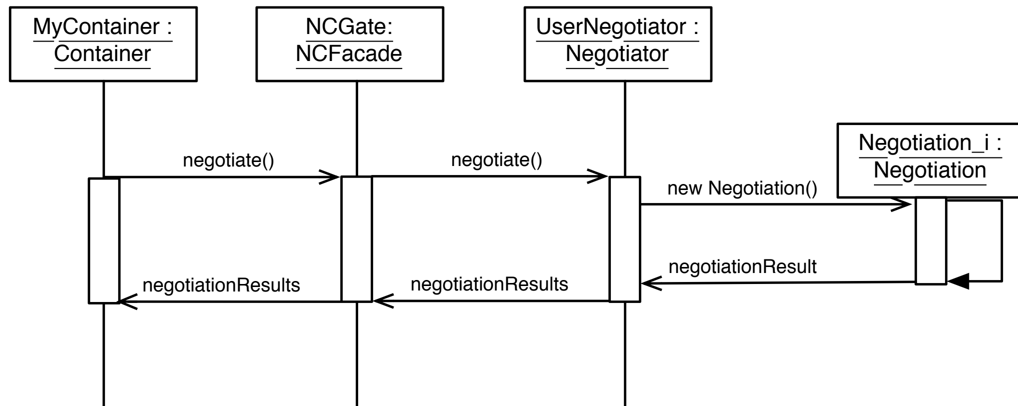

Figure 7.10: Sequence diagram for the negotiation process

delay probabilities. This process is described in more detail in chapter 6.

7.3.1.3 RMcast Component

The RMcast Component offers the QoS-supportive RMcast service through a configurable protocol. Its structure is divided into sending and receiving part, which are well separated.

The sending part of the protocol is offered through the RMcast primitive upon user's request, and only after successful negotiation. As a consequence, its architecture is instantiated dynamically upon user request.

The RMdeliver operation, on the other hand, needs to handle reception of messages from other members from the very first moment when the local host becomes operative, with respect to the multicast responsibilities. The object handling this part is invoked exactly once at the very beginning of the system lifecycle. Instantiation of the handler for the RMdeliver operation is therefore

```
public interface Component {
    public void RMcast(Object obj);
    public void RMreceive();
    public void initRMcast(int rho, double eta);
}
```

Figure 7.11: Structure of the RMcast `Component` interface.

part of the instantiation of the container.

The structure of the interface exported by the RMcast Component is shown in Figure 7.11, and methods there defined have the following purpose:

- `RMcast(Object obj)`: applies the sending part of the protocol to the input parameter `obj`, as described in figure 4.1. As mentioned earlier, its usage depends on a successful anticipate negotiation to be performed before invocation. In case this dependency is not satisfied, any attempt to access it will result in the container throwing an exception.

- `RMreceive()`:allows the system to apply the receiving side of the protocol to incoming relevant traffic as described in figure 4.2.

- `initRMcast(int rho, double eta, double jitter)`: used to setup the initial environment for the RMcast operation. Invocation of this primitive triggers parameters generated in the negotiation phase, i.e. level of redundancy ρ and interval time between subsequent retransmissions η, to be established as global protocol parameters and therefore assumed in the RMcast environment. The above data is provided as

input parameter upon invocation.

The software architecture of the RMcast Component is shown in the diagram in Figure 7.12. The `RMdeliver` and `RMcast` interfaces are interaction points belonging to the same interface provided by the RMcast Component, and allow the container to interact with objects handling the RMdeliver and RMcast operations respectively.

The RMdeliver operation is handled by the `RMreceiver` object, shown in figure 7.12, in the following way. The `Receiver` object receives message copies through the `Communication` interface. These are forwarded to the `RMreceiver` object which applies the receiving part of the RMcast protocol. The message is then delivered to the user by this latter object through the `RMdeliver` interface. Both the `RMreceiver` and the `Receiver` objects are a single-instance object in the local system. Whatever the number of concurrent RMcast operations taking place among the group in a certain moment in time, all message copies are received by the same objects.

The `RMreceiver` object needs an interface to handle expiration of timeouts in an asynchronous fashion. In *alipes*, this is provided by delegating such duties to the the `TimeService` component through the `Timeout` interface. The latter component is external to the RMcast Component, and is described in detail in section 7.3.2.3.

Expiration of a timeout triggers the RMcast Component to try to appoint the local host as new broadcaster for the remainder of the current RMcast

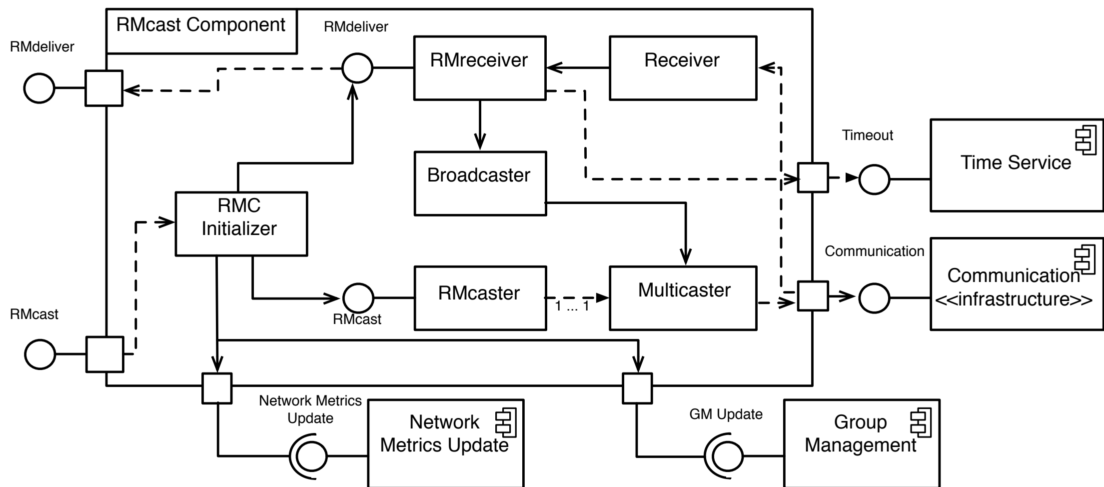

Figure 7.12: Component diagram for the RMcast Component

operation. This operation, in turn, involves broadcast to the group of the last received message copy as per protocol specification in chapter 4. In order to perform this operation asynchronously, the `RMReceiver` object delegates a `Broadcaster` object which, in turn, delegates the actual broadcast operation to a `Multicaster` object, which will be described shortly in the context of the RMcast operation.

The structure of the object handling the RMcast operation, which can be seen in figure 7.12, is slightly more complex than the receiving side. Requests of RMcast operations pass through the `RMCInitializer` object. This has the task of initializing the component upon instantiation and making sure that the environment is properly configured whenever an RMcast operation is requested. Activity of the `RMCInitializer` involves knowledge of up-to-date network conditions as part of the setting of the RMcast execution environment. As a consequence, it handles network metrics and group management updates

through the `Network Metrics Update` and `GM Update` interfaces respectively. As for the case of the Negotiation Component, this information is provided by the Network Metrics Update and the GM subsystems. When a message is forwarded, through the `RMcast` interface, the `RMCInitializer` checks in particular that global parameters are correctly set and, once all checks terminate successfully, delegates the `RMcaster` object for the actual RMcast operation.
Data to be RMcast is handled by the `RMcaster` object through the `RMcast` interface. This object is in charge of making sure that the multicast operation complies to the RMcast specifications (described in chapter 4), and delegates the actual transmission to the broadcasting primitive. The `RMcaster` invokes $\rho + 1$ times the broadcasting primitive, making sure that each transmission is made after η time.
The actual broadcast operation is performed by the `Multicaster` object, which carries out the unreliable broadcasts by concurrent (unreliable) unicast transmissions of UDP datagrams. The transmission is carried out through the `Communication` interface, which delegates the communication to the corresponding infrastructure.
The sequence diagram in figure 7.13 shows how the RMcast operation is performed. The `Initializer` object forwards the request of RMcast operation to the `RMcaster` object which reliably multicasts the message based on the RMcast protocol specifications. This implies using a `Multicaster` object to perform the $\rho + 1$ through the `Communication` infrastructure.

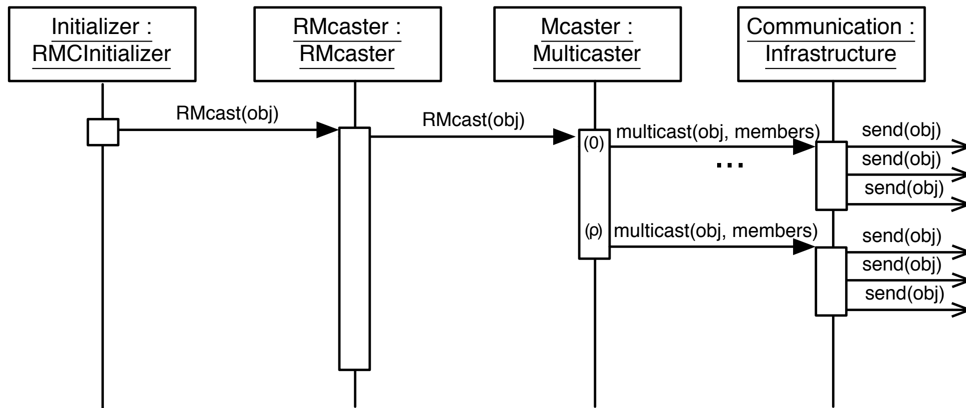

Figure 7.13: Sequence diagram for the RMcaster

7.3.1.4 Network Measurement Component

The Network Measurement Component measures the following:

- average packet delay;
- average packet loss;
- average packet jitter;
- packet delay statistical pattern;
- experimental evaluation of conditional probabilities.

As explained in the previous chapter, the Network Measurement Component considers the path towards each destination as a *channel*. The component probes each of these by means of the three-way ping algorithm in order to obtain the RTT, which is then halved to have the packet delay towards that particular destination. The value obtained is considered an estimation of the packet delay towards that particular destination, and is stored on a table of

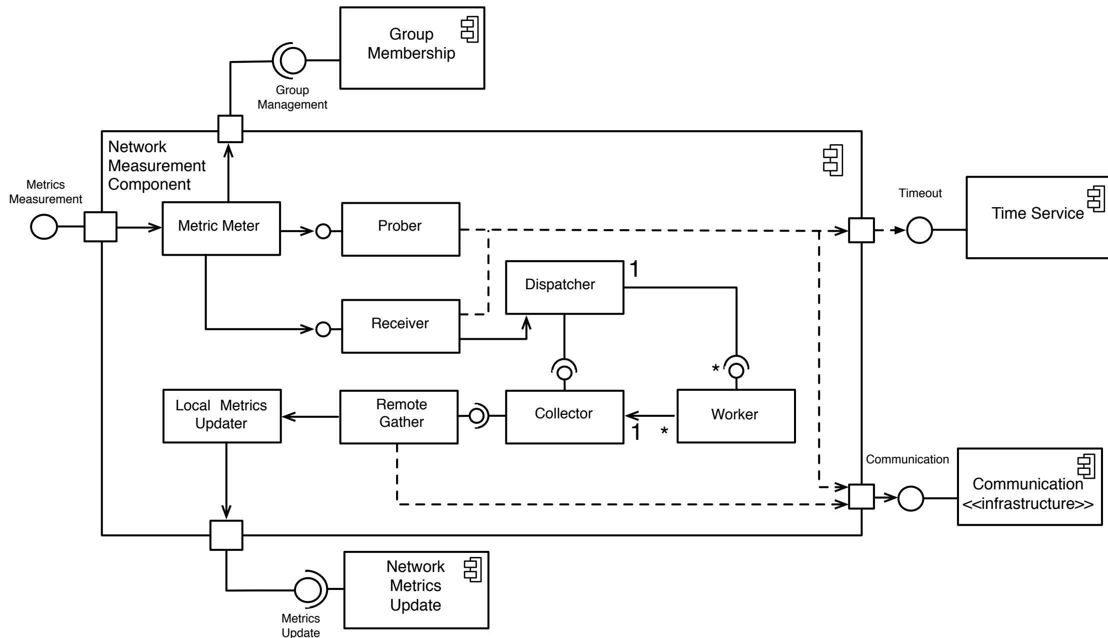

Figure 7.14: component diagram for the Network Measurement Component.

samples. Every (configurable) fixed amount of time, information on the table is processed in order to obtain average metrics.

