



INSTITUTO
SUPERIOR
TÉCNICO

UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Topology Management for Unstructured Overlay Networks

João Carlos Antunes Leitão

Supervisor: Doctor Luís Eduardo Teixeira Rodrigues

Thesis approved in public session to obtain the PhD Degree in
Information Systems and Computer Engineering

Jury final classification: Pass with Distinction

Jury:

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

Doctor Luís Eduardo Teixeira Rodrigues

Doctor Anne-Marie Kermarrec

Doctor Teresa Maria Sá Ferreira Vazão Vasques

Doctor Henrique João Lopes Domingos

2012



INSTITUTO
SUPERIOR
TÉCNICO

UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Topology Management for Unstructured Overlay Networks

João Carlos Antunes Leitão

Supervisor: Doctor Luís Eduardo Teixeira Rodrigues

Thesis approved in public session to obtain the PhD Degree in
Information Systems and Computer Engineering

Jury final classification: Pass with Distinction

Jury:

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

- Doctor **Luís Eduardo Teixeira Rodrigues**, *Professor Catedrático do Instituto Superior Técnico, da Universidade Técnica de Lisboa*
Doctor **Anne-Marie Kermarrec**, *Directrice de recherche, do Institut National de Recherche en Informatique et en Automatique, France*
Doctor **Teresa Maria Sá Ferreira Vazão Vasques**, *Professora Associada do Instituto Superior Técnico, da Universidade Técnica de Lisboa*
Doctor **Henrique João Lopes Domingos**, *Professor Auxiliar da Faculdade de Ciências e Tecnologia, da Universidade Nova de Lisboa*

Funding Institutions:

FCT: Fundação para a Ciência e Tecnologia
INESC-ID: Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa

2012

Acknowledgements

The work described in this pages would never be possible to be conducted without the support of many people, to whom I am in debt. First of all, I have to thank my advisor, Professor Luís Rodrigues, for providing me all the conditions and the environment to explore new ideas, and to look for excellence in my research. His motivation, his good nature, and his principles remain to this day an inspiration, and the opportunities that he offered me to real understand what is to be a researcher were essential to make me the person I am today.

Evidently I have to thank all my co-authors, Robbert van Renesse, José Orlando Pereira, Benoit Garbinato, Mouna Allani, Rui Oliveira, Nuno A. Carvalho, Liliana Rosa, João Alveirinho, Mário Ferreira, João Marques, and João Paiva, for the opportunity to collaborate and learn from them. Working with them has provided me the opportunity to broaden the spectrum of my research and assisted me in growing as a scientist.

The Distributed Systems Group (GSD) of INESC-ID was, and still is, a great environment to work. Great environments are a result of the people that make it. Therefore, I wish to thank the present and past elements of the GSD group with whom I have shared ideas and experiences for more than four years. With no particular order (and hopping not to forget anyone) my sincere thanks to (in no particular order): André Pessoa, Carlos Ribeiro, Carlos Torrão, Cristina Fonseca, Diogo Mónica, Diogo Paulo, José Mocito, João Barreto, João Ferreira, João Garcia, João Matos, João Nuno Silva, Luís Veiga, Maria Couceiro, Mauro Silva, Miguel Branco, Miguel Correia, Nuno Carvalho, Nuno Machado, Oksana Denysyuk, Paolo Romano, Paulo Ferreira, Pedro Pedrosa, Rodrigo Rodrigues, Rui Joaquim, Sérgio Garrau, and Xavier Vilaça.

The INESC-ID laboratory, where I have conducted the work presented in the thesis, has a great work environment. It would never be possible to list everyone in INESC-ID to whom I feel grateful. However, a special word of appreciation to David Matos, Luísa Coheur, and Rito Silva, for their good spirits, help, discussions, and friendship. Also, a big thanks for the administrative people of INESC-ID which were always helpful and many times have brighten

my days, in particular and with no particular order: Vanda Fidalgo, Teresa Mimoso, Manuela Sado, Paula Monteiro, Elisabete Rodrigues, and Aurélia Constantino.

A special word of appreciation to all my friends (listing them all would be impossible), that supported me, helped me, and really were patient enough to stay by my side when I complained about the hardships of life. A special word of appreciation to Alexandre Chícharo that also revised the text presented here.

Also, a big thanks to my family, in particular to my mother Natalina Leitão, and my brother Paulo Leitão that always without any doubt supported me and my decisions. A special word for my late grandfather Manuel Claudino, which every other week would ask me “Have you finished school yet?”. I never really explained to him that I was not attending school as he thought in his mind, but I know that if I could say to him now “I’m done!” he would have given me his best smile. His intelligence and his capacity to continually work and make efforts will never stop being an inspiration.

Evidently, the work here was also affected by so many other people in my life, even if in different measures. Because of that I have to thank my ex-colleagues in CIISEG, the LaSIGE laboratory at FCUL (in particular my old room companions, in particular Hugo Bastos, and the elements of the now extinct DiALNP research group) for everything, and my undergraduate teachers at FCUL. Also, my thanks to the member of the Zen Shin Kan Iaido Clube de Lisboa (and my Sensei Joaquim Mendes in particular), for the excellent environment they provided for me to *let go some steam* during the course of my PhD. In particular, to Pedro Gomes and Maria Luís for everything inside and outside the dojo.

A final word of appreciation to the Fundação para a Ciência e Tecnologia (FCT), for their financial support.

The work presented in the thesis was supported by the Fundação para a Ciência e Tecnologia (FCT) through:

- The PhD grant with reference “SFRH/BD/35913/2007”;
- The FCT founded project “Redico” (PTDC/EIA/71752/2006);
- The FCT founded project “HPCI” (PTDC/EIA-EIA/102212/2008);
- The FCT founded project “ADAAS” (CMU-PT/ELE/0030/2009);
- The INESC-ID multi-annual funding through the PIDDAC Program fund grant;

To all those that contributed to the work described in these pages, and to all those that will read this in the future and find it useful.

Abstract

The peer-to-peer (P2P) paradigm has emerged as a viable alternative to overcome limitations of the client-server model namely, in terms of scalability, fault-tolerance, and even operational costs. This paradigm has gained significant popularity with its successful application in the context of file sharing applications. The success of these applications is illustrated by systems such as Napster, Emule, Gnutella, and recently, BitTorrent. In order to ensure the scalability of these solutions many P2P services operate on top of unstructured overlay networks, which are logical networks deployed at the application level. Unstructured overlay networks establish random neighboring associations among participants of the system. Although the random nature of these overlays is desirable by many P2P services, the resulting topology may present sub-optimal characteristics, for instance from the point of view of link latency. This may have a significant impact on the performance of P2P services executed over these overlays.

This thesis focuses on the design and evaluation of techniques that manage the topology of unstructured overlay networks, to understand if and how it is possible to manage the topology of unstructured overlays in such a way that the random nature and low overhead that characterizes these overlays is not lost, while being able to impose some relaxed constraints over the topology to benefit the operation of specific P2P services.

To answer this question, four different approaches to manage the topology of unstructured overlays are proposed and evaluated in the thesis. Each approach is evaluated in the context of a distinct P2P service that serves as a case study for measuring its benefits. In more detail, this thesis presents: *i)* CellFarm, a new overlay that combines properties of unstructured and structured overlays, to achieve a highly resilient topology composed of cliques of nodes, highly connected among themselves, that supports efficient replication for P2P systems; *ii)* X-BOT, a protocol to bias the topology of unstructured overlay networks given any criteria X ; *iii)* Thicket, a protocol that efficiently embeds multiple interior-node-disjoint trees over a single unstructured overlay; and finally, *iv)* OpenFire, a protocol for balancing rumor mongering exchanges in networks populated by Firewalls and NAT-boxes.

Resumo

O paradigma entre-pares (P2P, do Inglês *peer-to-peer*) surge como uma alternativa viável para ultrapassar as limitações do modelo cliente-servidor nomeadamente, no que diz respeito à capacidade de escala, tolerância a falhas, e até mesmo em termos de custos monetários operacionais. Este paradigma ficou popularizado com o sucesso de aplicações de partilha de ficheiros. O sucesso e relevância desta classe de aplicações é ilustrado por exemplos como os sistemas Napster, Emule, Gnutella, e mais recentemente o protocolo de distribuição entre-pares BitTorrent. De forma a garantir a capacidade de escala dos sistemas baseados no paradigma entre-pares, é frequente estes sistemas operarem sobre uma rede sobreposta não estruturada: uma rede lógica estabelecida ao nível da aplicação. As redes sobrepostas não estruturadas estabelecem relações de vizinhança aleatórias entre os participantes do sistema. Apesar da natureza aleatória destas redes ser útil à operação de vários serviços entre-pares, a topologia resultante pode apresentar um conjunto de características sub óptimas por exemplo, a topologia da rede sobreposta pode ser definida, maioritariamente, por ligações que apresentam elevada latência. Consequentemente, a topologia das redes sobrepostas pode ter um impacto significativo no desempenho de serviços entre-pares executados sobre estas.

Esta tese foca-se no desenvolvimento e avaliação de técnicas utilizadas para gerir a topologia de redes sobrepostas não estruturadas. Adicionalmente, esta tese pretende avaliar se é possível gerir a topologia destas redes de tal forma que a sua natureza aleatória, e o baixo custo de manutenção, característicos das redes sobrepostas não estruturadas, não sejam comprometidos, apesar de se imporem constrangimentos adicionais sobre a topologia da rede, por forma a beneficiar a operação de serviços entre-pares.

De forma a responder a esta questão, a tese propõe e avalia um conjunto de quatro mecanismos para gerir a topologia de redes sobrepostas não estruturadas. Cada um dos mecanismos propostos é avaliado no contexto de um serviço entre-pares distinto, que serve de caso de estudo de forma a ilustrar os benefícios que são possíveis de alcançar através da solução proposta. Em maior detalhe a tese apresenta: *i*) CellFarm, uma nova rede sobreposta não estruturada cujo

desenho combina propriedades de redes sobrepostas estruturadas e não estruturadas, o que lhe permite alcançar uma topologia composta por cliques de nós, que possuem um grande e variado número de ligações entre si. A topologia do CellFarm constitui um abstracto adequado para suportar replicação eficiente em sistemas entre-pares; *ii*) X-BOT, um protocolo capaz de enviesar a topologia de uma rede sobreposta não estruturada tendo como critéria uma medida de desempenho arbitrária X ; *iii*) Thicket, um protocolo capaz de inscrever múltiplas árvores de disseminação sobre uma única rede sobreposta não estruturada, de tal forma que a larga maioria dos nós apenas comporta-se como nó interior numa única árvore; e finalmente, *iv*) OpenFire, um protocolo capaz de balancear as trocas de rumores entre nós num ambiente populado por *Firewalls* e *NATs*.

Palavras Chave Keywords

Keywords

Peer-to-Peer Systems

Unstructured Overlay Networks

Overlay Networks Topologies

Topology Management

Gossip Protocols

Palavras Chave

Sistemas Entre-Pares

Redes Sobrepostas Não Estruturadas

Topologia de Redes Sobrepostas

Gestão de Topologias

Protocols de Rumor

Index

1	Introduction	1
1.1	Problem Statement	3
1.2	Contributions Summary	9
1.2.1	CellFarm	9
1.2.2	X-BOT	9
1.2.3	Thicket	10
1.2.4	Open Fire	10
1.3	Results Summary	11
1.4	Ramifications and Collaborations	11
1.5	Research History	12
1.6	Thesis Structure	13
2	Fundamental Concepts and State of Art	15
2.1	Peer-to-Peer Services	15
2.1.1	Resource Location	16
2.1.1.1	Architectures	17
2.1.1.2	Type of queries	20
2.1.1.3	Query Dissemination Strategies	21
2.1.1.3.1	Flooding	21
2.1.1.3.2	Random Walks	22

2.1.2	Gossip-based Dissemination Protocols	23
2.1.2.1	Relevant Parameters	23
2.1.2.2	Communication Modes	25
2.1.3	Rumor Mongering	27
2.1.3.1	State Reconciliation	27
2.1.3.2	Behavior Under Unbalanced Networks	28
2.2	Performance Metrics	30
2.2.1	Direct Metrics	30
2.2.2	Indirect Metrics	32
2.2.2.1	Resource Location	32
2.2.2.2	Gossip-based Dissemination	33
2.2.2.3	Rumor Mongering	34
2.2.3	Overlay Topology and Application-Level Performance Metrics	34
2.3	Overlay Networks	35
2.3.1	Structured Overlay Networks	37
2.3.2	Unstructured Overlay Networks	38
2.4	Topology Management of Unstructured Overlay Networks	41
2.4.1	Management at the Overlay Layer	41
2.4.1.1	Main Approaches	42
2.4.1.2	Strengths and Limitations	42
2.4.1.3	Previous Work	43
2.4.2	Management at the P2P Service Layer	45
2.4.2.1	Main Approaches	46
2.4.2.2	Strengths and Limitations	47
2.4.2.3	Previous Work	48

3	Control the Topology: CellFarm	51
3.1	Motivation and Goals	51
3.1.1	Motivation	51
3.1.2	Goals	54
3.2	The CellFarm Protocol	55
3.2.1	Rationale	56
3.2.2	Algorithm	58
3.2.2.1	Join Procedure	59
3.2.2.2	CellFarm Divide Procedure	61
3.2.2.3	Collapse Procedure	64
3.2.2.4	External Neighboring Procedure	65
3.2.2.5	Anti-entropy Procedure	66
3.2.3	Increasing Fault-Tolerance	67
3.3	Case Study	67
3.3.1	Search Strategies	68
3.3.2	Baseline Strategy	69
3.3.3	Cell-Aware Strategy	69
3.4	Evaluation	71
3.4.1	Experimental Setting	71
3.4.1.1	Tested Topologies	72
3.4.1.2	Query Flood Strategy	73
3.4.1.3	Number of Experiments	73
3.4.2	Overlay Properties	74
3.4.2.1	Overlay Characterization in Steady State	74

3.4.2.2	Fault-Tolerance	77
3.4.2.3	Churn Scenario	79
3.4.3	CellFarm Support for Resource Location	80
3.5	Related Work	82
3.6	Discussion	84
4	Bias the Topology: X-BOT	87
4.1	Motivation and Goals	87
4.1.1	Motivation	87
4.1.2	Goals	88
4.2	The X-BOT Protocol	89
4.2.1	Rationale	89
4.2.2	Algorithm	92
4.2.2.1	Step 1	94
4.2.2.2	Step 2	94
4.2.2.3	Step 3	95
4.2.2.4	Step 4	96
4.2.2.5	Variant	96
4.3	Oracles	97
4.3.1	Oracle Implementations	97
4.3.2	Combining Oracles	99
4.3.3	On the Use of Inconsistent Oracles	100
4.3.4	Oracle Cost	100
4.4	Configuring X-BOT	101
4.4.1	Active View Size:	101

4.4.2	Passive View Size Control Parameter (k):	101
4.4.3	Period Between Optimizations (PBO):	102
4.4.4	Passive Scan Length (π):	102
4.4.5	Number of Unbiased Neighbors (μ):	103
4.5	Evaluation	103
4.5.1	Experimental Setting	103
4.5.2	X-BOT Performance	105
4.5.2.1	X-BOT Individual Evaluation	106
4.5.2.1.1	Overlay cost	106
4.5.2.1.2	Effect on the Average link cost	107
4.5.2.1.3	Clustering coefficient and average shortest path	109
4.5.2.2	Comparative Evaluation	111
4.5.2.2.1	Overlay Cost	111
4.5.2.2.2	Scenario with several Internet Service Providers	114
4.5.3	X-BOT Support for Broadcast	116
4.5.3.1	Steady State	117
4.5.3.2	Fault Tolerance	118
4.6	X-BOT Properties	119
4.6.1	Complexity	120
4.6.2	Avoiding Local Minima Configurations	120
4.6.3	Ensuring Low Clustering Coefficient	121
4.6.4	Ensuring Low Average Shortest Path	122
4.7	Related Work	122

5	Embed the Topology: Thicket	127
5.1	Motivation and Goals	127
5.1.1	Motivation	127
5.1.2	Goals	128
5.2	Preliminaries	129
5.2.1	Underlying Unstructured Overlay Network	129
5.2.2	Plumtree Protocol	131
5.2.3	Tree Construction	132
5.2.4	Tree Repair	132
5.2.5	Network Dynamics	133
5.3	The Thicket Protocol	134
5.3.1	Rationale	134
5.3.2	Algorithm	136
5.3.3	Architecture	136
5.3.4	Tree Construction	139
5.3.5	Tree Repair	140
5.3.6	Tree Reconfiguration	142
5.3.7	Network Dynamics	143
5.3.8	Configuring Thicket	143
5.4	Case Study	144
5.4.1	P2P Steaming Service	144
5.4.2	Prototype Implementation	145
5.4.2.1	Components	145
5.4.2.2	Practical Challenges	146

5.5	Evaluation	148
5.5.1	Experimental Setting	148
5.5.1.1	Simulation Setting	148
5.5.1.2	PlanetLab Setting	149
5.5.2	Thicket Performance	150
5.5.2.1	Simulation Results	150
5.5.2.1.1	Stable Environment	151
5.5.2.1.2	Fault-Tolerance	152
5.5.2.2	PlanetLab Deployment Results	153
5.5.2.2.1	Stable Environment	153
5.5.2.2.2	Faulty Environment	154
5.5.3	Thicket Support for Streaming	155
5.5.3.1	Simulation Results	155
5.5.3.1.1	Stable Environment:	155
5.5.3.1.2	Fault-Tolerance:	157
5.5.3.2	PlanetLab Deployment Results	158
5.5.3.2.1	Stable Environment	158
5.5.3.2.2	Faulty Scenario	159
5.6	Related Work	160
6	Enrich the Topology: OpenFire	165
6.1	Motivation and Goals	165
6.1.1	Motivation	165
6.1.2	Goals	167
6.2	The Open Fire Protocol	167

6.2.1	Rationale	167
6.2.2	Underlying Unstructured Overlay Network	169
6.2.3	Protocol	170
6.3	Case Study	173
6.4	Evaluation	173
6.4.1	Experimental Setting	173
6.4.2	Experimental Results	174
6.5	Related Work	177
7	Conclusions and Future Work	181
7.1	Conclusions	181
7.2	Future Work	186

List of Figures

- 1.1 A generic architecture for peer-to-peer applications. 7
- 1.2 Approaches for managing and leveraging the topology of unstructured overlay networks and mapping to the thesis structure. 8
- 3.1 The CellFarm overlay. Smaller (red) dots represent physical nodes, while larger (yellow) nodes represent virtual nodes. 54
- 3.2 Cell size distribution of CellFarm in a steady state scenario obtained through simulation and a prototype deployment over PlanetLab. 75
- 3.3 Overlay comparison of in-degree distribution and cost for maintaining the topology between CellFarm and an unstructured overlay managed by an improved version of the Scamp protocol. 77
- 3.4 Comparison of Overlay Connectivity and Recovery Time after massive node failures between CellFarm and an unstructured overlay based on an improved version of Scamp. 79
- 3.5 Overlay Connectivity evolution under different churn rates for CellFarm and an unstructured overlay based on an improved version of Scamp. 80
- 4.1 Typical *X*-BOT optimization round involving 4 peers in the unstructured overlay network. 92
- 4.2 Evolution of overlay cost for *X*-BOT when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios. 107
- 4.3 Evolution of clustering coefficient in the *X*-BOT overlay when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios. . . 110

4.4	Evolution of average shortest path in the <i>X</i> -BOT overlay when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios. . .	111
4.5	Cost of the overlay that results from the operation of <i>X</i> -BOT and the remaining tested solutions for the three experimental settings.	112
4.6	In-degree of the overlay that results from the operation of <i>X</i> -BOT and the remaining tested solutions for the three experimental settings.	113
4.7	Fraction of sub-optimal links for different protocols and different number of ISPs in the overlay that results from the operation of <i>X</i> -BOT and the remaining tested solutions	115
4.8	Message dissemination latency using the overlays generated by <i>X</i> -BOT and the remaining tested solutions for each scenario.	117
4.9	Resilience to node failures in terms of connectivity and time required to recover for <i>X</i> -BOT, GoCast, and Araneola.	118
5.1	<i>K</i> -interior node distribution over 5 trees for the NUTS and BOLTS strategies. . .	135
5.2	Results concerning the properties of Thicket in a stable environment.	150
5.3	Performance results for Thicket properties in a catastrophic scenario.	152
5.4	PlanetLab experimental results for the Thicket protocol.	153
5.5	<i>K</i> -interior node distribution obtained by Thicket in a catastrophic scenario. . . .	154
5.6	Forwarding load obtained by Thicket in a catastrophic scenario.	155
5.7	Performance results of the case study when leveraging Thicket in a stable environment.	156
5.8	Reliability of a P2P streaming application leveraging the design of Thicket in a faulty scenario.	157
5.9	Delivery time of data segments for the Thicket prototype in a stable environment.	159
5.10	Delivery time of data segments for the Thicket prototype in a catastrophic scenario.	160
5.11	Reliability of a P2P streaming application leveraging the design of Thicket in a catastrophic scenario.	160

6.1	Max rumor mongering exchanges / peer with OpenFire.	175
6.2	Maximum latency for the rumor mongering when relying on OpenFire	176
6.3	Max forwarded messages / peer for the operation of OpenFire.	176

List of Tables

3.1	Comparison of graph properties between CellFarm and an unstructured solution after stabilization.	75
3.2	Query dissemination latency obtained when using CellFarm and an unstructured overlay based on an optimized version of Scamp.	81
3.3	Query dissemination communication cost of the resource location systems leveraging CellFarm and an unstructured overlay based on an optimized version of Scamp.	82
4.1	Comparison of average link cost, average biased link cost, and average unbiased link cost for several values of parameter μ in <i>X</i> -BOT and in a random overlay network with similar properties.	108
4.2	Comparison of overlay properties of the overlay generated by <i>X</i> -BOT and the remaining tested solutions.	114
4.3	Percentage of nodes in the largest connected cluster for <i>X</i> -BOT and the remaining protocol for different numbers of ISPs.	116
4.4	Comparison of Broadcast Latency and Reliability using overlays generated by <i>X</i> -BOT and the remaining tested solutions.	118

1 Introduction

The peer-to-peer (P2P) paradigm has emerged as a viable approach to overcome limitations of the client-server model namely, in terms of scalability, fault-tolerance, and even operational costs. In a nutshell, scalability is addressed by having each participant in the system (typically designated by *peer*) contributing with its own resources, either in the form of processing power, bandwidth, disk space, etc. Fault-tolerance is improved by avoiding the existence of a single point of failure. Finally, the operational cost of the system can be lowered, by avoiding the need for powerful and expensive servers or centralized infrastructures capable of handling a large number of clients simultaneously.

This paradigm has gained practical relevance in the context of file sharing applications, such as Napster¹, Emule (Kulbak & Bickson, 2005), Gnutella², and more recently BitTorrent (Cohen, 2008). Other widely used applications such as Skype (Baset & Schulzrinne, 2004) and TOR (Dingledine et al., 2004) have also resorted to P2P-based solutions to ensure the scalability of some of their components. More recently, IPTV systems based on the P2P paradigm, such as the PPLive system (Hei et al., 2007), have been deployed and are currently in production with a high commercial success.

Some early P2P systems assume that each participant has access to the full membership information. An example of this is the application-level broadcast system proposed by Chu et al. (2002). This assumption is still valid today in small and medium sized systems. For instance, several one-hop distributed hash tables have been proposed (A. Gupta et al., 2004; Leong & Li, 2004; Risson et al., 2006), which also assume that each participant has access to the complete system filiation. One-hop DHTs have been employed with success for supporting storage systems designed for cloud computing environments, such as Amazon's Dynamo (DeCandia et al., 2007) and Facebook's Cassandra (Lakshman & Malik, 2010).

¹<http://www.napster.com>

²<http://www.gnutella.com>

While these solutions have proven to scale in stable environments, such as large data-centers, they present scalability limitations in large-scale open environments (such as the Internet), where the session time of participants may be short, resulting in a high filiation dynamism. In such systems, individual peers join, leave, and fail concurrently, sometimes at a high rate (a phenomena that is usually designated by *churn* (Stutzbach & Rejaie, 2006)). As a result, the bookkeeping overhead required to maintain complete views of the system up-to-date is prohibitively high (J. Li et al., 2005).

To overcome this limitation, most P2P systems rely on some form of membership protocol, that exposes to each participant a local partial view of the system. These partial views contain the identifiers³ of a (small) fraction of all peers in the system, with whom that participant can interact directly (*e.g.* through the exchange of messages). Ideally, the size of such partial views should grow logarithmically with the number of participants in the system.

Partial views encode neighboring relations among peers and their closure forms an *overlay network* (Stoica et al., 2001; Rowstron & Druschel, 2001; Leitão et al., 2007b; Voulgaris et al., 2005; Ganesh et al., 2003), a logical network that operates at the application level. Typically, each participant in the system is only aware of its overlay neighbors and never communicates directly with other peers. Overlay networks usually belong to one of two classes, according to the mechanisms employed to build and maintain the overlay topology. These classes are named *structured* and *unstructured* (overlay networks).

Structured overlay networks (Stoica et al., 2001; Rowstron & Druschel, 2001; Zhao et al., 2004; I. Gupta et al., 2003) impose constraints to the neighboring relationships that may be established among peers. These neighboring relations are required to follow a global coordination strategy, often based on random unique identifiers selected independently by each participant. As a result, the overlay converges to a topology known a priori; services and applications that use the overlay may leverage on the characteristics of that topology. Typical examples of structured overlay networks are distributed hash tables (DHT). DHTs offer a routing functionality over the node identifier space, using *keys* that belong to that same address space. This enables the system to route messages through the overlay links to the node with an identifier that is “closest” to the destination key. Routing can be performed in a number of steps

³A node identifier usually includes information required to contact the node, such as a pair $\{ip : port\}$ and, in some cases, also includes an unique (or probabilistically unique) identifier, such as a bit string.

that is logarithmic with the number of participants. Due to their efficient routing, structured overlay networks have been used to support many distributed services, including exact match resource location systems (Balakrishnan et al., 2003), distributed storage solutions (Druschel & Rowstron, 2001), and publish subscribe services (Rowstron et al., 2001).

In contrast, unstructured overlay networks (Ganesh et al., 2003; Voulgaris et al., 2005; Leitão et al., 2007b) are characterized by a much more relaxed topology where neighboring associations are random in nature. Due to this random nature, these overlays are usually easier to build and have a lower maintenance overhead when compared with their structured counterparts. Since they are not required to follow any global coordination method, their maintenance may rely on independent or pairwise decisions from participants. By imposing few constraints, such overlays are better at dealing with the filiation dynamics of large-scale distributed systems, as well as with churn scenarios. Typically, unstructured overlay networks are used to support distributed services based on gossip protocols, where nodes exchange information through pairwise (random) interactions. One such service, which has received significant attention from the scientific community in the past, is application level multicast (Birman et al., 1999; Kermarrec et al., 2003; Eugster et al., 2003; Leitão et al., 2007a). Unstructured overlays have also been employed to support other distributed services, such as: resource location (Chawathe et al., 2003; Tsoumakos & Rousopoulos, 2006; Iamnitchi et al., 2002), anti-entropy mechanisms (Demers et al., 1987; Renesse et al., 2003), data aggregation (Jelasity & Montresor, 2004), publish-subscribe (Eugster & Guerraoui, 2002), among others (Renesse et al., 1998; Koldehofe, 2003; H. Li et al., 2006).

1.1 Problem Statement

The properties and nature of the overlay network topology have a high impact on the performance of P2P services and applications executed on top of them. In order to develop distributed services based on the P2P paradigm it is therefore of paramount relevance to study and devise new mechanisms for creating and managing overlay networks, that better match the application requirements.

Overlay topology has been subject to intensive research for the particular case of distributed hash tables (Liben-Nowell et al., 2002; Gummadi et al., 2003; Ghodsi et al., 2007; Stoica

et al., 2001; Rowstron & Druschel, 2001; Maymounkov & Mazières, 2002; Rhea et al., 2004; Dabek et al., 2004). However, fewer results have been obtained for unstructured overlay networks, which more relaxed topologic constraints offers the potential for better fault-tolerance and lower maintenance overhead.

Furthermore, only a few works have explored mechanisms to bridge the gap between unstructured and structured overlay networks (Maniymaran et al., 2007; Carvalho et al., 2007; Leitão et al., 2007a). These works have shown that combining features of structured and unstructured overlay networks is a promising path to devise new and more efficient overlay topologies. The topic of improving overlay network topologies, particularly in the context of gossip-based protocols and unstructured overlay networks, has been identified as a relevant research field by Birman (Birman, 2007).

Considering this context, the thesis addresses the following question:

What kind of techniques can be devised to provide some degree of structure to overlay networks, such that the performance of the P2P applications can be improved without compromising the robustness and low cost of random overlays?

The work presented in the thesis also addresses additional questions, which are related to the main focus of the thesis on managing the topology of unstructured overlay networks.

The topology of unstructured and structured overlay networks is managed following two different approaches. On one hand unstructured overlay topologies are defined at random, by having nodes establishing neighboring relations among them in an uncoordinated and random fashion. This results in a highly flexible topology, able to easily reconfigure itself in face of membership dynamics (*e.g.*, churn). However, the topology itself cannot be easily leveraged to offer additional support to services, such as application-level routing. On the other hand, structured overlay networks have topologies which are defined by following hard constraints that take into consideration the identifiers of peers in the system. This makes the topology of these overlays highly inflexible, however it allows the topology to offer additional functionality to services, such as highly efficient application-level routing.

The exiting gap between the topologies of unstructured and structured overlay networks motivates additional research to combine, in a single overlay network, aspects from the management schemes usually employed by each overlay type. This would enable the design of

overlays that present the flexibility, and consequent robustness, of unstructured overlays while at the same time offering additional support and functionalities to P2P services executed on top of them. More precisely, the thesis will also address the following question:

- Is it possible to design overlay networks that fill the gap between structured and unstructured solutions design and functionality?

A common problem found in overlay networks in general, and in particular in unstructured overlay networks, is that of the *topology mismatch* (Hsiao et al., 2009), where the logical topology defined by the overlay is oblivious to the properties of the underlying network. This usually results in the overlay network owning several sub-optimal links leading to (potentially high) penalties over the performance of P2P services that leverage the topology offered by the overlay. Previous work has addressed how to overcome the topology mismatch problem, by either taking into consideration some underlay performance criteria, such as link latency (Massoulié et al., 2003; Tang & Ward, 2005; Melamed & Keidar, 2004). However, this does not avoid the topology of an unstructured overlay to be inadequate when considering a P2P service or application specific requirements.

This motivates additional research to devise a solution that is able to adapt the topology of unstructured overlay networks in a more flexible fashion, allowing the topology of the overlay to better match either underlay performance criteria or high level application requirements, while at the same time protecting the relevant properties of the overlay, namely the connectivity, balanced node degree, and low clustering coefficient. To address this challenge the work presented in the thesis tries to find an answer for the following question:

- Can a generic scheme be designed to adapt the topology of unstructured overlay networks that can take into consideration an arbitrary criteria, while at the same time protecting the relevant properties of random overlays?

Multimedia streaming over the Internet can highly benefit from schemes that leverage on spanning trees, as to avoid the consumption of network resources to disseminate (potentially unbounded) amounts of redundant information. Previous work (Leitão et al., 2007a) has already shown how to combine tree-based topologies with unstructured overlays. Unfortunately, relying on a single spanning tree results in a sub-optimal use of available resources in a P2P

system. This motivates additional research to devise efficient mechanisms to combine several spanning tree structures over a single unstructured overlay network. Such an approach would enable all peers in the system to share the load of the dissemination process in a more balanced fashion, while also offering the opportunity to use a larger portion of available network resources in the system. To tackle this challenge the thesis will also address the following question:

- Can one devise mechanisms to combine the properties of tree-based and unstructured overlays ensuring both robustness to node failures and an adequate load distribution across all peers?

Finally, a question which is relevant to the correctness of unstructured overlay network's topology and the correctness of services executed on top of them, is the presence of Firewall and NAT boxes in the underlay. Such components, limit the communication patterns that can be established among peers of a P2P system. The large portion of P2P solutions found in the literature assume a uniform communication pattern. The presence of Firewalls and NAT boxes result in biased interactions among peers, where peers which are located in the public network (*i.e.* which are not behind a Firewall or a NAT) will be contacted much more often than the remaining peers. To address these scenarios, the thesis will also focus on the following question:

- Is it possible to devise solutions which mask the presence of Firewalls and NAT boxes without exposing the topology of the underlay to the upper layers?

To address the questions presented above, we model the architecture of P2P systems as depicted in Figure 1.1. This model, based on the work presented by Aberer et al. (2005), captures the architecture of a P2P system, from the point of view of a single peer, as a composition of four layers. In the following we present a brief description of each layer.

Network Layer The network layer is responsible for exposing an interface to the transport layer offered by the operative system. In particular, this layer is responsible for queuing messages to be sent to other peers in the system, and also to receive and deliver messages received from other peers to the relevant protocols above. Additionally, and in the particular case where TCP is used as the transport protocol, this layer should notify the Overlay

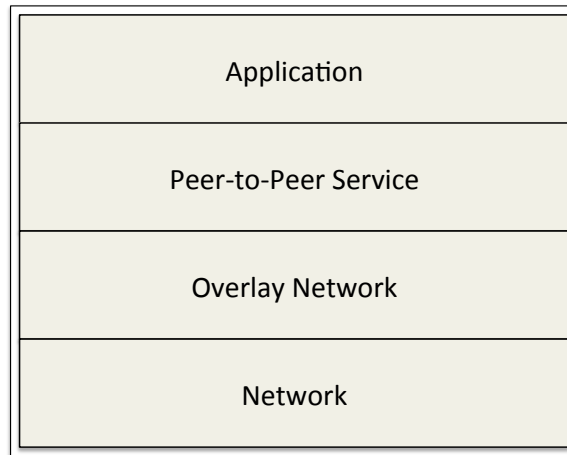


Figure 1.1: A generic architecture for peer-to-peer applications.

Network layer whenever a TCP connection to a peer closes, offering an unreliable failure detection service.

Overlay Network The overlay network layer is responsible for managing the logical network that connects all peers in the system. This is achieved by executing a distributed membership protocol to provide at each peer a set of neighbors, which correspond to overlay links. This layer should expose the overlay neighbors to the layer above and should be able to notify those protocols when changes occur in the (local) overlay topology.

P2P Service This layer is responsible for executing particular P2P services, by taking advantage on the P2P overlay maintained by the layer below it. Services may include gossip-dissemination services, routing mechanisms, anti-entropy protocols, among others. Services provided by this layer are exposed to the Application layer above.

Application The application layer is responsible for implementing the application logic and eventually exposing an interface to the user. This layer relies on the P2P Service layer to operate.

Taking into consideration the generic architecture presented above, we envision four different approaches to manage and leverage the topology of unstructured overlay networks as depicted in Figure 1.2. In the thesis we consider and explore the following approaches operating at the Overlay Network and P2P Service layers:

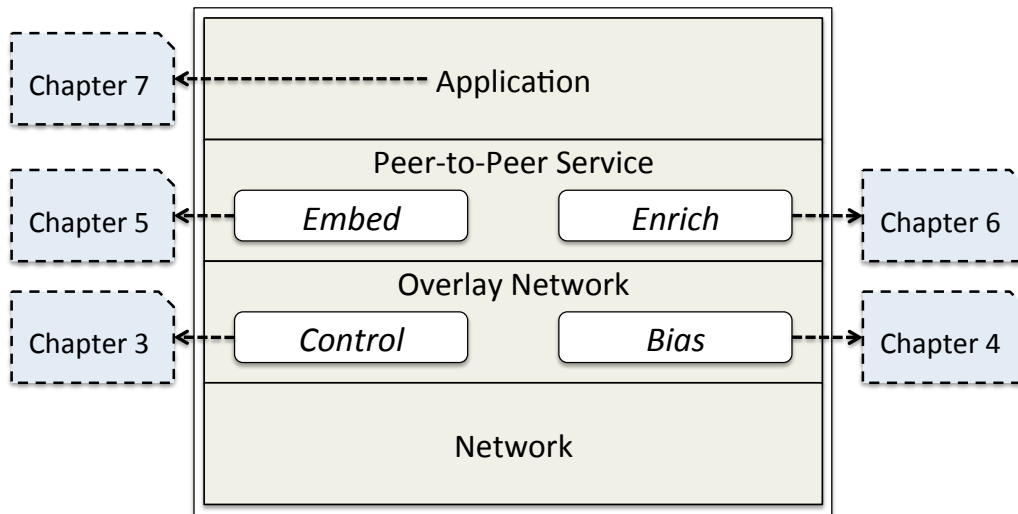


Figure 1.2: Approaches for managing and leveraging the topology of unstructured overlay networks and mapping to the thesis structure.

- At the Overlay Layer:

Control This technique is based on introducing soft-constraints to the neighboring relations that nodes can establish among them during the construction of the overlay topology. The goal is to maintain a high enough level of randomness as to ensure that the topology is flexible enough to deal with churn and has a low maintenance overhead, while ensuring topological properties that can be exploited by P2P services executed on top of the resulting overlay.

Bias This technique is based on constructing an overlay network with a random topology (using any of the available solutions found in the literature) and then (iteratively) replacing some of the original overlay links in order to improve a performance criteria, for instance the link latency. This allows to achieve an overlay network which is random in nature, but that is optimized to benefit the performance of P2P services executed on top of it.

- At the P2P Service Layer:

Embed This technique is based on embedding a secondary topology on top of a pure unstructured (random) overlay network. Links that are selected to form the secondary

topology are then leveraged by the P2P service to improve its operation, while the remaining unstructured overlay links can be used to transmit control information, or to support the operation of the service when the secondary topology becomes compromised due to global membership dynamics.

Enrich This technique is based on maintaining, at the P2P service layer, additional links between peers executing the distributed protocol. These links can be leveraged to improve the operation of the service. These links are not considered as being part of the underlying overlay network, and can be leveraged by peers, for instance, to exchange information with specific participants that might not be their direct overlay neighbors accordingly to the logic of the underlying membership protocol.

1.2 Contributions Summary

This thesis explores the four alternatives identified above, which are materialized in the following four contributions:

1.2.1 CellFarm

CellFarm is a novel protocol which operates at the *overlay network layer* that explores a *control approach* to manage the topology of an unstructured overlay network. The solution relies on low-cost overlay construction and maintenance mechanisms which are typically employed for unstructured overlay networks, such as gossip-based mechanisms and random walks, to build an overlay where participants organize themselves in controlled size, fully-connected clusters of nodes, where we name each cluster a *Cell*. Cells in turn are kept highly connected among themselves in a random fashion. Such an overlay is proposed as a building block for providing data replication and load distribution in P2P architectures, whereas each Cell operates as a virtual node in the system. We evaluate the benefits of CellFarm using a P2P unstructured resource location service.