The Network Measurement Component exports two interfaces, as shown in the component diagram of Figure 7.14. The first, identified in the figure as `MetricsMeasurement`, exports a single method to be invoked in order to start monitoring the network. The second, named `MetricsUpdate` in the figure, provides methods to share updates of network metrics.

The fundamental object in the structure of the Network Measurement Component is the *Metric Meter* object, which provides network behavioral information on a per-packet basis.

The Metrics Meter delegates the 3-way ping probing sessions described in chapter 6 to `Prober` and `Receiver` objects. To this end, it provides both with

the list of destinations whose connecting channel is to be monitored. The list is kept consistent by updates through the `GMUpdater` component. This interaction is realized through the `GroupManagement` interface, as depicted by the interaction point with the same name in figure 7.14.

The `Prober` simply selects a channel from the list and targets a unicast probe packet (`PHASE1` in the description on chapter 6). The actual transmission relies on the lower level communication facilities, indicated by the `Communication` infrastructure in figure 7.14 and realized in *alipes* by means of UDP[95] datagrams. Transmission of the probe packet starts the probing session, which triggers recording of the sending time and setup of the loss timeout mentioned in chapter 6. Similarly to the RMcast Component, the Network Measurement Component makes use of a `TimeService` subsystem to handle expiration of timeouts asynchronously, and is shown in figure 7.14 by the interaction through the `Timeout` interface.

The receiving side of the probing session is handled by a `Receiver` object, instantiated along with the `Prober`, which is in charge of receiving probe packets and acknowledging them when needed. When the packet received is a probe packet (`PHASE1`), the `Receiver` simply acknowledges with a `PHASE2` packet and records sending time of this latter packet. This action triggers start of probing session of the host acknowledging the `PHASE1` packet, with the setup of the corresponding loss timeout. When the packet received is a `PHASE2` packet, on the other hand, the `Receiver` records reception time (as this concludes own probing session), and further acknowledges with a `PHASE3` packet, allowing the

destination process to complete its own probing session.

Once a session on a channel is completed, the `Receiver` delegates a `Dispatcher` the processing of gathered data. This has the task of selecting the `Worker` for the coresponding channel and forwarding the data for the processing.

`Worker`s are in charge of performing the actual calculation of the RTT from data gathered on a probing session and, to this end, are instantiated on a per-channel basis. A worker calculates the average delay time towards the other end of the channel it has been instantiated for and, once calculated, forwards them to a `Collector` which, as the name suggests, collects data for each channel and selects local worst values for the relevant metrics.

The stochastic model contained in the Negotiation Component is evaluated under pessimistic assumptions as specified in 4.5, and therefore the group-wide worst values need to be considered. Local worst values are then disseminated among other group members in order to select the group-wide worst. This task is performed by the `Remote Gather`, which achieves this by using the `Communication` infrastructure. Group-wide worst values are determined by comparing, upon reception, other members' worst values with own local ones and taking the worst. This update process is concluded, soon after determination of the current group-wide worst values, by the notification to all other local system components. This last task is delegated to the `LocalMetricsUpdate` object, which realizes the service offered through the `MetricsUpdate` interface to provide the Network Metrics Update component the metrics to be spread out among other local system components.

7.3.2 Non-core components

The `Util` package contains subsystems providing a set of functionalities which, albeit necessary to the correct execution of the RMcast system, are intended to the use as "plugin" subsystems. As mentioned previously, this choice is motivated by the need to decouple definition of these object from their actual implementation. The remainder of this section provides a more detailed description of each of these components.

7.3.2.1 XML parser

The `XML` package contains objects and classes which allow *alipes* to engage exchange of data structured based on the XML[23] specification. In particular, the package contains an XML parser capable of extracting the logical structure of data in a *Document Object Model*[11] (DOM) format. Information can then be easily extracted from the DOM object and used by the RMcast system.

The XML format foresees data to be structured based on a hierarchy of XML *tags* which in turn, needs to be defined in a *Data Type Document*[23] or XMLSchema[46] against which the parser validates the structure of the XML file. Obviously, *alipes* provides a basic (XML-based) mini language that can be used to communicate with the system when XML is the method chosen.

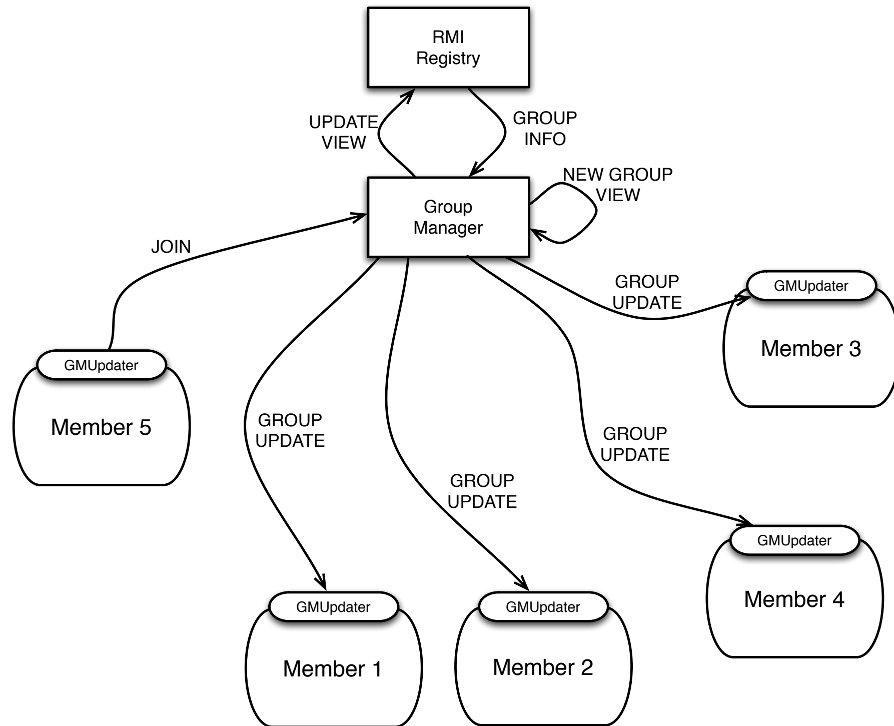

Figure 7.15: Group Management protocol

7.3.2.2 Group Membership Protocol

The `RMIGroupMembership` package implements a basic Group Membership (GM) protocol. Its aim is to build updated group views each time an event such as joins/leaves modifies the structure of the group, and to share these new group views among other group members so as to keep them consistent. Figure 7.15 shows the dynamics allowing the GM protocol to keep each member's group view consistent.

The GM protocol is composed of a *Group Manager* and *Group Membership Updater* (indicated as `GMUpdater`in figure 7.15). The former object governs a centralized RMI[94] registry, which allows persistent storage of information

on group. Information stored includes members' identity and location. In addition, the Group manager is also in charge of building group views to be shared among group members. Such group views are built each time an event outdates the previous view (such as new joins, leaves, etc.).

Up-to-date group views are shared among group members through the Group Membership Updater. The updater is instantiated by the container of each group member, and has the role of receiving new group views and passing them to the container for update of the local system components.

Instantiation of *alipes* on a host involves, in first instance, joining a group as new member. This action can be seen performed, in figure 7.15, by Member 5. The action of joining a group is performed by notifying presence to the Group Manager, whose address is made available *a priori*. This operation typically implies the new member to register a unique id in the registry, and the canonical $IPAddr : port$ format is used. Every time a new host joins, the Group Manager stores this information in the RMI registry and updates its group view. The new group view formed in this way is then shared among other group members by transmitting the view itself to each member's Group Membership Updater.

Similar dynamics is employed when a member leaves the group voluntarily: a leave notification is automatically sent to the Group Manager by the `Container` object when the user invokes the `killMe()` method on a node. As in the case of a new member joining, this action triggers a modification of the group structure. A new group view is then created and sent to the Group

Membership Updater of each member.

The Group Manager periodically pings the `GMUpdater` of each member for liveness, and this allows to detect members leaving the group due to failures. At destination, views are received by the local instance of the `GMUpdater`. Instantiation of this latter object on a member is part of the instantiation of the Container. Its structure is composed by a UDP[95] socket listening on a specific, globally known, port. Reception of a new group view triggers this latter to be shared among all local components by propagating invocation of the group update primitive locally.

The Group Membership protocol is an external facility, and is designed to be executed by means of a separate Java Virtual Machine. Obviously, the Group Manager needs to be executed before each member for an execution to be correct, and in *alipes* a text client provides a usable example on how initialization of the whole system should be done.

7.3.2.3 Time and Network Update Metrics Update subsystems

The *Time Service* and *Network Metrics Update* subsystems share many structural concepts. Their structure is based on an event-driven, publish/subscribe[42] paradigm and implies clients interested in the occurrence of a specific event to deposit a `listener` in a list. In the case of the Time Service, the event of interest is expiration of a timeout, while in the case of the Network Metrics Update is production of a new set of (up-to-date) network metrics.

The listener required by the Time Service to provide notification of timeout

expiration is called *Timeout Listener*. Besides the timeout to be notified expiration of, timeout listeners specify owner, creation timestamp and a method to call in case the timeout expires. When a client wants to be notified of expiration of a timeout, it creates a Timeout Listener and asks the Time Service to insert it into the list of active listeners.

In the Time Service, listeners are received by a thread that stores them in the list. Every (configurable) fixed amount of time, typically in the order of milliseconds, another thread wakes up and compares the current time with the time the timeout is due to expire. If the current time is bigger or equal to the expiration time, the timeout is retained to expire. Upon this occurrence, the owner is notified through the method specified, and the listener is discarded.

The Time Service is designed as a logical separate auxiliary subsystem instantiated locally by each member. This choice eases the system from having to compensate for differences in local clocks, which might arise in scenarios where a similar system is offered through a remote centralized or distributed solution.

The *Network Metrics Update* component, on the other hand, provides updates on network metrics to system components previously notifying their interest in this type of event. Unlike the previous service, the Network Metrics Update foresees the use of persistent listeners, which are deposited in the listeners list once and never deleted. As a consequence, subscription to the use of this service is required only once. In addition, listeners are limited in number, as they represent components awaiting for notification.

As different components need to be notified of variation in different metrics,

the Negotiation Component and the RMcast Component use different listeners, namely the *Negotiation Metrics Listener* and the *RMcast Metrics Listener*. They fundamentally differ only in the type of information they ask to be notified of, and in the destination notification must be directed to. The Network Measurement Component, on the other hand, provides the Network Update Service with the information which will be notified to other components. The component therefore uses the Network Update Service differently, and foresees invocation of a method to update values in the Network Update subsystem, which will realize the change in values and notify other local system components.

7.3.3 Testing in a Real Scenario

The *alipes* prototype has been tested in a real world scenario for its capabilities to maintain the negotiated guarantees. This testing phase has been conducted on the PlanetLab[2] (PL) platform, already briefly introduce in chapter 6.
In the next subsections we provide a more complete description of PlanetLab and, after, a description of the testing phase.

7.3.3.1 Environment and methodology

Planet Lab[2] (PL) is world-wide testbed and has worldwide industrial and academic affiliations. Each of the member institutions provides two or more hosts for PL users. Hosts are connected via the Internet on a best effort basis, and the Linux operating system on top of each host provides only basic

No.	Affiliation and Location
1	Motorola Inc., Seattle, USA
2	Newcastle University, Newcastle, UK
3	Indian Institute of Information Technology, Bangalore, India
4	University of Bologna, Italy
5	University of Technology at Sidney, Australia
6	RNP, Rio de Janeiro, Brazil
7	Cornell University, Itaha, USA
8	The University of Hong Kong, Hong Kong

Figure 7.16: Nodes affiliations and numbering.

communication primitives. Utilization of PL implies creation of a *slice*, which needs to be later populated with *nodes*. Slices are pseudo-domain abstractions similar to *Virtual LAN*s which are populated with nodes hosted on machines geographically distant. PL provided an excellent testbed platform for our purposes, as geographical distance among nodes allows a network behavior that is hard predict and failures can happen at any time without notice.

To the end of testing *alipes*, we have created a slice and populated it with eight nodes belonging to member affiliations. Enumeration of the participating nodes is shown in table 7.16, and corresponding geographical location of machines hosting the nodes is shown in figure 7.17.

All the nodes in our slice were included in a group by installing *alipes* on each of them and allowing the GM protocol there included to handle GM-related aspects.

Throughout testing, node number 4, in Bologna, Italy, hosted the Group Manager; besides the standard communication duties, its tasks were thus to receive joining notifications from new members, build group views and provide other

members with such views. All members were engaged in a one-to-many communication by means of *alipes*-based facilities. In each operation, we allowed the originator to change based on a round-robin policy.

Tests consisted of sessions whose results are collected on a per-message basis. Any node can act as a message originator. An automated client simulated the user requesting a negotiation every minute and sessions were formed by a total of 100 requests.

All negotiation requests had reliability fixed as 99.99% and a randomly chosen latency type (either absolute or relative) and delay. Note that the above reliability level refers to the precision the user asks the protocol to maintain on the provision of the specified timeliness bounds, rather than the delivery guarantee level which the protocol natively offers in the multicast operation.

Successful negotiations allowed the host to act as originator in the consequent multicast operation. Values for user latency requests are uniformly generated in a realistic interval time, which takes into account geographical distance of the hosts.

In table 7.18 we report a sample set of requests performed for a simulated user, with corresponding results. The table shows a numbered itemization of 15 requests, and for each of these it shows:

Run Run number. Simple count for the showed requests;

Neg Lat Latency amount. Randomly generated amount of user-requested latency;

Type Latency type. Randomly generated type of user-requested latency. Its

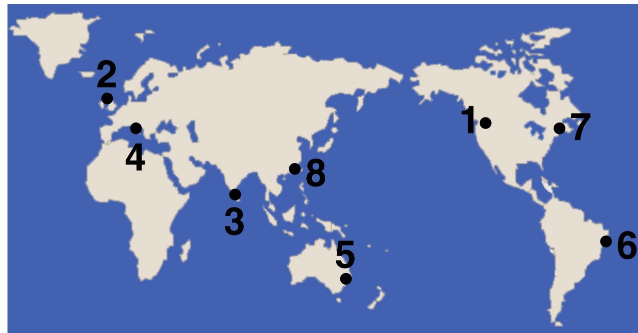

Figure 7.17: Nodes location.

value can either be ABS, indicating absolute latency, or REL, indicating relative latency. In case of absolute latency, the user is requesting delivery to all operative destinations within a latency bound from the the moment when the originator invokes the RMcasting primitive. On the other hand, a relative latency type indicates the user requesting delivery to all correct destinations within a specific latency bound from the moment when the first destination receives the message;

Accepted Negotiation result. States whether the negotiation is successful or not (i.e. whether user requests have been accepted or not). Values in this column can be either YES, for successful negotiations, or NO, for non-successful ones;

Success RMcast operation result. States whether the resulting RMcast operation has been successful in maintaining the agreed QoS guarantees or not. Values in this columns can be either YES, to indicate that the RMcast operation terminated successfully, or NO, to indicate that the RMcast operation has not been successful. The RMcast operation is retained

to be successful when the protocol delivers the message to all correct destinations within the specified timeliness bound. On the other hand, an operation is not successful when the protocol fails to either deliver the message to all correct destinations or fails to maintain the timeliness bound;

Perf Lat Performance time. Shows the interval time between the originator invoking the RMcasting primitive to the first moment thereafter when *all* destinations delivered the message. In the ideal case, this column should contain values smaller than or equal to values in the *Neg Lat* column. In fact, this would confirm underestimation of real protocol's performance in negotiation phase. Note how values in this column represent an upper bound to the absolute and relative latencies experienced in the actual real-world execution, and how comparison with values in the *Neg Lat* is possible;

Error Relative error. Displays a percentage representing the underestimation of the latency time negotiated with respect of the latency actually experienced in the real execution. This value is calculated as *(Neg Lat - Perf Lat)/Neg Lat*. Ideally it should contain positive percentage figures, meaning that the actual performance is faster than what predicted in negotiation phase. However, too large positive values would indicate that the negotiation is not accurate;

ρ Calculated level of redundancy. Whenever the negotiation is successful,

this column displays the level of redundancy needed to achieve the agreed QoS level;

Bcasts Total number of broadcasts carried out. Whenever the negotiation is successful, the column displays the number of broadcasts totally carried out in the correspondent RMcast operation. The value includes broadcasts carried out due to protocol specifications and are therefore expected not to fall beyond the correspondent $\rho + 1$ transmission;

Together with the reliability request, fixed at 99.99% as mentioned earlier, the *Neg Lat* and *Type* columns define the user-requested QoS level. For instance, run number 1 in the table specifies a user requesting a multicast operation to be completed within 779 milliseconds of relative latency with 99.99% reliability guarantee. In this case, the relative nature of the latency guarantee implies evaluation of the U_S probability (equation 4.11), and the successful negotiation in the table indicates that evaluation of the analytical model confirmed *at least* 99.99% confidence on the delivery of the message to all correct destinations within 779 milliseconds can be guaranteed. On the other hand, in run number 7 a user requests a guarantee of 116 milliseconds in terms of absolute latency, together with the fixed 99.99% reliability guarantee. In this case, the absolute nature of the requested latency triggers calculation of the r_D probability (equation 4.7), and the NO value in the *Accepted* column indicates that the system has been unable to guarantee message delivery to all correct destinations within 116 milliseconds with 99.99% confidence. However, note that the

classic multicast delivery guarantee, which implies delivery of a message to all or none of the correct destinations, is natively taken into account.

A first look at the table clearly shows that *all* RMcast operations performed have terminated successfully maintaing the promised guarantees. In addition, on a total of 15 runs, in 11 occasions *alipes* has been able to accept the requested QoS level. In other words, the table shows that in 74% of the total runs *alipes* has performed successful negotiations, and in each of these it has been able to maintain the negotiated guarantees. The remainder of this section will describe results of test runs reported in table 7.18 in detail, focusing on *alipes*' capability to maintain QoS levels.

7.3.3.2 Latency

alipes capabilities of maintaining latency guarantees have been measured in the following way. After a successful negotiation, the time needed to complete the RMcast operation is calculated and compared with the latency amount originally negotiated. The time needed to complete the communication process is calculated as the interval time between originator's invocation of the RMcasting primitive and the last destination delivering the message. The latency so obtained is thus compared with the negotiated latency regardless of this latter being absolute or relative.

The rationale behind this technique is based on the considerations that (i) the latency so calculated is the absolute latency, and (ii) the absolute latency represents an upper bound on evaluation of protocol performance by latency.

In fact, consider definitions of absolute and relative latencies given in section 4.3.1:

Absolute Latency the interval time between the originator invoking the $RMcast(m)$ primitive and the first instant thereafter when all operative destinations deliver the message m;

Relative Latency the interval time between the first destination delivering the message and the first instant thereafter when all destinations deliver the same message.

Consideration (i) can be clearly seen by comparing the above definition of absolute latency with the way we calculate the latency in execution of *alipes* on the PL slice. Consideration (ii), on the other hand, derives directly from the definitions above. In fact, the absolute latency refers to the interval time between the originator invoking the RMcast primitive and the instant when the last destination delivers the message. On the other hand, calculation of the relative latency starts when the first destination delivers the message, which obviously is at a later time than the originator invoking the RMcast primitive, and finishes, as well as for the absolute latency, when the last destination delivers the message.

Showing that the calculated latency is still smaller than the negotiated latency, regardless of this latter being absolute or relative, allows to calculate the relative error with which the negotiation underestimates the real protocol performance, and establishes goodness of the protocol performance. As a side

remark, when the calculated latency is compared with a negotiated latency which is relative, the comparison establishes a lower bound on the relative error. In other words, the value for the relative error showed in table can be considered to be the *minimum* relative error.

In the table, rows referring to communications performed, i.e. those ones exploiting a successful negotiation, clearly show that *alipes* terminates the RMcast successfully within a latency interval smaller than the negotiated one regardless this latter having absolute or relative nature. The *Error* column shows a quantification of this, with values varying from 4% to 14.1%. Although these figures might look quite small, they can be relevant if related to the size of the group. Besides, the small values indicate that negotiation provides accurate estimations of protocol real performance. Fluctuations in these percentages are due to unforeseen events degrading network performances during the correspondent RMcast operation, and the fact that *alipes* maintains agreed guarantees show effectiveness of its adaptation mechanisms.

7.4 *QoS-JGroups*

QoS-JGroups refers to the core system of *alipes* adapted and formatted based on protocol integration guidelines specified for the *JGroups Reliable Communication Toolkit*[13].

The purpose of integrating the single-packet RMcast protocol into *JGroups* is twofold. On one side, it provides an example of integration of the RMcast

Run	Neg Lat	Type	Accepted	Success	Perf Lat	Error	ρ	Bcasts
1	779	REL	YES	YES	701	10%	2	3
2	1361	ABS	YES	YES	1340	1.5%	2	3
3	828	ABS	YES	YES	815	1.5%	2	5
4	1060	REL	YES	YES	1052	7%	2	4
5	190	REL	NO					
6	774	ABS	YES	YES	678	1.24%	2	4
7	116	ABS	NO					
8	987	REL	YES	YES	947	4%	2	3
9	679	REL	YES	YES	644	5.1%	2	4
10	1149	ABS	YES	YES	1124	2.1%	2	4
11	750	ABS	YES	YES	722	3.7%	2	5
12	290	ABS	NO					
13	568	REL	YES	YES	550	3.2%	2	3
14	1150	ABS	YES	YES	1020	1.13%	2	4
15	298	REL	NO					

Figure 7.18: Results from tests on the Planet Lab slice

system into an already-existing system, where this latter can start benefitting of the new service straightaway and in a way completely transparent to the original system.

As second reason, integration of the RMcast system into *JGroups* suited excellently the needs of the TAPAS (*Trusted and QoS-Aware Provision of Application Services*)[3] EU-IST project.

The overall contribution of the TAPAS project was to develop novel methods, tools, algorithms and protocols that support the construction and provisioning of Internet application services. The project achieved the overall objective by developing QoS-enabled middleware services capable of meeting SLAs between application services and enhance component based middleware technologies such that components can be deployed and interact across organisational boundaries.

Partners of the project selected the *JBoss*[105] application server as referring

platform, and focused on enriching its architecture with a set of components to enhance trust and QoS-awareness.

This section is structured as follows: after a brief description of the architecture of *JBoss* an *JGroups*, in subsections 7.4.1 and 7.4.2 respectively, we shall describe the steps that led to integration of the RMcast system into *JGroups*, also describing the architecture of the *QoS-JGroups* so obtained and the changes to the original *JGroups*.

7.4.1 The *JBoss* application server

JBoss[105] is a well-known open source platform for developing and deploying enterprise Java applications, web applications, and portals. Developed entirely in Java, it offers the full range of *Java Enterprise Edition*[20] (J2EE) features as well as extended enterprise services such as clustering, caching and persistence. Its architecture can be divided into four primary layers:

- *Microkernel layer.* The core of the application server is a microkernel-based server. Its purpose is to provide a component model offering deployment, class-loading and full lifecycle management.

- *Services layer.* On top of the microkernel, an extensible *Service Oriented Architecture*[41] (SOA) offers a set of services. Standard services offered by this layer include transaction, messaging, mail, security and clustered services.

- *Aspect layer.* This layer offers an *Aspect Oriented Programming*[86]

(AOP) architecture, which allows behavior provided by the services to be included into any object by means of *interceptors*.

- *Application layer*. This topmost layer is the residing place of applications to be supported.

Among the services offered by *JBoss*, *clustering* is one of the most useful and widely used. This service allows a server to be replicated within a cluster of identical application servers enhancing scalability (by letting administrators to include custom load-balancing techniques with which to handle a higher number of server requests) and fault-tolerance (by redirecting requests to other servers in case the one chosen to handle the request becomes faulty).
In order for clustering to be fault-tolerant, it is fundamental to be able to keep consistency among replicated servers. In *JBoss* this task is handled by the *JGroups Reliable Communication Toolkit*[13] through provision of a reliable group communication service. The use of *JGroups* in the *JBoss* allows a reliable communication between replicated servers by including them in a group and broadcasting information relevant to cluster management to all members in the group. However, timeliness issues are not considered when keeping the cluster consistent (as well as in the rest of *JBoss*) and therefore the TAPAS project required to amend the *JGroups* in such a way to couple reliability with timeliness features.