1.2.2 X-BOT

X-BOT is a novel protocol which operates at the *overlay network layer* that explores a *bias approach* to manage the topology of an unstructured overlay network. X-BOT is able to bias the

topology of an unstructured overlay network given a generic efficiency criteria. The protocol operates iteratively in a decentralized fashion only requiring partial information about the system filiation. Additionally, contrary to previous state of art solutions, X-BOT is able to make the topology of the overlay evolve to more efficient configurations, while protecting the connectivity and node in-degree during the convergence process. We evaluate the benefits that can be extracted from the operation of X-BOT considering a P2P gossip-based broadcast service.

1.2.3 Thicket

Thicket is a novel protocol that operates at the *P2P service layer* to embed highly robust and efficient interior-node-disjoint trees over a single unstructured overlay network. This solution relies on the combination of eager-push and lazy-push gossip approaches to embed T spanning trees that cover all nodes over a single unstructured overlay network. The protocol ensures that a vast majority of peers in the system act as an interior node in a single spanning tree, which contributes for the load-distribution and also to the fault-tolerance of the system. Additionally, each peer maximum load (*i.e.*, the maximum number of downstream tree branches that depart from a node) is limited by a global configuration parameter. We have evaluated the advantages of Thicket in the context of a gossip-based streaming service.

1.2.4 Open Fire

OpenFire is a novel mechanism that operates at the *P2P service layer* by *enriching* the topology of an unstructured overlay network with additional links. The solution is able to balance rumor mongering exchanges in unbalanced overlay networks where nodes have variable in-degree due to the existence of Network Address Translation (NAT) boxes and Firewalls. The overlay topology is enriched through a low-cost single-sized cache, which is used to redirect rumor mongering requests among peers. This allows, in a very simple fashion, to ensure that in an unbalanced overlay each peer participates in a similar number of rumor mongering exchanges without any coordination among peers. We evaluate the benefits of OpenFire in the context of a gossip-based anti-entropy service.

1.3 Results Summary

Considering the contributions listed above, the main results present in the thesis are the following:

- Implementation of the CellFarm protocol and its evaluation through a combined use of simulation and a prototype deployment over the PlanetLab testbed.
- Design, implementation, and evaluation through simulation of a set of query dissemination mechanisms to support a P2P resource location service that leverages on the unique properties of the CellFarm overlay network.
- Implementation of the X-BOT protocol and its evaluation through the use of simulation operating in scenarios with different properties and taking into consideration distinct performance metrics.
- Implementation of Thicket and its evaluation through the combination of simulation and a prototype deployment over the PlanetLab testbed.
- Implementation of OpenFire and its evaluation using a simulation of this approach for unbalanced overlay networks in environments with NATs and Firewalls.

1.4 Ramifications and Collaborations

The main results of the thesis listed above have motivated additional research that was pursued through collaborations. We now list these additional contributions which relation with the main contributions of the thesis is briefly discussed in chapter 7.

Large-Scale Peer-to-Peer Autonomic Monitoring An autonomic monitoring infrastructure which relies on a P2P architecture based on unstructured overlay networks both to establish monitoring relations among components, and to disseminate data to specialized monitor consoles. This infrastructure is both robust to component failures and efficient.

RASM RASM is an application level broadcast scheme which relies on unstructured overlay networks to embed a spanning tree used to disseminate message in an expedite and efficient manner. RASM takes into consideration the expected reliability of both nodes

and overlay links when embedding the spanning tree to maximize the reliability of the dissemination mechanism. A forward error correction mechanism is used to disseminate redundant messages to mask potential omission due to the use of unreliable overlay links.

Curiata *Curiata* is a scalable and efficient resource location system that employs self-organizing techniques to integrate and combine the benefits of structured and unstructured approaches. The system supports flexible queries, like most unstructured solutions, while retaining the speed and efficiency provided by structured (DHT-based) solutions.

Rollerchain *Rollerchain* is a novel Distributed Hash Table that offers high availability of stored data in an efficient manner through the combination of unstructured and structured overlay networks. Rollerchain promotes load distribution and efficient replication of data in a DHT, independently of the distribution of stored data and node identifiers over the identifier space.

1.5 Research History

The work presented in the Thesis appears as a follow up to the work I have developed during my masters studies (Leitão, 2007) under the supervision of Prof. Luís Rodrigues and in cooperation with Prof. José Orlando Pereira, in the context of the FCT project P-SON (POSC / EIA / 60941 / 2004). The focus of this project was on devising protocols to generate probabilistically-structured overlay networks for supporting efficient epidemic multicast protocols. The research conducted in this period resulted in two contributions. The first was the Hybrid Partial View protocol (or simply HyParView) that creates and maintains an unstructured (or random) overlay network which is highly resilient to node failure, being able to recover from up to 80% of simultaneous node failures. This protocol employed the control approach discussed earlier to enforce symmetric overlay links and was originally introduced in Leitão et al. (2007b). The second contribution was the Plumtree protocol, which is a protocol that can efficiently embed a fault-tolerant spanning tree over an unstructured overlay network with properties similar to the ones guaranteed by HyParView. This protocol relied on the embed approach discussed earlier and is described in Leitão et al. (2007a).

Subsequently, I have been working in the main contributions of the PhD Thesis. I had the opportunity to work with other researchers at the national and international level and conse-

quently, some of the contributions and results described have been pursued in the context of those collaborations.

The CellFarm protocol was originally designed as an alternative to super-peer networks under the name Overnesia. The fundamental idea was to have regular peers to form clusters of nodes that could collaborate among them to act as super-peers. During this process we noticed that the solution offered a generic support for both load distribution and replication.

X-BOT was designed as a follow up of the HyParView protocol. The idea was to allow the overlay to include some awareness concerning the underlying network addressing the topology mismatch problem. This research was conducted in collaboration with José Pereira which was a co-author of HyParView, and also with an intern at the Distributed System Group of INESC-ID named João Marques.

The Thicket protocol resulted from a collaboration with Mário Ferreira, which was a master student from the Instituto Superior Técnico that conducted his master studies in the Distributed System Group under the supervision of Luís Rodrigues.

The scheme for balancing rumor mongering exchanges in unbalanced overlay networks resulted from a collaboration with Robbert van Renesse from Cornell University with which I had the opportunity to work, and learn, as the result of a short visit to Cornell University in the United States of America.

Additionally, the architecture for supporting autonomic P2P monitoring of large scale system resulted from a collaborative effort with fellow PhD student Liliana Rosa. The RASM protocol resulted from a collaboration with Mouna Allani, which was at the time a PhD student at the University of Lausanne (Switzerland) under the supervision of Benoît Garbinato. The Curiata system emerged from the contributions presented in the thesis, and a collaboration with João Alveirinho, a master students at the Distributed System Group of INESC-ID. The Rollerchain system emerged from collaborative work with João Paiva which is currently pursuing his PhD in the same research group under the supervision of Luís Rodrigues.

1.6 Thesis Structure

The remaining of the thesis has the following structure:

Chapter 2: introduces fundamental concepts which are relevant for the context of the contributions presented in the thesis;

Chapter 3: presents and evaluates CellFarm a novel overlay which explores the control approach at the overlay network layer;

Chapter 4: presents and evaluates *X-BOT* a novel protocol that explores the bias approach at the overlay network layer;

Chapter 5: presents and evaluates Thicket a novel protocol that explores the embedding approach at the P2P service layer label;

Chapter 6: presents and evaluates OpenFire, a novel solution for balancing gossip exchanges in unbalanced overlay networks that explores the enriching approach at the P2P service layer;

Chapter 7: concludes the thesis summarizing the results presented and derived from the thesis and discussing several pointers for future work.

Fundamental Concepts and State of Art



The thesis addresses new mechanisms that manage the topology of unstructured overlay networks to improve the performance of P2P services. In this chapter we present an overview on the fundamental concepts related to this topic and discuss the state of art on overlay networks.

This chapter is organized as follows: Section 2.1 discuss relevant examples of P2P services that typically operate over unstructured overlay networks. We list a number of relevant performance metrics, both direct (*i.e.*, associated with intrinsic properties of the overlay network topology) and indirect (*i.e.*, associated with performance metrics of services that execute on top of the overlay) in Section 2.2. These metrics will be used to validate and evaluate the benefits that can be extracted from the contributions proposed in the thesis. Section 2.3 identifies the differences between structured and unstructured overlay networks. Finally, Section 2.4 concludes this chapter with a discussion on existing techniques to manage the topology of unstructured overlay networks, both at the overlay layer and the P2P service layer. A summary on existing approaches found in the literature is also presented.

2.1 *Peer-to-Peer Services*

As discussed previously, the P2P paradigm has been used to implement several highly scalable and decentralized distributed services, such as reliable multicast (Birman et al., 1999; Kermarrec et al., 2003; Eugster et al., 2003; Hayden & Birman, 1996; Leitão et al., 2007b; H. Li et al., 2006; Pereira et al., 2003, 2004), data aggregation (Jelasity & Montresor, 2004; Kempe et al., 2003), publish-subscribe (Eugster & Guerraoui, 2002; Wong & Guha, 2008), failure detectors (Renesse et al., 1998; Johansen et al., 2006), slicing (Jelasity & Kermarrec, 2006), resource-location (Chawathe et al., 2003; Garbacki et al., 2007), distributed file systems (Druschel & Rowstron, 2001), and data management (DeCandia et al., 2007; Lakshman & Malik, 2010), among others.

The thesis focus on three relevant P2P services that are usually implemented on top of unstructured overlay networks. These particular services have been selected due to their relevance in the context of unstructured overlay networks, and also due to their diverse nature, which allows to illustrate the benefits of carefully designed unstructured overlay topologies in the context of different applications. We consider the following services: resource-location, gossip-based dissemination (both in the context of reliable broadcast and streaming), and rumor mongering. These services will serve as case-studies in the following chapters as a way to demonstrate the benefits that can be achieved by leveraging the contributions of the thesis, by evaluating the effect of the proposed mechanisms on particular performance metrics of these services (we will later discuss such metrics in section 2.2).

2.1.1 Resource Location

One of the most relevant P2P services is resource location. In a nutshell, resource location allows a participant in a P2P distributed system to obtain the identifiers of a set of peers that own a given resource. Note that a resource in this context can be a file, an entry in a distributed data base, free CPU time, etc.

A node that wishes to locate a given resource will first describe that resource using a query language. The resource location service is then responsible to route the query among peers in the system in order to gather information about the location of resources that match the issued query. Finally, the service should provide an answer to the issuer of the query.

Resource location systems became widely popular with the rise and fall of the Napster music sharing system¹ that appeared in 1999. Napster was closed due to its use of a centralized server for indexing all files shared by users. After this, several file sharing P2P systems have appeared and disappeared (mostly due to legal issues) which included some form of P2P resource location service. Some of the more popular system include Kazaa, eMule, the Gnutella network, among others.

In the following we discuss some of the architectures that have been used to support P2P resource location services. We then discuss the type of queries that can be supported by different architectures and the query dissemination strategies that are typically employed when

¹<http://music.napster.com/>

implementing this type of service.

2.1.1.1 Architectures

P2P resource location services can follow one of three main architectures as follows:

Centralized In this type of architecture a central server (or a group of servers) is responsible for maintaining a global index of all available resources in the system. For instance, in a file sharing application, the central server should hold a global index where it stores the set of peers that own each file currently available. When trying to locate a resource, a peer will issue a query directly to the central server, which will consult its local index and reply to the client. Notice that in this type of architecture the resource location *per se* is not distributed, however the access to resources (*e.g.*, requesting a copy of a file) is performed in a purely P2P fashion.

The original Napster system followed this architecture, and currently the BitTorrent protocol (Cohen, 2008) also leverages on centralized servers that index contents and serve torrent files to clients through Hypertext Transfer Protocol (HTTP).

Distributed over a DHT As an alternative to a centralized index server, some P2P resource location services rely on a DHT to maintain a distributed index of all resources available in the system. In this type of architecture, peers rely in the mapping functionality of DHT to register their resources, using an unique identifier (*e.g.*, a file name) to associate their own identifiers to each resource they own. When a peer wishes to locate a given resource, it uses that resource unique identifier to locate the node in the DHT which maintain the distributed registry of peers that currently own that resource. Pastry (Rowstron & Druschel, 2001) was originally proposed as an infrastructure to support efficient and distributed resource location over the Internet.

Distributed over an Unstructured Overlay Another alternative to the use of centralized index servers is to leverage an unstructured overlay network. In this type of architecture, each peer will maintain a local index of its own resources and in some cases, indexes for the resources of some of its neighbors. When a participant wishes to locate a given resource it disseminates a query among its peers. In order to ensure that the query is able to return

all possible results, the query must be disseminated to all participants. The first version of Gnutella was based on this approach.

Using a centralized server to maintain an index of all available resources in the system greatly simplifies the design of the resource location service. As participants in the system only have to contact the server in order to get the list of peers that own resources that match a given query. However, the existence of a centralized server also means that the system has a single point of failure. Additionally, the system has a high probability of presenting scalability limitations, as the central server may become a bottleneck in terms of network bandwidth, but also of processing capacity, as all queries are processed by the server. Note that, individual participants will have to register, and possibly unregister, their resources in order to make them available to other participants. This will require additional bandwidth and computational resources consumption.

Distributed solutions based on DHTs are both efficient and scalable. However, they require resources in the system to be identified by a unique identifier. This makes these approaches adequate for processing exact match queries where the node that performs the query knows exactly what it is looking for, and knows the unique identifier for that resource (we discuss this type of queries further ahead in more detail). Dealing with keyword based queries in this type of architecture requires additional mechanisms and consequently, will increase the complexity and operational costs of the system. Previous works, such as the work of Reynolds e Vahdat (2003), try to overcome this limitation by using inverted indexes, but introduce additional signaling and complexity due to the costs associated with the management of inverted indexes, and more elaborate routing of queries.

Solutions based on unstructured overlay networks usually require queries to be disseminated throughout the entire network in order to ensure that all matching resources are returned. This happens because there is no obvious way to limit the dissemination of queries to a subset of nodes and still achieve a good recall.² This results in limited scalability, specially in scenarios where the rate at which queries are issued by nodes is high. On the other hand, this approach provides a greater flexibility in the format of queries, as the resources that are targeted by a query can be described using any language.

²This is a typical performance metric defined as the fraction of resources that match a given query which are returned by the system.

Solutions based on unstructured overlay networks remain attractive, not only due to their increased flexibility and lack of a single point of failure but also because, as discussed previously, unstructured overlay networks are potentially more robust to churn scenarios. One way to try and overcome the scalability limitations of these approaches, is to rely in a hierarchical unstructured topology commonly named a super-peer network (Yang & Garcia-Molina, 2003). This approach works as follows:

Peers in the system which have more resources, or are deemed more stable (*e.g.*, higher probability for having a longer session time) according to some heuristic, organize themselves into an unstructured overlay network. These nodes are called *super-peers*. Other peers in the system (named *regular peers*) connect to one of these super-peers. Super-peers are responsible for maintaining a consolidated index of the resources owned by them and by regular-peers that are connected to them. When performing a query, a participant will route a query to its super-peer (if it is not a super-peer itself). This query is then flooded in the unstructured overlay network, such that all super-peers process and answer to the query if they (or some of the regular peers connected to them) own some resource that match it.

This approach can improve the scalability of the system, by reducing the number of nodes that have to receive and process each query. However, it also presents some limitations. Namely, it is not trivial to assert which peers are good candidates to become super-peers. Moreover, when a super-peer fails, or leaves the system, the regular peers that were connected to it become disconnected from the system, and have to select a new super-peer. These super-peers need to update their consolidated indexes, which leads to additional consumption of both bandwidth and computational power. Finally, this type of solution does not take advantage of the computational and network resources of regular peers, which do not contribute actively for the operation of the system. Super-peer-based approaches have been employed in the second version of Gnutella, Kazza, and in SOSPN system (Garbacki et al., 2007), among others.

Note that DHT-based architectures can also rely on a two-tier architecture to mitigate some of the negative effects of churn due to peers that remain in the system for a very brief period of time. In this approach, only nodes that are considered to be more stable (*i.e.*, super-peers) join the DHT, while the remaining nodes act as clients of super-peers. This type of hierarchical architecture is employed by the Skype system to maintain the directory of connected users.

However, Skype has become unavailable for somewhat long periods of time due to the failure of the super-peer DHT as the result of concurrent updates of clients running over Windows operative system, and also due to the concurrent failure of a large portion of super-peers due to a bug in a version of the Windows client. Both these incidents demonstrated that DHTs are susceptible to churn conditions (Arak, 2007; Rabbe, 2010).

2.1.1.2 Type of queries

Queries can employ different languages to describe the type of resources they target. There is no universally accepted framework to classify queries accordingly to their format, and the information they can carry to identify resources. In the context of the thesis we consider the following three types of queries that can be supported by resource location services:

Exact Match Queries This type of queries rely on resource unique identifiers. These queries allow to locate resources if their unique identifiers are known *a priori*. For instance, in a file sharing system, an exact match query can carry the name of the desired file.

Keyword Queries These queries usually carry a set of keywords that are associated with a set of resources. Typically these keywords can be combined using the logical operators “and” and “or”. These queries allow to return a list of resources, and peers that own those resources, which were classified with the keywords present in the query (respecting the logical relations specified in the query). For instance, in a file sharing system, a keyword query may carry a set of tags that users associate to the file such as *video*, *tv show*, *music*, etc.

Arbitrary Queries These queries contain a set of arbitrary properties associated with the resources that are available in the system. These properties may also be combined using several logical operators, such as “and”, “or”, negations, etc. For instance, in the case of a file sharing system, an arbitrary query may specify properties concerning keywords, contents, size and extension of the file, among others.

As discussed previously, DHT-based resource location services are very efficient in dealing with exact match queries. Keyword queries can also be supported by DHT-based resource location by using, for instance, the work of Reynolds e Vahdat (Reynolds & Vahdat, 2003). In sharp

contrast, arbitrary queries do not benefit from the DHT deterministic topology. Therefore, they are mainly supported by resource location services based on unstructured overlay networks.

2.1.1.3 Query Dissemination Strategies

The dissemination of queries in P2P resource location services can be performed using several strategies. In the particular case of centralized solutions, there is no need for a specialized mechanism to disseminate queries, as participants will simply send their queries to the central component and get an answer from it.

Architectures based on DHTs, usually rely on the application-level routing infrastructure provided by the underlying DHT to route queries to nodes that own information about resources that are relevant for the query. In the case of exact match queries this usually involves contacting a single peer, whose id is closest to the unique identifier of the targeted resource.³ For the case of keyword queries, one might require routing the query (or copies of the query) to several peers in the system, usually one for each of the keywords present in the query.

On the other hand when a P2P resource location service is executed on top of an unstructured overlay network, which is the case of particular interest for the work presented in the thesis, one can rely on several routing strategies to disseminate the query among participants.

There are two main classes of query dissemination strategies that are typically employed on resource location services based on unstructured overlay networks: Flooding, and Random Walks (Lv et al., 2002). In the literature there are several variants of these approaches proposed. In the context of the thesis we consider two variations of each strategy as follows:

2.1.1.3.1 Flooding

Complete Flooding This approach is based on disseminating each query to all participants in the system. Typically this is performed by relying on a push gossip-based dissemination scheme. Several systems have employed this approach, for instance the Coral content distribution network⁴.

³In many systems this usually means the peer which identifier is the closest to the output of a hash function over a human intelligible unique identifier.

⁴<http://www.coralcdn.org>

Flooding with Limited Horizon This is a variant of flooding where only a fraction of the overlay is flooded by the query. To this end, messages are disseminated with a conservative time to live (TTL) value which is decremented with each retransmission of the query. This allows to lower the overhead imposed by the query dissemination strategy, while potentially reducing the number of answers that are returned for each query. The use of flooding with a limited horizon is discussed by Tsoumakos e Roussopoulos (Tsoumakos & Roussopoulos, 2006).

2.1.1.3.2 Random Walks

Blind Random Walks In this approach a participant that wishes to disseminate a query initiates k random walks in the network. A random walk is performed by having a message being routed at random among overlay neighbors for a pre-determined number of hops (usually controlled by using a TTL parameter associated with the message). Each random walk will follow a random path on the overlay, and each peer visited by the random walk processes the associated query and replies to the issuer if some of its local resources match the query.

Guided Random Walks Contrary to blind random walks, guided random walks rely on information exchanged between overlay neighbors to bias the path of a random walk in the overlay. A common strategy to do this, is to have peers exchange among themselves information about their local indexes, for instance by using bloom filters (Bloom, 1970). A node can store, for each of its overlay neighbors, information about the contents directly available at that neighbor, and recursively, at that neighbor neighbors with a decreasing level of detail. Random walks are then routed at each step, by selecting the current node neighbor which, accordingly to the bloom filter, has a higher probability of owning resources that match the query. Variants of this technique have been proposed by Crespo e Garcia-Molina (2002) and Broder e Mitzenmacher (2004). Quasar (Wong & Guha, 2008) is another example of a system that relies on bloom filters to guide random walks over an overlay. Notice that this technique cannot be easily applied to arbitrary queries, as this would require highly complex data structures to be exchanged among overlay neighbors. Such process would present a non-negligible communication overhead.

2.1.2 Gossip-based Dissemination Protocols

Gossip-based dissemination aims at supporting the dissemination of messages produced by one, or many, participants among (all) other participants in the system through a collaborative process, where each node forwards messages received for the first time to a subset of the remaining participants (usually, their overlay neighbors).

Gossip-based dissemination protocols are a particular instance of the more generic class of gossip protocols as defined by Demers et al. (1987). The basic idea behind gossip-based dissemination protocols is to have all participants in the protocol collaborating equally to disseminate information mimicking the process through which rumors and epidemics spread over a population. To this end, when a peer wishes to disseminate a message, it selects t nodes at random - we name these nodes *gossip targets* - and forwards the message to them (t is a typical configuration parameter called *fanout*, which is discussed further ahead in the text). Upon receiving a message for the first time, each peer repeats this process: by selecting t gossip targets at random and forwarding the message to them.

If a node receives the same message twice - which is possible, as each node selects its gossip targets in an independent fashion (without being aware of gossip targets selected by other nodes) - it simply discards the message. To allow this, each node has to keep track of which messages it has already seen and delivered. The history of message identifiers may grow indefinitely during the execution of the protocol, unless some purging scheme is applied to garbage-collect obsolete entries. For strategies to purge message histories the reader should refer to the work by Koldehofe (2003).

The simple operation model of gossip protocols not only provides high scalability but also a high level of fault tolerance, as its intrinsic redundancy is able to mask network omissions as well as node failures.

2.1.2.1 Relevant Parameters

Gossip-based dissemination protocols can be parameterized in order to better control their operation and some relevant trade-offs. The two most relevant parameters associated with the configuration of gossip protocols can be described as follows:

Fanout: This is the number of nodes that are selected as gossip targets by a node at each gossip step in order to retransmit the message. There is a trade-off associated with this parameter between desired reliability and the amount of redundancy associated with the operation of the protocol. High fanout values ensure higher levels of fault tolerance (increasing the probability of atomic delivery, as defined by Kermarrec et al. (2003)) but also generate more redundant network traffic.

Maximum rounds: This is the maximum number of times a given gossip message is retransmitted by peers. Each message is transmitted with a *round value* - initially with a value of zero - which is increased each time a node retransmits the message. Nodes will only retransmit a message if its *round value* is smaller than a *maximum rounds* parameter.

Considering this parameter, gossip-based dissemination services can operate in two modes:

- *Unlimited mode:* In this mode of operation the parameter *maximum rounds* is undefined and there is no specific limit to the number of retransmissions executed for each gossip message.
- *Limited mode:* In this mode of operation the parameter *maximum rounds* is set to some integer value (higher than 0), effectively limiting the maximum hops executed by each message over the overlay.

By limiting the maximum number of gossip rounds one can also limit the maximum number of receivers. For instance, if the message is forwarded by flooding, in order to reach the entire system the maximum number of rounds must be set equal or higher to the network diameter.

There is an inherent trade-off between reliability and amount of redundancy associated with the use of maximum number of rounds parameter. In unlimited mode (or configuring the *maximum rounds* parameter with high values) there is a higher probability of achieving atomic delivery (*i.e.*, that all participants receive each disseminated message) but on the other hand, more redundant messages are produced (*i.e.*, messages that when received by a peer will not generate a delivery to the application layer). Furthermore, if memory is limited, there is the risk of the gossip propagation never finishing due to re-infection (Koldehofe, 2003).

2.1.2.2 Communication Modes

When implementing a P2P dissemination service based on a gossip protocol, peers can use several approaches to forward messages to their neighbors. In the context of the thesis the following three fundamental approaches are considered:

Eager-push approach: Peers send the full payload to their gossip targets as soon as they receive a message for the first time. This is an approach initiated by the sender.

Pull approach: Periodically, nodes query random selected peers for information concerning recently received, or available, messages. When they become aware of a message that they did not receive yet, they explicitly request the payload of that message to the neighbor that has it. This is a strategy that works best as a complement to a best-effort broadcast mechanism (*e.g.*, by first employing IP Multicast (Deering & Cheriton, 1990)). This approach is also employed in the work of Birman et al. (1999).

Lazy-push approach: When a node receives a message for the first time, it forwards only the message identifier (*e.g.*, a hash of the message contents, or some application level identifier carried by the message) instead of the full message payload. If peers receive an identifier of a message for which the payload has not been received yet, they explicitly request the payload from the sender in a similar fashion to the operation of the pull approach.

There is a trade-off associated with the use of eager-push, pull, or lazy-push strategies. Eager-push produces additional redundant traffic but it also achieves lower latency than the remaining strategies (as the remaining strategies require at least an extra round trip time to produce a message delivery to the application layer). From a latency perspective, the lazy-push approach conveys very similar results to pull approach.

Another important practical aspect to retain is that, contrary to pull/lazy push approaches, eager-push does not require the maintenance of local copies of delivered messages for (potential) later retransmission upon request by neighbors. Hence, pull/lazy push gossip approaches are more demanding in terms of memory requirements.

These fundamental approaches can also be combined to develop more complex gossip mechanisms, that usually try to benefit from the strong aspects of each individual approach.

As the reader can guess, the number of possible combinations is high, as one can use different algorithms or mechanisms to select when to employ or switch from one strategy to the other. In the following, we depict two of these hybrid strategies:

Eager-push and Pull approach: Gossip is executed in two distinct phases. A first phase uses eager-push gossip to disseminate messages in a best-effort manner to a vast majority of participants (in this phase, the configuration of the push gossip can be somewhat conservative). A second phase of pull gossip is used to recover from omissions that may occur in the first phase. The idea is to lower the amount of redundancy of the gossip process, without decreasing its efficiency. However, the use of pull gossip for recovery leads to an increase in the overall delivery latency (Carvalho et al., 2007).

Eager-push and Lazy-push: Gossip is executed by applying eager-push to a subset of the gossip targets of each node. The selection of the subset of peers can be made using several strategies (examples can be found in previous work by Leitão et al. (2007a) and Carvalho et al. (2007)). Lazy-push is used in the remaining gossip targets to recover from omissions and ensure that the reliability of the gossip process is not affected.

There are two relevant sub-classes of gossip-based dissemination services, which differ among them concerning the correctness criteria as well as the data type usually disseminated by them. We now discuss the differences between these two sub-classes.

Gossip-based broadcast service This particular sub-class of gossip-based dissemination services usually considers a system where any node may disseminate messages concurrently with any other node. Messages disseminated (mostly) do not have delivery order constraints, meaning that messages do not have to be delivered to the application in a pre-determined order. Although latency should be minimized, messages do not have strict time constraints to be delivered to all peers. An example of such a service is a dissemination service for RSS feeds to a large number of users.

Gossip-based streaming service This particular sub-class of gossip-based dissemination services are usually employed to disseminate multimedia data to a large number of consumers. In this case, peers consume the stream of data as they receive it from their peers. Due to this, messages usually have order constraints and also tight deadlines for delivery

to all peers (considering the moment at which the data is generated). Also, in most cases such as the streaming of a live event video, only a node will be disseminating data (this node is usually dubbed *source*). For some particular applications, more than a node may be able to broadcast content, but typically this does not happen in a concurrent fashion, for instance in a video conferencing application where only the video of the talking party is disseminated at each moment.

2.1.3 Rumor Mongering

Rumor mongering (as initially proposed in (Demers et al., 1987), also known as anti-entropy) is a particular instance of gossip protocols, and was one of the first proposals that considered the use of gossip as a building block for designing distributed systems. The main difference is that whereas in gossip-based dissemination protocols nodes collaborate among themselves to disseminate information that is produced by a source node, anti-entropy protocols are often focused on maintaining state information that is distributed and shared among a potential large group of peers (*e.g.*, the global filiation of the system), or in extracting aggregation values from a individual values maintained by each node (*e.g.*, the average communication load for each peer).

To this end, nodes periodically - every ΔT , often named the *gossip period* - engage in a exchange of information, where a peer (the *initiator*), will send a message to another peer (the *receiver*) which contains some information concerning its internal state. The receiver will then reply to the initiator with another message typically with a similar content *i.e.*, information about its own internal state. Then, both peers update their internal state by using the received information. Note that contrary to gossip-based dissemination protocols, in rumor mongering solutions peers engage in gossip exchanges at a fixed rate, independently of the production of events or state updates seen, or generated by, individual participants in the system.

2.1.3.1 State Reconciliation

Some rumor mongering services may use this initial message exchange only to determine if there are divergences in the internal state of nodes. If nodes detect that their states are divergent, they may be required to exchange additional messages between them in order to perform

state reconciliation. The mechanisms for performing state reconciliation may have high bandwidth requirements, depending on the complexity of the internal state maintained by nodes. Previous research efforts have focused on reducing these bandwidth requirements.

In the original Clearinghouse paper (Demers et al., 1987), the authors propose an iterative reconciliation technique, where nodes compare their internal states using hash functions, and exchange the most recent updates until their states become reconciled. Byers et al. have improved on this design by combining Bloom filters, Merckle trees, and Patricia trees (Byers et al., 2002). Minsky et al. have proposed a method based on characteristic polynomials (Minsky et al., 2003).

This class of services has received much attention from the community due to its simplicity, but also because solutions based on gossip exchanges are able to self-scale, as all participants will take, in average, the role of initiator and receiver once in each gossip period. Also, this simple procedure has been shown to result in a fast convergence of the node's state.

Rumor mongering services were originally designed by assuming that a peer is able to select a receiver uniformly at random among all participants in the system. A viable alternative for the particular case of large-scale dynamic environment, where a global view of the system is unavailable to nodes, is to have this service to operate on top of an unstructured overlay network. In particular the operation of rumor mongering may benefit from an unstructured overlay maintained by a cyclic algorithm⁵, at this allows peers to sample, and exchange information with, a larger number of distinct peers overtime. Intuitively, this approximates the behavior of rumor mongering to the one achieved when operating with access to full membership information.

2.1.3.2 Behavior Under Unbalanced Networks

As stated previously, rumor mongering were originally conceived to operate in scenarios where each peer has local access to the full membership of the system. In this particular case the behavior of the service is well balanced, given that in each gossip round, a participant will, in average, participate in a gossip exchange as initiator, and another one as receiver.

⁵The classes of algorithms for managing unstructured overlay networks are discussed further ahead in Section 2.3.2.

However, in a large-scale system, where one must rely on unstructured overlay networks to cope with membership dynamics, each peer only owns a partial view of the system. Moreover, unstructured overlay networks may not ensure that all participants are known by a similar number of nodes (*i.e.*, the overlay may exhibit an unbalanced in-degree distribution).

Additionally, in large scale systems over the Internet, it is not uncommon for communication to be asymmetric, notably due to the existence of firewalls and Network Address Translation (NAT) boxes, which limit the communication patterns that peers can establish. Depending on the operation of the protocol that maintains the unstructured overlay network, this might augment the unbalance over the in-degree of peers. However, independently of the effect over the overlay topology, this translates into a scenario where some peers cannot act as receivers for other peers. This leads to an unbalanced behavior for rumor mongering services, where nodes that are publicly available in the Internet, or which are more “popular” in the underlying unstructured overlay network (*i.e.*, which have an in-degree above the average), participate in a larger number of rumor mongering exchanges that involve complex state reconciliation operations.

Such unbalancing is undesirable, mostly because existing techniques to lower bandwidth consumption during state reconciliation, present high overheads in terms of the computations that are required to serialize and deserialize objects that are exchanged among peers during these operations. Additionally, cryptographic operations that may be required for performing the exchange (*e.g.* signing or encryption of message contents) significantly increase the consumption of computational resources (*i.e.*, CPU time).

For example, in a commercial Java-based deployment of Astrolabe (Renesse et al., 2003), that employs a rumor mongering aggregation service that uses Bloom filters and Merckle trees for reconciliation, nodes spend approximately 3% of their CPU time on the operations listed above. In a Planetlab deployment of Fireflies (Johansen et al., 2006), a secure gossip-based overlay network that uses the reconciliation technique of Minsky et al. (2003), as well as public key cryptography, nodes use approximately 10% of their CPU time to perform state reconciliation.

2.2 Performance Metrics

In order to evaluate the design of unstructured overlay networks, one has to take into consideration a set of performance metrics. More precisely, one has to consider overlay metrics, which measure direct properties of overlay topologies. Additionally, one should also consider some relevant application-level metrics, which are performance indicators of services executed on top of these overlays. Such indirect performance metrics allow to quantify the impact of the overlay topology over the operation of a particular P2P service that leverage their design.

2.2.1 Direct Metrics

Connectivity The overlay network should be connected, *i.e.*, there should be at least one path from each node to all other nodes. If this property is not met, isolated nodes will not be able to cooperate in the execution of a P2P protocol. For instance, messages that are disseminated on top of an overlay network that is not connected will never be able to be delivered to some peers.

Although connectivity is a binary property (meaning that an overlay is either *connected* or *partitioned*), a more useful metric to quantify the connectivity of an overlay is the size of the *largest connected component* (usually measured in function of the percentage of nodes in the overlay that belong to the largest connected component). A connected overlay will present a largest connected component of 100%.

Degree Distribution In an undirected graph, the degree of a node is simply the number of arcs that are connected to that node. Given that neighbor sets owned by peers define a directed graph, it is relevant to make a distinction between the *in-degree* and the *out-degree* of a node. The in-degree of a node n is the number of nodes that have n 's identifier on (at least one) their neighbor sets; it provides a measure of the reachability of a node in the overlay. For promoting a good load distribution among peers in a system, the in-degree should be similar across all participants (*i.e.*, the overlay should exhibit a balanced in-degree distribution). The out-degree of a node n is the number of nodes in the neighbor sets maintained by n ; it is a measure of the node contribution to maintain the overlay network (The notion of neighbor sets is discussed further ahead in the text).

Moreover, if the probability of failure is uniformly distributed in the node space, for im-

proved fault-tolerance, both the in-degree and out-degree should be evenly distributed across all nodes executing the membership protocol. To avoid scenarios where the departure (or failure) of only a few (more connected) peers leads to the full disruption of the overlay connectivity, the out-degree should be similar across all participants (*i.e.*, the overlay should exhibit a balanced out-degree distribution).

Average Shortest Path A path between two nodes in an overlay network is a set of edges that a message has to cross from one node to the other. The average path length of an overlay is the average of all shortest paths between all pair of nodes in the overlay. This property is closely related to the overlay diameter. To promote efficient communication patterns over the overlay network, the average shortest path should present low values, as this metric is intimately related with the time (and number of hops required) for information to be disseminated across all peers.

Clustering Coefficient The clustering coefficient of a node is the number of links that exist in the overlay connecting that node's neighbors divided by the maximum number of links between those neighbors. This metric indicates a density of neighboring relations across the neighbors of a given node (having a value between 0 and 1). The clustering coefficient of an overlay is the average of clustering coefficients of all nodes. This metric has a high impact on the number of redundant messages received by nodes when exchanging information across overlay links. A high value of clustering coefficient will result in additional localized traffic, which may lead to additional latency for (most) P2P services. Clustering coefficient can also affect the fault-tolerant properties of the overlay network, given that areas of the overlay that exhibit higher values of clustering coefficient can more easily become isolated from the remaining peers in the system.

Overlay Cost We assume that a *cost* may be associated with each link of the overlay. The overlay cost is the sum of cost for all links that form the overlay. Costs may be associated to a concrete (underlay) network metric such as link latency. However, the link cost may also capture higher level utility functions; for instance, in a file sharing P2P system, it could be related to the semantic similarity of files shared by both link edges. Ideally, the overlay cost should be minimized to improve the overall performance of P2P systems.

2.2.2 Indirect Metrics

In this section we discuss some relevant performance indicators for the three P2P services discussed previously. These indicators will serve as indirect metrics for asserting the benefits that can be extracted from the overlay topologies that result from the application of the techniques proposed in the thesis.

2.2.2.1 Resource Location

P2P resource location services performance highly depends on the mechanisms employed to disseminate, or route, queries to relevant peers in the system, and ultimately on the amount of relevant resources that are located by the service in response to a particular query. Considering this, we focus on the following performance indicators:

Dissemination Cost The dissemination cost (of a query) is the number of messages that have to be exchanged among peers in order to route, or disseminate, a query over the overlay. Implementation of resource location systems that rely of flooding-based mechanisms typically present a high dissemination cost, whereas systems that operate through random walks usually present lower values. There is a trade-off between query dissemination cost and recall rate.

Processing Cost The processing cost (of a query) is the percentage of nodes in the system that are required to consult their local resource index for matching resources for a given query. This performance indicator can be artificially lowered by sacrificing the recall rate, by having some peers drop some queries avoiding to process them. To ensure scalability of the system to high rates of query injection, the processing cost should be kept as low as possible.

Recall Rate The recall rate of a resource location service is the percentage of resources returned by the service in response to a query, in relation to the total amount of resources in the system that match the query. Resource location services should aim at exhibiting a query recall rate as close to 100% as possible.

2.2.2.2 Gossip-based Dissemination

Gossip-based dissemination services performance are highly entwined with the time required to disseminate messages across all peers in the system, as well as the ability of the service to be reliable *i.e.*, being able to deliver messages to all participants respecting any delivery time constraints if they exist, despite membership dynamics that might occur. In the context of the thesis we focus on the following performance indicators:

Reliability Reliability is defined as the percentage of (correct) nodes in a system which delivers each disseminated message. A reliability value of 100% is indicative that the broadcast protocol was successful in delivering a given message to all active nodes or, in other words, that the broadcast process resulted in an atomic broadcast as defined by Kermarrec et al. (2003). Note that for the particular case of gossip-based streaming services, messages are only considered to be delivered with success if the time constraints associated with the operation of the service are respected (*i.e.*, if peers are able to deliver messages to the application layer in time for the content be presented to the user without an interruption). Typically, the goal of any gossip-based dissemination service is to obtain a reliability of 100% despite network omissions or node failures.

Last Delivery Hop The last delivery hop, or simply LDH, measures the number of times that the last message which is delivered by a gossip-based dissemination service was forwarded. The lower this value, the lower the latency of the protocol. If all links between nodes were to exhibit the same latency, the latency of a gossip broadcast transmission would simply be the last deliver hop multiplied by the *per hop* latency. Also, low LDH values contribute to improve the reliability of the service, as messages will be required to be forwarded fewer times which decreases the window in which failures can occur that might disrupt the dissemination process. This metric depends on the diameter of the overlay network used to disseminate messages.

Latency The latency of a P2P dissemination service, is the time that a message takes from the moment it is initially disseminated by the source to the reception by the last node in the system. For most applications, broadcast latency should be kept as low as possible. Notice that one can artificially lower the broadcast latency of a gossip-based dissemination system by avoiding to deliver the message to all participants, sacrificing the reliability.

Therefore, broadcast latency should be only compared between dissemination services that exhibit similar reliability values.

2.2.2.3 Rumor Mongering

The goal of rumor mongering services is to allow peers to maintain a consistent distributed state. This should be achieved with a minimal inconsistency window and ensuring that the load imposed over peers to ensure the consistency is evenly distributed to promote scalability. We consider the following performance indicators for this service:

Load Overhead In anti-entropy protocols, nodes are required to engage in periodic gossip exchanges which allow nodes to update their (local) state. The load overhead is defined as the maximum number of rumor mongering exchanges performed by a single node in the system in a given time interval ΔT , minus the average number of gossip exchanges performed by all peers in the system in that same time period. To avoid overloading individual nodes, the load overhead should be kept as low as possible.

Latency This performance criteria is related with the latency of gossip-based dissemination services. Anti-entropy latency is defined as the average time required for an event, or change in the internal status of a given peer, to be disseminated, and therefore visible, to all (relevant) peers in the system. This performance criteria is a dominating factor over the maximum inconsistency time window allowed by the rumor mongering service.

2.2.3 Overlay Topology and Application-Level Performance Metrics

One can expect that the properties of the topology of unstructured overlay networks that supports a given P2P service has a direct impact on the performance of that same service. For instance, excessive clustering coefficient increases the latency of gossip-based dissemination and rumor mongering services, and can also lead to an increase in the dissemination cost of resource location services particularly, for those that rely in flooding mechanisms to disseminate queries. An overlay in which most links exhibit high latency or reduced bandwidth will affect negatively most P2P services which rely on the overlay to select peers with whom communicate. An unbalanced in-degree distribution may make dissemination services less reliable and

increase the load overhead of rumor mongering protocols by focusing gossip exchanges on a sub-set of peers with higher in-degree values.

This creates a clear necessity for managing and improving the topology of unstructured overlay networks to ensure that correct operation and improve the performance of P2P services executed on top of them. In Section 2.4 we introduce some of the existing techniques to achieve such a goal.

2.3 *Overlay Networks*

An overlay network is defined as a network which is deployed on top of another network. The links that compose an overlay network are said to be logical, or virtual, as they are (in most cases) independent of the underlying network links and topology. This means that two direct neighbors in the overlay, may be separated by several hops in the underlay, and vice versa. The work presented in the thesis focus on a particular subset of overlay networks: those that operate at the application level of the TCP/IP protocol stack.

Overlays encode neighboring relationships among peers that are participating and collaborating in a given distributed protocol. These neighboring relations are usually captured through the use of local neighbor sets maintained by each peer p . Notice that some protocols might sub-divide the neighbor set into multiple subsets, as a way of creating a logical separation among neighbors, which are then used for different purposes during the execution of the distributed protocol.

Overlay networks simplify the design of P2P distributed protocols, by decoupling the management of system membership from the distributed protocol. Managing the membership of these systems is a complex task, in particular due to the fact that the system filiation may be subject to churn (which is defined as a fast paced sequence of concurrent join and leave operations executed by several nodes, as well as node failures (Stutzbach & Rejaie, 2006)).

Neighbor sets are typically maintained by a distributed membership protocol, which is responsible for dealing with filiation dynamics. The protocol should ensure that a node j that joins the system is able to fill its neighbor sets with the identifiers of (correct) peers with whom it can exchange messages and that, eventually, j 's identifier will be added to neighbor sets maintained by (some of the) other peers currently active in the system. Symmetrically, when a

node l leaves the system (or fails), the membership protocol should ensure that eventually, l 's identifier is removed from all neighbor sets.

When a new node, say n , wishes to join an existing overlay network, n will be required to contact another peer, say c , that is already part of the overlay network; c is therefore called *the contact node* of n . Typically, the membership protocol that is responsible for managing the overlay network will make a request to c , triggering the insertion of n in the neighbor sets of some nodes in the system and consequently, in the current overlay topology. Often, this procedure will return to n some peers identifiers which it uses to initialize its neighbor set. The semantics associated with the set of peer identifiers which are returned to n depends on the overlay logic and the membership protocol.

Overlay networks can be divided in two main classes by taking into consideration the mechanisms used to maintain local neighbor sets, which in turn defines the overlay topology. These classes are named respectively, *structured* and *unstructured* overlay networks. In the literature the definition of structured and unstructured overlay is often inconsistent. In the context of the thesis we consider the following definitions:

- **Structured overlay network:** An overlay network which relies on a global coordination scheme, based on unique identifiers of nodes. For instance, an overlay network that organizes nodes in a ring, ordered accordingly to their identifiers. Such schemes allow to deterministically infer the location of a node (*i.e.*, the neighbors of that node) in the overlay given the identifiers of nodes currently in the system. Structured overlay networks have a topology that is known a priori, and enforced by construction.
- **Unstructured overlay network:** An overlay network that has a random topology, such that it is impossible to predict where a node will be positioned, even knowing the full filiation of the system, the current overlay topology, and the identifier of the joining node. In these networks there is a larger degree of freedom when managing the overlay topology in the presence of changes in the system filiation, namely in churn scenarios.

In the following we briefly discuss some of the characteristics of structured and unstructured overlay networks and provide a set of examples from the literature to better illustrate the typical design of solutions for each overlay type.

2.3.1 Structured Overlay Networks

As discussed above, in the context of the thesis we refer to structured overlay networks as overlays where the topology is tightly controlled by a global coordination mechanism.

The most common example of a structured overlay network is a Distributed Hash Table (DHT), that enables the system to map any given key (in the identifier space used by nodes) to a peer that is active at that moment and which identifier is equal, or the closest, to the destination key.

DHTs have been used for supporting, in an efficient fashion, many large-scale distributed services, including resource location, publish-subscribe, monitoring, storage, etc. DHTs have been widely studied, leading to the proliferation of DHT designs, such as Chord (Stoica et al., 2001), Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao et al., 2004), Koorde (Kaashoek & Karger, 2003), Kademia (Maymounkov & Mazières, 2002), and CHR (Araújo et al., 2005), among others.

The popularity of this type of overlay network derives from its flexibility, provided by an application-level routing infrastructure, where a node can route a message to another node that is *responsible* for any particular key. Additionally, DHTs can offer this functionality while limiting the amount of membership information that each participant has to maintain. In fact, Rowstron e Druschel have previously shown that, while each node only has to be aware of approximately 1% of the peers in the system, routing a message in the overlay can be performed in a logarithmic number of steps with regard to the total size of the system filiation (Rowstron & Druschel, 2001). This allows DHTs to be scalable, and therefore being an adequate support for large-scale services. Additionally, because each node is required to keep only a small amount of information about the system filiation, join and leave operations only impact a small subset of peers in the system, making these overlay topologies somewhat adapted to some dynamic environments.

DHTs have also been extremely studied in what concerns their behavior in face of high dynamics in the filiation (Rhea et al., 2004) and in face of network partitions (Shafaat et al., 2007). Previous research has shown that the DHT maintenance presents a higher bandwidth consumption in face of filiation dynamics when compared with unstructured overlay networks (Blake & Rodrigues, 2003). Also, results reported by Blake e Rodrigues (2003) show that the av-

erage bandwidth available to typical users, and trends in the evolution of these values, limit the availability and scalability of P2P distributed storage systems based on DHTs, a fact exacerbated by churn conditions, which are frequent operational conditions for systems deployed over the Internet (Stutzbach & Rejaie, 2006). The authors extrapolate from their results that the current DHT designs may be unable to deal with high dynamics in low bandwidth networks.

Also, due to the fact that DHT topologies are strict, the join procedure of new elements assumes the correctness of the routing infrastructure offered by the DHT. Additionally, when a node detects that one of its peers has failed, it usually cannot replace the link to that peer by a link to another peer selected at random. Instead it will have to resort to the routing infrastructure to locate a suitable replacement to recover from that failure. This suggests that DHTs are susceptible to churn conditions, which can result in the disconnection of the DHT leading to the failure of the application level routing service, which may render it impossible to repair the overlay topology. As an illustration of this problem, Skype has been down more than once due to the failure of its support DHT (Arak, 2007; Rabbe, 2010).

2.3.2 Unstructured Overlay Networks

As discussed previously, unstructured overlay networks are characterized by neighboring associations which are established (mostly) at random among participants in a system. Due to this fact, the topologies of unstructured overlay networks are much more flexible, and impossible to predict, even if an external observer has global knowledge of the current system filiation.

Unstructured overlay networks have been widely used to support gossip-based protocols. As discussed previously, this class of protocols is based on random exchange of information among participants in the system. Although typically, the topology of unstructured overlays is random in nature, different P2P services may benefit from operating over topologies with specific characteristics, for instance some gossip protocols may benefit from the existence of peers with a high degree, which can act as hubs in the network, while other protocols may benefit from operating on top of overlay networks where the neighbors of each peer change over time. This has motivated the proposal of several designs of unstructured overlay networks, such as Scamp (Ganesh et al., 2003), Cyclon (Voulgaris et al., 2005), MON (Liang et al., 2005), and Bounce (Deshpande et al., 2006). Additionally, some gossip-based dissemination systems also include specially tailored membership protocols to maintain unstructured over-

lay networks with specific properties, such as the ones proposed by Carvalho et al. (2007) or Leitão et al. (2007b).

As in all overlay networks, unstructured overlay networks are defined by the closure of neighbor sets maintained locally by each peer. Because these neighbor sets only maintain a fraction of the full system filiation, it is fairly common to name these sets *partial views*. Also, distributed membership protocols that maintain these views are often said to provide a *peer sampling service* (Jelasity et al., 2004). To ensure scalability, the size of partial views should be much smaller than the size of the full system filiation (for instance, logarithmic with the number of peers).

Partial views can be maintained using different strategies, providing nodes with a more static or dynamic view of the system. In particular, we identify two main approaches to manage partial views of unstructured overlay networks:

Reactive strategy: In this type of approach, a partial view is only updated in response to some external event that affects the overlay (*i.e.* a node joining or leaving the system). In stable conditions, partial views remains unaltered. Scamp (Ganesh et al., 2003) is an example of a reactive protocol.⁶

Cyclic strategy: In this type of approach, a partial view is updated every ΔT time units, as a result of some periodic process that usually involves the exchange of information with one or more neighbors. Therefore, a partial view may be updated even if the global system membership is stable. Cyclon (Voulgaris et al., 2005) is an example of a cyclic protocol.

Reactive strategies usually rely on some failure detection mechanism to trigger the update of partial views when a node fails. If the failure detection mechanism is fast and accurate, reactive mechanisms can provide faster response to failures than cyclic approaches. This approach is also more efficient, as it avoids the constant communication overhead imposed by the cyclic strategy. In contrast, a cyclic strategy allows each peer to access a wider range of distinct nodes with whom they can exchange information, even if the global membership remains unaltered,

⁶To be precise, Scamp is not purely reactive as it includes a lease mechanism that forces nodes to rejoin periodically to deal with nodes that become isolated due to the departure of failure of other nodes.

as the contents of partial views are continually updated. Therefore, implementing gossip on top of reactive partial views is closer to implementing gossip in a system where peers have access to the complete filiation of the system.

Unstructured overlay networks are interesting because they can be maintained with lower overhead than their structured counterparts. Also, their random nature allows localized decisions (*i.e.*, they require less coordination) which offer the possibility to react in a more timely manner to filiation dynamics, such as concurrent joins, leaves, and failures, making these networks potentially more resilient to churn. Additionally, because neighboring associations are not constrained by node identifiers, or a global coordination strategy, there is a higher degree of freedom when managing the contents of neighbor sets.

Unfortunately the random nature of unstructured overlay networks also leads to some undesired features. Namely, it is not trivial to support efficient application routing, as there are no correlation among the identifiers of nodes that establish neighboring relations among them. Additionally, the existence of several redundant paths among peers, while improving the resilience of the overlay to node failures, promotes inefficient communication patterns among peers. This effect can be augmented by imbalance in the popularity of nodes, *i.e.*, where some peers identifiers are present in the neighboring sets of a much larger number of nodes than others (unbalanced distribution of in-degree and out-degree). Inevitably, such peers will be required to participate in more information exchanges, which in some cases may lead to the exhaustion of their computational resources or available bandwidth. Finally, as the overlay is build at random, it usually does not take into consideration the properties of links at the underlay level (*e.g.*, the properties of the IP network such as latency or number of hops that separate two peers). This may lead to inefficient communication patterns, a problem that is usually dubbed *topology mismatch* (Hsiao et al., 2009). Notice that these shortcomings are also shared, in some measure, by structured solutions. However, as the topology of unstructured overlays is more flexible, such disadvantages can potentially be more easily circumvented.

2.4 Topology Management of Unstructured Overlay Networks

As hinted in the introduction, the thesis addresses mechanisms to manage the topology of unstructured overlays that operate at the overlay layer (*i.e.*, by introducing changes directly over the protocols that build and maintain the overlay) and also at the P2P service layer (*i.e.*, by introducing changes on the behavior of protocols used for coordinating peers and offer a distributed service).

This section discusses the fundamental aspects of these two approaches providing a brief overview of their strength and limitations.

2.4.1 Management at the Overlay Layer

The fundamental idea behind topology management at the overlay layer, is to directly manipulate the protocol that builds and maintains the unstructured overlay, as to ensure that the neighboring associations established among peers result in a overlay that owns a set of properties that benefit P2P services operating on top of them.

This can be achieved, for instance, by ensuring or giving preference to overlay links that match a specific criteria *e.g.*, links that have a point-to-point latency below a given threshold or offer a minimum bandwidth. Such an approach can also be employed to enforce some topological performance indicator *e.g.*, ensuring that the average shortest path among any pair of peers never grows above a pre-determined value or that no peer in the overlay presents a clustering coefficient above a given threshold, ensuring therefore that the overlay clustering coefficient also remains below that threshold.

The goal is to provide such constraints directly at the overlay layer, so that P2P services can be designed without any knowledge or special concern about ensuring these properties. We identify two fundamental approaches to exploit this strategy and that we explore in the context of the work presented in the thesis.

2.4.1.1 Main Approaches

Managing the topology at the overlay layer can be achieved by having specific constraints concerning neighboring relations of each peer. This ensures that no overlay link is ever created which violates these constraints. Alternatively, one can also build an unstructured overlay using a random algorithm, and then improve its topology by swapping existing overlay links by new links among peers in the system following an optimization strategy such as giving preference to links with lower point-to-point latency values.

Considering this, we now provide a more concise definition for each of these two approaches:

Control This approach is based on adding constraints to the neighboring relations of peers. Therefore, no overlay link is established that violates these constraints. Note that contrary to DHTs, the constraints over neighboring relations typically do not depend on the identifiers of nodes. Contrary to DHTs, the overlay topologies that result from employing this approach over unstructured overlay networks are impossible to predict *a priori*, even if one has global knowledge concerning the system filiation.

Bias This approach is based on building an overlay with a random topology (*i.e.*, without enforcing any constraint among peers neighboring relations) and then iteratively improving the topology of the overlay by swapping existing overlay links by new links in such a way that a set of performance criteria are improved as a result of this biasing process. The biasing process is permanently executed as to ensure that the overlay topology can accommodate new participants while retaining an optimized topology.

2.4.1.2 Strengths and Limitations

Managing the topology of unstructured overlay networks at the overlay layer offers the possibility for improving the operation of P2P services in a transparent way for the service. This makes the design and implementation of these service significantly more simple, as the protocols do not have to take any special consideration concerning the overlay topology or system filiation. Additionally, by directly manipulating the links that form the overlay network, one can avoid sub-optimal links and therefore the potential for improving the operation of P2P applications or services is greater.

There are however some pitfalls that might disrupt the correctness of the unstructured overlay topology and consequently endangers the correctness of P2P services executed over it. In particular, management mechanisms that rely on a control approach might be unable to ensure the connectivity of the overlay, as well as ensuring a balanced in-degree distribution and low clustering coefficient, without violating topological constraints. For instance, consider a simplistic example where a maximum point-to-point latency constraint is in place. If a subset of nodes in the system are connected through high latency connections (*e.g.*, dial-up connections) the protocol will be unable to locate other peers in the system with whom such nodes can establish neighboring relations without creating overlay links with a point-to-point latency above that of the considered threshold. This results in such nodes being unable to join the overlay and consequently violating the connectivity property of the overlay.

Another relevant limitation of control approaches is that in a churn scenario, peers might be unable to locate nodes in the system with whom they can establish neighboring relations for recovering from the departure of previous neighbors in a timely fashion. This can lead to the catastrophic failure of the overlay network during a churn period, rendering it impossible to repair the overlay even if the system enters in steady state (*i.e.*, if the global filiation of the system becomes stable).

The bias process can easily overcome churn scenarios, as the base overlay is build at random, therefore under churn peers are free to establish neighboring associations at random (*i.e.*, without being required to respect any constraint) which makes the overlay more robust to these scenarios. However, if the iterative biasing process is performed in a naive fashion, the overlay properties, namely the connectivity and the balanced in-degree distribution, can still be violated. Additionally, in the previously discussed scenario where one bias the overlay topology to promote low point-to-point latency links, nodes with high latency connections can still become isolated from the remaining of their peers. Previous works (Tang & Ward, 2005; Melamed & Keidar, 2004) have proposed the maintenance of unbiased overlay links (*i.e.*, purely random or suboptimal overlay links) to overcome these situations.

2.4.1.3 Previous Work

Several previous works found in the literature have followed strategies for managing the topology of unstructured overlay networks that fall in the scope of control and bias approaches at

the overlay layer.

- Control approaches:
 - Araneola by Melamed e Keidar (2004) propose an unstructured overlay network where each peer selects its overlay neighbors in such a way that a configurable portion of its neighbors are close considering the underlay topology, improving the point-to-point communication patterns established by peers over the overlay.
 - The HiScamp overlay based on the design of Scamp (Ganesh et al., 2003) proposed by Ganesh et al. (2002) features a hierarchical unstructured overlay topology that tries to match the underlying autonomous system (AS) topology. In HiScamp only a single peer among the population of nodes in an AS owns an overlay link for another peer in each other existing AS. The goal of HiScamp is to promote efficient gossip-based dissemination where messages are required to transverse high-cost inter-AS links fewer times.
 - In previous work, we have proposed HyParView (Leitão et al., 2007b) which is an unstructured overlay network that features symmetric overlay links. It was shown that this highly contributes to ensure a balanced in-degree distribution across all peers in the system, which in turn makes the overlay topology highly resilient to concurrent node failures.
 - ITA (Papadakis et al., 2009) is a protocol that aims at addressing the topology mismatch problem in unstructured overlay networks. The authors rely on a sampling-based scheme to construct the overlay in such a way that it becomes latency-aware. ITA tries to protect relevant properties of unstructured overlays, while also aiming at distributing the load imposed over routers at the underlay level.
 - A recent work by Glendenning et al. (2011) proposes a DHT where each node is materialized by a fully connected clique of nodes. The filiation of cliques is random in nature and achieved by leveraging a centralized coordination scheme.
- Bias approaches:
 - Narada (Chu et al., 2002) is a system used for supporting P2P broadcast over an unstructured overlay which is biased to promote the use of low latency links. Narada

was designed by assuming that nodes have access to the full system filiation in order to maintain the overlay topology.

- GoCast is a dissemination system proposed by Tang e Ward (2005) that relies on an overlay network which is biased to promote both low latency and high latency links while at the same time ensuring a balanced out-degree and in-degree across all peers in the system.
- T-Man is an overlay topology management scheme proposed by Jelasity et al. (2009) which uses a gossip-based iterative optimization scheme which allows nodes to continually bias their local neighboring set accordingly to an utility function which can order a set of peers according to their utility as an overlay neighbor.
- Gia (Chawathe et al., 2003) is a resource location system that operates on top of an unstructured overlay network which is biased as the system evolves to give preference to neighboring associations where one of the peers is a high capacity node (*i.e.* a node with more bandwidth and processing power) to improve the performance of one-hop replication of resource indexes. The biased topology is then leveraged to efficiently route queries primarily to high capacity nodes.
- SOSPNet is another resource location system proposed by Garbacki et al. (2007) which leverages on a hierarchical unstructured overlay network (*i.e.* a super-peer network). In SOSPNet regular peers bias their links to super-peers accordingly to past query results obtained when issuing queries to those super-peers.

We further discuss these works and compare them with the relevant contributions of the thesis in the following chapters.

2.4.2 Management at the P2P Service Layer

The fundamental idea associated with unstructured overlay topology management at the P2P service layer, is to expose the overlay links maintained by the overlay layer to the service, and allow the service to adapt its behavior accordingly to the feedback concerning the use of these links during the execution of the distributed protocol.

This can be achieved by using two distinct approaches. The first is to adapt the use of the overlay links at the service layer. This can be achieved for instance by giving preference to

forward messages to a sub-set of the overlay links, or by employing different communication approaches to a sub-set of links. Alternatively, one can leverage the feedback obtained from the execution of the distributed protocol that materialize the P2P service to explicitly add additional overlay links which are managed in a way that is independent of the overlay logic materialized by the overlay maintenance protocol.

The goal is to allow the overlay protocol to operate independently of P2P services, and to leverage the P2P service logic to improve its performance by taking explicitly into consideration the underlying overlay topology. In the following we clearly define the two approaches considered in the context of the thesis to manage the topology of unstructured overlays at the P2P service layer.

2.4.2.1 Main Approaches

As discussed previously, managing the topology at the P2P service layer can be achieved through two distinct approaches. The first is to adapt the use of the links by the distributed protocol by taking into consideration the properties of, or the results obtained by the service from employing, different overlay links. This strategy can allow to embed highly efficient topologies on top of the random unstructured overlay topology, which can then be exploited by the P2P service to improve its performance. Alternatively, one can enrich the overlay topology by creating additional links at the P2P service layer which are maintained in an independent fashion of the unstructured overlay. As these links are selected by following the P2P service logic, they can assist in improving its performance independently of the protocol used to manage the underlying unstructured overlay.

Considering this, we provide the two following definitions for each of these approaches:

Embed This approach is based on applying different communication strategies for different overlay links accordingly to their properties or based on past feedback obtained through the previous use of those links. In particular, by using different communication approaches over the links provided by the overlay layer, one can embed a secondary topology over the topology of the unstructured overlay network. This allows the protocol, which materialized the P2P service, to rely on a more efficient topology to support its main operation. This can be achieved without requiring the protocol responsible for the

management of the unstructured overlay to be aware of the P2P service logic.

Enrich This approach is based on enriching the overlay network topology with additional links following the P2P service logic. These links are not visible to the underlying protocol that manages the unstructured overlay topology. Such additional links can be used to improve the operation of the P2P service. This requires the P2P service layer to maintain an additional set of neighbors to materialize the additional overlay links.

2.4.2.2 Strengths and Limitations

Managing the topology of unstructured overlay networks at the P2P service layer offers the possibility for improving the overlay topology in concordance with the specific requirements of the distributed protocol, that offers the service to the application layer. Additionally, this strategy enable one to improve the overlay without imposing any additional dynamic to the overlay topology, which highly contributes for the stability and protection of the unstructured overlay properties. Such an approach can easily be employed independently of the underlying unstructured overlay, as long as the protocol that manages the overlay ensures any property required for the correctness of the P2P service operation.

However, by employing such a strategy one may be limited in terms of the overall performance gain that can be obtained, as this approach is mostly limited to operate by using the links provided by the underlying overlay network. Additionally, sub-optimal links (considering a specific set of performance criteria) are never removed from the overlay network, which can hinder the performance of P2P services and applications executed over that overlay.

By employing a management mechanism that operates at the P2P service layer, the distributed protocol that offers the service must incorporate some explicit knowledge about the system filiation, which might add to the complexity of these protocols. Furthermore, this might require additional coordination mechanisms among peers to maintain either the embedded topology or the additional links, which can incur in non-negligible overhead effectively limiting the scalability of the overall system.

2.4.2.3 Previous Work

Some previous works found in the literature have followed approaches that are similar to the two discussed above in order to manage the topology of unstructured overlay networks. In the following we briefly introduce some of these works which will be further discussed (along with others) in the thesis, in the context of the contributions which are more related to them.

- Embed approaches:
 - Narada (Chu et al., 2002) also includes a mechanism to embed an efficient spanning tree over the biased unstructured overlay network maintained by the system to support efficient application level broadcast. The proposed solution however is not scalable, as it relies on full membership information of the system and on a distance vector algorithm to select which overlay links become part of the spanning tree.
 - Previous work proposed by Liang et al. (2005), named MON, proposes an overlay network management mechanism that effectively embeds both spanning trees and directed acyclic graphs over a large scale unstructured overlay network. This is achieved by disseminating specialized control messages over the unstructured overlay and having nodes coordinate among themselves to establish the embedded topology. However, MON was designed to build short lived embedded topologies and does not make any consideration concerning fault-tolerance.
 - GoCast (Tang & Ward, 2005) also includes a mechanism to embed an efficient spanning tree on top of the biased unstructured overlay network created by the protocol. This spanning tree is used to disseminate messages in an efficient fashion. However, the protocol relies on a distance routing algorithm to embed the spanning tree which is a solution that does not scale for large-scale systems deployed over the Internet.
 - In previous work we have proposed Plumtree (Leitão et al., 2007a) which combines eager-push and lazy-push gossip communication modes to effectively embed a highly efficient and robust spanning tree over an unstructured overlay network. The tree is formed by the closure of the overlay links where eager-push is employed, ensuring that disseminating messages can be performed with low latency. However, in this system the forwarding load is not evenly distributed across all peers, as only nodes that are interior in the tree are required to transmit message payloads.

- The work proposed by Carvalho et al. (2007) describes a system which allow the emergence of efficient embedded structures over an unstructured overlay network. This work also leverages the trade-offs between eager-push and lazy-push gossip communication approaches, and takes into consideration not only properties of the underlying overlay network but also the state of the dissemination process. It was shown that such an approach could allow the emergence of probabilistic embedded topologies that could highly benefit the performance of a gossip-based dissemination protocol.
- Enrich approaches:
 - The work proposed by Kermarrec et al. (2009) explores how to explicitly add overlay links to an unstructured overlay network in order to establish chains of links that can be used to circumvent NAT boxes on P2P systems deployed over the Internet. This is performed by taking into consideration which nodes can exchange messages directly, and by adding these additional links by monitoring the operation of the peer sampling service responsible for managing the unstructured overlay network. Unfortunately, the result from the operation of this solution may degenerate on long forwarding chains among peers, which although allowing peers behind NATs to exchange messages, also renders the communication between them highly inefficient.
 - FASE (Fonseca & Miranda, 2008) is a system for supporting efficient resource location over unstructured overlay network which operates by having pointers to resources stored by nodes deployed over the overlay following a space partition logic. In order to monitor the number of active pointers deployed in the overlay, the system relies on a backup node which establishes additional links at the P2P service layer to all peers storing the pointers. These links are used to monitor the correctness of these peers, and to deploy replacement pointers to the data item in the event of churn.

Summary

In this chapter we have described in some detail some relevant P2P services that are usually supported by unstructured overlay networks. These P2P services form the set of case studies

which will be employed in the thesis to validate and quantify the benefits that can be extracted from the main contributions discussed in the following chapters. Performance metrics, both for overlays and applications, which are used to evaluate the topology of unstructured overlay networks have also been presented and discussed.

We then introduced overlay network and distinguished among unstructured and structured designs. Finally, we have provided a brief overview of existing techniques to manage the topology of unstructured overlay networks, focusing on two classes of solutions: Those that operate directly at the overlay network layer, and those that operate at the P2P service layer. Finally, we briefly surveyed previous works that have explored the use of these techniques.

Control the Topology: CellFarm



This chapter introduces and evaluates CellFarm, a protocol that exploits the control approach for managing the topology of unstructured overlay networks at the overlay layer.

We start by providing a motivation for this contribution and defining its goals. We then describe the fundamental building blocks and algorithms of CellFarm. A case study used to illustrate the benefits that can be extracted from CellFarm is then presented. The chapter follows with a discussion on the relevant aspects of the contribution, and with an additional discussion on related work found in the literature.

3.1 Motivation and Goals

3.1.1 Motivation

The idea of organizing processes into groups to build dependable systems has been used for a long time in different ways. In fact, it is a cornerstone of most existing techniques that aim at achieving distributed fault-tolerance by software (Birman & Renesse, 1994; Powell, 1994). For instance, one can organize a set of processes such that they collectively implement a replicated fault-tolerant state machine (Schneider, 1990), or simply run a replicated set of web servers with the main goal of achieving load-distribution (Cardellini et al., 2002).

As a result, there is a large body of research on techniques to coordinate members of a process group to achieve dependable services (Mullender, 1993; Birman & Renesse, 1994). The problem of how processes are selected and organized to build these groups is a much less studied topic. Quite often, the members of a process group are simply statically defined at deployment time. In more sophisticated systems there is a configuration manager that has a global view of the system and instructs processes to dynamically join or leave a given process group, for instance, to preserve a predefined target replication degree (Liu et al., 2010; Sahota et al., 2009). Even if the configuration manager may be replicated for fault-tolerance (*e.g.*, using

techniques such as the ones proposed by Moura e Endler (2003)), it is typically implemented as a logically centralized component, that maintains the global state required to know which groups need to be re-configured and which processes are more suitable to have their tasks re-assigned (Liu et al., 2010; Sahota et al., 2009). These approaches, although presenting the potential to leverage on global knowledge to achieve an optimal configuration, are not scalable and cannot be applied effectively in the context of large-scale P2P systems deployed over the Internet and subjected to churn conditions.

In some recent systems of medium scale, group formation is performed in a decentralized manner using consistent hashing. For instance, the Infinispan¹ distributed in-memory cache, relies on consistent hashing to select which nodes keep replicas of each data item. Unfortunately, this scheme also does not present enough scalability to be applied to large-scale P2P systems. As it requires that all nodes have a consistent and complete view of the membership for the entire system such that, consistent hashing techniques can be employed.

Moreover, the idea of using process groups has also been proposed to build dependable large-scale distributed systems. In fact, it is now possible to find in the literature several examples of architectures that use process groups in the context of systems with very large number of nodes (in the order of thousands), where the membership can be highly dynamic, in a similar fashion to large-scale P2P systems. For completeness, it is worth to take into consideration some of these examples:

- The work of Sahota et al. (2009) uses process groups to build scalable P2P grid services that maintain a resilient and efficient information system to assist in the management of available resources in the grid infrastructure.
- In Kun et al. (2009) the notion of process groups is suggested as a way to improve search in unstructured systems, more specifically, as a way to optimize one-hop replication schemes.
- In Liu et al. (2010), a large-scale P2P streaming system is proposed where each logical node of the streaming tree is in fact composed by a process group of fully connected nodes, that collaborate to achieve dependability and load-balancing.

¹See www.jboss.org/infinispan.

- In Glendenning et al. (2011) a dependable DHT is proposed where each node of the DHT is implemented by a group of processes that run Paxos (Lamport, 1998) to coordinate their tasks.

Unfortunately, these previous approaches either do not address explicitly how process groups are assembled and maintained, or use specialized solutions that are tightly coupled with the particular services being implemented. Given that there is a significant number of large-scale (and particularly P2P) services that could benefit from the existence of a generic infrastructure to create and maintain groups of peers in dynamic large-scale systems, it becomes relevant to design such infrastructure as an independent service that could be used as a building block in different contexts.

Also, as we have seen, previous approaches to build and maintain process groups are not scalable, and cannot be easily extended to operate on systems with large number of processes or subject to dynamic membership (where the global state needs to be updated frequently).

At first sight, it may appear that some naive solutions would be enough to achieve such goal. For instance, we could imagine a solution that would organize processes in a Chord-like ring (Stoica et al., 2001), and then would split the ring in portions of the desired group size. However, it is not possible to ensure that groups maintain a given target size in face of churn without frequent global reorganizations of the groups, which is clearly a non-scalable mechanism for systems with thousand or more nodes. All the remaining similar solutions we have considered suffer from equivalent problems.

In face of these challenges, the work presented in this chapter addresses the problem of dependable group creation and maintenance in large-scale P2P environment. In particular we propose a generic service that allows nodes to structure themselves in an overlay network of fault-tolerant process groups. The algorithm explores a topology management based on a control approach, where constraints are imposed over the neighboring associations among peers to achieve a unique topology. The solution presented here is fully decentralized and self-organizing. The topology resulting from our algorithm is composed of cliques of peers (a process group), where the members of each clique maintain connections to a large variety of other cliques, ensuring the overall connectivity of the overlay (as depicted in Figure 3.1). This is achieved without relying on neighboring constraints which take into consideration the identifiers of peers in the system. We have named our system CellFarm. As we will explain further

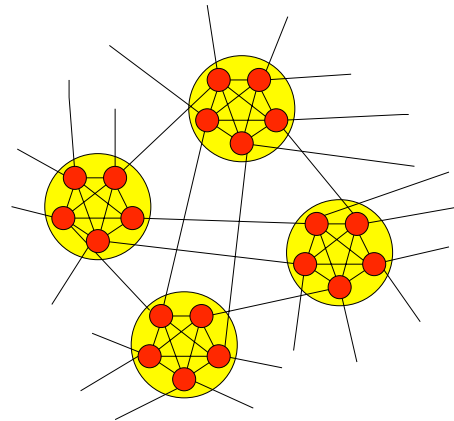


Figure 3.1: The CellFarm overlay. Smaller (red) dots represent physical nodes, while larger (yellow) nodes represent virtual nodes.

ahead in the text, the global connectivity of the overlay is essential, not only to potentially ensure the correct operation of P2P services executed on top of CellFarm, but also to ensure the correct operation of the algorithms employed to maintain the overlay topology despite failures, or even churn scenarios.

To show that an overlay such as CellFarm can be in fact used in a large variety of scenarios, we illustrate its application using a resource location service case study. This case study is interesting because the work of Kun et al. (2009) shows that search can be improved if process groups are provided, but give no algorithm to create or maintain these groups. We show that CellFarm can be in fact used with the same goal, as we apply the resulting unstructured overlay to perform one-hop replication of resource indexes replicated by nodes and exploit its topology to disseminate efficient queries that search resources over those indexes.

3.1.2 Goals

The goal of CellFarm is to provide a support infrastructure for easing the task of achieving fault-tolerance through replication and improve load distribution across peers in large-scale P2P systems.

In order to achieve these goals we have developed an unstructured overlay management protocol that exploits the control technique. In a nutshell, this translates into a protocol that imposes (soft) constraints over the neighboring relations established among peers in a fully decentralized fashion. As discussed in Chapter 2, the constraints imposed over the overlay

topology do not depend on node identifiers. This means that the resulting overlay topology is impossible to predict, even if one has global knowledge of the system filiation. In fact, as it will become clearer in the remaining of this chapter, the same set of peers may generate a large number of distinct topologies which respect all imposed constraints. This ensures that the overlay has enough flexibility to adapt itself in face of churn scenarios in a timely fashion, while ensuring at the same time the maintenance of the overlay properties and topology.

In more detail, CellFarm adds constraints to ensure that peers self-organize into cliques which in turn are highly and randomly connected among them. To promote the control of the in-degree distribution across peers, overlay links that materialize the cliques of peers and the connections among them are symmetric. This means that the overlay denoted by these links can be seen as an undirected graph. Additionally, this ensures that, if cliques have approximately the same size, and if each peer has the same number of overlay links to other cliques, all cliques have approximately the same reachability in the overlay *i.e.*, all cliques have a similar probability of being visited by a message randomly forwarded among peers. This is relevant to ensure the correctness of the mechanisms used to enforce the overlay topology.

Each clique of peers can then be used as a natural infrastructure for supporting state replication. Examples of state replication are the replication of resource indexes (one-hop replication) for resource location services, or even the state of computational tasks being computed on a public cycle-sharing infrastructure (similar to the Boinc platform (Anderson, 2004)). Additionally, as each peer maintains the same number of (incoming) overlay links from other cliques in the overlay, there is a potential for achieving a natural load distribution among peers of a single clique (*i.e.*, all peers in each clique will, on average, receive and process the same number of messages).

3.2 The CellFarm Protocol

In this section we provide the full specification of the CellFarm protocol. We start by introducing the rationale associated with the operation of the protocol, and then provide a high level description of the interaction among the different micro-protocol which are employed to control the topology. The complete specification of the operation of these micro-protocols is then provided and illustrated by pseudo-code for clarity when necessary.

3.2.1 Rationale

The main goal of CellFarm is to build and maintain a self-organizing unstructured overlay network that exports a view of process groups to P2P services executed on top of it. Each process group is achieved by a (fully-connected) cluster of peers named a Cell. The CellFarm overlay has the following characteristics:

- Each peer belongs to a single Cell and each Cell has a probabilistic unique identifier named *cID*;
- Each peer (eventually) knows the identity of all other members of its Cell (maintained by a local partial view of the system *i.e.*, a neighbor set named *iView*) and maintains a link to each of these neighbors (thus, Cells are fully connected);
- Each peer maintains a number of links to other Cells (through a second neighbor set named *eView*), which ensures the global connectivity of the CellFarm overlay. As we explain further ahead this is a requirement to ensure the correct operation of the protocol that maintains CellFarm topology.

The target size of each Cell is a protocol parameter; this allows this solution to be used by a wide range of P2P services and applications with different replication and load distribution requirements. A key aspect of our design is that CellFarm does not attempt to ensure that each Cell has *exactly* the target size. Such goal would be very hard to achieve in large scale dynamic environments (where multiple joins, leaves, and failures happen concurrently) without resorting to some sort of centralized coordination mechanism. Instead, the behavior of CellFarm is controlled by the following set of parameters:

- *Target Cell Size (CSTARGET)*: The target Cell size. Peers that belong to a Cell attempt to prevent the further growth of that Cell as soon as it reaches a size of *CSTARGET*.
- *Cell Max Size (CSMAX)*: If the size of a Cell becomes larger than threshold *CSMAX*, Cell members coordinate to split it into two new smaller Cells.
- *Cell Minimum Size (CSMIN)*: If the size of a Cell becomes smaller than threshold *CSMIN*, Cell members gradually try to abandon that Cell to join larger and more stable Cells, causing the graceful fading of the small Cell.

Besides controlling the size of each Cell, CellFarm strives to promote the creation of multi-

ple, distinct, inter-Cell connections. The existence of these diverse connections are essential to support the efficient exploration of the overlay through random walks, which is a key mechanism employed by the protocol that builds and maintains CellFarm. Additionally, this brings the following additional advantages: *i*) it makes the overlay robust to node and link failures; *ii*) it reduces the clustering in the connections among Cells and finally; *iii*) it lowers the diameter of the overlay (which eases the exploration of the overlay and also provides benefits for applications that disseminate messages, such as resource location systems).

For this purpose, each node maintains an external view (*i.e.*, a neighbor set) with identifiers of other peers located in different Cells (called the eView, whose size θ is also a protocol parameter). As it will become clear later in the text, the fact that each Cell has a probabilistic unique identifier eases the task of balancing inter-Cell neighboring relations among peers of a given Cell, improving several properties of the overlay topology.