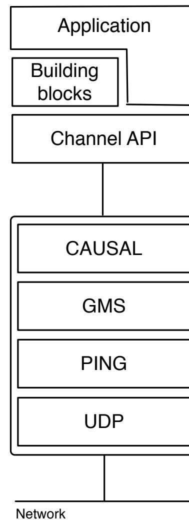

Figure 7.19: Architecture of *JGroups*

7.4.2 The *JGroups* Reliable Communication Toolkit

JGroups[13] is a toolkit for reliable multicast communication created by Bela Ban. Its architecture is based on the abstraction of *channel*, which connects the owner to the rest of the group. Architecture of *JGroups*, shown in Figure 7.19, relies on three main components, described in the following subsections.

7.4.2.1 *Channel API*

The channel API provides a set of libraries to allow creation, control and deletion of channels between the application on the local node and the rest of the group. The first step towards construction of a *JGroups*-enabled system is creating a channel. Creation is done by specifying a list of properties the user wants the channel to exploit. Properties on a channel are brought by protocols acting on that channel, and the user specifies the list of protocols the channel

is required to exploit. These are specified by means of a *definition string*, and figure 7.20, explained later in this section, shows the one instantiated for the default channel.

After creation, the channel needs to be connected to a group. An id for the group is passed as input parameter to the channel at instantiation phase. Successful joining to the specified group, together with the correct instantiation of the corresponding channel (i.e. instantiation of a channel owning all properties listed in the definition string), denotes successful connection of the channel itself.

When the channel is no longer needed, a disconnection procedure is started. The procedure finalizes and disposes all resources, such as protocols instantiated, and closes the channel. Closure of a channel, in particular, implies the host to voluntarily leave the group.

7.4.2.2 *Building Blocks*

Building blocks are layered on top of channels, and provide a set of more sophisticated APIs to give a higher abstraction of channels. Effectively, they do not act on channels, but rather on any transport interface. This allows to abstract the concept of channel away, allowing introduction of design patterns[52] to handle (and avoid) common architectural problems.

APIs in this component aim principally to provide more sophisticated communication entities, such as adapters and dispatchers[69], and data structures, such as distributed and replicated hash tables and distributed trees[30], while

providing distributed management for mutual exclusion.

7.4.2.3 *protocol stack*

The protocol stack contains definition of a set of properties to be used in channel creation phase. Each property is offered through a *protocol* object[1].

The standard set of properties covers a wide range of services, including transport (TCP[96] and UDP[95]), membership discovery, (NAK-based) reliable transmission, ordering (causal and total) and group membership. Properties, i.e. protocol objects, are specified in the channel definition phase through the definition string, which is passed as an input parameter to the successful creating the channel. The definition string effectively specifies a stack of protocols to be used in the channel, and both incoming and outgoing traffic will be handled by all protocols in this stack. Figure 7.20 shows the definition string for instantiation of the default channel. Referring to this figure, protocols are specified sequentially from the bottom-most to the top-most position in the stack. This implies, among other things, creation of dependencies between protocols in the stack.

At the bottom the user will need to specify a transport protocol, and for instance in figure 7.20 where the use of UDP[95] is specified. On top of this, the user will typically specify all other properties, all of which will rely on the specified transport protocol to carrie out external communication. In particular,

[1] The concept of protocol object is very similar to what described in chapter 2 for service composition frameworks, where it is sometimes called *micro-protocol*.

```
String props="UDP(mcast_addr=228.8.8.8;"+
mcast_port=45566;ip_ttl=32;mcast_send_buf_size=64000;"+
"mcast_recv_buf_size=64000):" +
"PING(timeout=2000;num_initial_members=3):" +
"MERGE2(min_interval=5000;max_interval=10000):" +
"FD_SOCK:" + "VERIFY_SUSPECT(timeout=1500):" +
"pbcast.NAKACK(max_xmit_size=8096;"+
"gc_lag=50 retransmit_timeout=600,1200,2400,4800):" +
"UNICAST(timeout=600,1200,2400,4800):" +
"pbcast.STABLE(desired_avg_gossip=20000):" +
"FRAG(frag_size=8096;down_thread=false;up_thread=false):" +
"pbcast.GMS(join_timeout=5000;join_retry_timeout=2000;"+
"shun=false;print_local_addr=true)";
```

Figure 7.20: Definition string for the default protocol stack

the string in figure 7.20 instantiates a channel employing a NAK-based probabilistic reliable message transmission (specified by the `pbcast.NAKACK`). The channel can fragment messages if needed, as specified by the `FRAG` protocol, and exploits fault tolerance by the use of a failure detector (specified by the `FD_SOCK` property). Finally, group membership is handled by a probabilistic GM protocol (specified as `pbcast.GMS`).

As mentioned earlier, instantiation of the stack has rigid constraints on the logical order with which protocol objects are defined, and therefore introduces inter-protocol dependencies whose satisfaction if mandatory for creation and connection of the channel. For example, the definition string will necessarily have to contain instantiation of the chosen communication protocol at the bottom of the stack, while the ordering protocol on top of it.

Any inconsistency in fulfilling dependencies and/or respecting constraints in the definition string results in the failure to instantiate the corresponding chan-

nel, and consequently a fatal error which prevents execution of the system.

Integration of our system into *JGroups* has required modification of a subset of protocols in the protocol stack, in particular:

`UDP` : offers a connectionless communication service based on datagrams[95]. Its configuration implies setting the size of buffers for incoming and outgoing traffic and, more importantly, the choice of the delivery model.

The user can choose between *IP multicast*[32] (the default method) and the multicast realized by means of multiple unicasts. In the former, messages are multicast by sending them to an *a priori* specified multicast IP address, which needs to be specified along with the port, while in the latter messages are delivered through simple unicast communications. The choice of the delivery model in this protocol also influences the `PING` protocol, which will soon be described.

`GMS` : handles group management. In a group, the oldest member becomes the group *coordinator*, and as such it is in charge of handling group management. On the group coordinator, the `GMS` protocol object is in charge of producing up-to-date group views. To this extent, it records members' joins and leaves, and verifies crash suspicions. Other group members localize the group coordinator through the use of the `PING` protocol object, next to be described. Every time an event modifies the structure of the group, the coordinator emits a new group view which is sent to all other members.

On the other hand, on a non-coordinator member the `GMS` protocol object

simply notifies its liveness and receives updated group views.

`PING` : retrieves the initial group membership by multicasting PING requests. The method used to perform this action depends on the delivery method chosen in `UDP`, as mentioned earlier. When the latter chooses to realize the multicast by means of IP multicast, PING requests are sent out to the group through the same multicast address. On the other hand, when `UDP` specifies the multicast operation to be realized through multiple unicasts, `PING` realizes the initial discovery of membership references through the use of a *Gossip Server*, whose purpose is to provide a central group-wide reference point for group membership. By sending PING requests to the Gossip Server, whose location needs to be passed as input parameters, the sender inherently asks to be included in the group view. The Gossip Server then typically accepts such requests and produces a new group view which is received by all members including the new one.

7.4.3 Integration: the `RMCAST` protocol object

The core of *alipes*, i.e. the implementation objects for the container, Negotiation, RMcast and Network Measurement components, has been extracted and integrated in the original *JGroups* as QoS-supportive reliable transport layer under the name `RMCAST`.

The `RMCAST` protocol object is positioned just above the transport protocol, as shown in figure 7.21. Consequently, in the definition string its usage needs to

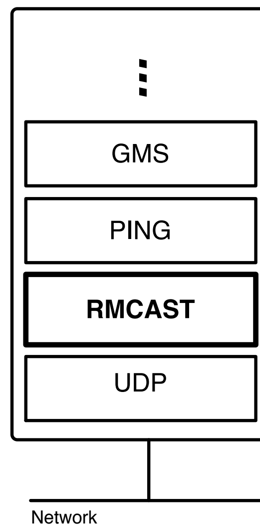

Figure 7.21: *JGroups* protocol stack with integration of the RMCAST protocol object

be declared just after the transport protocol. The usage of RMCAST introduces some dependencies that need to be satisfied. In detail:

- Transport Protocol: RMCAST requires the presence, in the system, of a protocol object offering a transport service. Given *alipes*' original orientation towards connectionless communication, the dependency requires the presence of UDP, that needs to be configured so as to realize the multicast operation by means of concurrent unicasts rather than through the use of IP multicast.

- Group Management: information about group is needed in RMCAST due to its involvement in more then one phase in the life cycle, as showed in figure 3.9. The dependency is satisfied by the presence of the GMS and PING protocol objects. Both these are required because of the depen-

dency of the former on the latter. In detail, `RMCAST` needs information concerning the size of the group and address of each member. The `GMS` protocol object is capable of providing both types of information. However, provision of the address of each member can be provided only by requiring the `PING` protocol object to use a Gossip Server-based group management. As a side remark, it is worth noting how this latter requirement also depends on the `UDP` protocol object to avoid the use of IP multicast.

Unsatisfying one of the aforementioned dependencies in the definition string triggers `RMCAST` to “vanish” from the protocol stack, i.e. to act by simply forwarding data without applying any logic.

`RMCAST` maintains a group view that is kept consistent with the group view maintained by the `GMS`. The group view is used internally to `RMCAST` in order to provide all group information needed for the RMcast protocol. Utilization of this view, rather than the one maintained by the `GMS` protocol object, allows to minimize the message traffic internal to the stack and draws its feasibility from the fact that the position of `RMCAST` in the protocol stack allows *all* GM information to be received.

In the new definition string, shown in figure 7.22, `RMCAST` is specifies the following parameters as fundamental for configuration:

- `reliability`: specifies the reliability QoS requirement. Its value can be any number r such that $0 < r < 1$.

- `latency`: specifies the latency delay QoS requirement. Its value is typically expressed in milliseconds, and any value u such that $u > 0$ is admitted.

- `type`: specifies the type the latency refers to. Its value can be either `absolute` or `relative`. The former refers to the latency bound guarantee defined in 4.3.1, while the latter refers to the relative latency bound, always defined in 4.3.1.

- `net_refresh`: specifies the interval time for refresh of data referring to monitoring of the network. Its value is expressed in milliseconds, and defines the slot of time within which data gathered from the sampling activity needs to be processes in order to calculate average metrics.

After instantiation, `RMCAST` starts execution by performing a negotiation that takes as parameters the values specified in the definition string. In case the negotiation results to fail, the user is notified about the impossibility to guarantee the specified QoS level on the multicast operation, and `RMCAST`, once again, becomes transparent to the execution of the system. On the other hand, in case the negotiation is successful, `RMCAST`'s architecture is instantiated as already described for *alipes*.

Execution of `RMCAST` implies a filtering activity in order to determine traffic that is relevant to its task. This filtering activity is realized as follows. Protocol objects at upper positions in the stack forward outgoing traffic to `RMCAST`, which checks the type of the traffic. When the type is detected to be

```
String props= "UDP(ip_mcast=true;mcast_addr=228.8.8.8;
     mcast_port=45566;ip_ttl=32;mcast_send_buf_size=64000;"+
     "mcast_recv_buf_size=64000):"+
    "RMCAST(reliability=0.99;latency=300;type=absolute;"+
    "confidence=0.99;net_refresh=120000):"+
    "PING(gossip_host=localhost;gossip_port=5555;
     gossip_refresh=1000;"+
    "timeout=2000;num_initial_members=2):"+
    [...]
```

Figure 7.22: New default definition string, with RMCAST integrated

of interest, it is processed internally and, once completed, forwarded to UDP for transmission. The same filtering is applied on the incoming traffic: UDP forwards RMCAST received packets. These are filtered based on their type, and processed internally based on the relevance with RMCAST's activity.

Data that relevant to RMCAST can essentially be divided into the two broad categories of RMcast data and GM data. The first refers to data upon which the RMcast protocol needs to be applied, while the second refers to data aiming to amend the local group view (and therefore needs to be take into account). The filtering activity has consequences which are different for each of the above categories. On the RMcast data, in fact, the filtering implies forwarding of the data based on the times and modes of the RMcast protocol. Therefore, a packet being forwarded from upper protocols in the stack and detected to be RMcast data is forwarded to the UDP protocol object exactly $(\rho+1)$ times each after η time, while a likewise packet being forwarded from the UDP protocol object is treated by being delivered (to the above protocol in the stack) and by being setup a timeout on reception of the next copy.

The filtering on the GM data, on the other hand, simply implies updating `RMCAST`'s internal group view. To this extent, packets containing GM data are cloned and stored to an internal buffer for processing. As a side remark, this allows not to jeopardize the original system's performance.