Each node maintains a link to all its internal neighbors (peers in the iView) and external neighbors (peers in the eView). These links are maintained using TCP connections, which are used to support the exchange of all messages between any pair of connecting nodes. The use of TCP is motivated by two reasons: *i*) it allows the communication between nodes to be network friendly as TCP flow control mechanisms are leveraged. Moreover, it allows to model the system without considering message losses between nodes; *ii*) TCP is used as an unreliable failure detector, as previously discussed by Leitão et al. (2007b). This is used, to expedite the detection of peers that departed or failed (*i.e.*, crashed), allowing the protocol to make the adequate adaptations to ensure the correctness of the overlay topology in a timely fashion.

Additionally, each node owns an additional neighbor set named global view or simply gView, containing additional random peer identifiers. This view is used as a backup list to regain the connectivity of the overlay in face of catastrophic failures, as well as to benefit the exploration of the overlay by each node. For instance, for establishing additional inter-Cell links. Notice that contrary to the remaining neighbor sets maintained by the CellFarm protocol, no TCP connections are maintained to peers in the gView. This is motivated by three reasons: *i*) the contents of these views are updated periodically; *ii*) these views are not symmetric; and, *iii*) the gView is not used to support direct communication among nodes, therefore no guarantees are provided concerning the correctness of nodes which identifiers are in those views (although

Algorithm 1: CellFarm Protocol Overview

```

1: upon event Init do
2:   trigger Join Procedure

3: every  $\Delta T_1 + \text{random}(\pi)$  do
4:   if #iView  $\geq$  CSMAX then trigger Divide Procedure
5:   else if #iView  $<$  CSMIN then trigger Collapse Procedure

6: every  $\Delta T_2$  do
7:   if #eView  $<$   $\theta$  then trigger External Neighboring Procedure

8: every  $\Delta T_3$  with a probability  $\rho$  do
9:   if #pdv = 0 then trigger Anti-entropy Procedure

```

the protocol that maintains them - a variant of the Cyclon protocol (Voulgaris et al., 2005) - ensures that eventually failed participants are removed from all correct nodes gViews).

3.2.2 Algorithm

The CellFarm overlay is maintained by five micro-protocols that are described below. Collectively, these micro-protocols strive to maintain the following properties (obviously, these properties may be temporarily violated when the system is unstable, for instance, under high churn, but are re-attained when the system stabilizes in a timely fashion): *i*) each peer belongs to a single Cell; *ii*) each node has a link to each and every other peer in its own Cell; *iii*) each peer has θ links to nodes in different Cells; *iv*) no Cell is smaller than CSMIN and; *v*) no Cell is larger than CSMAX.

Algorithm 1 provides a macroscopic perspective on the operation of the algorithm. The five micro-protocols that maintain CellFarm execute the following procedures: a *Join Procedure* used to join peers to the overlay; a *Divide Procedure* used to prevent the size of any Cell to exceed the CSMAX threshold; a *Collapse Procedure* used to gracefully eliminate Cells of small size; an *External Neighboring Procedure* in charge of promoting the creation of heterogeneous inter-Cell links; and finally, a *Cell Anti-entropy Procedure* used to maintain the consistency of the intra-Cell information which can also be leveraged to assist with the replication of data among nodes of a Cell by a P2P service operating on top of CellFarm.

There are some dependencies among these micro-protocols. In particular, to reduce the cost and increase the robustness and parallelism of the join procedure, the Cell size is allowed to temporarily exceed its upper threshold, in face of multiple concurrent join requests. This trig-

gers (at some random time, to avoid global synchronization) the execution of the divide procedure which will result in the creation of two new Cells with a size close or equal to C_{TARGET} . However, node failures may bring a Cell size below the lower threshold triggering the Cell collapse procedure. The thresholds that triggers the divide and collapse procedures should be configured so that system oscillations are avoided. Failures also affect the number of correct external neighbors maintained by each node. In this case, the external neighboring procedure is used to locate new external neighbors, such that each node reaches the target *external degree* (θ). Finally, under churn nodes may have inconsistent views of their current Cell composition. To address such scenarios (and also to help Cell members to balance their external neighboring relations) every node periodically executes, with a given probability ρ , a gossip-based anti-entropy procedure where it exchanges information with a random Cell neighbor (notice that this procedure operates in a similar fashion to a rumor mongering service). The *pdv* set denoted in the algorithm represents the *pending division vector*, which is used to temporarily store division proposal issued by other elements of the Cell. Verifying that this set is empty allows the protocol to avoid spending resources to manage the filiation of a Cell which is about to be divided into two new Cells. The use of this set is discussed in detail further ahead in this chapter in section 3.2.2.2. In the following text a more detailed description of each component of the CellFarm overlay management protocol is provided.

3.2.2.1 Join Procedure

The first peer to join the overlay only has to generate a random Cell identifier (Alg. 2, lines 3 – 4). Any additional peer, say *new*, sends a JOIN request to a node that is already part of the overlay, called the *contact* peer (Alg. 2, lines 5 – 6). If the contact node is part of a Cell which size is below C_{TARGET} , it immediately accepts the new node into that Cell (Alg. 2, lines 8 – 10). Otherwise, the request is forwarded in the overlay using a limited length random walk, that is preferably forwarded through links connecting nodes in different Cells (as to increase the number of different Cells visited by it). The random walk terminates when a Cell with size smaller than C_{TARGET} is visited (Alg. 2, line 16). If the time to live of the random walk expires before an appropriate Cell is found, the new node is added to the Cell where the random walk terminates, regardless of its size (*i.e.*, even if the Cell size is above C_{TARGET}).

When a particular peer, say *new*, is accepted into a Cell, a JOINREPLY message is sent to

Algorithm 2: Join Procedure

```

1: upon event Init do
2:    $cID \leftarrow \perp$ 
3:   if  $contact = \perp$ 
4:      $cID \leftarrow \text{generate unique ID}$ 
5:   else
6:     trigger Send(JOIN, contact)

7: upon event Receive(JOIN, newNode) do
8:   if  $\#iView < CSTARGET$  then
9:     trigger Send(JOINREPLY, sender,  $cID$ , iView)
10:     $iView \leftarrow iView \cup \{sender\}$ 
11:   else
12:     $n \leftarrow \text{selectRandom}(eView \cup iView)$ 
13:    trigger Send(FORWARDJOIN, n, sender, MAXTTL)

14: upon event Receive(FORWARDJOIN, sender, newNode, ttl) do
15:    $ttl \leftarrow ttl - 1$ 
16:   if  $\#iView < CSTARGET$  or  $ttl = 0$  then
17:     trigger Send(JOINREPLY, newNode,  $cID$ , iView)
18:      $iView \leftarrow iView \cup \{newNode\}$ 
19:   else
20:      $n \leftarrow \text{selectRandom}((eView \cup iView) \setminus sender)$ 
21:     trigger Send(FORWARDJOIN, n, newNode, ttl)

22: upon event Receive(JOINREPLY, sender, id, view) do
23:    $cID \leftarrow id$ 
24:    $iView \leftarrow view \cup \{sender\}$ 
25:   forall  $n \in view$  do
26:     trigger Send(NEIGHBORINGREQUEST, n,  $cID$ )

27: upon event Receive(NEIGHBORINGREQUEST, sender, id) do
28:   if  $cID = id$  then
29:      $iView \leftarrow iView \cup \{sender\}$ 
30:   else
31:     trigger Send(DISCONNECTREQUEST, sender)

```

it by the peer that accepts the join request *i.e.*, the node where the random walk terminates (Alg. 2, lines 9 – 11 and 17 – 18). Upon receiving the JOINREPLY message, peer *new* uses the information contained in it to update its *cID* identifier and to establish neighboring relations with all remaining members of the Cell by sending NEIGHBORINGREQUEST messages (Alg. 2, lines 23 – 26). Nodes that receive this message validate if the new node is joining the correct Cell (by comparing their own Cell identifier with the identifier carried in the message), in which case they add its identifier to their local *iView*. Otherwise they refuse the request by replying with a DISCONNECTREQUEST message (Alg. 3, lines 27 – 31). Upon the reception of a DISCONNECTREQUEST message, a node removes the sender of the message from the *iView* or *eView* and closes the TCP connection maintained to that node.

As the reader can probably guess, if the identifier of that Cell is updated between the transmission of the JOINREPLY and the peer *new* establishing the overlay links for its Cell neighbors,

new's requests will be rejected by those peers resulting in *new* becoming disconnected from the overlay. However, such scenario, despite being very improbable, is circumvented by having the peer rejoin the overlay (using an element of its gView as a contact peer).

3.2.2.2 CellFarm Divide Procedure

Algorithm 3: Divide Procedure

```

1: upon event CHECKCELLSIZE TIMER do
2:   if  $cID \neq \perp$  and  $pdv = \emptyset$  then
3:     if  $\#iView \geq CSMAX$  and  $\nexists n: n \in iView: n.nID < nID$  then
4:        $ID_a \leftarrow \text{get new unique id}$ 
5:        $ID_b \leftarrow \text{get new unique id}$ 
6:        $a \leftarrow \{myself\} \cup \text{SelectHalf}(iView)$ 
7:        $b \leftarrow iView \setminus \{a\}$ 
8:        $pdv \leftarrow \{\text{CELLDIVISION}(myself, cID, ID_a, ID_b, a, b)\}$ 
9:       forall  $n \in iView$  do
10:        trigger  $\text{Send}(\text{CELLDIVISION}, n, cID, ID_a, ID_b, a, b)$ 
11:        setup timer ( $\text{EXECUTECELLDIVISION TIMER}, RTT * 2$ )

12: upon event EXECUTECELLDIVISION TIMER do
13:   if  $pdv \neq \emptyset$  then
14:      $s \leftarrow s \in pdv \rightarrow \nexists x: x \in pdv \wedge x.sender.nID < s.sender.nID$ 
15:     if  $myself \in s.a$  then
16:       forall  $n \in iView$  do
17:         if  $n \in s.a$  then
18:           trigger  $\text{Send}(\text{CELLUPDATE}, n, cID, s.ID_a, true)$ 
19:         else if  $\text{position}(myself, s.a) = \text{position}(n, s.b)$  then
20:           trigger  $\text{Send}(\text{CELLUPDATE}, n, cID, s.ID_a, false)$ 
21:            $iView \leftarrow iView \setminus \{n\}$ 
22:            $eView \leftarrow eView \cup \{n\}$ 
23:         else
24:           trigger  $\text{Send}(\text{DISCONNECTREQUEST}, n)$ 
25:            $iView \leftarrow iView \setminus \{n\}$ 
26:          $cID \leftarrow s.ID_a$ 
27:     else if  $myself \in s.b$  then
28:       forall  $n \in iView$  do
29:         if  $n \in s.b$  then
30:           trigger  $\text{Send}(\text{CELLUPDATE}, n, cID, s.ID_b, true)$ 
31:         else if  $\text{position}(myself, s.b) = \text{position}(n, s.a)$  then
32:           trigger  $\text{Send}(\text{CELLUPDATE}, (n), cID, s.ID_b, false)$ 
33:            $iView \leftarrow iView \setminus \{n\}$ 
34:            $eView \leftarrow eView \cup \{n\}$ 
35:         else
36:           trigger  $\text{Send}(\text{DISCONNECTREQUEST}, n)$ 
37:            $iView \leftarrow iView \setminus \{n\}$ 
38:          $cID \leftarrow s.ID_b$ 
39:      $pdv \leftarrow \emptyset$ 

```

This procedure is triggered when the size of a Cell exceeds the threshold $CSMAX$ and its purpose is to split a Cell into two smaller Cells. The intuition for using Cell division as the mechanism to generate new Cells is to avoid the creation of a large number of small (potentially unitary) Cells. By splitting a large Cell in two (similar to what living cells do) one can

Algorithm 4: Cellfarm Divide Procedure (Continuation)

```

40: upon event Receive(CELLUPDATE, sender, IDold, IDnew, isCell) do
41:   if isCell = true then
42:     if cID ≠ IDnew then
43:       if pdv = ∅ or cID ≠ IDold then
44:         trigger Send(DISCONNECTREQUEST, sender)
45:         iView ← iView \ {sender}
46:     else
47:       if sender ∈ eView then
48:         if sender.cID ≠ IDnew then
49:           if IDnew = cID then
50:             eView ← eView \ {sender}
51:             iView ← iView ∪ {sender}
52:           else
53:             update local information on cID of sender
54:         if pdv = ∅ or sender ∉ iView then
55:           trigger Send(DISCONNECTREQUEST, sender)
56:           iView ← iView \ {sender}
57:   if pdv ≠ ∅ then
58:     trigger EXECUTECELLDIVISION TIMER

59: upon event Receive(CELLDIVISION, sender, ID, IDa, IDb, a, b) do
60:   if cID = ID then
61:     if pdv = ∅ then
62:       setup timer (EXECUTECELLDIVISION TIMER, RTT *2)
63:       pdv ← pdv ∪ {CELLDIVISION (sender, cID, IDa, IDb, a, b)}

```

generate two stable and independent Cells, with a low impact on the overlay topology and global connectivity. This allows the operations being executed and the data stored at the original Cell to be carried out by one of the new Cells with no overhead, while the remaining new Cell can be used to perform other operations and/or store new data.

The procedure is initiated by the Cell member with the smallest identifier when it detects that the size of its local iView is equal or above the parameter CS_{MAX} (remember that similar to what is done in classical DHTs, each node in our system owns a probabilistic unique identifier despite the fact that these identifiers are not used to define the overlay topology). This condition is verified independently and periodically by each peer every interval $\Delta T_1 + \text{random}(\pi)$ (Alg. 1, lines 3 – 5). This peer then generates two random Cell identifiers (namely ID_a and ID_b) and splits the current Cell membership into two ordered sets a and b , sending this information to the remaining Cell members in a CELLDIVISION message (Alg. 3, lines 4 – 11).

When a CELLDIVISION message is received it is put in *quarantine*, for a period of time that should be greater than twice the maximum round trip time (RTT) between the sender and every other peer of that Cell². The message is stored in a set named *pending division vector*,

²As TCP connections are maintained to all peers in a node's iView, the RTT value can be easily calculated by

or simply *pdv*. The *quarantine* period aims at avoiding that multiple concurrent Cell divisions are initiated when the Cell view is not fully consistent (and more than one member of the Cell believes to have the lowest node identifier). At the end of the quarantine period, the CELLDIVISION message issued by the node with smallest identifier is processed by each peer in the Cell while the remaining are discarded (Alg. 3, lines 12 – 39). Notice that the quarantine period might be hard to calculate in highly dynamic environments. However in situations where this mechanism fails (which was shown experimentally to be very improbable), it only results in the temporary disconnection of a very small number of peers. These nodes can rejoin the overlay, for instance by re-executing the join procedure using a participant extracted from the gView as a contact node.

When a CELLDIVISION message is processed by a node d , it adopts the Cell division proposal included in that message. Thus, it updates its local Cell identifier (cID). This update results in the peer updating its cID to ID_a if its identifier is found in set a , and to ID_b if its identifier is instead included in the set b of the division proposal. Moreover, d sends a CELLUPDATE message to all nodes of its new Cell to accelerate the convergence of the algorithm. Additionally, node d sends a DISCONNECTREQUEST to all nodes in his current iView that do not belong to its new Cell, except to the node d' that occupies the same position in the complementary Cell membership set enclosed in the CELLDIVISION message (sets a and b described previously). To node d' , d sends a request to establish an inter-Cell link; this ensures that the two new Cells remain well connected (this is achieved by sending a CELLUPDATE message with a special flag activated). Notice that peers may be required to disconnect from an external neighbor to add the previous Cell neighbor to the eView set, in the case where they already had θ external neighbors. The large number of external connections created between both Cells is reduced as a result of the anti-entropy procedure described further ahead.

Because the Cell identifier of peers is updated as the result of the division process, at the end of this process each peer also send to their external neighbors a notification of their new Cell identifier (with the exception of the peer added to the eView from the iView due to the division process as described above).

accessing the RTT estimates kept by TCP and selecting the largest value, although a conservative static configuration (e.g., 1 second) is enough to ensure the correct operation of the protocol.

Algorithm 5: CellFarm Collapse Procedure

```

1: upon event CHECKCELLSIZE TIMER do
2:   if #iView < CSMIN then
3:     with a probability of:  $(1 - \#iView/CSMIN)$  do
4:        $n \leftarrow \text{selectRandom}(eView \cup gView \cup iView)$ 
5:       trigger Send(RELOCATEREQUEST,  $n$ , myself, cID, TTL)

6: upon event Receive(RELOCATEREQUEST, sender, node, ID, ttl) do
7:   ttl  $\leftarrow$  ttl - 1
8:   if cID  $\neq$  ID and #iView  $\leq$  CSTARGET then
9:     iView  $\leftarrow$  iView  $\cup$  {node}
10:    if ttl > 0 then
11:      trigger Send(RELOCATEREPLY, node, cID, iView)
12:    else if ttl > 0 then
13:       $n \leftarrow \text{selectRandom}(eView \cup iView)$ 
14:      trigger Send(RELOCATEREQUEST,  $n$ , node, ID, ttl)

15: upon event Receive(RELOCATEREPLY, sender, id, reloc.view) do
16:   if #iView < CSMIN then
17:     forall  $n \in iView$  do
18:       trigger Send(DISCONNECTREQUEST,  $n$ )
19:       iView  $\leftarrow$  iView  $\setminus$  { $n$ }
20:     forall  $n: n \in eView \wedge n.cID = id$  do
21:       trigger Send(CELLUPDATE,  $n$ , cID, id, false)
22:       eView  $\leftarrow$  eView  $\setminus$  { $n$ }
23:       iView  $\leftarrow$  iView  $\cup$  { $n$ }
24:     cID  $\leftarrow$  id
25:     forall  $n: n \in reloc.view \wedge n \notin iView$  do
26:       trigger Send(NEIGHBORINGREQUEST,  $n$ , cID)
27:   else
28:     trigger Send(DISCONNECTREQUEST, sender)

```

3.2.2.3 Collapse Procedure

This procedure is used to gracefully disband a Cell whose size has fallen below the threshold parameter CSMIN, by migrating its members to other (larger and therefore more stable) Cells. This procedure is decentralized. Each node periodically verifies the size of its current Cells (by inspecting the contents of its iView), if the size of the Cell is below CSMIN, it takes the initiative to relocate itself to another Cell, resulting in the collapse of the older one. To avoid the abrupt collapse of a Cell, nodes only decide to initiate the relocation procedure with a given probability p , which increases as the size of the Cell decreases. That probability is provided by the equation $(1 - \#iView/CSMIN)$ (Alg. 5, line 3). To do this, a RELOCATEREQUEST message is propagated in a manner similar to the JOIN request described above (Alg. 5, lines 6 – 14). Notice however, that if the local Cell size has become stable during the time required to execute this procedure (*i.e.*, if the size of the Cell has grown to a value equal or above CSMIN), the peer that issued the relocation request terminates the procedure by issuing a DISCONNECTREQUEST message to the node that replied to it with a RELOCATEREPLY message (Alg. 5, line 28).

Algorithm 6: CellFarm External Neighboring Procedure

```

1: upon event CHECKEXTERNALCONNECTIVITY TIMER do
2: if #eView <  $\theta$  then
3:    $k \leftarrow \emptyset$ 
4:   forall  $n \in \text{eView}$  do
5:      $k \leftarrow k \cup \{n.\text{cID}\}$ 
6:    $d \leftarrow \text{selectRandom}(\text{eView} \cup \text{iView} \text{ cup } \text{gView})$ 
7:   if eView =  $\emptyset$  then
8:     trigger Send(EXTERNALREQUEST,  $d$ , myself, cID,  $k$ , true, TTL)
9:   else
10:    trigger Send(EXTERNALREQUEST,  $d$ , myself, cID,  $k$ , false, TTL)

11: upon event Receive(EXTERNALREQUEST, sender, node, ID,  $k$ , empty, ttl) do
12:    $\text{ttl} \leftarrow \text{ttl} - 1$ 
13:   if cID  $\neq$  ID and cID  $\neq$   $k$  and #eView <  $\theta$  and  $\nexists n \in \text{eView}: n.\text{cID} = \text{ID}$  do
14:     eView  $\leftarrow$  eView  $\cup$  {node}
15:     trigger Send(EXTERNALREPLY, node, cID, ID)
16:   else if ttl > 0
17:      $d \leftarrow \text{selectRandom}(\text{eView} \cup \text{iView})$ 
18:     trigger Send(EXTERNALREQUEST,  $d$ , node, ID,  $k$ , empty, ttl)
19:   else if empty = true
20:      $n \leftarrow \text{selectRandom}(\text{eView})$ 
21:     trigger Send(DISCONNECTREQUEST,  $n$ )
22:     eView  $\leftarrow$  eView  $\setminus$  { $n$ }
23:     eView  $\leftarrow$  eView  $\cup$  {node}
24:     trigger Send(EXTERNALREPLY, node, cID, ID)

25: upon event Receive(EXTERNALREPLY, sender, ID, ID $k$ ) do
26:   if #eView =  $\theta$  then
27:     trigger Send(DISCONNECTREQUEST, sender)
28:   else
29:     eView  $\leftarrow$  eView  $\cup$  {sender}
30:     if ID $k$   $\neq$  cID then
31:       trigger Send(CELLUPDATE, sender, ID $k$ , cID, false)

```

If no suitable Cell is found in the random walk performed by the RELOCATEREQUEST message, the node remains in his current Cell. It will later attempt to re-execute this procedure if its current Cell size does not stabilizes meanwhile.

3.2.2.4 External Neighboring Procedure

To ensure that CellFarm remains highly connected, and that random walks performed to maintain the overlay have an increased chance of success, the protocol attempts to maintain at each node a pre-defined number θ of external neighbors, *i.e.*, links to peers located in other Cells³. Thus, a node that has fewer than θ external neighbors actively tries to establish external links by disseminating an EXTERNALREQUEST message using a fixed-length random walk, that tries to locate a suitable neighbor in a different Cell. A node is considered to be a suitable external

³To ensure the correct operation of random walks, θ should be equal or greater to 2 as to ensure that a random walk received from an external link can be forwarded, if necessary, through another external link.

neighbor if it belongs to a different Cell, has fewer than θ external neighbors, and does not have another external neighbor belonging to the Cell of the peer that issued the request, nor to other Cells to which that peer is already connected (in order to promote inter-Cell link heterogeneity). Alg. 6 depicts this mechanism.

In the particular case where the source of the EXTERNALREQUEST message has no external neighbor, a special flag (named *empty*) is set to true in the request propagated by the random walk (Alg. 6, lines 7 – 8). In this case, if the random walk terminates before a suitable neighbor is found, the last visited node becomes a neighbor of the source, even if it already has θ external neighbors and needs to disconnect from a random external neighbor to maintain a number of external neighbors equal to θ (Alg. 6, lines 19 – 24).

3.2.2.5 Anti-entropy Procedure

In face of concurrent joins and crashes, the iView maintained by different nodes in the same Cell may diverge. To increase the intra-Cell consistency, a gossip-based anti-entropy procedure is executed inside the Cell. Periodically, with a given probability ρ , every node n selects another node p in its Cell and sends to it a message containing its own view of the Cell membership. This allows p to detect missing nodes in its iView even with a small value of ρ ; in our experiments we determined that a ρ value of 0.1 is adequate. Moreover, if p detects some missing nodes in n 's iView it replies to n with a similar message. Notice that a similar overhead could be achieved by using a probability ρ of one and configuring a larger period of time between the execution of this procedure, however, our scheme allows to have faster convergence, avoiding spontaneous synchronization among nodes.

Additionally, the anti-entropy procedure is also exploited to promote the diversity of overlay links connecting different Cells, by allowing nodes in a Cell to coordinate among themselves to avoid the maintenance of redundant external neighboring relationships to peers in the same Cell. To that end, when executing the anti-entropy procedure, a node n , also sends to its peer a list containing the Cell identifiers of all its external neighbors. If a node p receives two consecutive anti-entropy messages that contain a Cell identifier of one of its external neighbors, say e , p removes node e from its eView by sending a DISCONNECTREQUEST message. The reception of two messages is required to promote some stability in the overlay network topology. This process allows to reduce the clustering coefficient among Cells which benefits the routing

of random walks in the overlay, and promotes a lower overlay diameter.

3.2.3 Increasing Fault-Tolerance

As described earlier, to increase the fault-tolerance of CellFarm, we use an approach similar to the one described by Leitão et al. (2007b), *i.e.*, we augment the state of each peer with a random partial view of the entire system, named the gView. The gView is maintained using a low cost background protocol, based on the exchange of shuffle messages among random pairs of nodes. These messages carry samples of node identifiers extracted from both the gView, iView, and eView of the sender. These identifiers are used to update the contents of the gView of the receiver (similar to the operation of the Cyclon protocol (Voulgaris et al., 2005)).

Moreover, whenever a node has to remove a correct peer from its iView or eView, or when it receives a request sent by a node which is not in one of those sets, that node identifier can be added to the gView. Notice that the gView can also be used to facilitate the exploration of the overlay network by peers trying, for instance, to find additional external neighbors (*i.e.* nodes that are associated to different Cells).

As discussed previously, the gView is not used to support direct communication between participants of the system. The gView is managed by a low complexity protocol and, contrary to the remaining partial views used by the CellFarm protocol (*i.e.*, the iView and the eView) no constraints are imposed on its contents.

3.3 Case Study

As noted previously, there are many examples in the literature of large-scale distributed systems where process groups are used to achieve higher reliability and better performance (by distributing the load among the group members). The previous cited examples include distributed grid computing (Liu et al., 2010), highly-resilient DHTs (Glendenning et al., 2011), and unstructured search (Kun et al., 2009). To experiment with CellFarm on all these applications is outside the scope of the thesis. Instead, we focus on a single case study namely, we use the notion of process groups provided by CellFarm to improve a resource location system based on an unstructured overlay network and on one-hop replication (Chawathe et al., 2003). Furthermore, we address the case where the service must exhibit full coverage *i.e.*, that each query

should be compared against the resource indexes of almost all nodes to ensure a quasi-perfect recall rate⁴.

We show that an unstructured overlay such as CellFarm makes the idea of using process groups to improve search, not only practically viable, but also flexible. In our scheme, peers are not restricted to store content based on the constraints of consistent hashing. Instead, nodes can store any content without restrictions. Then, as suggested in (Kun et al., 2009), nodes perform replication to the members of their process group. However, thanks to the topological properties of CellFarm, given that each Cell is fully connected, replication can be achieved by relying on a *structured* variant of one-hop replication.

3.3.1 Search Strategies

As noted in Chapter 2, there are basically two possible search strategies in unstructured systems: *flooding* and *random-walks*. Furthermore, these basic strategies can have *uninformed* or *informed* variants. In the basic uninformed variant, no additional information is maintained about the (potential) location of resources. The informed variants improve on those techniques by maintaining additional routing information (for instance, by leveraging the results of past queries (Yang & Garcia-Molina, 2002; Tsoumakos & Roussopoulos, 2003)).

Naturally, there are many aspects that affect the performance of informed variants: the data distribution (how many items are rare and how many items are popular), the query distribution (how many queries search for popular items, temporal locality, etc), the query routing strategy, the amount of routing information maintained at each node, and, of course, the overlay topology. Therefore, the degree of success of informed strategies is heavily dependent on the workload characterization.

To illustrate the benefits of the Cell-based topology of CellFarm to improve one-hop replication, we need to decouple our evaluation from the factors above. Therefore, we have opted to use an uninformed strategy, more precisely, flooding, since this technique is oblivious to factors such as data distribution or query workload characterization. As a result, differences in the performance of such strategy on different overlays result exclusively from the topological properties, which is the focus of this contribution.

⁴In this context, we consider that a query has full coverage, if it is compared against at least, 99.9% of all nodes resource indexes.

Therefore, we provide results for two systems that use one-hop replication and flooding. The first, denoted as the *baseline*, makes no use of the notion of process groups when replicating the indexes or performing the queries. The other, denoted *cell-aware*, exploits the fact that processes are organized in groups when performing replication and disseminating queries. Each of these systems is described below with more detail.

3.3.2 Baseline Strategy

In the baseline system, nodes are organized in a flat unstructured network where no notion of process group is provided. Each peer replicates its own resource index to all its direct overlay neighbors. Then, when a query is injected in the system by a given participant, it is sent to all its neighbors. In turn, these neighbors also forward the query to all their overlay neighbors. This process is repeated for a maximum number of times to avoid queries to loop indefinitely in the network; this is controlled by a parameter named *Query Time To Live* or simply QTTL. When a query is first injected in the system it is tagged with an initial QTTL value, which is decremented whenever the query is retransmitted (*i.e.*, forwarded in the overlay). A query that has reached a QTTL value of zero is no longer retransmitted.

Flooding protocols benefit from one-hop replication techniques, as this allows them to avoid the execution of the last forwarding hop, which accounts for a large fraction of the messages generated by these protocols (Chawathe et al., 2003).

Flooding ensures complete coverage without requiring any routing support as long as the QTTL value is equal or greater than the overlay diameter minus one. Unfortunately, it has a very high message cost as many redundant messages are generated. Therefore, practical systems that use flooding often sacrifice complete coverage by configuring QTTL with small values (therefore, limiting the exploration to the vicinity of the query source and limiting the recall rate of the resource location service).

3.3.3 Cell-Aware Strategy

The baseline strategy described above is oblivious to the overlay topology. We now propose an adaptation of this technique that exploit the notion of process groups and the unique properties of the CellFarm topology.

For this, we require nodes in each Cell to maintain a replicated consolidated index of all resources stored by them (which is a form of limited or *structured* one-hop replication in which resource indexes are only replicated to the overlay neighbors of a node which belong to the same Cell). The replication of indexes among nodes of any given Cell can be easily implemented by piggybacking information regarding the local indexes on the anti-entropy mechanism described in Section 3.2.2.5. This approach is simple and yields good results, as local indexes tend to have a comparatively low update rate. It would be possible to design more sophisticated resource index replication strategies, but these are orthogonal to the main focus of the thesis in general and this contribution in particular.

Our Cell-Aware routing strategy allows queries to visit every Cell in the overlay while minimizing the number of nodes of each Cell that are involved in the processing (and ideally the forwarding) of each query, therefore significantly improving the load-distribution of the system. For that purpose we tag queries with the identifiers of nodes (or Cells in the case of CellFarm) which already processed the query (this can be efficiently implemented using Bloom Filters)⁵.

In detail, Cell-Aware flooding operates as follows: When a node receives a query (for the first time) it decreases the QTTL. If the query was received from a neighbor in a different Cell (*i.e.* in that node's eView) the node processes the query, after which it adds its own Cell ID to the bloom filter associated with (that copy of) the query. If the QTTL is above zero, that node forwards the query to all participants in that node's eView that do not belong to Cells already visited by the query. If the QTTL is above one, that node also selects half of the neighbors in its iView (rounded up) to which it also forwards the query.

A node which receives a query from a neighbor in its own Cell (*i.e.* in that node's iView) never processes it, and only forwards it to all neighbors in its own eView that do not belong to Cells already visited by the query. If a node receives a query that was already processed or forwarded by it, it simply drops that message.

Notice that by sending the query to half the elements of the Cell we achieve two purposes: *i)* only half the nodes in the Cell are required to spend resources forwarding that query; and

⁵Evidently, there is not a global bloom filter for each query, each individual copy of a query contains a bloom filter filled the identifiers of nodes (or Cells) that were in the path performed by that specific copy of the query. The bloom filters of different copies of the query diverge each time the query is forwarded over the overlay.

ii) those nodes can store locally the identifier of the query, which allows them to discard any additional copy they might receive in the future. This will limit the amount of system resources that would be consumed for processing redundant copies of queries.

3.4 Evaluation

We have performed an experimental evaluation to assess the topological properties of CellFarm; the benefits of its topology for query routing in the context of the resource location service case study; and the resilience of the overlay to failures. For this purpose we have used a combination of extensive simulation and a prototype deployment over the PlanetLab testbed⁶, as follows:

3.4.1 Experimental Setting

For simulations, the PeerSim simulator (Montresor & Jelasity, 2009) was employed. For this experimental setup implementations of both the CellFarm protocol, as well as an unstructured overlay enriched with one-hop replication of resource indexes stored in individual nodes were developed and tested. All these implementations use the event driven engine of PeerSim. All experiments with PeerSim were conducted in a system composed of 10.000 nodes. PeerSim simulations use a virtual clock that coordinates the delivery of events to nodes (and protocols). The virtual time is measured in time units (TU); each TU is equivalent to 1 millisecond in a real deployment. Message delay was configured to be uniformly distributed between 100 and 1,000 time units (TU) accordingly to typical latencies observed in the PlanetLab testbed.

For the PlanetLab experimental deployment, a prototype of CellFarm in the Java language was developed. The prototype was implemented following the pseudo-code and descriptions presented in this chapter, and required approximately 1700 lines of code. For these experiments 251 PlanetLab nodes scattered throughout the World were used for deploying and asserting the performance of the prototype.

⁶<http://www.planet-lab.org/>

3.4.1.1 Tested Topologies

To provide a better understanding on the advantages and disadvantages of CellFarm, the performance results obtained with CellFarm are compared with those obtained with a flat unstructured overlay. It is relevant to remember the reader that although CellFarm relies on probabilistic unique node identifiers (similar to DHT's), its topology is random in nature (as nodes are not restricted to establish neighboring relations by node identifiers) thus, the comparison with an unstructured overlay is the most adequate. This evaluation section includes comparative results between the CellFarm overlay and an unstructured overlay constructed by an enriched version of the Scamp protocol (Ganesh et al., 2003).

The use of Scamp is motivated by the following observations: *i)* the complete specification of Scamp is published; *ii)* Scamp maintains stable neighboring relations, being adequate to the use of TCP as transport protocol; and *iii)* Scamp produces an overlay which has similar properties to overlays typically used in unstructured resource location systems.

To make the comparison with CellFarm fair, the basic operation of Scamp was adapted to ensure that it could rely on the same mechanism that are employed for maintaining CellFarm. These mechanisms do not change the topological properties of Scamp, but make the signaling cost, required to maintain the overlay, comparable with that of CellFarm. The changes performed are as follows:

- Scamp was modified to use TCP both to provide point-to-point reliable communication as well as to use TCP as an unreliable failure detector, removing the need for the heartbeat messages usually employed by Scamp.

- Scamp was enriched with a global view similar to the one employed by CellFarm, which can be used to recover from failures (by selecting nodes to whom send a Subscription request).

- Scamp was also enriched with a scheme similar to the one employed by CellFarm to manage the contents of its gView.

Both protocols rely on a gView of size 30. In the following, we provide some details about the configuration of these protocols.

CellFarm Configuration The CellFarm parameters that control the Cell size were configured as follows: `CSTARGET` was set to 11; `CSMAX` was set to 16; and `CSMIN` was set to 8. The param-

eter θ was set to 4, so that each node tries to maintain 4 external neighbors. All random walks used by overlay maintenance protocols have a time to live of 10 hops. Additionally the time parameters associated with the periodic tasks of nodes were set as follows: ΔT_1 and ΔT_2 were set to $20s/20.000$ TU, while ΔT_3 was set to $10s/10.000$ TU. The random factor π varies between 0 and $20s/20.000$ TU (Alg. 1 depicts the effect of these parameters in the CellFarm protocol operation).

Scamp Configuration The Scamp protocol was configured with a c value of 3 (this parameter is related to the fault tolerance of the Scamp protocol). This ensures that each node maintains an average number of neighbors which is slightly above to the number maintained by CellFarm. To ensure fairness, the periodic operations of Scamp were configured to use similar time intervals to those employed by CellFarm (presented above).

3.4.1.2 Query Flood Strategy

To measure the effect of the CellFarm topology we have implemented two resource location services that differ on the query routing mechanism employed. One relied on the baseline flood protocol and the other used the cells-aware variant of the query routing algorithm, as described previously.

The fact that peers require some amount of time to process a query is modeled in simulations by introducing a processing delay (PD) after the query is received and before the query is forwarded. This delay is not fixed; instead, it increases linearly with the number of resource indexes replicated by a node. We have set the PD to $0.1s/100$ TU for each (individual peer) index, in both tested scenarios.

The query dissemination protocols were configured to use a QTTL of 5 for both CellFarm and the unstructured overlay. We have experimentally determined that this was the minimum value for the QTTL parameter, which enabled each system to achieve a query coverage equal or above 99.9% (hence, an approximately perfect recall rate for the resource location service).

3.4.1.3 Number of Experiments

All results presented in here represent an average of results extracted from 5 independent executions of each experience. Confidence intervals are omitted from figures for readability, how-

ever these intervals were calculated for a confidence of 95% and they were similar across all experiments and protocols.

3.4.2 Overlay Properties

We start by presenting experimental results for the properties of the CellFarm overlay. We first show how these properties evolve in steady state *i.e.*, where no change to the global membership of the system occurs. We then discuss the properties of the resulting overlay under two dynamic scenarios. The first evaluates the robustness of the overlay in a scenario where large fractions of peers fail simultaneously. The second studies the effects of churn over the overlay topology.

3.4.2.1 Overlay Characterization in Steady State

In here the topological properties of the overlay when CellFarm operates in steady state are evaluated and discussed.

Cell Size Distribution: We start by presenting the resulting distribution of the Cell size in a typical CellFarm overlay. We show results both from simulations and from a PlanetLab deployment. Our goal is to validate the design of CellFarm, and show that simulations effectively capture the protocol behavior in a real World deployment.

Figure 3.2(a) depicts the Cell size distribution obtained in the simulations. As one can notice, the most common Cell size is defined by parameter `CSTARGET`. Also, no Cell has a size below `CSMIN` nor a size above `CSMAX`. This validates the operation of our protocol. Additionally, notice that more Cells have sizes above the target size. This is a desirable property, as one expects that the utility of Cells decreases when their size falls below `CSTARGET`. In fact our architecture strives to ensure that the largest majority of Cells have a size that falls between `CSTARGET` and `CSMAX`.