7.4.4 A word on testing

The `RMCAST` protocol object has been obtained by simply extracting the core protocol of *alipes* in its complete structure and formatting this latter in a way compliant to guidelines given in *JGroups*. This process did not require any change in the original structure of the architecture of *alipes*, and only involved small changes in the way the system is instantiated and primitives invoked. In other words, `RMCAST` and the core of *alipes* have an identical structure, with the only difference that *alipes* is encapsulated into an outer middleware suite, while `RMCAST` is encapsulated into a format that complies to the implementation guidelines specified in *JGroups*.

All characteristics, functionalities and structures found and described for *alipes* are present in `RMCAST`, and we therefore claim that, without loss of generality, results showed and commented in 7.3.3 for *alipes* can be considered valid for `RMCAST`.

7.5 Criticisms

Although both prototypes are effective in providing the service they are expected to, criticisms can be made to both implementations. Generally speaking, implementations presented here have been developed with the only purpose of testing the protocol suite on a real world scenario, in the case of *alipes*, and providing an example of integration on an already-working system. Therefore, both can be improved in many directions:

Architectural structure: the main structure, designed for *alipes* and "inherited" by `RMCAST`, is thought to be as simple as possible. The size of each component's architecture might therefore result cumbersome with the many objects to instantiate and execute, especially for what concerns the Network Measurement Component. The consequence of this is essentially that both prototypes are not optimized for reducing consumption of resources.

Communication: many of the intra-component interactions are performed, in both implementations, via message-passing paradigm through the use of synchronized queues. Besides this type of communication behaves excellently in normal conditions, queues have limited capacity, and when traffic reaches extremely high levels this might cause abnormal behavior.
This problem might be prevented by substituting synchronized queues with more sophisticated, dynamic in size, data structures for temporary storage.

7.6 Concluding Remarks

The protocol described in chapter 4 has been implemented in two separate prototypes. The first is a middleware solution released in a system named *alipes*. The second, which derives from the first, complies to the protocol format of the *JGroups Reliable Communication Toolkit*.

alipes handles inter-component coordination by means of a `Container` object, which allows transparent user-access to sensitive primitives also preventing misuse of the RMcasting and RMdelivering primitives based on the negotiation response. Intra-component communication is realized through message passing paradigm via synchronized queues, while inter-component communication is achieved by a combined use of the Container and a set of event-driven subsystems. *alipes* also provides a basic group management facility which dynamically provides information about group size and identity of members, and can be used as plugin by means of interfacing with several technologies such as XML.

Testing, conducted geographically distant through the PlantLab testbed, showed that *alipes* is actually capable of providing and maintaining reliability and timeliness guarantees negotiated in advance.

In the second prototype, the system has been integrated as protocol object in the stack of the *JGroups Reliable Communication Toolkit*, under the name `RMCAST`. Albeit the core structure and architecture remains unchanged, integration of this latter in *JGroups* implied slight changes to the protocol core of

the *alipes* system, mainly aimed to use *JGroups* native facilities such as group membership. This prototype is awaiting for a proper exhaustive testing, and the use of `RMCAST` is expected to bring a certain number of benefits such as:

- timely and reliable communication over (best-effort) wide area networks without the need of connection-oriented protocols such as `TCP` with a significant save in terms of network resources, as testing of the *alipes* suggests,.

- ease of use, as native management of reliability, timeliness and network adaptation allows reduction of the number of protocol objects to instantiate in the definition string, and consequently reduction in terms of dependencies to be satisfied.

As an important consequence of all the aforementioned reasons, the *JGroups* development team is currently considering a permanent integration of `RMCAST` in the *JGroups* suite.

Chapter 8

Conclusions

8.1 Discussion

In this dissertation we have presented our research in design and development of a QoS-Negotiable Reliable Multicast protocol, which represents a first step towards construction of a QoS-Supportive Group Communication System offering more sophisticated services. In particular, the system here presented offers a multicast operation capable of providing negotiable QoS performance guarantees. QoS attributes here considered are reliability (intended as of all-or-nothing delivery guarantee) and timeliness (intended as delay bounds on the successful termination of the operation).

Providing reliable and timely communication is a complex task, especially when lower level unreliable communication primitives are used and communication is conducted across the Internet infrastructure. In chapter 2 we have shown that timeliness and reliability have never been considered as equally important attributes in the design of a multicast protocol. We have seen how systems offering reliability guarantees are designed to achieve so through

deterministic techniques[6, 102], probabilistic techniques[14, 31, 44] or a combination of the two[74, 93]. We have also described how timeliness is usually provided in a variety of ways and forms[21, 60, 5, 79, 77]. However, we have put emphasis on the fact that, to the best of our knowledge, none of the systems currently developed considers *both* reliability and timeliness as equally important QoS attributes to be provided in the communication process. This, to the author's opinion, makes the system subject of this thesis novel and unique.

In chapter 3 we have described the theoretical model our system refers to, along with its architecture. We have highlighted how the classical synchronous and asynchronous models do not capture the inherent probabilistic nature of the QoS-support in communications performed across (best-effort) Internet boundaries. We have then advocated the use of a *Probabilistic Asynchronous*[45] (PA) model as theoretical design model for our RMcast system. We have described how this latter model satisfies the needs of providing probabilistic guarantees in the communication context.

We then described the system architecture. We advocated the use of a QoS management interface as a way to allow lower level subsystems, such as the *Communication Subsystem* (CS), to export a behavior that can be retained predictable in the long term by other subsystems. We described how this led to designing the structure of our QoS-Supportive Reliable Multicast System as composed by three components named *Negotiation Component*, *RMcast Component* and *Network Measurement Component*.

We have identified each component's primary task as negotiating QoS levels

with the user for the first, providing an adaptive RMcast configurable protocol for the second, and measuring performance metrics of the underlying CS for the third, while also describing the inter-component interaction model.

We have described how the negotiation process involves evaluation of a stochastic model of the RMcast protocol contained in the RMcast Component, and how such evaluation generates configuration parameters to adapt execution of the RMcast protocol,to the achievement of the QoS to be offered. We have described how the model is based on pessimistic assumptions so as to ensure that the system will provide a service level which is higher than the one negotiated.

We have showed how both the stochastic model and the RMcast protocol assume knowledge of current network conditions, and how the Network Measurement Component provides such information in form of average metrics describing network reliability, stability and timeliness characteristics.

In chapter 4 we provided a formal description of the RMcast protocol under the assumption that messages do not require fragmentation in order to be multicast. We defined the protocol's main features as *Redundancy*, which implies the originator to carry out ρ redundant transmissions, each separated by η time, *Responsiveness*, which implies receivers to take over responsibility for completion of the RMcast operation in case of originator crash, and *Selection*, that designates exactly one process to take over the broadcasting responsibilities in case of such devolution. We described how the design based on such features allows the RMcast protocol to provide guarantees on the delivery of

a message to all operative destinations with a probability which can be made arbitrarily close to 1 if the originator does not crash, while guaranteeing that multicast message delivery can be predicted with a probability that is very high and can be evaluated in advance when the originator crashes. We named these *Validity* and *Agreement* respectively.

We also formally described, in chapter 4, the analytical model used for negotiation purposes, and showed its accuracy through comparison with simulations in same environmental conditions.

In chapter 5 we have relaxed the “non-fragmentation” assumption taken in chapter 4 and proposed two approaches to extend QoS guarantees to the RMcast of messages who need to be fragmented into an arbitrary number of packets. In particular, we have assumed the message m to be divided into π packets, and proposed two ways to provide the same QoS guarantees described in chapter 4 on the multicast of the entire set of packets. We named the two extension approaches as *Per-Message* and *Per-Packet*. The first implies packets to be considered as a single logical one over which the logic of the protocol in chapter 4 is applied. The second, on the other hand, implies the π packets to be considered singularly and being transmitted by means of independent instances of the protocol described in chapter 4.

We have described both approaches, putting emphasis on each one’s *pros* and *cons* and, whereas possible providing optimizations. In doing so, we amended the Per-Message approach with two properties, named *Retention* and *Composition*, to the extent of reducing the loss rate exacerbated by the use of this

approach.

Amendments involved also the Per-Packet approach: we suggested independent instances to be handled by a single thread in a pseudo-concurrent way, in contexts where the machine hosting the system cannot support instances to be handled by dedicated threads.

We derived each approach's stochastic model starting from the one described in chapter 4, and assessed accuracy and additional cost aspects through simulations.

In chapter 6 we described the monitoring and measurement techniques employed by the Network Measurement Component to provide information describing current network conditions. This information is provided through statistical metrics describing reliability, stability and timeliness characteristics of the network.

We described the role of this component in scenarios where the CS works on a best-effort basis, and commented the added value of the use of the component in scenarios where the CS it is assumed to be controlled by an *Internet Service Provider* (ISP).

We have described how the *three-way ping* algorithm used to calculate the RTT towards destinations allows to reduce the additional overhead of this operation, and how from this information the component calculates the QoS metrics of interest. Besides, in this chapter we describe the two techniques the Network Measurement component employs for converting conditional probabilities in the stochastic model into numerically tractable equations. We named these

ways as *experimental* and *statistical*, and described how the former calculates conditional probabilities by calculating the percentage of samples actually satisfying the probability, while the latter performs a $\chi - square$ test to obtain a statistical pattern for the samples, to the extent of converting conditional probabilities in equations corresponding to the calculated pattern.

In chapter 7 we have described two prototype implementations for the concepts described in chapter 4. We have described the main prototype, developed as a middleware suite named *alipes*. We have advocated the choice of a model coordinated by a *Container* object, to allow transparent access to sensitive primitives and, at the same time, to prevent their misuse.

We have described testing of *alipes* in a real scenario by means of the *Planet Lab*[2] worldwide testbed, and shown results proofing that *alipes* is effective in providing real QoS support for Internet-scale GC over best effort networks.

Through the second prototype implementation, we provided an example of how the core of *alipes* can be extrapolated and integrated into an already-existing system, such as the *JGroups Reliable Communication Toolkit*[13]. We have described the integration process and the way integration can be made seamless to the original JGroups system. Finally, we mentioned the benefits of the *QoS-JGroups* system, obtained by integration of the core of *alipes* into the original JGroups. We described how *QoS-JGroups* has been of advantage to the TAPAS EU-IST project, where this system provided the basis for QoS-supportive clustered services, and mentioned the interest of the *JGroups* development team considering *QoS-JGroups* for a stable release. Both these

interests provide a further example of real-world usage of our QoS-Supportive Reliable Multicast System.

8.2 Take-home Message

Analysis of a reliable multicast system becomes more complex when the main protocol has probabilistic behavior. Its non-deterministic nature requires careful analysis of every aspect of the protocol dynamics, while however providing some unexpected results that is definitely interesting and worth studying and analyzing.

On the simulation side, a solid theoretical model for the simulator to be implemented is fundamental. The model needs to be "tailored" to the actual protocol to be simulated. Nonetheless, it has to guarantee a certain degree of extensibility and modularity. The former guarantees the possibility to change global parameters in a way transparent to the actual algorithm simulated, while the latter allows variation in the main algorithm to remain circumscribed to the context of small modules.

Once designed, stochastically defined and simulated, implementation and execution on a real-world scenario provides a unique chance to find out how accurate the previous work has been. The choice of standard design and implementation tools ease the development and deployment on already-esisting systems. In the specific case of the design and development tools used for implementation of the two approaches here presented, the perfect UML-Java

interoperability eased development of the architecture, leaving the main focus on the development of components' internals and inter-process communication. Finally, comparison of results obtained from testing with results obtained from simulations is fundamental to determine faults and bugs in either of the mechanics. However, the environment used for testing needs to be reflected in the slightest detail in simulations as well, and even small differences might influence the final results consistently.

8.3 Directions of Further Research

The RMcast system described in this dissertation provides a robust communication service. However, as already mentioned, it needs to be seen under the light of a first step towards construction of systems offering more sophisticated QoS guarantees for group communication. As such, further research on the system and its architecture can focus on a variety of directions:

- *Architectural extensions*: the system architecture can be extended so as to allow provision of more sophisticated services. In fact, following the idea of service composition frameworks mentioned in chapter 2, more sophisticated QoS guarantees can be provided by adding corresponding protocols *on top* of the base layer represented by our system. Possible extensions might target provision of more sophisticated guarantees such as multicast ordering schemes (causal, FIFO or total). However, according to what said in chapter 3, in order for this to be feasible each extension

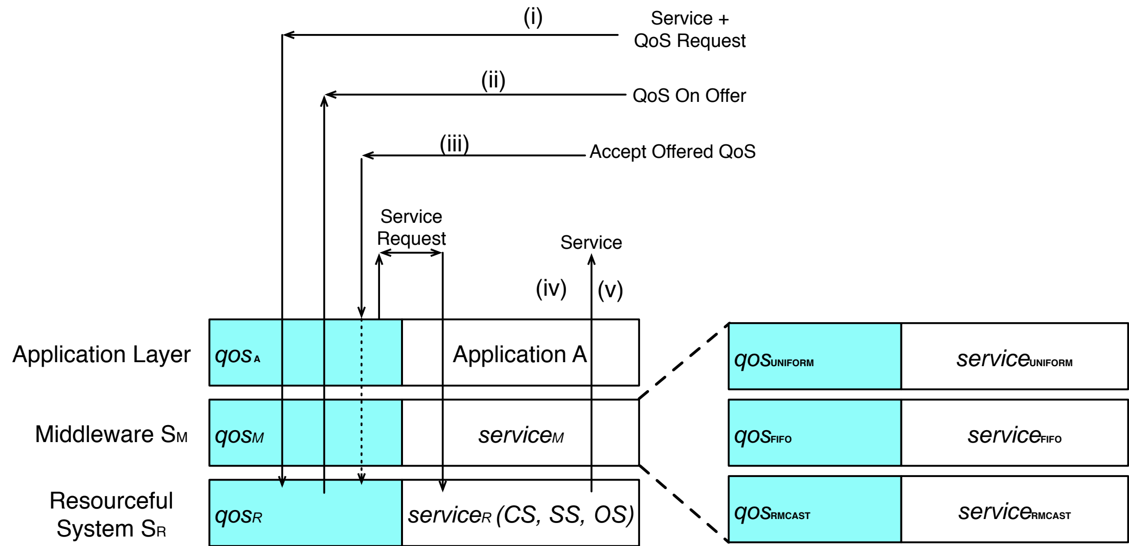

Figure 8.1: Architectural extension of the QoS-Adaptive middleware architecture.

layer must export a QoS management interface. As an example, figure 8.1 shows integration of a protocol offering a uniform FIFO ordering service. In this figure, each subsystem is composed by a QoS management and a service interface, indicated as qos_M and $service_M$ respectively, according to the architecture structure described in chapter 3. The middleware subsystem would provide the uniform FIFO reliable multicast service by allowing the unordered reliable multicast service, offered by the layer containing the qos_{RMCAST} and $service_{RMCAST}$ interfaces, as transport layer, upon which the logic for FIFO ordering is applied by means of the layer containing the the qos_{FIFO} and $service_{FIFO}$ interfaces. Ordering is then extended to uniform FIFO by application of a further layer, shown in the figure as containing the $qos_{UNIFORM}$ and $service_{UNIFORM}$ interfaces, which applies the uniformity logic to the FIFO-ordered messages

and delivers them to the user.

As described in chapter 3, the service interface of each middleware subsystem would be implemented by a fault-tolerant protocol designed to be configurable with parameters whose setting will allow it to achieve a desired QoS level. The QoS management interface of each layer would evaluate the QoS offered by the corresponding service interface and derive parameters also considering the QoS offered by the lower subsystem. In addition to such type of extensions, the QoS-Supportive middleware architecture can be extended in such a way to make behavior of subsystems directly handling resources predictable in the long term. In fact, as described in chapter 3 the system subject of this thesis provides a QoS management interface for the CS only. Providing other resourceful subsystems, such as the *Storage Subsystem* (SS), with a similar management interface would allow the system to make predictions on the delay times due to handling of the information at both source and destination, enhancing accuracy of guarantees provided.

- *Enhancement of the negotiation process*: The negotiation process could be enhanced in more than one way. Support for a wider set of QoS attributes is the most appealing way to do so, and might be done by considering negotiable attributes such as, for example, available bandwidth or security level. When new negotiation parameters refer to network characteristics, this would obviously affect the structure of the Network

Measurement service, which would have to provide network information in terms of relevant QoS metrics.

The negotiation process might also be enhanced to introduce mechanisms that account for *traffic shaping* techniques to allow more flexible allocation of concurrent QoS requests. In fact, at present the Negotiation component always considers the total amount of available resources as the basis upon which to negotiate QoS levels with the user application. This inevitably leads to a degradation of negotiable levels in situations where it is needed to handle more than one negotiation sequentially.

Another interesting way to enhance the negotiation process would be to consider internal resources. In fact, resources such as machine average load or computational power available locally are not currently not considered in the negotiation process, albeit they might influence execution of the multicast operation and, in some case, undermine achievement of the negotiated QoS levels. Inclusion of these factors in the negotiation process would allow more accurate negotiations, and is very tight to the provision of resourceful subsystems with QoS management interfaces mentioned in the architectural extensions above.

- *Integration with IP Multicast*: the use of IP Multicast[32] requires support in terms of network infrastructure and therefore, as mentioned in chapter 2, its use cannot be assumed in communications across Internet boundaries. However, whereas such support is present, an interesting di-

rection for future research focus would be to adapt the system so as to use the IP Multicast delivery scheme. The system would benefit from low-level handling of information multiplexing, which would improve system performance also easing the middleware level from related issues such as synchronization.

- *Implementation and testing of extensions*: prototypes described in chapter 7 only implement the single-packet protocol described in chapter 4, and therefore are capable of sending messages of standard size.

  Next step towards development of a fully usable system would be implementation and testing of extensions allowing the system to deal with messages of arbitrary size described in chapter 5. In particular, it would be interesting to conduct the testing phase by using the RMcast system in soft real-time contexts such as multimedia applications.

8.4 Concluding Remarks

In this thesis we have presented and studied a QoS-Supportive RMcast system. Its novelty lies in the design aimed to consider *both* reliability and timeliness as equally important QoS attributes in the communication. In addition, the system provides negotiation facilities based on an stochastic model by means of which system performance can be estimated anticipately to service provision. This results in QoS attributes to be provided in a negotiable form, giving the user application practical guarantees about maximum achievable QoS levels.

The system offers a Reliable Multicast operation through a fault-tolerant protocol of adaptive design, configurable around a set of parameters which influence execution in order to achieve the desired QoS level and whose values are generated in the negotiation phase. Support for messages of arbitrary size is provided through two extensions to the original model, which treat messages differently and consequently are shown to be more effective in different contexts. Together, they cover a wide range of possible contexts.

The system also features a Network Measurement Component, which provides network QoS information to the extent of allowing estimation of service behavior to be based on up-to-date network performance levels.

Experiments carried out on implementations show that utilization of the system is effective in providing a fast and timely transport layer, while simulations carried out allow to infer a wide range of application which might benefit from usage of the system.

Main contributions of the system subject of this thesis can be summarized as follows:

- design of a Reliable Multicast system which considers reliability and timeliness as equally important QoS attributes in the multicast operation (chapter 3);
- design of a protocol of adaptive logic and configurable parameters, and extension for usage with messages of arbitrary size (chapters 4 and 5);
- derivation of analytical expressions for negotiating QoS metrics whose

effectiveness is validated through simulations (chapters 4 and 5);

- design of a distributed network monitoring and measurement subsystem (chapter 6);
- development and testing of two workinh prototypes of the system (chapter 7).

Appendix A

A Gossip-based Group Management Protocol

A.1 Introduction

In this appendix we describe a Group Management (GM) protocol that can be used in conjunction with the RMcast system described in this dissertation to provide each process with up-to-date group views.

In the protocol, at regular interval of times each process disseminates a token containing its own id. If the token is not lost, each process receiving the token adds the sender to its own group view, if not already included. If the token is not received, on the contrary, the process interprets the missed reception as the sender having left the group and, consequently, removes it from its group view, if previously included. Over time, each operative process will receive tokens from all other operative processes and build a group view.

Dissemination of tokens is done through a gossip protocol, and accuracy of group views depends directly on the coverage capability of the gossip protocol.

For this reason, in this appendix we shall describe the gossip protocol and study its coverage capabilities.

The gossip protocol here presented is based on the encounter-based broadcast protocol[29] by Cooper et al.. This protocol was originally designed as reliable multicast protocol on Mobile Ad Hoc network (MANET) environments, and what we shall describe and study in the remainder of this appendix is an adaptation of its original design to the use as GM protocol in WAN wired networks.

A.2 Design features

The gossip protocol has features to maximize the probabilities of covering the group entirely while minimizing the cost of message and storage overhead. In this protocol, each process gossips a previously generated token with a bounded number (at least τ) of randomly selected processes and τ is a protocol parameter. A process i maintains *view* V_i on the group G as the set of all operative nodes in G. A process' view may not be accurate: it may contain a process that may have left G or crashed. We assume that V_i contains at least f operative processes of G. The protocol exports a $SendToken(tok)$ primitive. Invocation of this primitive is made by the *originator* (of *tok*) while processes receiving the token are called the *destinations* (of *tok*). Tokens are assigned an identifier $tok.id$, that uniquely identifies that specific token and is assigned by the originator just before being gossiped.

A.2.1 Gossip

The originator gossips the token *tok* with $\lceil \tau(1-q)^{-1} \rceil$ destination processes selected randomly from its V, where q is the transmission loss probability. The protocol makes gossiping effective in two ways. First, gossip targets are chosen judiciously, excluding those that are known to have received *tok*: a process maintains a set $Received(tok.id)$ containing all destination processes which it knows to have received *tok*. Obviously, $Received(tok.id)$ is initialized to $\{tok.sender\} \cup \{own_id\} \cup \{tok.originator\}$, when the process receives *tok* for the first time. Processes for gossiping are randomly selected from $V - Received(tok.id)$.

Secondly, each process gossips in τ rounds with each round separated by time-out intervals which are exponentially distributed with mean ξ. In each round, a process gossips with $\lceil (1-q)^{-1} \rceil$ processes selected from $V - Received(tok.id)$. Staggering of gossips into multiple rounds permits $Received(tok.id)$ to increase between rounds, avoiding gossip with destinations that are already known to have *tok*. When $V - Received(tok.id) = \oslash$, the originator has gossiped to all known destinations in its view: the gossip thus terminates, and the originator simply waits to receive tokens from all other processes in the group.

At destination, each process receiving *tok* includes the sender in its own view, if not already included, and sends the received token to $\lceil \tau(1-q)^{-1} \rceil$ destination processes, selected randomly as described above.

Achievement of the maximum coverage in the gossiping is heavily influenced by

the value of parameter τ. Experiments conducted by Cooper et al., described in [29], show that the encounter-based protocol does indeed achieve coverages close to 1 on MANET environments subject to variable density, with particularly good performance in environments where simple flooding performs poorly. In addition, they also show that the propagation time, i.e. the time needed to achieve maximum coverage, is decreased in high density environments.

A.3 Analytical approximations

Consider an "idealized" system with n processes which never cease to propagate the messages they receive ($\tau = \infty$). Let T be the random variable representing a message *propagation time*, i.e., the interval between the $SendToken(tok)$ primitive is invoked by some node, and the first instant thereafter at which all operative nodes have received it. When $\tau = \infty$, T is finite with probability 1. It is then of interest to estimate its average value, $E(T)$. That quantity will also be used in choosing a suitable value for τ, when designing a practicable τ-gossip protocol.

An estimate for $E(T)$ will be obtained under the following simplifying assumptions:

(a) Each node executes gossip rounds separated intervals which are exponentially distributed with mean ξ.

(b) At each gossip round, one other process receives the gossip.

(c) The process that receives a gossip in a given round is equally likely to be any of the other processes; that is, the probability that process i will next choose process j, $j \neq i$, is equal to $1/(n-1)$, regardless of past history.

Assumption (a) is enforced by the protocol. Assumption (b) is optimistic. Recall that a process chooses $\lceil (1-q)^{-1} \rceil$ gossip targets at each round, and all the chosen targets may already be crashed. A remedy will be to choose $\lceil (1-q)^{-1}(f) \rceil$ gossip targets, and this would ensure that not all the chosen ones are inoperative. Assumption (c) is loosely based on the fact that all processes are statistically identical. If the initial contents of the processes' V are uniformly distributed, the assumption is justifiable at the first gossip round, although it may well be violated in subsequent ones. However, this assumption provides the simplification necessary for analytical tractability. Its effect on the performance measures will be evaluated in the simulation experiments.