Figure 3.2(b) presents the distribution of Cell size obtained over the PlanetLab testbed: one can see that all Cell sizes respect the configuration. Additionally, the distribution of Cell size across the system is consistent with the one observed in simulation. These results show that simulations are indeed able to capture the behavior of the protocol in a real deployment, and

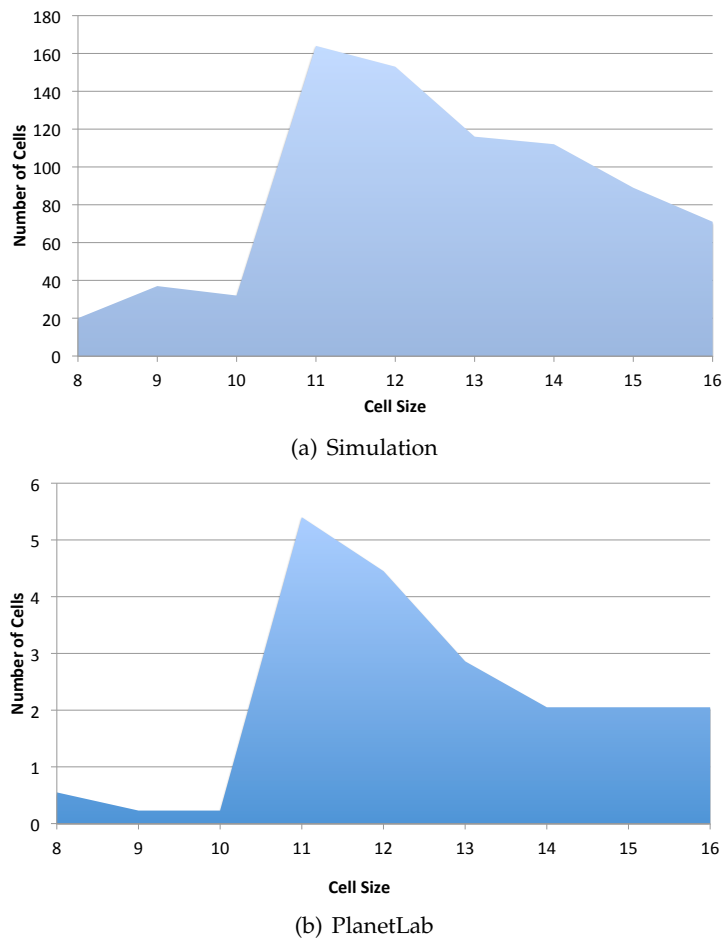


Figure 3.2: Cell size distribution of CellFarm in a steady state scenario obtained through simulation and a prototype deployment over PlanetLab.

also that the solution is viable in real scenarios. Furthermore, we have verified the correctness of all constraints imposed over nodes partial views (iView and eView), for instance in terms of symmetry, and these were verified in both experimental setups, validating the design of the protocol on enforcing all relevant constraints over the neighboring relations established among peers.

	Average clustering coefficient	Average shortest path
CellFarm	0.34	4.25
Unstructured	0.01	3.40

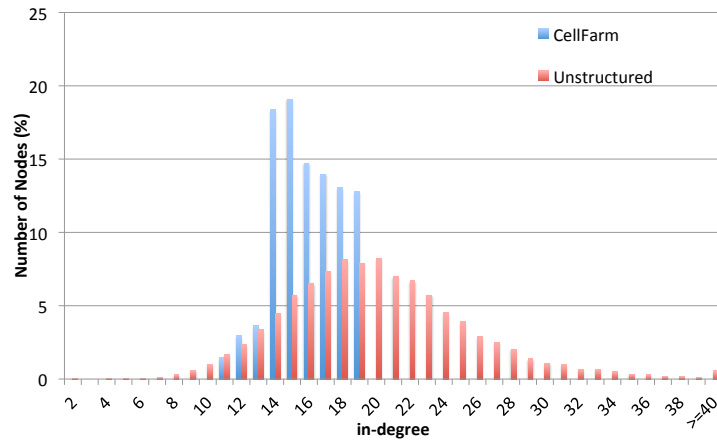
Table 3.1: Comparison of graph properties between CellFarm and an unstructured solution after stabilization.

Clustering Coefficient and Average Shortest Path: Table 3.1 presents the values of two relevant topology metrics discussed previously on Chapter 2, namely the average clustering coefficient and average shortest path.

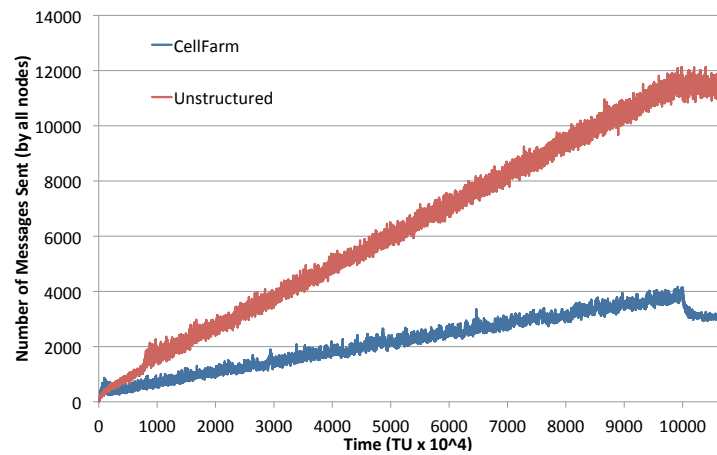
As expected, given that a Cell is a cluster of nodes, the clustering coefficient of CellFarm is larger than that of the unstructured overlay network. Still, the fact that CellFarm promotes the diversity of links connecting different Cells mitigates the effect of the increase over the clustering coefficient and the average shortest path (ASP). As a result, the ASP value for CellFarm is only marginally above of the unstructured topology. This is a relevant aspect, not only for the correct operation of CellFarm, but also to ensure an efficient routing of queries in the case study considered in this chapter, as it will be shown further ahead in the text.

In-Degree: Figure 3.3(a) presents the in-degree distribution of peers in both systems. Remember that the in-degree is a relevant metric as it provides a measure of the reachability of a node in an overlay network. In the context of the case study, this metric also captures the probability of a peer being required to contribute to the dissemination and processing of queries. Ideally, all nodes in the system would have the same in-degree value, as this would ensure a more uniform load distribution. The figure shows that CellFarm is the algorithm that presents a more balanced in-degree distribution. In fact, all nodes in CellFarm have an in-degree that varies between 11 and 19. In sharp contrast, the unstructured system exhibits in-degree values that vary between 2 and 57. This indicates that CellFarm design promotes a better load distribution across peers when compared with a pure unstructured approach where no control technique is employed to impose soft constraints over the neighboring relations of nodes.

Message Cost: Figure 3.3(b) depicts the total number of messages exchanged by peers during simulations for building and maintaining each of the considered overlays. There are two distinct phases of the protocols represented in the figure. In the first phase of the simulation (up to $10.000 * 10^4$ TU) nodes are joining the overlay. After that point the system operates in steady state. The figure shows that when nodes are still joining the system, both systems show an increase in the global communication cost required for maintain the topology of the overlay. Naturally, in steady state this overhead drops slightly. In both cases, CellFarm presents a much lower communication cost when compared with the unstructured alternative. This is mostly due to two factors: *i*) CellFarm employs a reactive management scheme, where nodes



(a) In-degree distribution



(b) Message cost

Figure 3.3: Overlay comparison of in-degree distribution and cost for maintaining the topology between CellFarm and an unstructured overlay managed by an improved version of the Scamp protocol.

only execute some maintenance operations when some external event (such as a node joining or leaving) occurs; whereas Scamp requires peers to rejoin the system periodically in order to deal with node departures and rebalancing the in-degree of participants; and *ii*) CellFarm management in steady state is based on an anti-entropy mechanism that can be configured to operate with low overhead.

3.4.2.2 Fault-Tolerance

The simulation test setup was employed to evaluate the effect of simultaneous node failures in the overlay topology for each system (following a similar methodology to the one employed

in (Ganesh et al., 2003)). To this end, the overlays were initialized as described previously and subsequently, failures (*i.e.*, crashes) were induced over a fraction of existing peer simultaneously, ranging from 10% to 70% node failures. Immediately after the failures, the percentage of correct peers in the largest connected component was measured. Afterwards, in the cases where the overlay connectivity was lost, we measured the number of simulation cycles required for the recovery mechanisms of each particular overlay to recover the global connectivity of the overlay topology.

Additionally, the impact of these failures over the in-degree distribution of each overlay, as well as the Cell size distribution for the particular case of CellFarm, was verified. This verification was performed after each overlay was able to reconfigure itself to recover its connectivity.

We assume that in such a system, more overloaded nodes have an increased probability of failing. Therefore we select nodes which have a larger number of neighbors, since such nodes are required to use additional bandwidth and computational resources to support the operation of the system.

Size of Largest Connected Component: Figure 3.4(a) depicts the percentage of correct peers in the largest connected component of the overlay, for each tested overlay, immediately after node failures. Notice that although both systems are able to sustain a significant amount of concurrent failures, the unstructured solution connectivity is lost when more than 40% of nodes fail concurrently (these results are consistent with results published in (Ganesh et al., 2003)). CellFarm however, is able to sustain up to 60% of concurrent node failures. This happens because CellFarm management is based on reactive mechanisms, and highly robust gossip-based mechanisms. These also justifies the results depicted in Figure 3.4(b), which show the recovery times of both solutions (*i.e.*, the time required to reconfigure the overlay topology). Notice that, for failures of 70%, the unstructured solution is unable to regain global connectivity.

In-Degree Distribution and Clustering Coefficient: We have also observed the effects of simultaneous failures in the in-degree distribution for both overlays, as well as the Cell size distribution of CellFarm. CellFarm is able to reconfigure itself in order to keep Cell size and in-degree distribution similar to those exhibited in steady state. The same is not true for the unstructured system. The unstructured overlay exhibits a generalized drop in the in-degree of the surviving nodes. This happens because the failures leave the overlay connected, even if

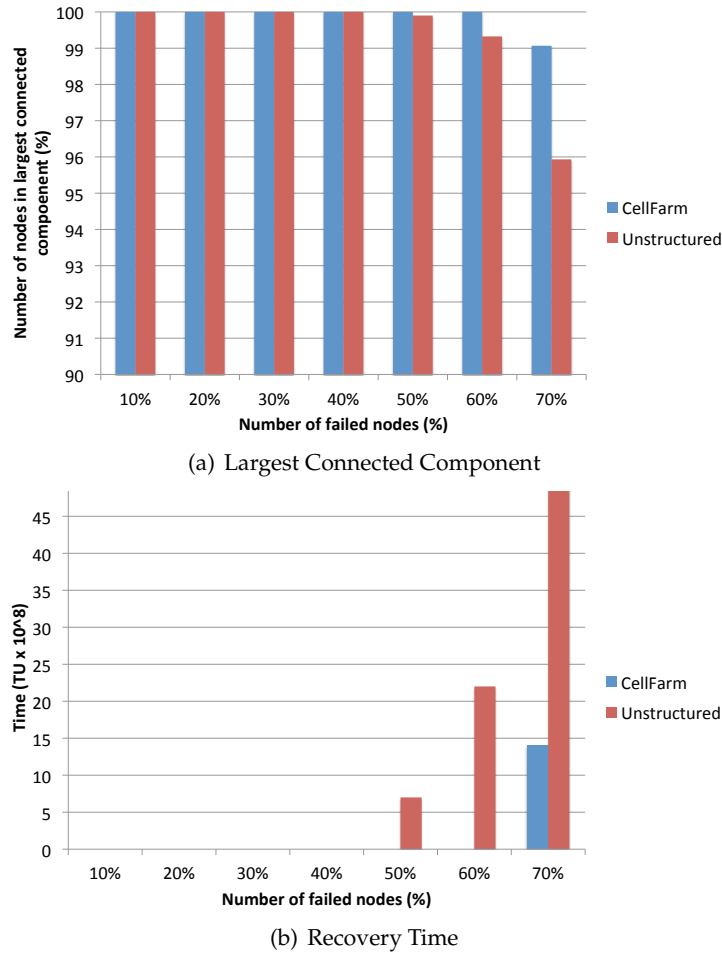


Figure 3.4: Comparison of Overlay Connectivity and Recovery Time after massive node failures between CellFarm and an unstructured overlay based on an improved version of Scamp.

by few links. Therefore, only a few nodes are forced to reexecute the join procedure. As a result, the loss of redundant links is not compensated during recovery, resulting in a post-failure overlay with reduced connectivity and increased overlay diameter.

3.4.2.3 Churn Scenario

Any overlay designed to operate in a large scale environment (*e.g.* over the Internet) should ensure that its connectivity is not hampered in the presence of churn.

To illustrate the robustness of CellFarm to churn in Figure 3.5 we depict the size of the largest component of both overlays after a period of churn. Churn was induced, after the stabilization of the overlays, in the system for a period of $1 * 10^6 TU$ where every $2 * 10^4 TU$ a

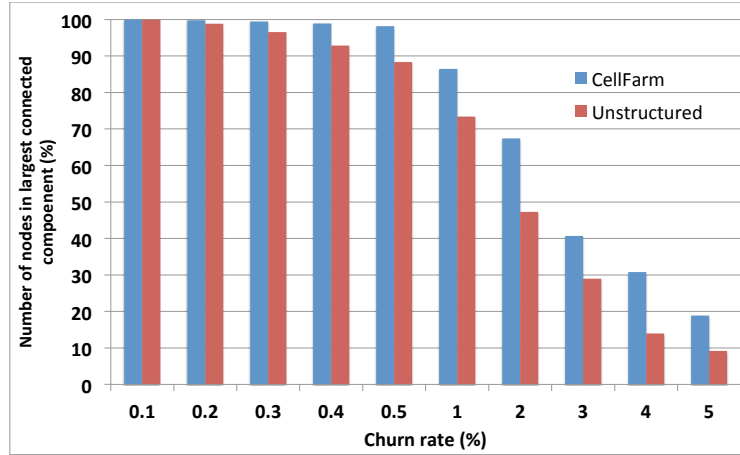


Figure 3.5: Overlay Connectivity evolution under different churn rates for CellFarm and an unstructured overlay based on an improved version of Scamp.

percentage c of nodes in the system concurrently left and were replaced by a similar number of new nodes. After this we left the overlay stabilize for $1 * 10^6 TU$ and extracted the presented results. We took into consideration different churn rates (*i.e.* c) in our experiences ranging from 0.1% to 5%. We took into consideration such aggressive churn rates to really stress the overlay topologies. Results presented in Figure 3.5 clearly shows that CellFarm is much more robust to churn. This is due to the reactive nature of the algorithm which maintains the CellFarm topology.

3.4.3 CellFarm Support for Resource Location

In this section the performance of the P2P resource location service using the previously discussed flood-based query routing mechanisms is presented and discussed. The main concern of this experimental work is to evaluate the scalability of each considered architecture since the overload of nodes, due to the injection of multiple concurrent queries, may cause the disruption of the system due to resource exhaustion. Additionally, the load distribution achieved by each solution is also evaluated. All these experiments were performed through simulations, as this allowed to measure the effects of CellFarm over a much larger number of peers.

We simulate the behavior of both systems when a single query is injected in the system,

and also when 100 queries are injected (by random nodes) concurrently. We then measured the time required by queries to be disseminated through out the system until they achieve a full coverage. Queries were only injected in each system after the stabilization of the respective overlay topologies.

Results are summarized in Table 3.2. For the case where a single query is injected, one can observe that CellFarm outperforms the unstructured alternative slightly. This happens because CellFarm uses a structured one-hop replication scheme by leveraging the Cell based topology which exports to the service a notion of process groups. This allows each peer to avoid processing redundant queries *i.e.*, queries that were already processed by peers, which collectively, already replicate all resource indexes present at the local peer consolidated index.

Although the latency gain for a single query is small, when the load in the system increases the benefits become much more notorious. When 100 concurrent queries are injected in the system, CellFarm latency for disseminating all queries is only 33% of the value exhibited by the unstructured solution. In fact, the unstructured solution spends a significative amount of resources in processing redundant queries, which delays the dissemination of queries. In sharp contrast, CellFarm empowers the cell-aware routing strategy to distribute the load among nodes, avoiding nodes to become resource exhausted.

Table 3.3 depicts the global message cost of disseminating a single query in each system, and the maximum number of messages sent by a single node to route 100 simultaneous queries using the flood routing strategies. As expected the message cost in the unstructured topology is higher than that of CellFarm. This happens because the structured one-hop replication strategy used in the CellFarm allows to avoid the transmission of a significative number of redundant messages.

Notice that, the CellFarm topology offers the potential to perform query dissemination (and processing) with better load distribution. This is illustrated by the maximum number of

	Average latency for a single query (TU)	Maximum Latency for 100 queries (TU)
CellFarm	8,163.0	55,220.0
Unstructured	10,037.0	147,016.84

Table 3.2: Query dissemination latency obtained when using CellFarm and an unstructured overlay based on an optimized version of Scamp.

messages that a single node is required to send while 100 independent queries are issued in parallel. Table 3.3 shows that CellFarm has the best load distribution, which is highlighted by the lower number of messages that a single node is required to forward when compared with the unstructured alternative (nearly half). This lowers the chance that nodes become overloaded, and consequently fail, disrupting the operation of the P2P system.

3.5 Related Work

As stated previously, CellFarm provides an unique topology where nodes self-organize into process groups formed by cliques of peers. Some recent works have also explored similar topologies in different contexts. In Sahota et al. (2009) the authors propose PIndex, a grouped peer-to-peer network that is used to support scalable grid information services. Similarly to CellFarm, PIndex organizes nodes in fully connected groups of peers, interconnected among themselves, that are used for supporting load balancing, and to improve fault and churn resilience. Unfortunately, the topology is built and maintained using a centralized approach.

In the work by Liu et al. (2010) the authors present MN-Tree, a fault-tolerant tree-based overlay, where each vertex in the tree is materialized by a fully connected cluster of nodes similar to those provided by CellFarm. These clusters of nodes allow the tree to be resilient to node failures. However, unlike our approach, clusters of nodes are built using a distributed protocol that is tightly coupled with the tree topology, requiring nodes to maintain information concerning all the nodes below it in the tree and all peers located between its position in the tree and the root, limiting the scalability of this solution.

GBIR (Kun et al., 2009) uses process groups to improve search. Several shortcomings of GBIR have already been addressed earlier in this chapter. Additional limitations are as follows. First, the number of groups is defined *a priori*, and its parametrization requires an accurate

	Total messages sent for a single query	maximum messages sent by a single node for 100 queries
CellFarm	10,440.40	44.2
Unstructured	49013,40	80.4

Table 3.3: Query dissemination communication cost of the resource location systems leveraging CellFarm and an unstructured overlay based on an optimized version of Scamp.

estimate of the system membership and size; in our approach, the number of groups adapts automatically to changes in the system size. Second, the approach makes no effort to maintain groups close to a target size (the group size depends on the properties of the hash function and on node identifiers distribution); unless the number of groups is relatively low, the variability in their size may be significantly larger. Finally, their replication strategy assumes that nodes are likely to have members from most of the groups among their neighbors. This assumption further restricts their approach to a strategy where there are few and very large replication groups; this restriction does not exist in CellFarm.

Scatter (Glendenning et al., 2011) is an efficient and resilient DHT where each DHT node is materialized through a fully connected group of nodes. Scatter topology is used for supporting data replication, relying on Paxos (Lamport, 1998) for maintaining a DHT ring composed of strongly consistent replication groups. Contrary to the contribution presented here, Scatter heavily relies on consensus to implement every operation in the system including the management of the overlay topology, therefore it has higher signaling costs and may block under heavy churn. In sharp contrast, CellFarm relies on a set of micro-protocols based on random walks and robust gossip-based anti-entropy mechanisms, which allows the overlay to retain its relevant properties independently of membership dynamics.

None of these previous works present a solution that builds an overlay network with the properties of CellFarm in a fully decentralized and self-organizing manner, by imposing soft constraints over the topology of unstructured overlay networks. Additionally, there is no work in the literature that presents a flood-based query routing protocol for a P2P resource location service that is able to exhibit the properties of the one introduced here from the point of view of communication cost and load distribution, while ensuring complete convergence of the system. To the best of our knowledge there is no other distributed protocol that allows to build an overlay network with the unique properties exhibited by CellFarm.

In the context of unstructured resource location, Gia (Chawathe et al., 2003) is a system based on a flat overlay that adapts the topology according to the node capacity. CellFarm is not driven by any specific node characteristic. Gia enforces index replication to all one-hop neighbors. Similar to Gia, the resource location approach proposed here for operating on top of CellFarm also replicates indexes to a subset of one-hop neighbors, the *iView*. However, unlike Gia, there is a tighter control on the number of nodes that replicate each peer resources

index; additionally, we leverage the cell-based topology of CellFarm to make routing decisions, allowing to lower the load imposed on nodes to process a single query without hampering the query coverage, and consequently the recall rate of the resource location service.

Some query routing protocols, such as the ones proposed by Tsoumakos e Roussopoulos (2003), route queries using biased random walks that rely on additional information provided by the system, to increase the probability of locating resources that match a given query. In the work of Broder e Mitzenmacher (2004) the authors suggest the use of bloom filters to provide information for biasing query routing in P2P Systems. These techniques are complementary to, and can be combined with, the algorithms designed for the particular case study discussed in this chapter. In particular, such approach can be adapted to devise new query routing strategies that can exploit the unique topology of CellFarm.

3.6 Discussion

CellFarm is a protocol that builds and manages an unstructured overlay network, that contrary to most existing examples found in the literature, includes mechanisms to enforce a set of constraints over the neighboring relations of peers. These constraints allowed the overlay topology to have topological properties, in particular having peers self organizing in cliques which are highly connected among themselves.

This was achieved without sacrificing the *plasticity* of the overlay topology (unlike what most DHTs effectively do). This means that the topology can be adapted to cope with the dynamics of a large-scale environment, namely to scenarios where churn occurs, as peers have a high degree of freedom when selecting their overlay neighbors.

The results presented previously support the observation that following this control technique to manage the topology of unstructured overlays, allows to build overlays that have a low maintenance cost, while exhibiting a high fault-tolerance and exporting topological properties that can ease the design and boost the performance of P2P services executed on top of them. In the particular case of the contribution presented in this chapter, we have demonstrated this by evaluating the performance of a P2P resource location service which exploits the topology of CellFarm.

Interestingly, in order to leverage the topology of CellFarm the resource location service

requires to be aware of the topological properties of the underlying overlay. This is not surprisingly as the operation of the service has to be adapted to cope with the existence of Cells to support the structured one-hop replication mechanism, as well as to make local decisions concerning processing of a query by taking into consideration the overlay link through which that query reached the local node.

Summary

In this chapter the CellFarm overlay was presented and evaluated. CellFarm is a protocol that builds and maintains an overlay network where nodes self-organize in fully connected clusters of nodes, which are highly connected among themselves. Such process groups, or Cells, can be used as a building block to support efficient replication and load-distribution in large scale systems, particularly those with dynamic memberships. CellFarm illustrates the benefits that can be achieved by exploring a control technique to manage the topology of unstructured overlay networks at the overlay layer. In practice, the unique properties of CellFarm were achieved by imposing soft constraints over the neighboring relations that peers can establish while still allowing the topology to be flexible enough to deal with churn scenarios.

As a case study, it was demonstrated how the unique topology of CellFarm can be leveraged to improve the operation of a P2P resource location service based on one-hop replication. We proposed a Cells-aware routing strategy that, by taking into consideration the structured one-hop replication made available by CellFarm, can lower the communication overhead of these systems while providing superior load distribution among peers.

In the next chapter we present X-BOT, a protocol that is able to improve the topology of unstructured overlay networks by exploring a bias technique which also operates at the overlay network layer.

Publications

A preliminary version of the contribution presented in this chapter was made available to the research community through the following technical report:

Overnesia: a Robust Overlay Network for Virtual Super-Peers. João Leitão and Luís Rodrigues. Technical Report 36/2009, INESC-ID, July 2009 (Available in: <http://www.>

inesc-id.pt/ficheiros/publicacoes/5510.pdf).

Bias the Topology: X-BOT

This chapter introduces and evaluates *X-BOT*, a protocol that exploits the bias approach for managing the topology of unstructured overlay networks at the overlay layer.

We start by providing a motivation for this contribution and defining its goals in Section 4.1. Section 4.2 describes the *X-BOT* algorithm. Section 4.3 discusses the different Oracles that can be used with *X-BOT* and Section 4.4 discusses the protocol configuration. An evaluation of *X-BOT* is presented in Section 4.5. Some relevant properties of *X-BOT* are discussed in Section 4.6 and finally, Section 4.7 compares *X-BOT* with other approaches.

4.1 Motivation and Goals

4.1.1 Motivation

As discussed previously in the thesis, gossip protocols often rely on a *peer sampling service* (Jelasity et al., 2004), which is a membership protocol which provides locally, to each peer, a small subset (called a *partial view*) of the full membership list; in this case, participants in the system use their local partial views to select peers with whom they exchange messages. These partial views encode the unstructured overlay network over which messages can be disseminated using a gossip-based protocol. Members of the partial view may be selected at random from the entire membership; the random nature of unstructured overlay fit well with the operation of gossip-based protocols. However, the random selection of peers may lead to scenarios where many of the overlay links are suboptimal, for instance, with regard to a given efficiency criteria, such as network bandwidth or latency. Unfortunately, the inefficiency of the overlay has a direct negative impact in the performance of P2P services that operate on top of the unstructured overlay (as a case study we will focus on the particular case of gossip-based broadcast services in the context of the contribution presented in this chapter). Moreover, P2P services may benefit from having the ability to bias the underlying overlay network topology to

match some high level specific criteria. For instance, in file sharing services, one may organize the overlay such that neighbors share similar content. Overlay efficiency has been recognized as a key research topic for gossip-based protocols (Birman, 2007).

4.1.2 Goals

From the analysis of relevant unstructured overlay network properties and observation of previous approaches (which will be addressed in detail in Section 4.7), we define a set of guidelines that should be respected when biasing the topology of these overlays.

Limited Number of Peers Each adaptation to the overlay topology should involve only a small limited number of participants. This is a requisite to ensure the scalability of the biasing mechanism and to avoid excessive coordination overhead.

Limited Information The biasing of the overlay topology should only rely in localized information. This is required to lower the overhead of the bias process and reduce the communication among participants implicitly improving the scalability of the solution. Additionally, the existence of inaccurate information should not completely disrupt the operation of the biasing mechanism.

Maintain Node Degree To avoid an increasing risk of overlay disconnection, the degree of peers that participate in the biasing of the overlay should remain constant. If this requisite is not met, the execution of multiple adaptation steps may compromise the overlay connectivity.

Maintain Unbiased Links To preserve the network diameter and prevent clustering from increasing, a portion of the overlay links should be kept without being biased. This ensures that some level of randomness remains in the overlay topology.

Considering the identified guide-lines, we have designed *X-BOT*, a new protocol to **B**ias the **O**verlay **T**opology according to some target efficiency criteria **X**, for instance, to better match the topology of the underlying network. *X-BOT* is completely decentralized and explores a solution for managing the topology of unstructured overlay networks that operates at the overlay layer by leveraging the *bias* technique.

Additionally, and unlike prior works, *X-BOT* owns the following set of relevant characteristics: *i*) it operates only with local information and it does not require peers to have *a priori* knowledge about their target location on the target topology; *ii*) it employs a new coordinated 4-node optimization technique that allows to achieve better overlay configurations; *iii*) the protocol strives to preserve the degree of peers that participate in an optimization step, which is fundamental to preserve the connectivity of the overlay; *iv*) every modification performed to the overlay increases its efficiency; this is feasible due to the dynamic nature of our model, which avoids the overlay from stabilizing in local minima; *v*) the optimization performed by *X-BOT* preserves several key properties of the overlay such as a low clustering coefficient and low overlay diameter; *vi*) our scheme is highly flexible, as we rely on a companion *oracle* to estimate the link cost and, therefore, our algorithm can bias the network according to different efficiency criteria that are encoded in specific cost metrics, just by employing the appropriate oracle.

4.2 The *X-BOT* Protocol

This section provides the full description of the *X-BOT* protocol. It starts by presenting a rationale for the design of the protocol, followed by the description of the *X-BOT* algorithm complemented by pseudo-code.

4.2.1 Rationale

The architecture of *X-BOT* is inspired by the HyParView protocol¹ (Leitão et al., 2007b). Unlike previous protocols for building and maintaining unstructured overlay networks, HyParView relies on two distinct partial views (*i.e.*, neighbor sets) which are maintained using different strategies and for distinct purposes (in fact, the protocol is said to be Hybrid because it combines these two different strategies). Considering this, *X-BOT* used the following architecture.

A small symmetric *active view* with $(fanout+1)$ size is used to support communications among participants (*e.g.*, disseminate broadcast messages among peers), therefore this is the partial view which materializes the unstructured overlay network exposed to P2P services above. The active view is maintained using a reactive strategy, which means it is only updated

¹For *Hybrid Partial View*.

in response to some external event that affects the overlay (e.g. a node joining or leaving). Each node maintains a TCP connection to each neighbor in its active view. The use of TCP allows the selection of a small fanout for disseminating messages through gossip-based protocols, as it masks network omissions. Moreover, TCP is used as an unreliable failure detector, which facilitates the implementation of the reactive maintenance strategy. The symmetry of these views helps to preserve the overlay connectivity, by providing each peer with direct control over its own in-degree. In the particular case of *X-BOT*, the partial view is the target of the biasing mechanism, as we detail further ahead.

Each peer also maintains a larger *passive view*, k times larger than the active view, whereas k is a constant related with the desired level of fault tolerance. The passive view is maintained by a cyclic strategy, meaning that periodically each peer performs a shuffle operation with one random peer in the overlay that results in an update of both participants passive views. This mechanism operates in a similar fashion to one employed for managing *gView* of *CellFarm* discussed in the previous chapter. This partial view is used for fault tolerance, as it works like a backup list of peers that ensure a constant out-degree for nodes.

Additionally, we assume that all peers have access to a local *oracle*. Oracles are components that export a `getLinkCost (Peer p)` interface, which returns the link cost between the invoking peer and the given target node p in the system (since there is a single link to each neighbor, the terminology link cost and peer cost are used interchangeably when referring to the output of the oracle). In section 4.3 additional details and some examples of oracle implementations are discussed.

Although *X-BOT* maintains an active view and a passive view similar to the *HyParView* protocol, *X-BOT* relaxes stability in order to be able to continuously and iteratively attempt to improve the overlay topology according to some efficiency metric embedded in the companion oracle. This allows the topology of the unstructured overlay to self adapt in order to better match the requirements of P2P services, executed on top of it. Periodically, each node executes a biasing procedure, in which it has the opportunity to start one, or more, *optimization rounds*. In an optimization round a peer attempts to swap one member of its active view with one neighbor extracted from its passive view, effectively replacing a previously existing overlay link, by a new and *better* link. While executing the biasing procedure, a peer uses its local oracle to obtain an estimate of the link cost to some randomly selected peers of its passive

view. The number of nodes scanned in each execution of the biasing procedure is a protocol parameter called *Passive Scan Length* and simply denoted π . Implicitly, this parameter limits the maximum number of optimization rounds triggered by a peer in a single execution of the biasing procedure.

The passive view is not biased. However, the passive view should be continuously updated during the system operation, so that it reflects the changes in the global membership (e.g. peers that leave the system, are eventually purged from all passive views, and peers that join the system eventually appear in some of the passive views). Therefore, passive views are a source of peers that can potentially be upgraded to the active view, to effectively bias the overlay to a better topology. This also assists in avoiding our algorithm from falling into local minima configurations, as each time a peer executes the biasing procedure it may sample different candidates.

X-BOT strives to preserve the connectivity of the overlay. This has two implications in the operation of the protocol: *i*) peers only make an effort to optimize their active views when they have a full active view (i.e., no bias is applied to active views until maximal connectivity is ensured). Furthermore, each peer attempts to maintain some unbiased neighbors, as explained ahead; *ii*) the protocol strives to preserve the degree of nodes that participate in an optimization procedure. Moreover, we ensure that despite the biasing of the overlay topology, the symmetry of the active views is also preserved, which is instrumental to ensure a good distribution of in-degree, which in turn has a significant impact on the connectivity of the overlay. Typically, each optimization round involves 4 peers in the system.

Unbiased Neighbors By eagerly imposing a bias on the topology of the overlay, one may easily break some of the key desirable properties of a random overlay, such as the low clustering coefficient, low average path length, or connectivity (Tang & Ward, 2005). The negative effect of such bias can be even more notorious in an architecture such as the one employed in *X-BOT*, that relies on small active views. To avoid such pitfall, we do not bias all members of active views. Instead, each peer maintains both “high-cost” (unbiased) and “low-cost” (biased) neighbors. The number of “high-cost” neighbors kept by each peer is a protocol parameter named *Unbiased Neighbors* and simply denoted μ .

Unfortunately, it is not trivial to decide which peers have a “high-cost,” given that peers are not required to have global knowledge of the system, not only regarding membership in-

formation but also regarding global metrics, such as the overlay cost or average link cost. To circumvent this problem, active views are maintained sorted by link cost (the first element of each active view is the neighbor with the largest link cost). A node never attempts to bias the first μ members of its active view (meaning that the μ higher cost overlay links of a peer are protected from the biasing process).

Notice that this scheme implicitly imposes a passive biasing to unbiased neighbors, as the unbiased neighbors kept by each peer in *X-BOT* are the elements of that peer's active view that present the higher cost accordingly to the local oracle. We say that this is a passive biasing process because peers do not actively exchange their unbiased neighbors to promote more distant neighbors.

4.2.2 Algorithm

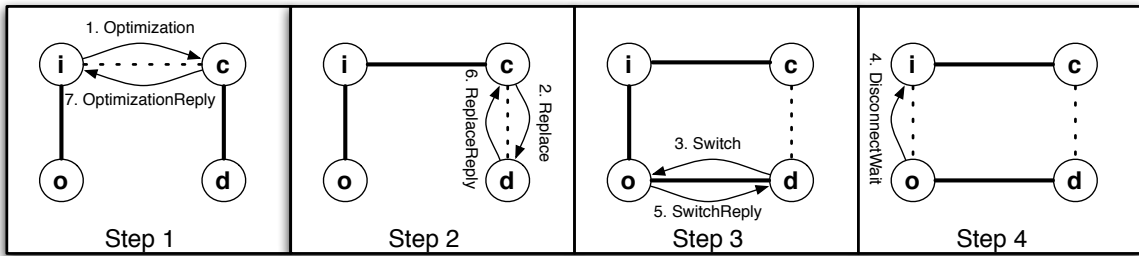


Figure 4.1: Typical *X-BOT* optimization round involving 4 peers in the unstructured overlay network.

The *X-BOT* algorithm is presented in *Algorithm 7*. The operation of a typical optimization round is illustrated in Figure 4.1. Notice that the pseudo-code of the algorithm has been simplified for clarity. For instance, insertions of peers into passive views and the mechanisms required to ensure the symmetry of active views have been omitted from the listing. A typical optimization round involves 4 peers, and each round is composed of 4 steps, executed by each peer participating in the optimization. We use the following notation to identify each of the participating nodes. Node *i* (**initiator**): is the peer that starts the optimization round. Node *o* (**old**): is a peer from *i*'s active view which is replaced during the optimization process. Node *c* (**candidate**): is a peer from *i*'s passive view which is upgraded to *i*'s active view during the course of the optimization round. Node *d* (**disconnected**): is the peer that is removed from

Algorithm 7: X-BOT: Main algorithm in pseudo-code

```

1:  every  $\Delta T$  do
2:    if isFull(activeView) then
3:      candidates  $\leftarrow$  randomSample(passiveView, PSL)
4:      for  $i := UN ; i < \text{sizeof}(\text{activeView}) ; i := i + 1$ 
5:         $o \leftarrow \text{activeView}[i]$ 
6:        while candidates  $\neq \{\}$  do
7:           $c \leftarrow \text{removeFirst}(\text{candidates})$ 
8:          if isBetter( $o, c$ ) then
9:            Send(OPTIMIZATION,  $c, o, \text{myself}$ )
10:         break

11: upon Receive(OPTIMIZATIONREPLY, answer,  $o, d, c$ ) do
12:   if answer then
13:     if  $o \in \text{activeView}$  do
14:       if  $d \neq \text{null}$  then
15:         Send(DISCONNECTWAIT,  $o, \text{myself}$ )
16:       else
17:         Send(DISCONNECT,  $o, \text{myself}$ )
18:         activeView  $\leftarrow \text{activeView} \setminus \{o\}$ 
19:         passiveView  $\leftarrow \text{passiveView} \setminus \{c\}$ 
20:         activeView  $\leftarrow \text{activeView} \cup \{c\}$ 

21: upon Receive(OPTIMIZATION,  $o, \text{peer}$ ) do
22:   if  $\neg \text{isFull}(\text{activeView})$  then
23:     activeView  $\leftarrow \text{activeView} \cup \{\text{peer}\}$ 
24:     Send(OPTIMIZATIONREPLY, true,  $o, \text{null}, \text{myself}$ )
25:   else
26:      $d \leftarrow \text{activeView}[UNOPT]$ 
27:     Send(REPLACE,  $d, o, \text{peer}, \text{myself}$ )

28: upon Receive(REPLACEREPLY, answer,  $i, o, d$ ) do
29:   if answer then
30:     activeView  $\leftarrow \text{activeView} \setminus \{d\}$ 
31:     activeView  $\leftarrow \text{activeView} \cup \{i\}$ 
32:     Send(OPTIMIZATIONREPLY, answer,  $o, d, \text{myself}$ )

33: upon Receive(REPLACE,  $o, i, c$ ) do
34:   if  $\neg \text{isBetter}(\text{peer}, o)$  then
35:     Send(REPLACEREPLY,  $c, \text{false}, i, o, \text{myself}$ )
36:   else
37:     Send(SWITCH,  $o, i, c, \text{myself}$ )

38: upon Receive(SWITCHREPLY, answer,  $i, c, o$ ) do
39:   if answer then
40:     activeView  $\leftarrow \text{activeView} \setminus \{c\}$ 
41:     activeView  $\leftarrow \text{activeView} \cup \{o\}$ 
42:     Send(REPLACEREPLY, answer,  $i, \text{myself}$ )

43: upon Receive(SWITCH,  $i, c, d$ ) do
44:   if  $i \in \text{activeView}$  or received(DISCONNECTWAIT from  $i$ ) then
45:     Send(DISCONNECTWAIT,  $i, \text{myself}$ )
46:     activeView  $\leftarrow \text{activeView} \setminus \{i\}$ 
47:     activeView  $\leftarrow \text{activeView} \cup \{d\}$ 
48:     Send(SWITCHREPLY, answer,  $di, c, \text{myself}$ )

49: isBetter( $old, new$ )
50:   return Oracle.getCost( $old$ ) > Oracle.getCost( $new$ )

```

the candidate's active view in order to accept i in it while preserving the node degree for all 4 participating peers.

We now describe in detail each of the 4 steps that compose an optimization round.

4.2.2.1 Step 1

Step 1 is executed by node i (Algorithm 7, lines 1 – 20) and its purpose is to contact one, or more, potential candidates to participate in a set of optimization rounds².

This step starts with the random selection of, at most, π nodes from the i 's passive view. This random sample defines a set of candidates for executing optimization rounds. To check if a target node is a suitable candidate, i iterates over its active view, consulting the oracle to compare the cost of its neighbors with the cost of the target (Algorithm 7, lines 49 – 50). When a suitable candidate c is found, which presents a possibility for improving a given neighbor o , i sends to c an OPTIMIZATION message, stating its interest in exchanging o for c in its own active view. The reception of that message will trigger the execution of Step 2 at node c .