Let $X = \{X(t)\,;\, t \geq 0\}$ be the Markov process whose state at any given time is the number of processes that have already received the message. The initial state of X is $X(0) = 1$ (only the originator has 'received' m). The random variable T is the first passage time of X from state 1 to state n. Suppose that X is in state k, i.e. k processes have received the message and $n-k$ have not. If any of the former k processes gossips with any of the latter $n-k$, the Markov process will jump to state $k+1$. Since each process gossips at rate $1/\xi$, and the probability of gossiping successfully with any other process is $1/(n-1)$,

the transition rate of X from state k to state $k+1$, $r_{k,k+1}$, is equal to

$$r_k = \left[\frac{k}{\xi}\right]\left[\frac{n-k}{n-1}\right] . \tag{A.1}$$

In other words, the average time that X remains in state k is

$$\frac{1}{r_k} = \frac{(n-1)\xi}{k(n-k)} . \tag{A.2}$$

Hence, the average first passage time from state 1 to state n is given by

$$E(T) = (n-1)\xi \sum_{k=1}^{n-1} \frac{1}{k(n-k)} . \tag{A.3}$$

This last expression can be simplified by rewriting the terms under the summation sign in the form

$$\frac{1}{k(n-k)} = \frac{1}{n}\left[\frac{1}{k} + \frac{1}{n-k}\right] .$$

The two resulting sums are in fact identical. Therefore,

$$E(T) = \frac{2(n-1)\xi}{n} \sum_{k=1}^{n-1} \frac{1}{k} = \frac{2(n-1)\xi H_{n-1}}{n} , \tag{A.4}$$

where H_n is the nth harmonic number. When n is large, the latter is approximately equal to

$$H_n \approx \ln n + \gamma ,$$

where $\gamma = 0.5772...$ is Euler-Mascheroni's number. Also, when n is large, $(n-1)/n \approx 1$ and $\ln(n-1) \approx \ln n$. We have thus arrived at the following estimate, valid under assumptions (a), (b) and (c):

Theorem A.3.1 *In a large network where messages are not discarded, the average propagation period for a message is approximately equal to*

$$E(T) \approx 2\xi(\ln n + \gamma) \ . \tag{A.5}$$

An immediate corollary of Proposition 1 is that, during the propagation period T, the originator performs an average of $2(\ln n + \gamma)$ gossip rounds. Other processes, who receive the message later on, tend to make fewer gossip rounds. Thus, choosing the encounter threshold, τ, to have the value

$$\tau = 2\lceil \ln n + \gamma \rceil \ , \tag{A.6}$$

should ensure that, when the protocol terminates, most nodes will have received the message. This suggestion will be tested experimentally.

Note 1. An attractive aspect of equation (A.6) is that the only parameter appearing in it is the number of processes, n. The value of ξ does not matter, as long as assumptions (a), (b) and (c) are satisfied reasonably well. In fact, the value of ξ serves as basis for determining the interval time between subsequent gossip rounds. As a consequence, it affects the message propagation time and not the coverage rate.

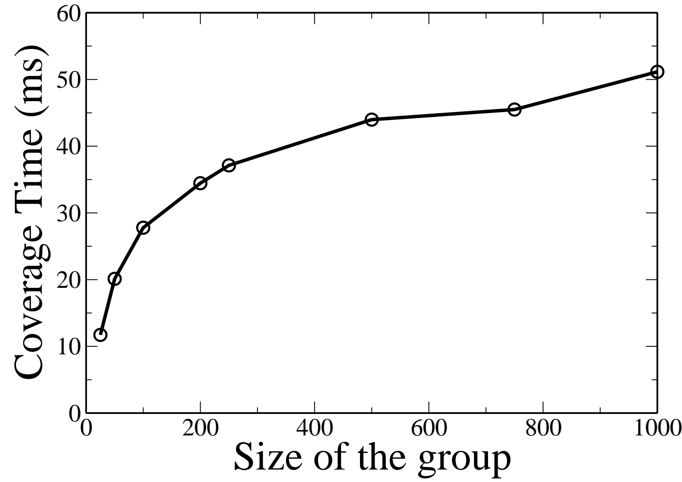

Figure A.1: Coverage time for the gossip protocol when $\xi = 2.5$ ms and τ is variable with the group size

Note 2. Since, under τ-propagation, every process that receives a message broadcasts it τ times, the total number of broadcasts per message is on the order of $O(n\tau)$. Hence, if τ is chosen according to (A.6), the total number of broadcasts per message is on the order of $O(n \ln n)$.

A.4 Simulation Experiments

The GM protocol here presented has been simulated to the extent of estimating its effectiveness in covering the group. Results obtained from experiments have been divided into two sets. In the first one, the scalability of the protocol is observed by studying how the *coverage time*, intended as the time needed by the protocol to reach *all* group members, changes with respect of the group size. In fact, although the protocol might theoretically not achieve coverage

Nodes	τ
25	7
50	8
100	10
200	11
250	12
500	13
1000	14

Figure A.2: Value of τ based on the group size

of *all* members, in all simulations whose results will be shown here it achieved 100% coverage. In the second set, on the other hand, we study how the length of the ξ timeout affects the coverage.

Figure A.1 shows results from the first set of experiments. The graph shows the coverage time, in the vertical axis, matched with the group size in the horizontal axis. In all experiments of this set we have fixed the value for the timeout at $\xi = 2.5$ milliseconds. The coverage time is expressed in milliseconds, while the group size varies between 25 and 1000 nodes. Note that experiments with different number of nodes required different values of τ, as explained in Note 1 and showed in equation A.6. Table A.2 shows the values of τ used for each group size chosen for conducting experiments.

The graph shows that for groups of size smaller than 200 nodes, the coverage time increases with the size of the group in a way approximately exponential. However, this behavior changes for groups of bigger size, where at sensitive increases of group size correspond smaller and smaller increases in coverage times. This behavior finds explanation by considering the effect of concurrency

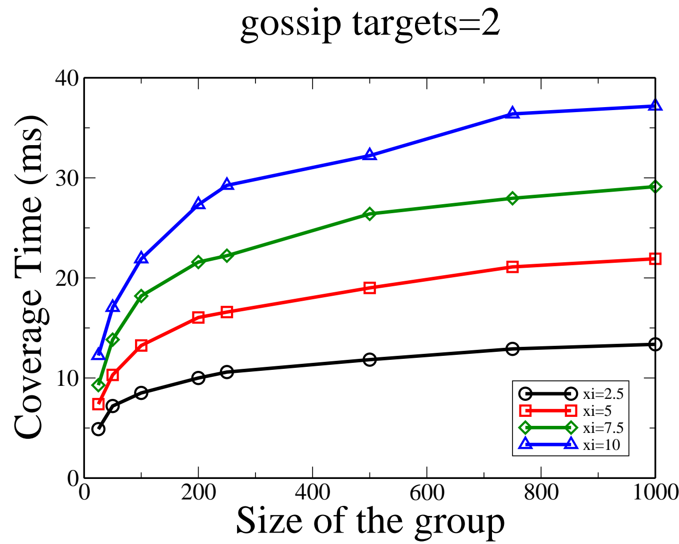

Figure A.3: Performance with various values of ξ, when $\tau = 8$ and gossip targets=2.

in gossips, where processes receiving the token from the originator increase the distribution speed of the token. The obvious conclusion we can draw from what the graph shows is that the protocol performs better with large groups, where concurrency can be better exploited.

Figure A.3 shows results from the second set of experiments, that we use to show how the length of the interval time between subsequent gossip rounds, i.e. ξ, influences the coverage time. In this set we performed essentially the same tests as for the previous set except for the value of ξ, which is let vary as 2.5 ms, 5 ms, 7.5 ms and 10 ms. For each of this values, the protocol has been executed on a group whose size is made growing from 25 to 1000 as for the first set of experiments. In this set of graphs, as well as in the set of graphs previously described, the coverage time, in the vertical axis, is matched with

a variable group size, in the horizontal axis.

The first observation is that the coverage time decreases with the value of ξ. However, a closer look to the graph allows some interesting considerations about these variations. Smaller values of ξ trigger smaller differences in variations of the coverage time when size of the group grows. In fact, when $\xi = 2.5$ milliseconds and the size of the group grows from 25 to 50 members, the corresponding coverage times differs of less then 2 milliseconds. On the contrary, when $\xi = 10$ milliseconds, a similar group growth causes the corresponding coverage times to differ of more than 5 milliseconds. In addition, it is possible to note from the graph how the coverage time stabilizes faster for smaller values of ξ. When this latter parameter is set to 2.5 milliseconds, in fact, the coverage time seems to experience a growth less than proportional to the group size, whereas in the case of ξ being 10 milliseconds the same line seems to suffer more for growth of the group. All these factors lead to the consideration that the protocol seems to sustain better group growth when gossips are carried out faster, and by allowing this there is a clear advantage in scalability.

Bibliography

[1] The Internet 2 Consortium.

[2] PlanetLab: An Open Platform For Developing, Deploying And Accessing Planetary-Scale Services.

[3] TAPAS: Trusted and QoS-Aware Provision of Application Services., 2005.

[4] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin. Lightweight Multicast for Real-Time Process Groups. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 250–259, June 1996.

[5] T. F. Abdelzaher, E. M. Atkins, and K. Shin. Qos negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Software Engineering*, 2000.

[6] B. Adamson, C. Bormann, M. Handley, and J. Macker. Negative-Acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol. Request for Comments 3940 (RFC 3940), November 2004.

[7] R. Al-Ali, A. Hafid, O. Rana, and D. Walker. QoS Adaptation in Service-Oriented Grids. In *Proceedings of the 1st International Workshop on Middleware for Grid Computing (MGC2003)*. ACM/IFIP/USENIX Middleware 2003, June 2003.

[8] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Request for Comments 2581 (RFC 2581), April1999.

[9] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a Communication Sub-System for High Availability. In *Proceeding of the 22nd Symposium of Fault-Tolerance Computing*, pages 76–84, July 1992.

[10] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Improving QoS Routing Performance Under Inaccurate Link-State Information. In *Proceedings of the 16th International Teletraffic Congress (ITC'16)*, June 1999.

[11] V. Apparao, S. Byrne, M. Champion, S. Isaacs, A. L. Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) level 1 specification.

[12] F. Baker, C. Iturralde, F. L. Faucher, and B. Davie. Aggregation of RSVP for IPv4 and IPv6 Reservations. Request for Comments 3175, September 2001.

[13] B. Ban. JGroups - A Toolkit for Reliable Multicast Communication.

[14] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsk. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[15] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[16] K. P. Birman, A. Schiper, and P. Stephenson. Lightweight Casual and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

[17] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society, Los Alamitos, CA, 1994.

[18] S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Internet Draft, December 1998.

[19] R. Bless and K. Wehrle. IP Multicast in Differentiated Services Networks. Internet Draft, February 2003.

[20] S. Bodoff, E. Armstrong, J. Ball, and D. Bode-Carson. *The J2EE Tutorial*. Addison-Wesley Professional, 2004.

[21] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – version 1 functional specification. Request for Comments 2205 (RFC 2205), September 1997.

[22] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (third edition).

[23] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1. W3C Recommendation, February 2004.

[24] J. breslau and S. Shenker. Is Service Priority Useful in Networks? In *Proceeding of ACM SIGMetrics*, pages 66–67, June 1998.

[25] S. Brim, B. Carpenter, A. Jean-Marie, and C. Diot. Per Hop Behavior Identification Codes. Request for Comments 2836 (RFC 2836), May 2000.

[26] N. Brown and C. Kindel. Distributed Component Object Model Protocol - DCOM/1.0. Network Working group Internet Draft, January 1998.

[27] M. Campione and K. Walrath. *The Java(TM) Tutorial*. Addison Wesley, 1998.

[28] W.-K. Chen, M. Hiltunen, and R. Schlichting. Constructing Adaptive Software in Distributed Systems. In *Proceedings of the 21st International Conference on Distributed Computing Systems*. IEEE Computer Society, April 2001.

[29] D. Cooper, P. Ezhilchelvan, and I. Mitrani. High Coverage Broadcasting for Mobile Ad-Hoc Networks. In K. Mitrou, K. Kontovasilis, and

G. Rouskas, editors, *Lecture Notes in Computer Science*, volume 3042 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2004.

[30] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, second edition.* The MIT Press, 2001.

[31] P. Costa and G. P. Picco. Semi-Probabilistic Content-Based Publish-Subscribe. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS05)*, pages 575–585. IEEE Computer Society, June 2005.