This step ends with the reception of an OPTIMIZATIONREPLY message from node c (Algorithm 7, lines 11 – 20), or with the suspicion that c has failed (*i.e.*, crashed). If node c accepts the exchange, then i will add c to the active view. If o is still in its active view,³ i will send a DISCONNECTWAIT or DISCONNECT message to o . The difference between these messages is simple: DISCONNECT, only removes the sender from the active view (as described in previous work (Leitão et al., 2007b)) while DISCONNECTWAIT also notifies node o that it should maintain (until an internal timeout expires) that free slot in the active view, which will be used in step 4 to establish a new overlay link between node o and node d . Node i chooses which message to send, based on information received from c , specifically, if c had to remove some node from its active view in order to insert i in its active view.

4.2.2.2 Step 2

Step 2 is initiated by node c with the reception of an OPTIMIZATION message from node i (Algorithm 7, lines 21 – 27) and ends when c replies to i with an OPTIMIZATIONREPLY message.

If c does not have a full active view, it immediately replies to i by sending an OPTIMIZATIONREPLY message accepting the exchange, and notifying i that no other node was involved

²More than an optimization round might be triggered in the context of this step as discussed previously.

³Note that o might have already disconnected from i as a result of the execution of step 4 as described ahead.

in the optimization. In this case i will disconnect itself from o and insert c in its active view. Note that in this particular scenario, our algorithm does not preserve the degree of node o , although it preserves the number of links in the overlay. However, this is an uncommon scenario given that, according to our experiments, in steady state usually more than 97% of nodes in the system have full active views. On the other hand, if c has a full active view, c has to select some neighbor d from its active view to exchange for i . Therefore c sends to d a REPLACE message, stating its desire to remove d from its active view; this message also indicates to d that it can connect to o in exchange. The REPLACE message also carries information concerning the identification of the initiator of the optimization procedure. In order to decrease the average link cost, the selection of d is deterministic, in such a way that d is the neighbor of c with the higher cost (excluding, naturally, the first μ protected members that materialize the overlay unbiased links).

In the latter case, to conclude this step, c has to receive a REPLACEREPLY message from node d (Algorithm 7, lines 28 – 32) or suspect that d has failed (in which case, node c acts as if it had a free slot in the active view from the beginning of this step). If d accepts the exchange, c will remove d from its active view and replace it with i . If d declines the exchange, c does not update its own active view. In either case, c will notify i of d 's answer using the OPTIMIZATIONREPLY message.

4.2.2.3 Step 3

This step begins with the reception by node d of a REPLACE message (Algorithm 7, lines 33–37) and ends when a REPLACEREPLY message is sent back to node c .

A REPLACE message explicitly requests node d to exchange node c with node o in its active view. Node d consults its local oracle to assess if o has a lower link cost than c . Note that our algorithm only requires 2 of the 4 involved peers in an optimization round to consult the oracle in order to assess the merit of the proposed peer exchange. This is enough as we assume that link costs are approximately symmetric (in Section 4.5 we evaluate the performance of the protocol in a realistic scenario were this property is not guaranteed) and effectively, we are exchanging 2 existing links in the overlay, for 2 new links. We exchange the link between i and o for a link between i and c and the link between d and c with the link between d and o . Therefore, only nodes i and d need to assess the gain resulting from these exchanges. The conservative

use of the oracle provides a benefit if the oracle implementation adds some overhead (e.g. due to the possible need of exchanging messages among oracles) whenever it is consulted by a peer.

Naturally, if node d verifies that there is no gain in the exchange of c for o , d aborts the exchange by notifying c . Otherwise, it will send a SWITCH message to node o notifying him to switch node i in its active view for himself. Moreover, it notifies node o . Finally, the answer received from o in a SWITCHREPLY (Algorithm 7, lines 38 – 42) is forwarded to c .

4.2.2.4 Step 4

This step is executed by node o upon the reception of a SWITCH message (Algorithm 7, lines 43 – 48) and ends when o sends a SWITCHREPLY to d . This step is required only to ensure the symmetry of active views. The behavior of node o in this step is deterministic. For clarity, in Algorithm 7 we only depict 2 of the constraints that are checked before accepting the exchange. The complete list of constraints that have to be checked can be found in the full description of the HyParView protocol (Leitão, 2007). After checking all constraints (related for instance, with active view symmetry and non-repetition of node identifiers between the active and passive views), node o sends a DISCONNECTWAIT message to i and adds d to its active view. This concludes an optimization round.

4.2.2.5 Variant

Algorithm 8: Alternative for the isBetter procedure

```

isBetter( $old, new$ )
   $cOld := Oracle.getLinkCost(old)$ 
   $cNew := Oracle.getLinkCost(new)$ 
  return  $cOld > cNew \wedge \frac{(cOld - cNew)}{cOld} \geq THRESHOLD$ 

```

Depending on the implementation and efficiency metrics computed by the oracle, it can provide values that are not completely accurate. However, this will not have a significant impact in the overlay correctness. In fact in scenarios where the oracle provides random cost values, the resulting topology will have the same characteristics of a random unbiased overlay while remaining connected and with an adequate in-degree distribution.

In scenarios where the (average) accuracy of the oracle is known (for instance a work by

Karwaczyński et al. (2007), the author states that longest IP prefix and latency has an approximate correlation of -0.85), the *X*-BOT protocol can be easily adapted to this, by changing the **isBetter** evaluation function to include some hysteresis, namely, a link *new* should only be considered as having a lower cost than other link *old*, if the difference between the cost (obtained through the oracle) offers a gain above a given *Threshold*, which should be a protocol parameter that takes into consideration the accuracy of the deployed oracle. Algorithm 8 depicts the required changes to the **isBetter** function for clarity of the presentation.

4.3 Oracles

The class of oracles employed by *X*-BOT rely on *local knowledge*, *i.e.*, each peer makes optimization decisions based on local information regarding the distance to its own overlay neighbors. Furthermore, as an optimization round, only 4 nodes are involved. Thus the optimization step is also localized. This highly contributes to the scalability of *X*-BOT.

Naturally, this prevents *X*-BOT from generating topologies that require global knowledge. For instance, to bias a network towards a complex topology such as a Torus⁴, would require peers to be aware of the location of many other elements in the overlay or to have *a priori* knowledge of their final positioning in the overlay topology (such as in T-Man (Jelasity et al., 2009)). To bias the network towards such topologies is outside the scope of contribution presented in this chapter. However, as discussed in the next paragraphs, there are several meaningful oracles that fit in the class of oracles employed by *X*-BOT.

4.3.1 Oracle Implementations

Latency Oracle: One of the most intuitive oracles that can be devised is a latency oracle. This oracle operates by collecting and storing measures of round trip times (RTT) between peers, using specific probe messages exchanged by the oracles. The oracle must be aware of the peers which are known at the local host, and it measures the RTT for each known node storing the last reading (or some weighted average), which can be directly used as the link cost value. The use of weighted averages makes the oracle less susceptible to error in individual measurements or to network traffic sudden variations (*i.e.*, spikes).

⁴<http://en.wikipedia.org/wiki/Torus>.

Moreover, if TCP connections are maintained among peers, the oracle can use the estimated RTT calculated by TCP as the link cost avoiding the explicit exchange of messages to perform these measurements.

Internet Service Provider Oracle: In a setting where exchanging messages with a peer associated to a different ISP has an increased financial cost compared to sending a message to a node in the same ISP, it might be useful to keep as many neighbors as possible from the local ISP. An oracle to this end could be built by maintaining information concerning the local ISP and a table of costs for each known ISP (this table could simply store a low value for the local ISP and a high value for all remaining). When the oracle becomes aware of a new peer, it simply exchanges a message with the remote oracle to identify local ISP's, and assert the cost for the link using its local cost table.

Stretch Oracle: It is possible to devise an oracle that returns the number of hops, in the physical network, required to materialize a given overlay link. This can be implemented using tools such as the Unix *traceroute* command, similarly to what is proposed in the Araneola protocol (Melamed & Keidar, 2004). Notice that the underlying topology of the Internet is relatively static, meaning that traceroute measurements do not need to be refreshed frequently, allowing the oracle to have a negligible overhead.

Such an oracle would allow to minimize the stretch of the overlay network, *i.e.*, the ratio between the number of hops in the underlying IP network and the number of hops in the overlay for a given path. Additionally, as reported in the work of Rostami e Habibi (2007) there is an approximately linear correlation between the physical distance of two peers in the physical network, and the latency of communication between those peers, which implies that such an oracle could contribute to lower communication latency among peers as well as the link stress due to the use of the overlay.

Landmark Oracle: A Landmark Oracle is able to place nodes in a coordinate system. This allows to associate costs to links based on the Euclidean distance between the vertices, *i.e.*, a given peer may derive the link cost from its own coordinates and the coordinates of another peer in the system.

Such oracle can be implemented measuring the round-trip-time (RTT) of each node to a globally agreed, and well known, set of servers, which are labelled *landmarks* (*e.g.* DNS root servers). However, this is valid for any set of well know, and geographically disperse servers),

as suggested by Ratnasamy, Handley, et al. (2001). Previous work by Ratnasamy, Francis, et al. (2001) has shown that nodes with similar RTT measures to selected landmark servers have a higher probability of being geographically closer.

IP-based Clustering Oracle: An IP-based Oracle assigns costs to links based on the IP addresses of peers (assigning low costs to links in the same network and higher costs to links that cross different autonomous systems). Such an oracle allows to improve the locality of neighboring relations in unstructured overlay networks with very reduced overhead.

Implementations of an IP-based Oracle have been proposed in the works of Karwaczyński (2007) and Karwaczyński et al. (2007). These implementations are inexpensive as they do not require the exchange of messages among peers. Such oracles operate by taking into consideration the IP of peers to extrapolate the link cost between two peers. For instance, using the length of the match of common IP prefixes to calculate a measure of proximity between two peers. Additionally, other static information, such as the one employed in Skipnet (Harvey et al., 2003) (which leverages in DNS names) can also be employed to enrich the operation of the oracle and improve the precision of the link costs provided by the oracle.

Content Similarity Oracle: A Content Similarity Oracle assigns to links a cost that is inversely proportional to the similarity of the content stored by the peers at its vertices. Such an oracle can be used to bias the network such that nodes with similar content are located in the same region in the overlay. This may help in implementing resource location services, by enabling new strategies for routing and guiding queries to the overlay regions where the desired resources are more likely to exist.

Such an oracle can be implemented, for instance, by classifying resources in a set of (possibly finite) categories, and then locally calculate the percentage of stored resources that fall in each category. In order to compute the link cost, oracles could exchange a data structure containing the percentage of resources that fall in the c most significative categories, and then compute a similarity rate that can be directly used as the link cost.

4.3.2 Combining Oracles

The oracles that have been discussed in the previous paragraphs can be combined to create more complex oracles which allow one to take a combination of different criteria into consider-

ation when optimizing the overlay. For instance the Content Similarity Oracle and the Latency Oracle can be combined to favor the selection of neighbors that have simultaneously similar content and are physically close.

To build such an oracle the output of each contributing oracle would have to be normalized to a pre-determined scale, taking into consideration the maximum (expected) cost value returned by each type of oracle. Subsequently, the output of the combined oracle can be defined as a weighted average of the output of each contributing oracle. Exploring such an approach is however, outside the scope of the thesis.

4.3.3 On the Use of Inconsistent Oracles

All the work reported in this chapter assumes that the system is pre-configured such that all peers use the same oracle. One could imagine scenarios where different nodes in the same overlay would have different (local) goals and that this would still allow the emergence of some global stable topology. Unfortunately, it is possible to show that if a set of independent agents in a system have conflicting goals, the convergence of the system is not guaranteed. Exploring adequate mechanisms that enable different peers to simultaneously rely on distinct oracles, that take into consideration distinct performance criteria, in a single overlay network will be addressed in future work.

4.3.4 Oracle Cost

Some oracle implementations may require the explicit exchange of messages among distinct peers, to enable the oracle to locally compute and provide a cost for an existing or potential new overlay link. For instance, consider the simple latency oracle, that was previously described, that exchanges *ping* messages to compute an approximation of the latency between peers.

This could introduce some undesired additional overhead to the operation of X-BOT. However, in X-BOT each optimization round only requires two nodes to consult their local oracles. Moreover, the number of times that a node consults its oracle in the first step of our protocol can be limited by the π parameter; experimental results obtained during the design and development of X-BOT have shown that π can be configured with a small value (such as 2).

Finally, the oracle can also perform the required message exchanges in background, decoupling the traffic generated by the oracle from the traffic produced by the operation of X-BOT.

4.4 Configuring X-BOT

X-BOT has several configuration parameters that affect its operation. This section addresses the effect of each parameter in the behavior of the protocol and provide some insights on how to adequately configure these parameters.

4.4.1 Active View Size:

This parameter controls the number of peers maintained the active view. Active views in X-BOT are symmetric and have a size optimized to support gossip-based dissemination protocols, *i.e.*, their size is configured to be $fanout + 1$, where $fanout$ is the parameter used by the gossip-based broadcast protocol. It has been shown that, in order to ensure a high probability of atomic delivery of messages for gossip-based processes, the fanout should be in the order of $\ln(N)$, where N is the number of peers in the system (Eugster et al., 2004). However, these theoretical results do not assume an overlay that ensures global connectivity, and furthermore assume that some messages may be lost during the dissemination process. Additionally, it is well known that in an undirected k -regular random graph, 3 is the minimum degree to ensure global connectivity (Bollobás, 1981). Taking into consideration these results, we suggest that for the particular case study addressed in this chapter, active views should have a size that falls between 3 and $\ln(N)$. We have opted to use a size of $\log(N) + 1$, which experimentally has shown to ensure global connectivity of the overlay with a probability close to 1.

4.4.2 Passive View Size Control Parameter (k):

The k parameter controls the size of the passive views maintained by X-BOT. In fact, in our system, the passive view size is k times larger than the active view. As discussed previously, the passive view is used for two complementary purposes, namely: *i*) fault-tolerance; and *ii*) as a source to extract unbiased samples of other nodes in the overlay, that X-BOT can use to bias the local active views.

From the point of view of fault-tolerance the size of the passive view should be at least $\ln(N)$, where N is the total number of peers in the system. However, a larger value would yield a larger sample of peers to support the operation of X-BOT and consequently, offer additional opportunities to bias the active view of peers. On the other hand, a much larger size may induce a non-negligible overhead, as more information need to be maintained by each peer.

In our experiments the value of k was set to 6, as this allows to have a view of size 30 which although being larger than $\ln(N)$, does not impose a noticeable overhead in the protocol. We have determined experimentally that this offers a good trade-off between the minimum size required for fault-tolerance, and also the necessity of X-BOT to have access to a varied sample of peers.

4.4.3 Period Between Optimizations (PBO):

The period between optimizations (or simply PBO) determines the time between each attempt by the X-BOT protocol to locate one (or more) suitable neighbor to further bias its active view *i.e.*, the time between the execution of the biasing procedure. It also determines the rate at which the active view of a node may be updated (*i.e.*, changed).

Large values of PBO slow the convergence of the overlay topology, leading potentially to larger periods of operation with a sub-optimal overlay configuration. On the other hand, large PBO values promote the stability of the overlay. Furthermore, low PBO values increase the X-BOT overhead, given that a more frequent operation induces additional message exchanges and more oracle invocations. In any case, the PBO value should be a multiple of the passive view update period; such that when a new optimization round start the passive views has been updated with new potential candidates for optimization. We suggest that PBO should be configured to be at least twice the period between the execution of the procedure which updates the passive views of participants.

4.4.4 Passive Scan Length (π):

The passive scan length (or simply π) determines the number of peers extracted from the passive view to serve as candidates to bias the active view by the X-BOT protocol every PBO time period. The link cost to these peers is then measured (through the companion oracle), and if

their cost is below that of the most expensive active view neighbor (excluding the protected μ most expensive neighbors) kept by that peer, the optimization process is triggered. This parameter also determines the maximum number of elements of an active view that can be updated in a single optimization round of *X-BOT*.

The value of π should be lower (or equal) to that of half the nodes in the active view that can be biased by *X-BOT* (which is equal to the size of the active view minus the number of unbiased neighbors). Setting π to a small value has two advantages: *i*) it promotes some stability in the overlay, avoiding to exchange the majority of neighbors in the active view of a single peer in the context of a single execution of the optimization procedure, and *ii*) it lowers the cost of the overall optimization process. Results presented in the evaluation section show that a conservative configuration of π allows to achieve fast convergence and yields a good level of optimization for the overlay.

4.4.5 Number of Unbiased Neighbors (μ):

This parameter controls the number of elements in an active view that are never targeted by the optimization process of *X-BOT*. Experimental results presented in section 4.5 show the impact of this parameter on several properties of the unstructured overlay networks; from those results it is possible to observe that a μ value of 1 provides good results in all scenarios (these results are consistent with those presented by the authors of GoCast (Tang & Ward, 2005)).

4.5 Evaluation

4.5.1 Experimental Setting

We conducted an extensive experimental evaluation of *X-BOT* comparing its performance with that of GoCast (Tang & Ward, 2005), Araneola (Melamed & Keidar, 2004), and T-Man (Jelasity et al., 2009) using the PeerSim simulator (Montresor & Jelasity, 2009). To that purpose, all considered protocols were implemented in the simulator using the cycle-based engine of PeerSim. Furthermore, all protocols were adapted to leverage the same oracle as *X-BOT* to ensure a fair comparison. Also, to assess that our implementations of GoCast, Araneola, and T-Man were accurate, we did compare their performance with the results that have been published in the

corresponding papers (these results are omitted from the thesis as they do not contribute to illustrate the benefits of the contributions presented here). In the experiments reported here, we use the following scenarios, which allowed us to assess the benefits of X-BOT in environments with different characteristics:

Cartesian Scenario: This scenario uses a network of 10,000 nodes organized in a cartesian plan (a 100x100 square), where two direct neighbors are at a distance of 1. We model the cost of a link between two nodes, as being equal to the distance between those nodes in the cartesian space. This scenario is interesting as it offers a high potential for improving the topology of a random overlay. Moreover the link costs in this scenario are symmetric and present a gaussian distribution.

PlanetLab Scenario: This scenario is composed of 341 nodes in which the cost of a link is defined according to the *all pair pings* trace⁵ that contains ping times measured among a set of PlanetLab nodes. Each simulated node represents a PlanetLab node and the cost between any two nodes, n_i and n_j , is set as half of the round trip time between these two nodes as measured from n_i according to the PlanetLab traces. Notice that in this scenario, link costs are not necessarily symmetric. This scenario allows us to observe the performance of X-BOT in a realistic setting.

Inet-3.0 Scenario: This scenario is composed of 10,000 nodes. In this scenario we used the network generator Inet-3.0 (Winick & Jamin, 2002) to distribute 10,000 nodes over a network of autonomous systems using the default parameters of Inet-3.0.⁶ We then computed the shortest path for every pair of nodes, and defined the latency of the corresponding link to be equal to the distance between routers that are transversed by the shortest path connecting the end-points of the link. The link cost was set to be the latency between the end-points of a link. This scenario allows to evaluate the performance of protocols in a large-scale realistic scenario, where links between nodes present a wider range of possible link cost values.

⁵A repository with these traces can be found in: http://pdos.csail.mit.edu/~strib/pl_app/.

⁶In fact we generated 5 different network topologies, each generated topology was used in each individual simulation run.

4.5.2 X-BOT Performance

In this section an experimental study on the properties of the overlays that result from the operation of X-BOT, while maintaining distinct numbers of unbiased numbers (*e.g.* using different values for the μ parameter), is provided. We provide these results as a proof of concept, allowing us to show the improvement capabilities and the flexibility of our approach. Moreover, these results provide some insight on the effect of X-BOT's optimization procedure over the clustering coefficient of the overlay.

We follow by providing experimental results for configurations where all protocols attempt to maintain a single random/unbiased neighbor (both our experiments and previous work by Tang e Ward (2005) suggest that this is the most useful configuration for most scenarios and, in particular, for scenarios where the overlay is used to support gossip-based broadcast services).

In all experiments we conducted, the size of the passive views maintained by X-BOT was set to 30 (*i.e.*, $k = 6$). The remaining protocols benefit from a (similar) random partial view maintained by a companion membership protocol. To ensure a fair comparison, we also set the size of these views to 30. Moreover, we initialized these views with contents extracted from the passive views of HyParView after 250 simulation cycles. Simple maintenance routines for these views (similar to the ones employed by X-BOT) were also added to the remaining protocols. Furthermore, because T-Man lacks a join procedure, we initialized its views with contents extracted from the active views of HyParView extracted in similar conditions, which provided T-Man with a random initial configuration similar to the one of X-BOT.

Furthermore, for all simulations presented in this chapter, X-BOT was configured as follows: *Period between optimizations* (PBO) was set to 2 simulation cycles. This increases the probability of having new peers in the passive view of a node in each optimization step, as described previously. The shuffle procedure used to update the contents of passive views was executed once by each peer in every simulation cycle. *Passive scan length* (π) was set to 2, so each time a node executes the step 1 of the optimization algorithm, it tests, at most, 2 peers from its passive view. This also limits to 2 the maximum number of neighbors which can be exchanged in a single execution of the optimization procedure.

Finally, the initial (external) partial views provided to Araneola and GoCast were sorted by link cost. This is the most favorable configuration for these protocols. However, this strategy

requires additional invocations to the oracle, which might incur in additional overhead if the oracle implementation requires the exchange of messages. All results reported in the chapter are an average of 5 independent runs of each simulation. Confidence intervals were verified and were non-significant. These were omitted from the figures for improved readability.

4.5.2.1 X-BOT Individual Evaluation

In this section we evaluate a set of relevant metrics that focus on properties of the overlay topology that results from the operation of X-BOT. Experiments were conducted by executing X-BOT for 1,000 simulation cycles and observing the evolution of the overlay and its final configuration.

In this set of experiments we are mainly concerned with evaluating the effect of keeping a different number of *Unbiased Neighbors*. To this end, we tested the protocol using values for the μ parameter that range from 0 (none) to 5 (all). Figure 4.2 plots the evolution of the overlay cost, Figure 4.3 reports on the clustering coefficient and finally, the evolution of the average shortest path is shown in Figure 4.4. Each figure reports the evolution of these metrics for X-BOT as the protocol operates in both the cartesian, PlanetLab, and Inet-3.0 scenarios. We now discuss these results in detail.

4.5.2.1.1 Overlay cost Figures 4.2(a), 4.2(b), and 4.2(c) show respectively the evolution of the overlay cost in the cartesian, PlanetLab, and Inet-3.0 scenarios. In all scenarios the protocol shows a similar behavior. As the protocol keeps less unbiased neighbors (*e.g.* as the μ parameter lowers) the final cost of the overlay becomes lower. Notice that the overlay cost in all scenarios with a μ value of 5 is constant. In fact, in this case the overlay is not biased, and this case is used as a baseline to assert the benefits of X-BOT comparing with an unbiased unstructured overlay network.

Although the protocol behavior is consistent in all scenarios, the reduction in the overlay cost is not. For instance, whereas in the cartesian scenario with μ value of 0, the overlay cost is reduced by a value on the the order of 1×10^7 presenting a final overlay cost close to 3.5×10^5 , the same configuration in the PlanetLab scenario allows the overlay to lower its cost by a value close to 1×10^8 , exhibiting a final cost in the order of 3×10^8 . In the Inet-3.0 scenario the overlay cost drops by approximately 1.4×10^8 . This is due to the nature of the underlying network

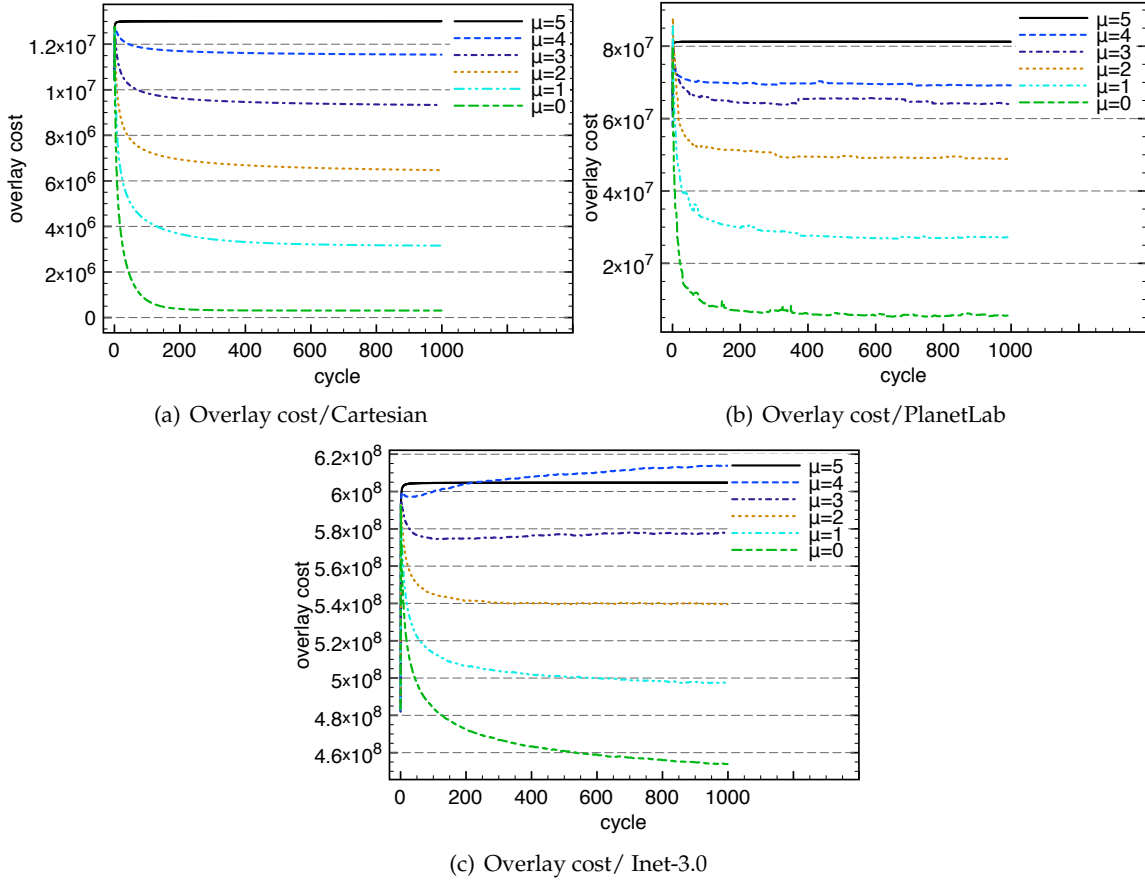


Figure 4.2: Evolution of overlay cost for X -BOT when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios.

and the metric being computed by the oracle. In fact, distinct physical topologies and different oracles offer distinct potentials for improving the topology of the overlay network.

Interestingly enough, in the Inet-3.0 scenario with a μ value of 4 (only one neighbor is biased) the overlay cost slightly rises. This is due to the fact that the operation of X -BOT implicitly bias *unbiased* neighbors to promote distant connections in order to improve the overlay diameter. This scenario is the one where link costs present a wider range of values, which leads, with a single biased neighbor, the overlay topology to feature higher cost links as further detailed below.

4.5.2.1.2 Effect on the Average link cost We now present additional experimental data that illustrates the effect on the average link cost of the overlay (making a distinction between average biased and average unbiased link cost). In these experiments we focus on the operation

	Cartesian Scenario		
	Average Link Link Cost	Average Biased Link Cost	Average Unbiased Link Cost
Random Overlay	520.05	–	–
X-BOT ($\mu = 4$)	461.36	24.73	570.62
X-BOT ($\mu = 3$)	372.85	22.87	606.47
X-BOT ($\mu = 2$)	258.44	18.17	619.05
X-BOT ($\mu = 1$)	125.93	12.35	581.79
	PlanetLab Scenario		
	Average Link Cost	Average Biased Link Cost	Average Unbiased Link Cost
Random Overlay	100,865.86	–	–
X-BOT ($\mu = 4$)	88,990.38	2,225.70	110,596.92
X-BOT ($\mu = 3$)	77,511.78	3,000.66	127,111.88
X-BOT ($\mu = 2$)	57,683.03	3,795.05	138,421.72
X-BOT ($\mu = 1$)	31,456.59	4,825.36	137,902.78
	Inet-3.0 Scenario		
	Average Link Cost	Average Biased Link Cost	Average Unbiased Link Cost
Random Overlay	27404.44	–	–
X-BOT ($\mu = 4$)	27,857.47	21,793.35	29,373.62
X-BOT ($\mu = 3$)	26,406.80	21,432.64	29,722.99
X-BOT ($\mu = 2$)	24,871.94	21,404.65	30,073.13
X-BOT ($\mu = 1$)	23,186.46	21,370.48	30,450.88

Table 4.1: Comparison of average link cost, average biased link cost, and average unbiased link cost for several values of parameter μ in X-BOT and in a random overlay network with similar properties.

of X-BOT when configured with a μ value varying between 1 and 4. Results were obtained by executing the protocol for 1,000 simulation cycles after all peers had joined the system. At the end of the simulation the average cost of unbiased and biased links, kept by peers as a result of X-BOT's operation was measured. The average link cost of the initial random overlay is also presented for providing a baseline. Table 4.1 summarizes the obtained results for 5 independent runs of each experience.

The results show that, for all test scenarios, with smaller values of μ we obtain lower values in the average link cost when compared with an overlay built at random. This is not a surprising effect, as it is the result of the X-BOT optimization on the (larger number of) biased links. This is consistent with the results previously reported in Figure 4.2.

Also, in most scenarios, with smaller values of μ , we obtain higher values for the average link cost of unbiased links. This is due to the fact that, as explained previously, X-BOT tends to

avoid biasing the links that have a greater cost. There are however exceptions to this behavior. In particular, in the cartesian and Inet-3.0 scenarios, the average cost of unbiased links is higher with $\mu = 2$ than with $\mu = 1$. This happens because in those topologies each node has many peers which are *closer* in the distance metric considered for each scenario. Therefore, the random topology that serves as the base for *X-BOT* operation already included many links with a low cost, and *X-BOT* does not actively try to find distant peers to keep as unbiased neighbors.

Interestingly, in the PlanetLab scenario, as the number of unbiased neighbors kept by each node gets lower, the average link cost of unbiased neighbors slightly increases. This happens because in this scenario nodes have few *close* neighbors considering the metric encoded in the companion oracle. Therefore, as nodes try to find *closer* neighbors they have to select peers which, despite the fact that are among the closest available peers, still have a slightly high link cost. Despite this, the average biased link cost is still dramatically lower than the average link cost of a random overlay for the same scenario, clearly showing the benefits of our overlay topology management scheme.

4.5.2.1.3 Clustering coefficient and average shortest path Figures 4.3(a), 4.3(b), and 4.3(c) depict results for clustering coefficient in the cartesian, PlanetLab, and Inet-3.0 scenarios respectively. As expected, biasing all elements in the partial view of all nodes highly increases the clustering coefficient of the overlay. On the other hand, maintaining a single unbiased neighbor is enough to greatly mitigate this effect.

Interestingly, in the cartesian scenario with 4 unbiased neighbors and in the Inet-3.0 scenario with 2 to 4 unbiased neighbors the resulting clustering coefficient drops to values below that of the unbiased overlay (*i.e.*, the random overlay denoted by the line labelled $\mu = 5$). This phenomenon can be explained as follows. By maintaining active views sorted by cost, the selected unbiased neighbors are those with a larger cost. In other words, our algorithm, at no extra cost, promotes the maintenance of *long distance* nodes in each view.

The reader should also notice that Figure 4.3(a) has a logarithmic scale. In fact, the increase of clustering coefficient is much higher in this scenario. This is not a surprisingly phenomenon, as in this scenario the cost function computed by the oracle is related with the distance between nodes, which is a transitive property. This leads us to the conclusion that when biasing an overlay by relaying on a transitive performance metric one should relay in additional unbiased neighbors.

We present similar results concerning the average shortest path of the resulting overlays in Figures 4.4(a), 4.4(b), and 4.4(c). As noted earlier, there is a strong correlation between the clustering coefficient and the average shortest path of an unstructured overlay network. Therefore, these plots show results that are consistent with the previous ones.

When no unbiased neighbor is kept, the average shortest path of the overlay rapidly rises to values off the scale in the cartesian scenario (Figure 4.4(a)). This happens due to the nature of the cartesian scenario, in a consistent manner with experimental results previously reported here.

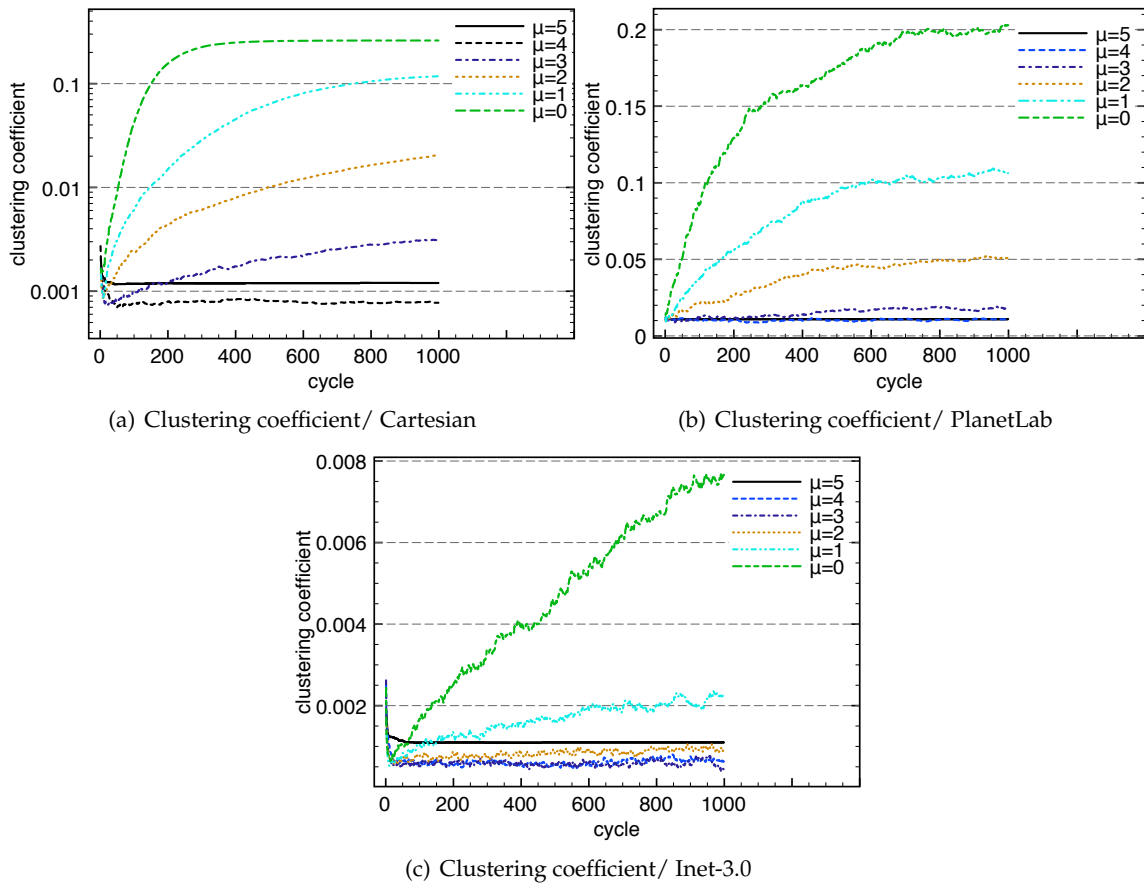


Figure 4.3: Evolution of clustering coefficient in the X-BOT overlay when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios.

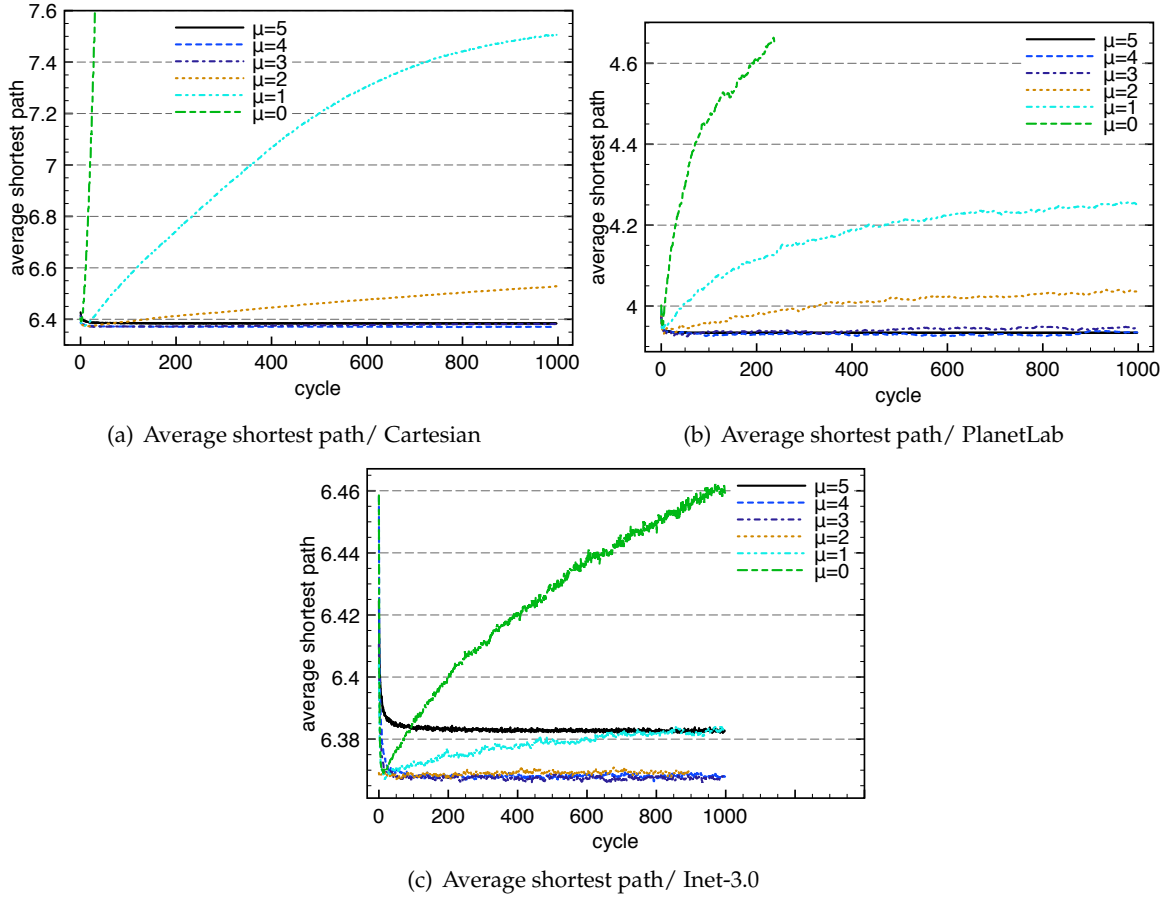


Figure 4.4: Evolution of average shortest path in the X -BOT overlay when varying the number of unbiased neighbors for the Cartesian, PlanetLab, and Inet-3.0 scenarios.

4.5.2.2 Comparative Evaluation

In this section we extend the previous experiments to provide comparative figures of the X -BOT performance against other protocols. The experimental work discussed in this section was conducted following a methodology similar to the one described in the previous section. As stated earlier, we focus our evaluation in scenarios where only one unbiased/random neighbor is maintained by each of the considered protocols.

4.5.2.2.1 Overlay Cost We give particular emphasis to the overlay cost metric, as it allows to assert the benefits of employing each of the protocols. Figures 4.5(a), 4.5(b), and 4.5(c) depict, respectively, the overlay cost for the cartesian, PlanetLab, and Inet-3.0 scenarios for all protocols. Compared with both Araneola and GoCast, X -BOT achieves a lower overlay cost. This can be explained as follows: Araneola is a reactive protocol, in the sense that once the

neighbor set of a peer stabilizes (*i.e.* by matching all protocol requirements) it will never be updated again until some external event happens (*e.g.* a neighbor fails). Therefore Araneola does not explore the full optimization potential of the environment. GoCast, on the other hand, is able to iteratively improve the overlay topology, unfortunately the protocol does not ensure a constant degree of nodes (see Figures 4.6(a), 4.6(b), and 4.6(c)), which results in the creation of additional links that increase the overlay cost. Moreover, considering that the average cost of each link maintained by GoCast is higher than the cost of the links maintained by *X-BOT*, it is possible to observe that the 4-node coordination optimization strategy at the core of *X-BOT* offers better results than the GoCast approach.

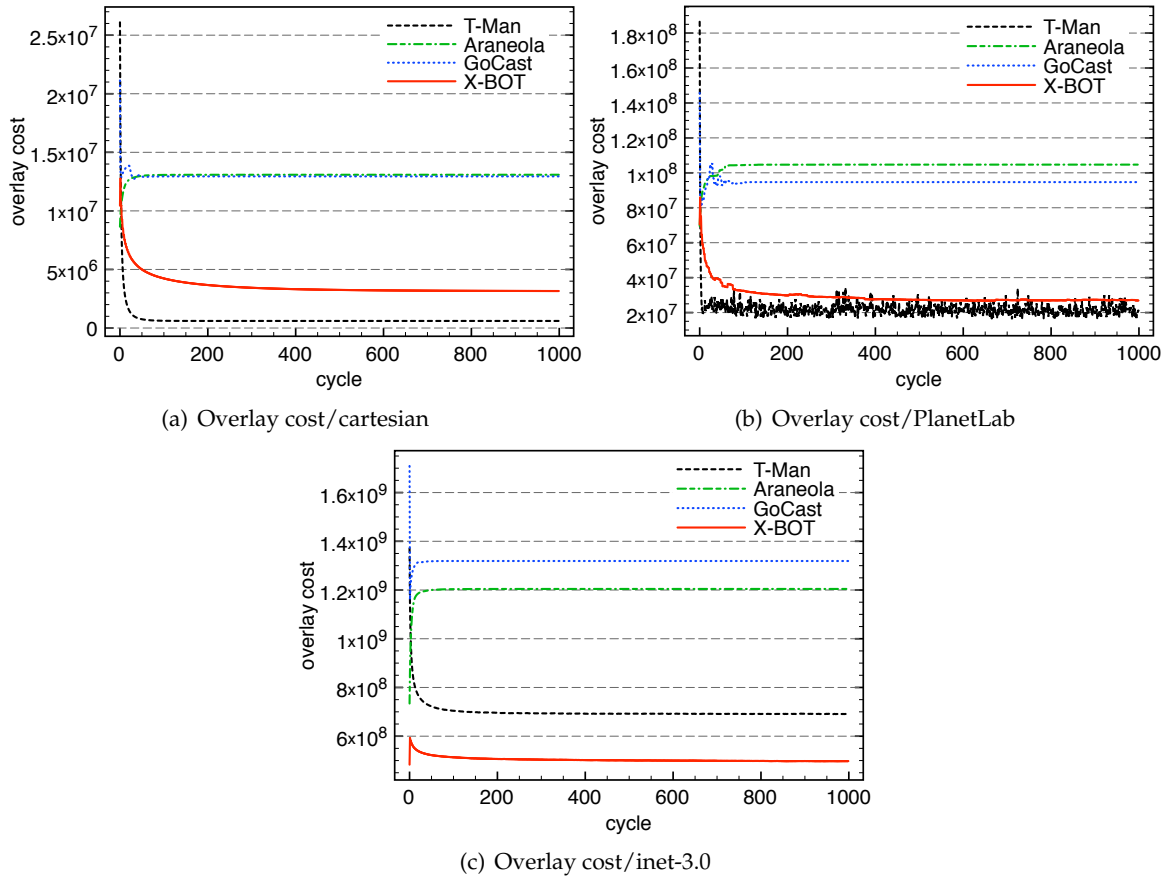


Figure 4.5: Cost of the overlay that results from the operation of *X-BOT* and the remaining tested solutions for the three experimental settings.

T-Man achieves a performance that is similar to *X-BOT* in the PlanetLab scenario and can even achieve a more efficient overlay topology configuration than *X-BOT* in the cartesian and Inet-3.0 scenarios. This is due to its aggressive optimization strategy. Unfortunately, this ag-

gressive strategy has severe drawbacks, namely it has a negative impact in the overlay connectivity. Figures 4.6(a), 4.6(b), and 4.6(c) depict the in-degree distribution of the overlay that results from the operation of each protocol. T-Man generates overlays where several peers exhibit an in-degree of 0 (which is specially noticeable in the Inet-3.0 scenario where many peers are distant from the core of the network) while other peers have a very high in-degree (as high as 120 neighbors in the PlanetLab scenario, and 1277 neighbors in the Inet-3.0 setting). As a result, T-Man is unable to preserve the connectivity of the overlay, which has a negative impact on P2P services being executed on top of the overlay (such as gossip-based broadcast services). Notice that in this scenario, and for load balancing during message dissemination, each peer should ideally, have the same in-degree value, as this ensures that each peer has to send and receive (on average) a similar number of messages.

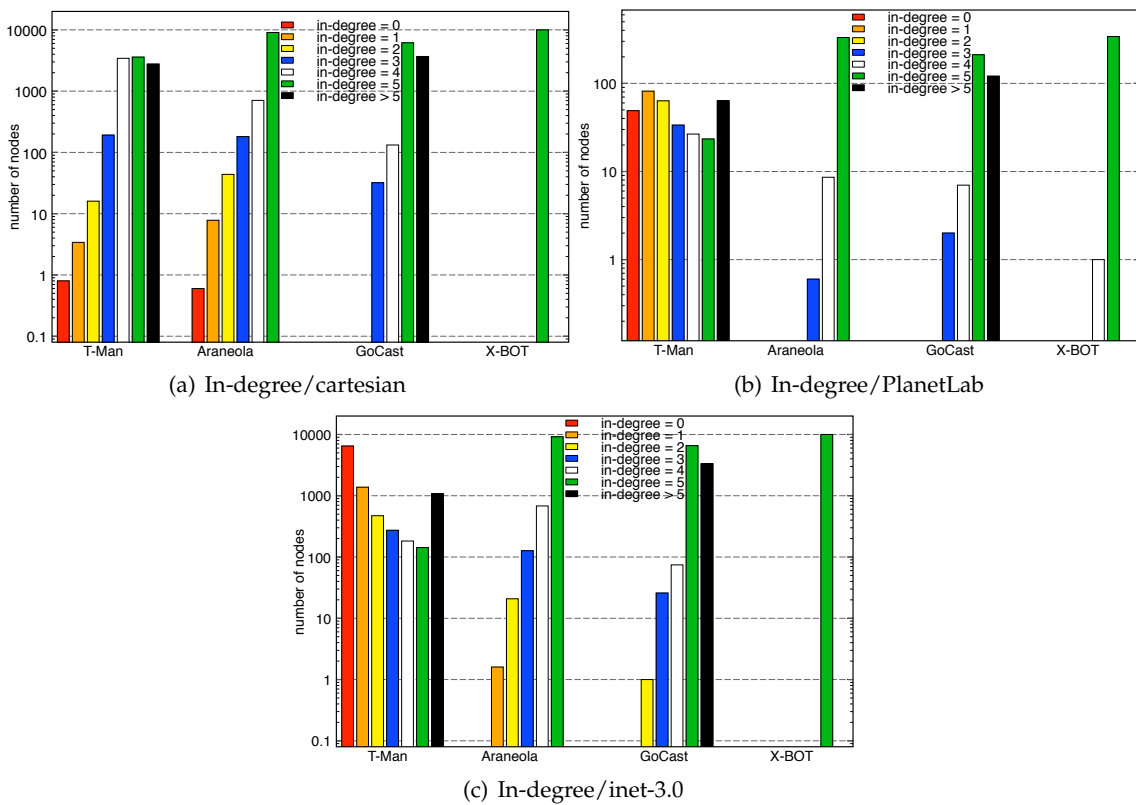


Figure 4.6: In-degree of the overlay that results from the operation of X-BOT and the remaining tested solutions for the three experimental settings.

Table 4.2 shows the resulting clustering coefficient (CC) and average shortest path (ASP) for all protocols in all scenarios. Notice that T-Man does not present a value for average shortest path, as none of the executions of the protocol was able to maintain the overlay connected. X-

BOT and GoCast offer the best results, although GoCast achieves these results at the expense of maintaining several peers with a node degree much higher than the target value.

Despite the fact that *X*-BOT presents a somewhat higher clustering coefficient when compared with GoCast (in all scenarios), in terms of average shortest path *X*-BOT has values only slightly above those of GoCast. A fact that is specially noticeable in the Inet-3.0 scenario, which is the scenario that better models the operation of protocols over a large-scale network with properties similar to those found in the Internet. This happens because contrary to GoCast, *X*-BOT implicitly biases unbiased neighbors to promote distant links, resulting in an overlay with lower diameter. This allows *X*-BOT to better support (several) P2P services, such as gossip-based broadcast services as shown by the results presented further ahead in the text.

4.5.2.2.2 Scenario with several Internet Service Providers In order to illustrate the flexibility of *X*-BOT, we evaluated the performance of the protocol in an additional setting. This setting is similar to the cartesian scenario with the difference that we associate to each peer in the system one Internet Service Provider (ISP). The assignment has been performed such that the number of nodes belonging to each internet service provider is approximately the same.

We rely on an implementation of an ISP oracle that operates by providing a cost of 0 units to any peer belonging to the same internet service provider, and a large constant value (in our case 1,000) to all remaining peers in the system. We performed experiments with *X*-BOT in scenarios where a different number (ranging from 2 to 10) of internet service providers coexist. In order to extract comparative measures, we performed similar experiments with T-Man, Araneola, and GoCast.

In such a specific scenario we rely on a metric that better captures the efficiency of the resulting overlay networks. This metric, that we dubbed Inter-ISP link density is defined as

	Cartesian		PlanetLab		Inet-3.0	
	CC	ASP	CC	ASP	CC	ASP
T-Man	0.271	∞	0.350	∞	0.197	∞
Araneola	0.193	9.808	0.070	4.235	0.1048	9.66
GoCast	0.0009	6.021	0.024	3.793	0.0004	6.03
X-BOT	0.117	7.506	0.098	4.230	0.0023	6.38

Table 4.2: Comparison of overlay properties of the overlay generated by *X*-BOT and the remaining tested solutions.

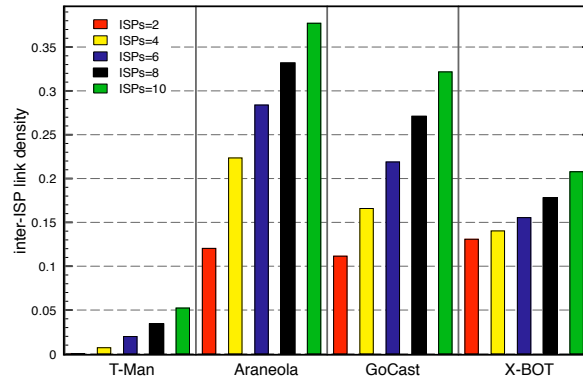


Figure 4.7: Fraction of sub-optimal links for different protocols and different number of ISPs in the overlay that results from the operation of *X-BOT* and the remaining tested solutions

the ratio between the number of overlay links that connect nodes in two distinct ISPs and all existing links in the overlay having a value between 0 and 1.

Intuitively, an efficient overlay in this scenario should present a low inter-ISP link density. However, it should be above zero to ensure that the overlay remains connected. In a scenario where nodes are associated to more than one ISP, a value of inter-ISP link density of 0 is only possible to achieve if the overlay becomes partitioned. Our experimental results show that a value between 0.1 and 0.2 offers good connectivity while minimizing the number of links that connect peers in distinct ISPs.

Figure 4.7 summarizes the experimental results obtained in this scenario, for all protocols, and for a variable numbers of coexisting ISPs ranging from 2 to 10. With the exception of T-Man, *X-BOT* is the protocol that achieves lower values for inter-ISP link density. Moreover, for all protocols the inter-ISP link density rises with the number of coexisting ISPs. This is expected as the number of nodes in each ISP becomes lower with the addition of other ISPs leading to a situation where it becomes harder for a node to find peers that are also connected through its ISP. Notice that this effect is less visible in *X-BOT* than Araneola or GoCast. This results from the *X-BOT* strategy that periodically employs its 4-node coordinated optimization technique to continually evolve the overlay topology to a better configuration.

T-Man is able to obtain lower inter-ISP link density values by sacrificing the overlay connectivity. The percentage of nodes in the largest connected cluster in the resulting overlay network is depicted in Table 4.3. Notice that the percentage of nodes in T-Man is always far below that of 100%, and this value lowers with the number of ISPs. In fact this happens because

Number of ISPs	T-Man	Araneola	GoCast	X-BOT
2	50.011	99.994	100.0	100.0
3	30.380	99.994	100.0	100.0
4	25.050	99.992	100.0	100.0
5	20.082	99.996	100.0	100.0
6	16.780	99.996	100.0	100.0
7	14.184	99.998	100.0	100.0
8	12.664	99.994	100.0	100.0
9	11.282	99.998	100.0	100.0
10	10.156	99.998	100.0	100.0

Table 4.3: Percentage of nodes in the largest connected cluster for X-BOT and the remaining protocol for different numbers of ISPs.

the aggressive strategy of T-Man leads to a situation where nodes associated to a given ISP i , are only connected to other nodes in i , or nodes in other ISPs which neighbors are all associated to that same ISP, resulting in the overlay becoming partitioned in a number of components similar to the number of coexisting ISPs.

4.5.3 X-BOT Support for Broadcast

In this section we evaluate the performance of the a P2P broadcast service when operating on top of the overlays that result from the operation of the different protocols. This service is provided mostly by a gossip protocol where each peer forwards each broadcast message received by the first time to *fanout* of its overlay neighbors selected at random (excluding the peer from which it received the message originally). In the particular case of X-BOT which has a tighter control over the overlay degree, and can ensure that the overlay remains connected even with small-sized active view, this gossip protocol almost behaves like a flood protocol. This allows to better capture the effect of having a large amount of lower latency overlay links on the performance of the P2P broadcast service.

Considering that the target node degree in the overlay is 5 we set the *fanout* value of the gossip protocol to 4 (i.e., the largest fanout that prevents a message from being sent more than once through any given overlay link). As the results presented earlier in this chapter, reported results are an average of 5 independent experiments. Link latency is captured by the link cost. The event-based engine of PeerSim was used to support the operation of the broadcast service

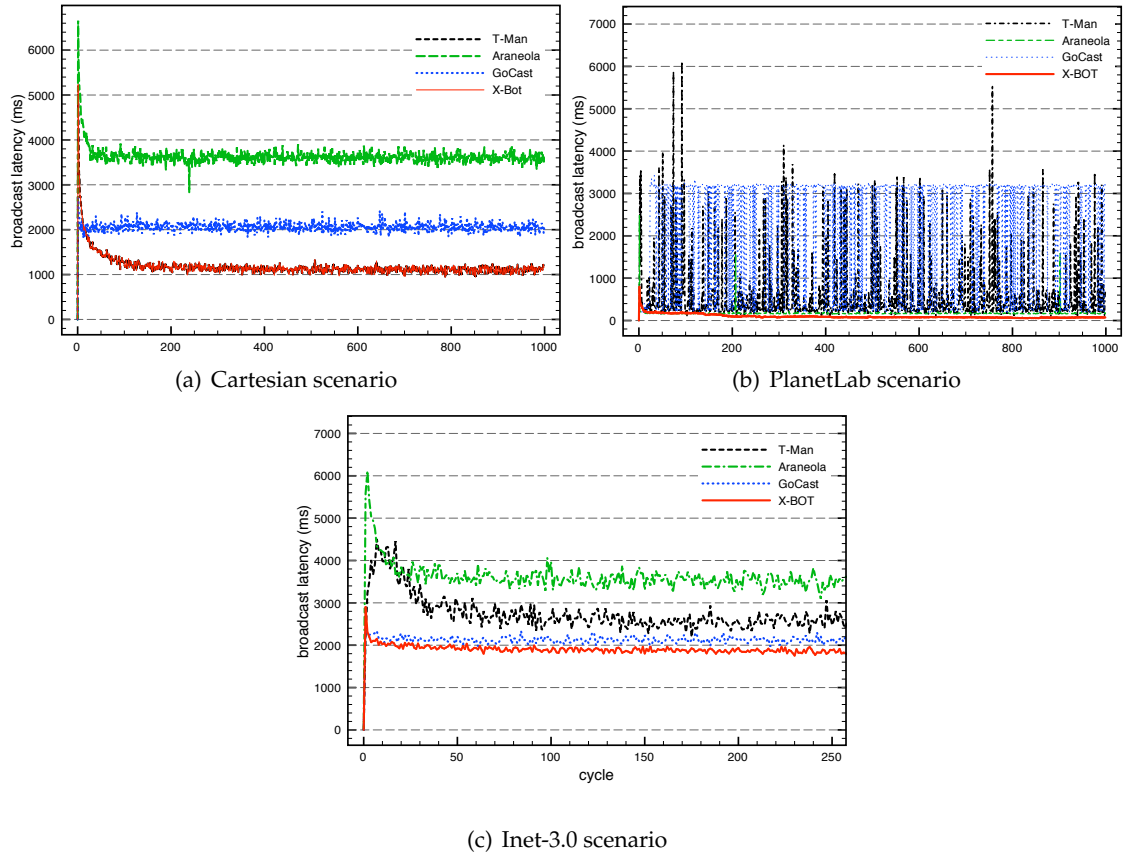


Figure 4.8: Message dissemination latency using the overlays generated by *X-BOT* and the remaining tested solutions for each scenario.

and introduce latency to message dissemination based on that metric.

4.5.3.1 Steady State

Figures 4.8(a), 4.8(b), and 4.8(c) depict the broadcast latency (*i.e.* the amount of time required to deliver a message to the maximum of participants) for each protocol. Only T-Man is able to provide better latency than *X-BOT*. This only happens because T-Man is not able to provide a reliability of 100% as its overlay becomes disconnected. This is notorious in the PlanetLab and Inet-3.0 scenarios, which have a non-gaussian link cost distribution (see Table 4.4). The good performance of *X-BOT* is due to its capacity to improve the overlay efficiency while preserving the in-degree distribution and connectivity (as shown previously). Notice that in the PlanetLab scenario, GoCast and T-Man latency exhibits several spikes. This is due to adaptations of the overlay which affect the node degree distribution, which can increase the overlay diameter

resulting in additional latency.

To provide a better comparison among protocols, Table 4.4 shows both the latency and the reliability values obtained for the execution of the broadcast service. In all three considered scenarios, *X-BOT* offers better reliability with a lower latency. This clearly shows that *X-BOT*, when equipped with a latency oracle, offers a better support to implement gossip-based broadcast services. Notice that, as described above, *T-Man* is unable to achieve a reliability of 100% due to the fact that the overlay becomes disconnected as a result of the protocol operation.

Cartesian Scenario				
	T-Man	Araneola	GoCast	X-BOT
Latency (ms)	1238.0	3701.8	2032.8	1165.2
Reliability (%)	99.328	99.994	100	100
PlanetLab Scenario				
	T-Man	Araneola	GoCast	X-BOT
Latency (ms)	711.8	162.8	3180.8	72.0
Reliability (%)	49.0322	100	100	100
Inet-3.0 Scenario				
	T-Man	Araneola	GoCast	X-BOT
Latency (ms)	2545.2	3517.0	2108.0	1879.8
Reliability (%)	13.806	100	99.99996	100

Table 4.4: Comparison of Broadcast Latency and Reliability using overlays generated by *X-BOT* and the remaining tested solutions.

4.5.3.2 Fault Tolerance

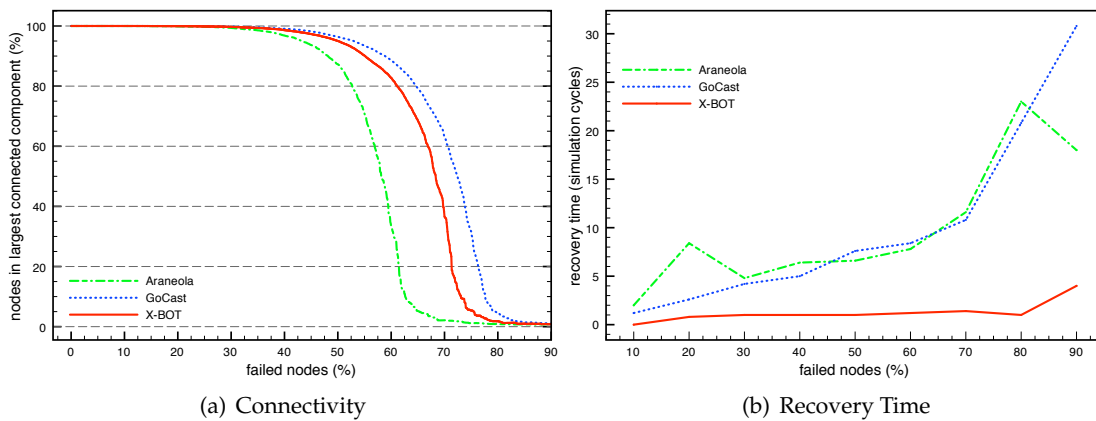


Figure 4.9: Resilience to node failures in terms of connectivity and time required to recover for *X-BOT*, *GoCast*, and *Araneola*.

We now evaluate the resilience and healing capabilities exhibited by each protocol in face

of node failures. We assume that nodes can fail by crashing, and that TCP enables a peer to detect these failures after a small amount of time (in our experimental setup this occurs in the following simulation cycle). We focus this experimental study on the Cartesian scenario, as the behavior of the protocols from the point of view of fault tolerance is highly independent of the properties of the underlying network. Moreover, results for the T-Man protocol were omitted given that the overlay becomes disconnect in steady state.

Figure 4.9(a) plots the percentage of correct nodes in the largest connected overlay component as nodes fail one by one. In these simulations protocols were not allowed to take any corrective measures. *X-BOT* offers better connectivity in face of failures when compared with *Araneola*. However, *GoCast* connectivity surpasses that of *X-BOT*. This is not surprising, giving that *GoCast* maintains a significant number of nodes with degree above 5. Unfortunately, as discussed previously, this feature has a negative impact on the performance of gossip-based broadcast services.

We then evaluated the time required for each protocol to recover from massive failures that range from 10% to 90% of simultaneous nodes crashes. To this end, we measured the number of simulation cycles required, in average, for the broadcast service to regain its previous (or maximum) reliability values after the induction of failures. Whereas in general *X-BOT* always regains a reliability of 100%, the same is not true for other protocols. Results are depicted in Figure 4.9(b). *X-BOT* is able to recover from failures much faster. This is due to the design of *X-BOT* which, unlike *GoCast* or *Araneola*, promotes connectivity, by avoiding optimization rounds when peers do not have a full active view.

4.6 *X-BOT Properties*

This section provides additional insights on some of the *X-BOT* properties. In particular, the complexity of the protocol from the point of view of communication overhead is addressed. Also, the section presents informal arguments, claiming that the probability of *X-BOT* falling into local minima configurations is small. Finally, the section highlights the features of *X-BOT* that help in providing low clustering coefficient and low average shortest path to the resulting overlay.

4.6.1 Complexity

A complete X-BOT optimization round requires the sequential exchange of seven messages. Furthermore, in the most common run each node involved in the optimization only has to send and receive at most two messages (exceptions are runs where faults occur which lower this cost). Given that the optimization of the overlay can be executed as a background activity, the cost of the adaptive mechanisms can be easily tuned to become negligible when compared with the application traffic. This can be performed by adjusting the Period Between Optimizations parameter.

4.6.2 Avoiding Local Minima Configurations

This section provides informal arguments to backup the claim that the X-BOT avoids local minima with a probability greater than zero.

Assume that the overlay is in a given configuration, say \mathcal{C} , and there is some other possible configuration, say \mathcal{C}' , which has a lower overlay cost. Since X-BOT switches links in pairs to preserve the original degrees of the nodes, there must exist 4 nodes, a, b, c, d , such that links l_{ab} and l_{cd} in configuration \mathcal{C} are replaced by links l_{ac} and l_{bd} in configuration \mathcal{C}' such that:

$$\text{LinkCost}(l_{ac}) + \text{LinkCost}(l_{bd}) < \text{LinkCost}(l_{ab}) + \text{LinkCost}(l_{cd})$$

Assume that $\text{LinkCost}(l_{ac}) < \text{LinkCost}(l_{ab})$. In order for node a to trigger an optimization round trying to replace his current link to b for a new link to c , c must be in a 's passive view, and be sampled by its local oracle. As long as c is in a 's passive view, there is always a probability greater than zero that c is selected to be sampled by the local oracle by the construction of the protocol. Therefore, in order to avoid the overlay from staying in configuration \mathcal{C} , without ever switching to configuration \mathcal{C}' , it is enough to ensure that eventually c will be part of a 's passive view.

Passive views are updated periodically through the exchange of samples of peers extracted both from active and passive views. These samples are exchanged through random walks, that are forwarded across neighbors in the active view of nodes. As experimental results reported in the paper have shown, X-BOT ensures that the in-degree of all peers in the system is approximately the same for all nodes in the system (considering the overlay denoted by the closure

of all peers active views). Furthermore, when nodes initiate the process of updating their passive view they include their identifier in the sample they exchange. Therefore, while c is active, there will always be some passive views in the system that contain c 's identifier. As these views are shuffled, eventually c will be in a 's passive view with some probability greater than zero.

This would allow node a to sample the link cost to c using its local oracle, and therefore trigger an optimization round that would replace his current link with b for a new link with c and the overlay to move from configuration \mathcal{C} to \mathcal{C}' .

Note however that X-BOT is a localized algorithm. Therefore it cannot perform optimizations and achieve overlay topologies that require global knowledge to be achieved.

4.6.3 Ensuring Low Clustering Coefficient

Experimental results have shown that X-BOT is able to maintain a small clustering coefficient. This section provides the rationale that justifies why the clustering coefficient only increases slightly despite the fact that the overlay topology is biased to promote some form of locality (taking into consideration an efficiency criteria encoded in the companion oracle). There are two main factors that contribute to maintain the clustering coefficient of the biased overlay network (relatively) low:

- The maintenance of unbiased neighbors which are implicitly biased to be distant neighbors selected at random (remember that the selection of the unbiased neighbors of a node is performed when nodes are filling their active views; during this time X-BOT do not execute its optimization protocol). As a result, the probability that two neighboring nodes to share the same unbiased nodes is very low (this is supported by experimental observations);
- Biased neighbors are not deterministically selected. On the contrary, X-BOT relies on a continually changing random sample of nodes (the passive view) to locate suitable candidates for its biasing procedure. Therefore, there is a high probability that neighboring peers sample different candidates during the execution of X-BOT and therefore, select distinct peers for their respective active views.

4.6.4 Ensuring Low Average Shortest Path

Experimental results presented earlier in this chapter have shown that the average shortest path between any pair of nodes in the overlay network that results from the operation of X-BOT is very similar to that of a random overlay.

This happens due to the existence of unbiased neighbors that, as discussed above, promote the existence of distant neighbors. Notice that if each node maintains a distant link, there are still a significant number of these links, that can be used to access remote areas of the overlay in a single hop. Also, because these links are selected at random, there is a high probability that neighboring nodes have unbiased nodes which are positioned in different regions of the overlay. This contributes to the existence of short paths between any pair of nodes in the overlay.

4.7 Related Work

The adaptation of unstructured overlay topologies to better match the underlying physical topology (to address the topology mismatch problem) or to better match applications requirements have been studied before. In this section we survey, and compare with X-BOT, some of the most relevant works in the field.

Narada (Chu et al., 2002) includes self-organizing protocols to construct and maintain overlay networks. Their approach is based in a utility function that is applied periodically to (some) peers; the output of the utility function is used to take local decisions concerning the addition and removal of links to the overlay. Because Narada is targeted at small and medium scale systems, it operates using full membership information and, therefore, scales poorly.

A work by I. Gupta et al. (2006) also aims at increasing gossip efficiency by eliminating overlay links that transverse a given physical link multiple times. However, it relies on the fact that peers can be organized in a hierarchical manner, for instance, by grouping peers by network domains (e.g. Local Area Networks, Subnets or even Autonomous Systems). X-BOT does not require knowledge concerning the physical location of nodes nor the maintenance of any hierarchy among peers, being therefore a more generic solution.

The Localiser algorithm (Massoulie et al., 2003) is an algorithm that aims both at optimizing

unstructured overlays according to a proximity criterion and to promote the balancing of node degrees. Localiser is based on a Metropolis scheme where nodes, iteratively, strive to minimize an energy function, by swapping connections. Although this algorithm strives to balance the degree of nodes in the system, unlike *X-BOT*, it does not ensure a constant degree in nodes that participate in the optimization. Furthermore, the Localiser algorithm is more likely to fall in local minima of the energy function, due to the fact the pool of peers available to generate overlay optimization is limited. Therefore, sometimes the algorithm has to perform adaptations that increase the energy function, which compromises the stability of the overlay. Moreover, Localiser does not attempt to preserve low clustering and small diameter.

The work of Rostami e Habibi (2007) proposes a mathematical model for measuring the degree of matching between an overlay network topology and the underlying physical topology (Rostami & Habibi, 2007). Similar to *X-BOT*, this metric is based on a link cost notion that can either be calculated by measuring the latency between two peers or the number of physical hops that separate two overlay neighbors. The authors propose an heuristic and an algorithm to lower the mismatch between the overlay topology and the physical network topology. However, and contrary to *X-BOT*, their algorithm requires nodes to exchange their complete partial views. This creates a significant instability in the overlay, and may disrupt the operation of protocols that are executed on top of it.

Hsiao et al. (2009) also propose an approach to address the topology mismatch problem in unstructured overlay networks. Their solution however is only tailored for minimizing the latency between neighboring peers. Additionally, and contrary to *X-BOT*, their solution requires each peer to have an estimate of the total number of peers in the system. To allow the protocol to gather the information necessary to support its operation, each peer is required to sample potentially hundred of nodes latencies before joining the overlay. The authors propose the use of a mechanism based on global coordinates to infer potential latency values between peers to mitigate the high cost associated with the execution of their solution.

GoCast (Tang & Ward, 2005) and Araneola (Melamed & Keidar, 2004) also include mechanisms to bias the topology of overlay networks maintaining symmetric partial views supported by TCP connections. GoCast builds an overlay which is optimized to maintain both random (distant) neighbors and close neighbors while balancing node degree in such a way that degree of nodes converge to a given pre-established value D and varies only between $D-2$ and $D+2$.

Araneola is similar to GoCast in the sense that it also builds an unstructured overlay network that presents several properties of k -regular graphs. Similar to GoCast, Araneola controls its topology to take into consideration some network metric, ensuring that better links are kept with a larger probability. However, these protocols rely on mechanisms to bias the overlay that are more complex and where each node makes independent decisions. In sharp contrast *X-BOT* uses a coordinated 4-node optimization technique that is simpler and allows to improve the overlay topology while maintaining a better node degree distribution. This chapter has presented comparative results for the performance of these systems with that of *X-BOT*, which have clearly showed the benefits of *X-BOT*.

T-Man (Jelasy et al., 2009) is a generic topology management scheme for overlay networks that is able to evolve a given overlay topology to a desired target topology (such as a Torus or a ring). This is achieved by having neighbors periodically exchanging their partial views. Both nodes update their partial views by merging these views and selecting the best c nodes, where c is the size of a partial view. The selection is based on a single ranking function which captures the desired topology, in the sense that it enables each node in the system to extract clues on its optimal position and the correct neighbors in the overlay network. Unlike *X-BOT*, T-Man does not aim at protecting relevant properties of the original overlay, nor it ensures the stability of in-degree of nodes during the optimization of the overlay (which may result in the overlay becoming partitioned as illustrated by the results previously presented in this chapter).

Summary

In this chapter we proposed and evaluated *X-BOT*, a new protocol that allows an unstructured overlay network to bias its topology according to a target efficiency criteria. The challenge addressed by this contribution was to improve the overlay topology by selecting better links without losing the relevant properties of the original overlay (such as the low clustering coefficient, in-degree distribution, and high robustness to peer failures and fast recovery) by exploring a topology management technique operating at the overlay network layer which bias the topology iteratively using a 4-node coordination mechanism.

Experimental results have demonstrated that *X-BOT* is able to improve the overlay topol-

ogy of an unstructured overlay network to more efficient configurations than previous approaches in several distinct scenarios. A significant feature of *X-BOT* is its ability to promote overlay connectivity, by preserving node degrees and avoid excessive clustering among peers in the overlay. As a result, *X-BOT* is able to support efficient and resilient gossip-based broadcast services when equipped with an appropriate oracle (e.g. a latency oracle). Moreover, *X-BOT* is able to recover from failures faster than previous proposed solutions.

In the next chapter the Thicket protocol is introduced. Contrary to *X-BOT* and CellFarm which operate at the overlay layer, Thicket explores an unstructured overlay topology management technique named *embed* which operates at the P2P service layer.

Publications

The work presented in this chapter has been published through the following publications:

On the Structure of Unstructured Overlay Networks (fast abstract). J. Leitão, J. Pereira and L. Rodrigues. In Supplement of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Anchorage, Alaska, USA, June, 2008.

On Adding Structure to Unstructured Overlay Networks. J. Leitão, N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. In Handbook of Peer-to-Peer Networking, X. Shen, H. Yu, J. Buford, M. Akon (Eds.), Springer 2010. pp. 327-365. ISBN: 978-0-387-09750-3.

X-BOT: A Protocol for Resilient Optimization of Unstructured Overlays. J. Leitão, J. P. Marques, J. Pereira and L. Rodrigues. Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems, Niagara Falls, New York, U.S.A., Sep, 2009. pp. 236–245.

X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks. J. Leitão, J. P. Marques, J. Pereira, and L. Rodrigues. IEEE Transactions on Parallel and Distributed Systems (Published online January 2012).

5 Embed the Topology: Thicket

This chapter introduces and evaluates Thicket, a protocol that relies on the embed approach for managing the topology of unstructured overlay networks at the P2P service layer. The remaining of this chapter is organized as follows: Section 5.1 motivates this contribution and defines the goals of Thicket; Section 5.2 introduces fundamental concepts for understanding the design of Thicket, namely it describes the assumptions made concerning the properties of the underlying unstructured overlay networks and briefly discusses the design of Plumtree, which is a starting point for the design of Thicket. Thicket is described in Section 5.3 and the case study which is used to demonstrate the benefits of Thicket is introduced in Section 5.4. Experimental results that evaluate the performance of Thicket, and the effect of Thicket in the case study are presented in Section 5.5. Finally, related work is presented and discussed in Section 5.6.

5.1 Motivation and Goals

5.1.1 Motivation

Scalable, efficient, and robust mechanisms to support the dissemination of information to a very large number of participants are extremely relevant for a wide range of applications, including, among others, large scale monitoring and control infrastructures (Liang et al., 2005), query dissemination (Baltoni et al., 2010), live multimedia streaming (Frey et al., 2010), and IP Television (IPTV) services (Hei et al., 2007). The contribution presented in this chapter aims at improving dissemination services such as these, where there is time constraints to deliver disseminated data to all participants. This is achieved by exploiting decentralized P2P cooperation among all participants (as opposed to solutions that use centralized components or assume the availability of an underlying IP-multicast service). The P2P approach has already proved successfully in circumventing the difficulties faced when attempting to deploy global

IP-multicast support (Deering & Cheriton, 1990; Diot et al., 2000).

More precisely, this contribution combines features of gossip-based dissemination with features of systems based on multicast trees. From gossip-based systems we use the ability to operate on top of low-cost unstructured overlays and the resiliency inherent to the epidemic propagation. From tree-based systems we exploit the efficient use of resources, eliminating most of the undesired redundancy of the pure gossip-based approach (Eugster et al., 2003). Note that, tree-based approaches suffer from two major drawbacks: *i*) interior nodes support a much higher load than leaf nodes; *ii*) the failure of a single interior node is able to break the tree compromising the reliability of the dissemination service. These limitations are circumvented not only by incorporating gossip mechanisms to quickly repair the tree but also by using multiple trees, such that each node is interior in just one, or few, trees and a leaf node in the remaining. Multiple trees allow to achieve load distribution and also offer the opportunity to send (controlled amounts of redundant) information over the different trees for continuity of service if one of the trees breaks (*i.e.*, during tree repair).

The resulting approach can be used in applications such as live-streaming. By leveraging on network coding techniques (Frey et al., 2010) it is possible to split the original data stream in several slices and send these slices through different trees. Slices may encode enough redundancy such that, if a node temporarily misses messages from one of the trees, it is still able to decode the original stream using the remaining slices received from the correct trees.

5.1.2 Goals

Several works have explored solutions for embedding spanning trees over overlay network. These solutions cover a significative design space. In one hand, centralized solutions have been proposed which embed a single tree, take for example the Bayeux system (Zhuang et al., 2001) and also to embed multiple trees for instance, the CoopNet system (Padmanabhan et al., 2002). In the other hand, several works explore a decentralized approach either over structured or unstructured overlay networks.

In the case of structured overlay networks, Scribe (Rowstron et al., 2001) is a system which embeds a single spanning tree over the Pastry DHT (Rowstron & Druschel, 2001). An example of a system which embed multiple spanning trees over a single structured overlay network can be found in the Splitstream protocol (Castro et al., 2003). For the particular case of unstructured

overlay networks, Mon (Liang et al., 2005) and Plumtree (Leitão et al., 2007a) are examples of protocol that embed a single spanning tree.

Contrary to the previous examples, we aim at devising a solution which explores a relatively unexplored region of the design space, where we rely on highly robust unstructured overlay networks which present a low maintenance overhead to embed several interior-node-disjoining spanning trees in a decentralized fashion, as to contribute to the reliability of dissemination schemes. To the best of our knowledge, Chunkyspread (Venkataraman et al., 2006) is the only other solution which has explored this particular area of the design space. We will discuss the relation of this work with Thicket further ahead in the chapter.

In summary, we aim at designing a solution that combines the following features: *i*) it embeds trees in a peer-to-peer overlay, as this offers a good trade-off between efficiency and robustness; *ii*) is fully decentralized; *iii*) is able to build multiple-tree that have few interior nodes in common; and *iv*) can operate on top of unstructured overlays.