[32] S. Deering. Host Extensions for IP Multicasting. Request for Comments 1112 (RFC 1112), August 1989.

[33] C. Delporte-Gallet and H. Fauconnier. An Example of Real-Time Group Communication System. In *Proceedings of the 21th IEEE International Conference on Distributed Computing Systems Workshops*, pages 5–10. IEEE Computer Society, April 2001.

[34] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic Algorithms for Replicated Database Maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, 1987.

[35] A. Di Ferdinando and P. Ezhilchelvan. Integration of a Group Communication Protocol into JGroups. TAPAS EU-IST Project Deliverable D13: Second Year Evaluation and Assessment Report, March 2004.

[36] A. Di Ferdinando, P. Ezhilchelvan, M. Dales, and J. Crowcroft. A QoS-Negotiable Middleware System for Reliably Multicasting Messages of Arbitrary Size. In *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-TIme Distributed Computing (ISORC 2006)*, April 2006.

[37] A. Di Ferdinando, P. Ezhilchelvan, and I. Mitrani. Design and Evaluation of a QoS-Adaptive System for Reliable Multicasting. Submitted to IEEE Transactions on Dependable and Secure Computing.

[38] A. Di Ferdinando, P. Ezhilchelvan, and I. Mitrani. Design and Evaluation of a QoS-Adaptive System for Reliable Multicasting. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2004)*, pages 31–40. IEEE Computer Society, October 2004.

[39] A. Di Ferdinando, P. Ezhilchelvan, and I. Mitrani. Performance Evaluation of a QoS-Adaptive Reliable Multicast Protocol. Technical report, University of Newcastle, April 2004.

[40] A. Di Ferdinando, P. Ezhilchelvan, I. Mitrani, and G. Morgan. Qos-adaptive group communication. TAPAS EU-IST Project Deliverable D8. http://tapas.sourceforge.net/deliverables/D8.pdf, May 2004.

[41] T. Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design.* Prentice Hall, 2005.

[42] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[43] P. Eugster and R. Guerraoui. Probabilistic Multicast. In *Proceeding of the International Conference on Dependable Systems and Networks (DSN2002)*, pages 313–322, 2002.

[44] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast, July 2001.

[45] P. Ezhilchelvan and S. Shrivastava. *Architecting Dependable Systems*, chapter A Model and a Design Approach to Building QoS Adaptive Systems, pages 221–246. Number LNCS 3069. Springer, 2004.

[46] D. Fallside and P.Walmsley. XML Schema Part 0 Primer Second Edition.

[47] V. Firoiu, J.-Y. Le-Boudec, D. Towsley, and Z.-L. Zhang. Theories and Models for Internet Quality of Service. *Proceeding of the IEEE*, 90(9):1565–1591, September 2002.

[48] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[49] M. Fowler and K.Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison Wesley, 1999.

[50] A. O. Freier, P. Karlton, and P. C. Cocher. The SSL Protocol Version 3.0. IETF Internet Draft, November 1996.

[51] X. Fu. Analysis on RSVP Regarding Multicast. Internet Draft, December 2002.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

[53] E. Gelenbe. Sensible Decisions Based on QoS. *Computational Management Science*, 1(1):1–14, December 2003.

[54] E. Gelenbe, R. Lent, A. Di Ferdinando, and A. Manzalini. An Autonomic Networked Auction System. In *International Transactions on Systems Science and Applications*, 2006.

[55] E. Gelenbe, R. Lent, and A. Nunez. Self-Aware Networks and QoS. In *Proceedings of the IEEE*, volume 92, pages 1478–1489, September 2004.

[56] O. M. Group. The Common Object Request Broker: Architecture and Specification - version 2.6, December 2001.

[57] R. Guerin, D. Williams, and A. Orda. Qos routing mechanisms and ospf expensions. In *Proceedings of GLOBECOMM*, 1997.

[58] V. Hadzillacos and S. Toueg. *Distributed Systems*, chapter Fault-tolerant Broadcasts and Related Problems, pages 97–146. Addison-Wesley, 1993.

[59] V. Hadzillacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems*, chapter Distributed Systems, pages 97–146. Addison-Wesley, 1993.

[60] A. Hafid, G. V. Bochmann, and B. Kerherve. A Quality of Service Negotiation Procedure for Distributed Multimedia Presentational Applications. In *Proceeding of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC-5)*, 1996.

[61] J. Harris, H. Stocker, and W. Harris. *Handbook of Mathematics and Computational Science*. Springer, 1998.

[62] M. Hayden. *The Ensemble System*. PhD thesis, Department of Computing Science, Cornell University, 1998.

[63] M. Hayden and K. Birman. Probabilistic Broadcast. technical Report TR 96-1606, Department of Computing Science, Cornell University, 1996.

[64] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. Request for Comments 2597 (RFC 2597), June 1999.

[65] A. Hiles. *The Complete Guide to IT Service Level Agreements: Aligning IT Service to Business Needs*. Rothstein Associates, 2002.

[66] M. Hiltunen, R. Schlichting, X. Han, M. Cardozo, and R. Das. Real-Time Dependable Channel: Customizing QoS Attributes for Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, June 1999.

[67] M. Hofman. Enabling Group Communication in Global Networks. *Proceedings of Global Networking*, 1997.

[68] H. Holbrook, S. Singhal, and D. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of ACM SIGCOMM '95*. IEEE Computer Society, August 1995.

[69] A. Holub. *Holub on patterns. Learning Design Patterns by looking at code.* apress, 2003.

[70] N. Hutchinson and L. Peterson. The x-kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[71] V. Jacobson, K.Nichols, and K. Poduri. An Expedited Forwarding PHB. Request for Comments 2598 (RFC 2598), June 1999.

[72] W. Jianxin, C. Jianer, and C. Songqiao. An Effective Randomized QoS Routing Algorithm on Nework with Inaccurate Parameters. *Journal of Computer Science and technology*, 17(1):38–46, January 200.

[73] W. Jianxin, W. Weiping, C. Jianer, and C. Songqiao. A Randomized QoS Routing Algorithm On Networks with Inaccurate Link-State Information. In *Proceeding of the International Conference on Communication technologies (ICCT2000)*, August 2000.

[74] M. Kadansky, D. Chiu, and J. Wesley. Tree-Based Reliable Multicast (TRAM). Internet Draft, May 1999.

[75] C. Kappler. A QoS Model for Signaling IntServ Controlled-Load Service with NSIS. Internet Draft, May 2005.

[76] A.-M. Kermarrec, L. Massoulie, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.

[77] K.Nichols and S.Blake. Differentiated Services Operational Model and Definitions. Internet Draft, February 1998.

[78] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed Dystem. *Communications of the ACM*, 7(3):189–208, July 1978.

[79] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On Quality of Service Optimization with Discrete QoS Options. In *Proceeding of IEEE Real-time Technology Application Symposium*, June 1999.

[80] S. Liang. *Java(TM) Native Interface: Programmer's Guide and Specification*. Addison-Wesley Professional, 1999.

[81] K. Lidl, J. Osborne, and J. Malcome. Drinking from the firehose: Multicast USENET News. In *Proceeding of USENET Writer*, pages 33–45, Berkeley, CA, January 1994. USENIX Associates.

[82] M.-J.Lin and K.Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *Proceedings of European Dependable Computing Conference (EDCC-3)*, 2000.

[83] Q. Ma and P. Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. In *proceesings of IEEE International Conference on Network Protocols*, October 1997.

[84] Q. Ma, P. Steenkiste, and H. Zhang. Routing High-Bandwidth Traffic in Max-Min Fair Share Networks. In *Proceeding of ACM SIGCOMM'96*, pages 206–217, 1996.

[85] M. May, J. Bolot, A. Jean-Marie, and C. Diot. Simple Performance Models of Differentiated Services Schemes for the Internet. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1385–94, March 1999.

[86] J. McGovern, S. Tyagi, M. Stevens, and S. Mathew. *Java Web Services Architecture*. Morgan Kaufmann, 2003.

[87] S. Microsystems. Java DataBase Connectivity.

[88] M. Miley. Reinventing Business: Application Service Providers. *Oracle Magazine*, pages 48–52, December 2000.

[89] S. Mishra, L. Peterson, and R. Schlichting. A Communication Substrate for Fault-Tolerant Distributed Programs. *DIstributed Systems Engineering*, 17(1):87–103, December 1993.

[90] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: a Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, 39(4):54–63, April 1996.

[91] H. Naser, A. Leon-Garcia, and O. Aboul-Magd. Voice over Differentiated Services Architecture for the Internet. Internet Draft, December 1998.

[92] A. Orda. Routing with End-to-End QoS Guarantees in Broadband Networks. *IEEE/ACM Transactions on Networking*, 7:365–374, June 1999.

[93] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.

[94] E. Pitt, K. McNiff, and K. McNiff. *Java RMI: The Remote Method Invocation Guide*. Addison Wesley, 2001.

[95] J. Postel. User Datagram Protocol. Request for Comments 768 (RFC 768), August 1980.

[96] J. Postel. Transmission Control Protocol. Request for Comments 793 (RFC 793), September 1981.

[97] K. Potter-Kihlstrom, L. Moser, and P. Melliar-Smith. The SecureRing Group Communication System. *ACM Transactions on Information and System Security (TISSEC)*, 4(4):371–406, November 2001.

[98] B. Rajagopalan and H. Sandick. A Framework for QoS-Based Routing in the Internet. Request for Comments 2386 (RFC2386), August 1998.

[99] L. Rodrigues and P. Verissimo. xAMP: a Multi-Primitive Group Communication Service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, October 1992.

[100] A. Shaikh, J. Rexford, and K. Shin. Dynamics of Quality of Service Routing with Inaccurate Link-State Information. technical Report CSE-TR-350-97, University of Michigan, 1997.

[101] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. Request for Comments 2212 (RFC2212), September 1997.

[102] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, S. Sumanasekera, and L. Vicisano. PGM Reliable Transport Protocol Specification. Request for Comments 3208 (RFC 3208), December 2001.

[103] W. T. Strayer, B. J. Dempsey, and A. C. Weaver. XTP: The Xpress Transport Protocol, 1992.

[104] Q. Sun and D. Sturman. A Gossip-Based Reliable Multicast for Large-Scale High-Throughput Applications. In *Proceeding of the 27th IEEE International Conference on Dependable Systems and Networks (DSN2000)*, July 2000.

[105] T. J. Team. The JBoss Application Server.

[106] F. Travostino, E. Menze, and F. Reynolds. Paths: Programming with System Resources in Support of Real-Time Distributed Applications. In *Proceedings of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems*, February 1996.

[107] R. van Renesse, K. P. Birman, R. Friedman, M. Hayden, and D. Karr. A Framework for Protocol Composition in Horus. In *Proceedings of the 14th ACM Principles of Distributed Computing Conference*, pages 80–89, August 1995.

[108] R. van Renesse, K. P. Birman, and S. Maffeis. Horus, a Flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, April 1996.

[109] X. Wang and H. Schulzrinne. RNAP: A Resource Negotiation and Pricing Protocol. In *Proceedings of the International Workshop Network Operating Systems Support Digital Audio Video (NOSSDAV99*, 1999.

[110] X. Wang and H. Schulzrinne. An Integrated Resource Negotiation, Pricing, and QoS Adaptation Framework for Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 18(12):2514–2529, December 2000.

[111] Z. Wang and J. Crowcroft. Quality of Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, 1996.

[112] W. Weizhao, X. Y. Li, S. Z. N. Megiddo, X. Yinfeng, and Z. Binhai. Design DiffServ Multicast with Selfish Agents. In *Proceedings of the First International Conference on Algorithmic Applications in Management*, volume 3521, pages 214–223, 2005.

[113] B. Whetten, S. Paul, T. Montgomery, and N. Rastogi. The RMTP-II Protocol. Internet Draft, April 1998.

[114] G. Wong, M. Hiltunen, and R. Schlichting. A Configurable and Extensible Transport Protocol. In *Proceeding of the 20th Annual Conference of IEEE Communications and Computer Societies (INFOCOM 2001)*, pages 319–328. IEEE Computer Society, 2001.

[115] J. Wroclawski. Integrated Services (IntServ).

[116] Z. Xiao and K. P. Birman. A Randomized Error Recovery Algorithm for Reliable Multicast. In *INFOCOM*, pages 239–248, 2001.

[117] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. *Proceeding of ACM Multimedia*, 1995.