5.2 Preliminaries

In this section we discuss the assumptions that are made concerning the properties of the underlying unstructured overlay network which supports the operation of Thicket. Furthermore, we briefly discuss the Plumtree protocol which serves as a starting point for the design of Thicket. Note that the design of Plumtree makes the same assumptions concerning the properties of the underlying overlay as Thicket.

5.2.1 Underlying Unstructured Overlay Network

Both Plumtree and Thicket designs are based on the assumption that the underlying overlay owns the following essential properties:

Connectivity: The overlay must be connected. As discussed previously, this means that there should be at least one path in the overlay connecting any two pair of nodes in the system. Thus, all participants should have in their neighbor sets that materialize the unstructured overlay network, at least, another correct peer; all participants should be in the partial view of, at least, a correct peer and; all peers should belong to a single cluster.

Reactive membership: The neighbor sets provided by the protocol that manages the unstructured overlay network must remain unchanged in a stable environment. Changes to the contents of neighbor sets should only happen due to changes in the global system membership (*i.e.*, when new peers join the system or when peers fail). Additionally, such changes should be localized, affecting only a small number of participants in the system. The reason for this is that the stability of the spanning tree structures depend on the stability of the underlying unstructured overlay network topology; when a peer is added or removed to the neighbor set of another participant, it might produce changes over the set of links being employed for supporting a spanning tree. Therefore, spurious changes to the neighbor sets are undesirable and should be avoided.

Symmetric overlay links: If overlay links are symmetric, when peer b is in the neighbor set of peer a , then peer a must be in the neighbor set of peer b . This translates into an unstructured overlay network where all links are bi-directional, or in other words, in an unstructured overlay network that denotes an undirected graph. If the links that form the spanning tree are symmetric, a single spanning tree can be shared by multiple sources (*i.e.*, multiple peers can use it to disseminate their messages). Symmetric overlay links render the task of creating bi-directional trees easier, and reduce the amount of state that has to be maintained by each participant in the system. Additionally, previous work (Leitão et al., 2007b) has shown that ensuring symmetric partial views improves the overall connectivity and in-degree balancing of unstructured overlay networks, which results in more robust and balanced overlay topologies.

Scalable: Both Plumtree and Thicket are aimed at supporting large-scale distributed applications that can be deployed over the Internet. Therefore, the protocol that maintains the underlying unstructured overlay network should be able to operate correctly in such large scales (*i.e.* systems with several thousands of participants). This means that the overhead for maintaining the unstructured overlay correct (and their previously mentioned properties) should be either constant or grow logarithmically with the size of the system.

Additionally, both Thicket and Plumtree assume that the protocol that manages the unstructured overlay network is responsible for notifying the P2P service layer whenever there is a change on the neighbor set of the peer using, respectively, $NeighborUp(p)$ and $NeighborDown(p)$ calls (where p stands for the identifier of the peer being added or removed from that

participant neighbor set).

From the implementation point of view, we have used the HyParView protocol (Leitão et al., 2007b) for maintaining the unstructured overlay to support our solution. HyParView uses TCP connections to support all message exchanges between nodes. Although other reactive solutions could be used, in order to keep the discussion concrete, from now on we just assume that the remaining blocks of our solution are built on top of this protocol.

5.2.2 Plumtree Protocol

As discussed previously, the goal of Thicket is to achieve a dissemination solution that employs several interior-node-disjoint spanning trees embedded on top of a single unstructured overlay network. However, to achieve such goal, one has to devise an efficient and robust mechanism to embed a single spanning tree over the overlay. The design of Plumtree illustrates how one can employ the previously described approach to embed a single spanning tree over an unstructured overlay. Plumtree serves as an initial step for designing Thicket. Plumtree has two main components namely, a tree construction and a tree repair mechanism that we now describe.

The Plumtree approach is based on the following design principles. We disseminate messages as any pure gossip protocol, in the sense that, to broadcast a message, each node contacts f neighbors extracted from the underlying unstructured overlay network (where f is the protocol fanout). However, each peer uses a combination of eager push and lazy push gossip. Eager push is used just for a subset of the f overlay neighbors, while lazy push is used for the remaining peers. The links used for eager push are selected in such a way that their closure effectively builds a spanning tree embedded in the underlying unstructured overlay network. Lazy push links are used to ensure gossip reliability when nodes fail and also to quickly heal the tree. Furthermore, contrary to typical gossip solutions, the set of (random) eager and lazy peers is not changed at each gossip round. Instead, the same peers are used until failures are detected by the underlying overlay layer.

5.2.3 Tree Construction

Plumtree maintains two sets of peers: the *eagerPushPeers*, that contains identifiers of peers with whom the node uses eager-push and *lazyPushPeers*, with which it uses lazy-push. Initially, *eagerPushPeers* contains f random peers, that are obtained from the neighbor set provided by the underlying overlay layer, and *lazyPushPeers* is empty. Therefore, in the first rounds, the protocol operates as a pure eager-push gossip protocol.¹

We use a mechanism to construct the spanning tree that is similar to that proposed in (Jiang & Zhang, 2003). After the initialization of the *eagerPushPeers* set described above, participants embed the spanning tree by moving neighbors from *eagerPushPeers* to *lazyPushPeers*, in such a way that, as the protocol evolves, the overlay defined by the first set becomes a tree. When a duplicate is received, its sender is moved to the *lazyPushPeers*. Furthermore, a PRUNE message is sent to the sender of the duplicate message such that, in response, it also moves the link to the *lazyPushPeers*. This procedure ensures that, after a single message dissemination a single spanning tree has been embedded.

One interesting aspect of this process is that, in face of a stable network (*i.e.* with constant load), it will tend to generate a spanning tree that minimizes the message latency (as it only keeps the path that generates the first message reception at each peer).

As soon as overlay neighbors are added to the *lazyPushPeers* set, messages start being propagated using both eager and lazy push. Lazy push is implemented by sending SUMMARY messages, that only contain the broadcast ID of disseminated messages, to all elements of the *lazyPushPeers* set. Note however that, to reduce the amount of control traffic, SUMMARY messages do not need to be sent immediately. A scheduling policy is used to piggyback multiple message identifiers in a single control message. The only requirement for the scheduling policy is that all received message identifiers are (eventually) disseminated in a SUMMARY message.

5.2.4 Tree Repair

When a failure occurs, some tree branches may become disconnected. Therefore, eager push is not enough to sustain full system coverage in face of failures. The lazy push SUMMARY

¹The fanout value f must be selected such that the overlay defined by the *eagerPushPeers* of all nodes is connected.

messages exchanged through the remaining links of the unstructured overlay are not only used to recover missing messages but also to provide a quick mechanism to repair the embedded spanning tree.

When a peer receives a SUMMARY message, it simply marks the (payload) messages whose identifiers are contained in the message as missing. It then starts a timer, with a predefined timeout value, and waits for the missing messages to be received via eager push. The timeout value is a protocol parameter that should be configured considering the diameter of the overlay and a target maximum recovery latency, defined by the application requirements.

When the timer expires at a given peer p , p selects the first announcement it has received in a SUMMARY for each missing messages. It then sends a GRAFT message to the source of the corresponding SUMMARY message. The GRAFT message has a dual purpose. In first place, it triggers the transmission of the missing messages payload. In second place, it adds the corresponding underlying overlay link to the spanning tree, effectively recovering it. When a GRAFT message is sent, another timer is initialized to expire after a certain timeout, to ensure that the message will be requested to another neighbor if it is not received meanwhile. This second timeout value should be smaller than the first, in the order of an average round trip time to that neighbor.²

5.2.5 Network Dynamics

We now describe how Plumtree reacts to changes in the system membership. As discussed previously, these changes are notified by the underlying overlay layer using the *NeighborDown* and *NeighborUp* notifications. When a neighbor is detected to leave the overlay, it is simply removed from either the *eagerPushPeers* or *lazyPushPeers* set. Additionally, the record of any SUMMARY messages received from that peer are deleted from the missing history. When a new member is added to the neighbor set maintained by the underlying overlay layer, that new overlay neighbor is added to the *eagerPushPeers* set *i.e.*, the overlay link to that peer is considered a candidate to become part of the spanning tree. The interested reader can refer to the original paper (Leitão et al., 2007a), where the full description of Plumtree is provided.

²This value can be extracted from the internal data structures maintained by TCP.

5.3 The Thicket Protocol

5.3.1 Rationale

We now address the challenges in designing a decentralized algorithm for building T trees on top of an unstructured overlay. At first sight such a goal may appear easy to be achieved. In particular, it is tempting to consider an algorithm that is a trivial extension of Plumtree, namely, the following two alternative solutions appear as natural candidates:

- Following the approach employed by SplitStream (Castro et al., 2003), one could select T proxies of the root at random (for instance, by doing a random walk from the source node on top of the overlay network), and then build a different tree rooted at each of these proxies by leveraging the Plumtree protocol. This approach can also be seen as a simplified version of the Chunkspread protocol. We have named this approach the *Naive Unstructured splitStream*, or simply, NUTS.³

- One may also consider the simple solution that consists in creating T random unstructured overlays (by running T instances of HyParView in parallel) and then embedding a different tree over each one of these overlays. The intuition is that the inherent randomization in the construction of the unstructured overlays (and of the embedded trees) would be enough to create trees with enough diversity. We have named this approach *Basic multiple OverLay-TreeS*, or simply, BOLTS.

We have implemented these two naive strategies to assess how good they perform in practice. We analyze their resulting performance to extract some guidelines for the design of Thicket. In order to experiment the NUTS approach, we have constructed a single HyParView overlay and used Plumtree to create T trees rooted at random nodes in the overlay. To experiment the BOLTS strategy we have created T independent instances of the HyParView overlay (by letting nodes join each instance by different random orders) and then embedded a single tree in each of these instances using Plumtree.

We evaluated both strategies by simulating a system composed of 10,000 nodes, and aimed at embedding 5 independent spanning trees (we will describe the experimental setup em-

³We discuss the operation of both SplitStream and Chunkspread (Venkataraman et al., 2006) further ahead in section 5.6.

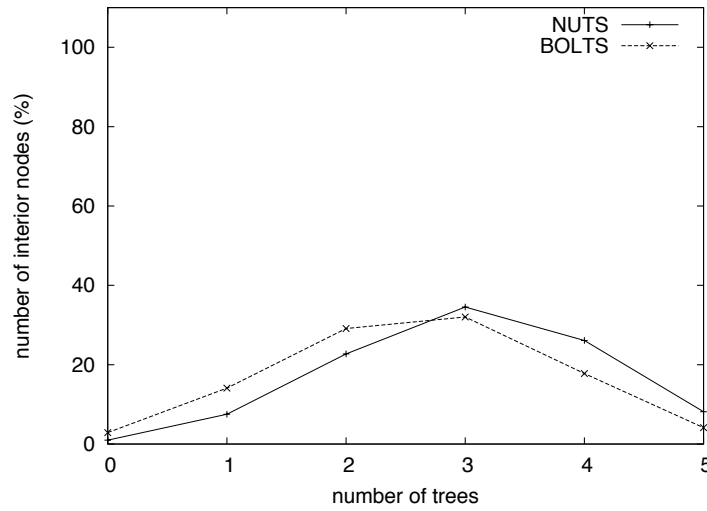


Figure 5.1: K -interior node distribution over 5 trees for the NUTS and BOLTS strategies.

ployed in detail in section 5.5). For NUTS we employed a single HyParView instance with a node degree of 25. For BOLTS we configured each of the HyParView instances to have a node degree of 5. The *fanout* value f used by the Plumtree instances was set to 5 which is related to the number of neighbors maintained by HyParView for that number of peers (Leitão et al., 2007b). These configurations ensure that each peer has an identical number of overlay links in both approaches. Figure 5.1 plots the percentage of nodes that are interior in 0, 1, 2, 3, 4, and 5 trees respectively.

The figure shows that, in both evaluated strategies, only a small fraction of peers (between 7% and 17%) are interior in a single tree. The majority of participants in the system are interior in either 2, 3, or 4 trees (with a small fraction being interior in all trees for both strategies). Notice that, for BOLTS, there are some peers that do not act as interior nodes in any tree (*i.e.*, 0). These peers do not contribute to the data dissemination process. This clearly shows that these strategies create (even in steady state) suboptimal configurations, where many participants are required to forward messages in more than a single tree. Additionally, this also indicates that the single failure of a peer can disrupt the operation of a significant number, or even all, spanning trees, which clearly compromises the reliability of the data dissemination process.

These results can be explained by the random and uncoordinated nature of the tree construction process, in which each tree is built in an independent way. In fact, although a large measure of randomness is implicit in the topology of unstructured overlay networks for the

Algorithm 9: Thicket: Data Structures & Initialization

```

1  data structure Tree
2    field activePeers : Set

3  data structure Load : int[]

4  upon event Init do
5    foreach  $t \in \text{trees}$  do
6       $t.\text{activePeers} \leftarrow \emptyset$ 
7       $\text{backupPeers} \leftarrow \text{getPeers}()$ 
8       $\text{announcements} \leftarrow \emptyset$ 
9       $\text{receivedMsgs} \leftarrow \emptyset$ 
10      $\text{loadEstimate}_p(t) \leftarrow \emptyset$ 

```

BOLTS solution, and the selection of peers is independent in the NUTS approach, there is still a significant probability that peers can be selected to be interior in several, or even all, trees.

5.3.2 Algorithm

Considering the results presented above, we now introduce Thicket, an algorithm which tackles the challenge of efficiently embedding several spanning trees over a single unstructured overlay network, while ensuring a fair load distribution in the system, by having most peers act as interior in a single tree.

5.3.3 Architecture

Thicket operates by employing a gossip-based technique to embed T interior-node-disjoint spanning trees (T is a protocol parameter, we discuss the adequate configuration of T further ahead in this chapter), where most peers are interior in a single tree and leaf in all other spanning trees. For that purpose, Thicket extends the operation of Plumtree in several aspects. In particular, Thicket uses the remaining overlay links, which are not used to embed any of the spanning trees, for the following purposes: *i*) ensure complete coverage of all existing trees *i.e.*, that all peers in the system are connected to all trees, notice that to ensure this, some participants may be required to be interior in more than a single tree; *ii*) detect and recover from tree partitions when peers either fail or leave the system; *iii*) ensure that tree heights are kept small, despite the existing dynamics on the system filiation; and finally, *iv*) that the forwarding load of each participant (for all trees where it operates as an interior node) is limited by a protocol parameter named *maxLoad*.

The last parameter, *maxLoad*, must be set according to the capacity of peers (given that it limits the forwarding load imposed on each participant). However, if the chosen value is too low, peers might be unable to coordinate among themselves to embed spanning trees with full coverage (*i.e.* that connect all participants). Following epidemic theory, *maxLoad* should be, at least, logarithmic with the number of peers in the system.

Algorithm 9 depicts the data structures maintained by Thicket, as well as its initialization procedure. Each peer p in Thicket keeps a set of *backupPeers_p*; with the identifiers of the neighbors that are not being used to receive (or forward) messages in any of the T trees. This set serves a purpose similar to the *lazyPushPeers* set of Plumtree. However, contrary to Plumtree, initially all (overlay) neighbors of p are contained in this set. Additionally, for each tree t maintained by Thicket, each peer p maintains a set *t.activePeers_p* with the identifiers of the neighbors from which it receives (or forwards to) data messages in t (similarly to the *eagerPushPeers* set maintained by Plumtree).

In tandem with Plumtree design, each peer p also maintains an *announcements_p* set, in which it stores control information received from peers which overlay links to p are not being employed to embed any of the T spanning trees (*i.e.*, overlay neighbors which are part of the *backupPeers_n* set). This information is used to detect and recover from tree partitions due to peer failures or departures. We will later explain in detail how the recovery procedure operates. To avoid routing loops, each peer p also maintains a *receivedMsgs_p* set, with identifiers of messages previously delivered and forwarded by a p .

Finally, to balance the load imposed on peers, *i.e.*, to ensure that most participants are only interior in a single tree and to limit the message forwarding load imposed on each individual peer, each peer p keeps an estimate of the forwarding load of its neighbors. For this purpose, every time a peer s sends a message to another participant, it includes a list of values denoting the number of overlay neighbors to which s has to forward messages in each tree.⁴ Since this information can be encoded efficiently, it is piggybacked in all data and control messages exchanged between neighbors. This allows every peer to keep fresh information about the load of its overlay neighbors without explicitly exchanging messages just for this purpose. Each node p maintains the most recent information received from its neighbor n for each tree t in the local

⁴We assume that tree identifiers are sequential numbers starting at zero. This list has a size of T . The number in position t represents the forwarding load of that peer in tree t (which is the size of *t.activePeers_p*, minus 1).

Algorithm 10: Thicket: Tree Construction

```

1  upon event Broadcast( $m$ ) do
2    tree  $\leftarrow$  nextTree()
3    muid  $\leftarrow$  (nextSqn(), tree)
4    if tree.activePeers =  $\emptyset$  then
5      call SourceTreeBranching(tree)
6    call Forward ( $m$ , muid, tree, myself)
7    trigger Deliver( $m$ )
8    receivedMsgs  $\leftarrow$  receivedMsgs  $\cup$  {muid}

9  upon event Receive (DATA,  $m$ , muid, load, tree, sender) do
10   if muid  $\notin$  receivedMsgs then
11     trigger Deliver( $m$ )
12     receivedMsgs  $\leftarrow$  receivedMsgs  $\cup$  {muid}
13     if  $\forall (id) \in$  missingFromTree(announcements, tree): id=muid then
14       cancel Timer(mID)
15       announcements  $\leftarrow$  removeMuid(muid, announcements)
16     if tree.activePeers =  $\emptyset$  then
17       if sender  $\in$  backupPeers then
18         tree.activePeers  $\leftarrow$  tree.activePeers  $\cup$  {sender}
19         backupPeers  $\leftarrow$  backupPeers  $\setminus$  {sender}
20         call treeBranching(tree)
21       call Forward ( $m$ , mID, round+1, tree, myself)
22       call Balance (mID, mask, tree, sender)
23     else
24       tree.activePeers  $\leftarrow$  tree.activePeers  $\setminus$  {sender}
25       backupPeers  $\leftarrow$  backupPeers  $\cup$  {sender}
26     trigger Send(PRUNE, sender, tree, myself)

27 procedure SourceTreeBranching (tree) do
28   peers  $\leftarrow$  getRandomPeers(backupPeers,  $f$ )
29   foreach  $p \in$  peers do
30     tree.activePeers  $\leftarrow$  tree.activePeers  $\cup$  { $p$ }
31     backupPeers  $\leftarrow$  backupPeers  $\setminus$  { $p$ }

32 procedure TreeBranching (tree) do
33   if  $\nexists t \in$  trees :  $|t.activePeers| > 1$  then
34     peers  $\leftarrow$  getRandomPeers(backupPeers,  $f - 1$ )
35     foreach  $p \in$  peers do
36       tree.activePeers  $\leftarrow$  tree.activePeers  $\cup$  { $p$ }
37       backupPeers  $\leftarrow$  backupPeers  $\setminus$  { $p$ }

38 every  $T$  seconds do
39   if  $\sum_t$  Load  $<$  maxLoad then
40     SUMMARY  $\leftarrow$  GetNewSummary (receivedMessages)
41     foreach  $p \in$  backupPeers do
42       trigger send(SUMMARY, Load)

43 procedure Forward ( $m$ , muid, tree, sender) do
44   foreach  $p \in$  tree.activePeers:  $p \neq$  sender do
45     trigger Send(DATA,  $p$ ,  $m$ , muid, Load, tree, myself)

46 upon event Receive (PRUNE, load, tree, sender) do
47   tree.ActivePeers  $\leftarrow$  tree.ActivePeers  $\setminus$  {sender}
48   BackupPeers  $\leftarrow$  BackupPeers  $\cup$  {sender}

```

variable $loadEstimate(n, t)_p$.

5.3.4 Tree Construction

Algorithm 10 depicts a simplified version of the pseudo-code for the tree construction procedure. We have omitted some obvious aspects from the pseudo-code (for instance the update of *loadEstimate*) to improve its readability.

The creation of each tree t is initiated by the source node (Algorithm 10, lines 1 – 8). To that end, and for each tree t , the source s selects f nodes at random from the *backupPeers_s* set and moves them to the *t.activePeers_s* set (procedure *SourceTreeBranching* denoted in Algorithm 10, lines 27 – 31). After this, the source initiates the dissemination of data messages in each tree t , by sending messages to the nodes in *t.activePeers_s* (by employing eager push) (Algorithm 10, line 6).

All messages are tagged with a unique identifier, *muid*, composed of the pair (*sqnb*, t), where *sqnb* is a sequence number and t the tree identifier (Algorithm 10, line 3). The *muids* of previously delivered (and forwarded) messages are stored in the *receivedMsgs_p* set of each peer p (Algorithm 10, line 12). Periodically, each peer p sends a SUMMARY of this set to all overlay neighbors in its *backupPeers_p* set (Algorithm 10, lines 38 – 42). Note that these messages also include load information used to update the *loadEstimate* data set.

When a peer p receives a data message from s through an overlay link being used to embed tree t , it first checks if tree t has already been created locally (Algorithm 10, line 16). The first message that is received through a given tree t triggers the local tree branching procedure for t (Algorithm 10, line 20). The construction step for an interior node is different from the one executed by the source node. First, p removes s from *backupPeers_p* and adds s to *t.activePeers_p*. Furthermore, if $\nexists t' : |t'.activePeers_p| > 1$ (i.e., the participant is not interior in some other tree t'), then p moves at most $f - 1$ peers from *backupPeers_p* to *t.activePeers_p* (procedure *TreeBranching* denoted in Algorithm 10, lines 32 – 37). Otherwise, if p is already an interior node in some other tree, it stops the branching process, becoming a leaf node in tree t (as noted by the lack of an *else* block in the procedure *TreeBranching* in Algorithm 10, lines 32 – 37).

The data message is then processed. If the message is not found to be a duplicate (by inspecting the *receivedMsgs_p* set), it is forwarded to the nodes in *t.activePeers_p \ {s}* (Algorithm 10, line 21). On the other hand, if the received message is a duplicate, the node moves s from *t.activePeers_p* to *backupPeers_p* and sends a PRUNE message back to s (Algorithm 10, lines 23 – 26).

Algorithm 11: Thicket: Tree Repair

```

1  upon event Receive (SUMMARY, load, sender) do
2    foreach (muid, p) ∈ SUMMARY: muid ∉ receivedMsgs do
3      if ∄ Timer(t) : t = muid.t then
4        setup timer Timer(muid.t, timeout)
5        announcements ← announcements ∪ {(muid, sender)}

6  upon event Timer(tree) do
7    (muid, p) ← removeBest(announcements, tree)
8    tree.activePeers ← tree.activePeers ∪ {p}
9    backupPeers ← backupPeers \ {p}
10   trigger Send(GRAFT, p, null, loadEstimatep, tree, myself)

11 upon event Receive (GRAFT, muid, load, tree, sender) do
12   if  $\sum_t Load < maxLoad \wedge sender \in tree.backupPeers \wedge$ 
13      $(|tree.activePeers| > 1 \vee load = Load)$  then
14     tree.activePeers ← tree.activePeers ∪ {sender}
15     backupPeers ← backupPeers \ {sender}
16   else
17     trigger Send(PRUNE, sender, Load, tree, myself)

18 procedure Balance (muid, load, tree, sender) do
19   if ∃ (id, p) ∈ announcements : id.t = tree then
20     newLoad ← IncTreeLoad(loadEstimatep, tree)
21     if  $nInterior(newLoad) < nInterior(load)$  then
22       trigger Send(GRAFT, n, null, loadEstimatep, t, myself)
23       trigger Send(PRUNE, sender, Load, tree, myself)

```

Upon receiving the PRUNE message, s will move p from $t.activePeers_s$ to $backupPeers_s$ (Algorithm 10, lines 46–48). This message operates in a similar fashion to PRUNE message employed in the design of Plumtree.

By executing this algorithm, participants in the system become interior in at most one spanning tree. The algorithm also promotes load balancing (as long as the number of data messages sent through each tree is similar). On the other hand, since this mechanism selects random peers for embedding each tree, there is a non-negligible probability that some participants are not connected to every tree at the end of this process. Such occurrences are addressed by the tree repair mechanism that is described in the following text.

5.3.5 Tree Repair

The goals of the tree repair mechanism are twofold: *i*) it ensures that all peers eventually become connected to all T spanning trees and, *ii*) it detects and recovers from tree partitions that might happen due to failure or departure of peers. This component relies on the SUMMARY messages disseminated periodically by each participant. We recall that SUMMARY messages contain the identifiers of disseminated data messages recently added to the *receivedMsgs* set.

More precisely, each SUMMARY message contains the identifiers of all messages received since the last SUMMARY message sent by that peer (Algorithm 10, lines 38 – 42).

When a peer p receives a SUMMARY message from an overlay neighbor s , it verifies if all message identifiers are recorded in its $receivedMsgs_p$ set (Algorithm 11, line 2). If no messages have been missed, the SUMMARY is simply discarded. Otherwise, a tuple $(muid, s)$ is stored in the $announcements_p$ set for each data message that has not been received yet (Algorithm 11, line 5). Furthermore, for each tree t where a message has been detected to be missing, a *repair timer* is initiated (Algorithm 11, lines 3 – 4); if the missing messages have not been received by the time this timer expires, peer p assumes that it has become disconnected from that tree and takes measures to repair it (Algorithm 11, lines 6 – 10).

The tree repair procedure is significantly different from the one employed in the design of Plumtree, as one has to take into consideration that peers should only be interior in a single tree and additionally, it should also be ensured that no participant is required to forward more messages than $maxLoad$ considering all trees where it acts as an interior node. Consider that peer p has received from a set of overlay neighbors S a SUMMARY message with the $muid$ of a data message detected to be lost in tree t . Peer p is going to select a single target overlay neighbor $s_t \in S$ to repair the tree t . The selection procedure uses the information kept by each participant concerning the load of their peers (taking into consideration the locally stored load estimates). Namely, s_t is selected at random among all peers in S for which the forwarding load is below the threshold ($maxLoad$) and that are estimated to be interior nodes in a smaller number of trees or that are already interior in t (this selection strategy is encoded in method `removeBest` which is employed in Algorithm ??, line 7).

After selecting s_t , p performs the following two steps: s_t is removed from $backupPeers_p$ and added to $t.activePeers_p$ and a GRAFT message is sent to s_t (Algorithm 11, lines 8 – 10). The GRAFT message includes the current view of n concerning the load of s_t (note that n 's information about s_t may be outdated, as this information is only propagated through piggyback in other messages exchanged among participants).

When s_t receives a GRAFT message from p for tree t , it first checks if p based its decision on up-to-date values for the load of s_t (i.e., if the current forwarding load of s_t matches the information owned by p) or if, despite eventual inaccuracies in the estimate, s_t can nevertheless satisfy the request of p without increasing the number of trees where it is interior nor

increasing its current forwarding load to values above $maxLoad$. If this is the case, s_t moves p to $t.activePeers_{s_t}$ (Algorithm 11, lines 12 – 15). Otherwise, s_t rejects the GRAFT message by replying to p with a PRUNE message and since load information is piggybacked to all messages, this will also update p 's information on s_t 's load (Algorithm 11, lines 16 – 17).

Finally, if p receives a PRUNE message back from s_t , p will move back s_t from $t.activePeers_p$ to the $backupPeers_p$ and attempt to repair t by picking a new target from the $announcements_p$ set.

Algorithm 11 depicts a simplified version of this procedure in pseudo-code.

5.3.6 Tree Reconfiguration

The tree construction and repair procedures described earlier are able to embed several spanning trees with complete coverage, where the large majority of peers are interior in a single spanning tree (this happens due to the repair mechanism, as confirmed by experimental results presented further ahead in section 5.5). This is true in a stable environment (*i.e.*, when there are no joins or leaves in the system). However, multiple executions of the repair mechanism may lead to sub-optimal configurations where several peers become interior in more than a single tree while still owning a load below $maxLoad$.

To circumvent this problem, we developed a reconfiguration procedure that operates as follows: When a peer p receives a non-redundant data message from a peer s in a tree t for which it had previously received an announcement from third peer a , it compares the estimated loads of s and a (Algorithm 10, line 22).

If $\sum_t loadEstimate(s, t)_p > \sum_t loadEstimate(a, t)_p$ and p can replace the position of s in tree t without becoming interior in more trees, peer p attempts to replace the link between s and p by a link between a and p in that spanning tree. For this purpose, p sends a PRUNE message to s and a GRAFT message to a (Algorithm 11, lines 18 – 23).

Note that the reconfiguration is only performed if the announcement from a is received by peer p before the data message from s . This ensures that a reconfiguration contributes to reduce the dissemination latency in the tree while avoiding the construction of cycles. Additionally, as participants whose forwarding load reaches $maxLoad$ become unable to help their overlay neighbors on repairing spanning trees, they cancel the periodic transmission of SUMMARY messages.

Algorithm 12: Thicket: Overlay Network Dynamics

```

1  upon event NeighborDown(node) do
2    foreach tree ∈ trees do
3      tree.ActivePeers ← tree.ActivePeers \ {node}
4      BackupPeers ← BackupPeers \ {node}
5    foreach (muid,s) ∈ announcements : s = node do
6      announcements ← announcements \ {(muid,s)}

7  upon event NeighborUp(node) do
8    BackupPeers ← BackupPeers ∪ {node}

```

5.3.7 Network Dynamics

Similar to what happens with Plumtree, the overlay layer is responsible for detecting changes in the neighbor set maintained locally and for notifying Thicket when these changes occur, using the *NeighborDown(p)* and *NeighborUp(p)* notifications (see Algorithm 12).

When a peer n receives a *NeighborDown(p)* notification it removes p from all $t.activePeers_n$ sets and also from the $backupPeers_n$ set. Additionally, all records of announcements sent by p are also deleted from the $announcements_n$ set. This might result in peer n becoming disconnected from some trees (most of the times from a single tree). The tree repair mechanism however is able to detect and recover from this scenario in an expedite fashion.

On the other hand, and in contrast with Plumtree, when a peer n receives a *NeighborUp(p)* notification, it is required to add p to the $backupPeers_n$ set. As a result, p will start exchanging SUMMARY messages with n (and vice-versa as n will receive a similar notification for p as the underlying overlay network is assumed to have symmetric links). As explained above, these messages will allow not only joining peers to become connected to all spanning trees, but also to leverage on new overlay neighbors to balance the load imposed over peers that were already part of the system (using the tree reconfiguration mechanism).

5.3.8 Configuring Thicket

Parameter T: Considering the number of trees created (T) and the protocol *fanout* (f), the maximum value for parameter T is intimately related with the parameter f . In fact, take the case where f is equal to 2. In this scenario each tree is a binary tree where half the participants are interior. Therefore, in such a scenario only 2 trees can be built using the same overlay

without having a peer acting as interior in more than a single tree hence, $T \leq 2$. Therefore, the maximum number of trees (T) is limited by the fanout (f) used in branching the trees⁵.

Unstructured Overlay Degree: The degree of the unstructured overlay network should at least be equal to f (for the tree where each peer acts as interior) plus a link for each additional tree ($T - 1$, these links are used to receive the messages disseminated through the remaining tree) however, this would render a decentralized mechanism to build such trees infeasible. Therefore, we rely on overlay degrees in the order of $f * T$, which provides each peer with access to enough (random) overlay links to find suitable configurations for its role in all trees.

5.4 Case Study

In this section we discuss the case study used to evaluate the benefits that can be extracted from Thicket design. In particular we focus on a P2P streaming service used to disseminate music encoded in mp3 format. Furthermore, we provide details on the prototype implementation of Thicket which includes an implementation of the P2P streaming service that serves as a case study for the contribution presented in this paper.

5.4.1 P2P Steaming Service

As a case study we consider a P2P streaming service which relies on several co-existing spanning trees to convey multimedia content. We consider that the dissemination process leverages the co-existing spanning trees to introduce redundancy in the disseminated data (by using forward error correction techniques). Therefore, assuming that T spanning trees are available, the P2P streaming service will encode each data segment to be disseminated across all participants in T distinct messages using forward error correction techniques (Præsthholm et al., 2007), such that any participant can decode the data segment with any $T - 1$ of the T messages disseminated.

Each of the T messages that encode a data segment are disseminated using different spanning trees, which enables participants that become temporarily disconnected from one of the spanning trees to still receive enough messages to decode the data segment.

⁵We have determined the value of f used in our evaluation experimentally. This value is related with the *fanout* of gossip protocols that operate over symmetric overlay networks (Leitão et al., 2007b).

We provide additional details on the operation of this P2P streaming service in the context of the prototype implementation description that follows.

5.4.2 Prototype Implementation

We have implemented a prototype of Thicket and the considered P2P streaming service, that serves as a case study, in the Java language. This prototype was employed to evaluate the contribution presented in this chapter in the PlaneLab testbed. This section describes some of the relevant components included in the prototype implementation and discusses some challenges faced during the development process.

5.4.2.1 Components

To deploy a fully distributed system using Thicket at its core, some additional components were implemented in order to support the of full operation of the system. We now describe each of these additional components in more detail.

Transport Layer. A non-blocking interface to the transport layer was used to handle message exchanges among different nodes participating in the system. This layer supplied an interface to send messages without blocking the application and also a set of callbacks to notify the above protocols of the reception of messages received from the network and to which they previously registered their interest in receiving. This layer relies on non-blocking TCP sockets. A single connection is maintained for each peer (present in the neighbor set at the overlay layer), and that connection is shared by HyParView and Thicket to exchange messages with other peers. To implement this an outgoing queue of messages is associated to each TCP connection maintained by this layer. The *java.nio* extension was used to avoid blocking the application when attempting to transmit a message. Messages remain in the outgoing queue until they are able to be transmitted. Additionally, this layer monitors TCP connections and notify the peer sampling service when a connection fails, which is used by that layer to trigger a local unreliable failure detector.

Streaming Layer. The Streaming Layer was implemented to run both at the source and at the recipients. In the source, the stream to be broadcasted is written to the Streaming layer which

stores data in a buffer until its size is large enough to fill a packet. At this time, the packet(s) are sent using the Thicket protocol. At the recipients, the Streaming Layer waits until it receives the data chunks from the network and inserts them in an internal buffer, preserving their correct order, waiting for a read operation. If a read is performed and the next chunk is not available the operation will block until the missing data is received.

Application. Two different types of applications were developed. First, a *FileSharing* application which uses the Thicket Protocol to stream file contents to all recipients. This application was mainly used to debug the prototype implementation and we do not present here results extracted using this application. We also implemented a *MP3Player* application that allows the broadcasting of a music file provided as an input to the sender, which is then played by all the participants in the system. We have used this second application to obtain the results reported in this chapter.

Forward Error Correction. In order to make the developed prototype more resilient to failure scenarios, the streaming layer encodes data segments using a Forward Error Correction (FEC) Library. With FEC, a segment composed of N chunks can be encoded in M chunks in a way that the reception of N of the encoded chunks permits the decoding of the original data segment. This is accomplished by introducing redundancy on the encoded chunks. This feature is useful to tolerate temporary disconnections of nodes to a subset of trees. Using FEC, the messages received from the remaining trees can still be used to obtain the original data.

5.4.2.2 Practical Challenges

During the development of the prototype, several practical challenges were encountered in the protocol operation that required special attention. We now discuss these issues and the measures that were taken to overcome them.

Tree Repair. Considering an incoming data stream, when a peer p becomes disconnected from a tree t , several messages will not be received and, consequently, the *muids* of those messages will be present in the SUMMARY messages received from p neighbors. At this point, p needs to recover the messages starting from the one with the lowest *sqnb*. Therefore, each node stores the next *sqnb* to be received in each tree in a variable *nextSqnb_t*. This information is used to

choose the neighbor during the repairing process, a neighbor p is chosen to repair tree t if there is an *announcement* entry with $muid(sqnb, t)$ for it, such that $sqnb = nextSqnb_t$. When n repairs the branch to p for a tree t , n will send all data messages with sequence number between the requested value in the respective GRAFT message and the $nextSqnb_t$ of n .

Garbage Collection of Obsolete Messages. To determine when a received message will no longer be useful and can be discarded, an interval of time Δt was set for each message to remain stored in the *receivedMsgs* set. To purge obsolete messages, the *muids* of the messages in the *receivedMsgs* set are inserted in a queue. Then, every Δt the number of messages in the queue is recorded and, in the next execution of the periodic handler those messages are removed, leaving only the messages that have been inserted after its previous execution. This process guarantees that the messages are maintained at least for Δt . Algorithm 13 depicts the pseudo-code for this procedure.

Algorithm 13: Thicket: Message Garbage Collection

```

1   $n \leftarrow 0$ 
2  every  $\Delta t$  seconds do
3       $receivedMsgs \leftarrow removeFirst(receivedMsgs, n)$ 
4       $n \leftarrow |receivedMsgs|$ 

```

Storing Announcements. Every data message received by a peer results in an entry in the next SUMMARY message sent to each of the *backupPeers_p* set. Considering the rates of streaming applications this could lead to a large amount of data to be managed by the receivers of SUMMARY messages. To better manage this control data, announcements are stored in intervals composed of only two integers. An interval is stored for each neighbor and each tree. This way the memory requirements for storing the relevant data to be maintained at the announcements set is significantly reduced.

Storing Outdated Data Flows as Announcements. Due to the highly unstable delays observed in the Internet, a peer p may receive messages through a tree t from two different peers interleaved in a way that at least one message from each neighbor results in a duplicate. This phenomena triggers the transmission of a PRUNE message for both peers leaving p disconnected from t . To avoid this scenario, Thicket uses the $currentFlow_p(t)$ variable to detect if an

incoming data flow is outdated (which happens if a PRUNE message was already issued to the flow sender). If a fresh message is then received from a peer to which a branch was previously dropped, the message is simply treated as an announcement and will only be delivered if the corresponding timer expires.

5.5 Evaluation

We performed an extensive experimental study on the performance of Thicket using the PeerSim simulator (Montresor & Jelasity, 2009). Additionally, we have conducted a deployment of a Thicket prototype over hundreds of nodes scattered throughout the world in the PlanetLab testbed. This chapter presents the experimental results obtained using both approaches.

5.5.1 Experimental Setting

This section discusses the experimental setup employed in our experiences.

5.5.1.1 Simulation Setting

We have implemented Thicket for the PeerSim simulator. In order to extract comparative figures we compare the performance of Thicket with that of Plumtree (that serves as baseline for Thicket) as well as the NUTS and BOLTS alternatives previously introduced. For fairness, all protocols were executed on top of the same unstructured overlay, maintained by the HyParView protocol. HyParView is able to recover from concurrent failures as large as 80% of all nodes. Since HyParView uses TCP to maintain connections between overlay neighbors, we do not model message losses in our system. TCP is also used as an unreliable failure detector.

All the experiments have been conducted in a configuration with 10.000 nodes. Results presented here are an average of 10 independent executions of each experiment. All tested protocols, with the exception of Plumtree, were configured to generate $T = 5$ trees. Additionally, Thicket establishes trees using a gossip *fanout* of $f = 5$ and NUTS initiates the *eagerPushPeers* set of each spanning tree with 5 random selected overlay neighbors. Thicket, Plumtree, and NUTS operate on top of an unstructured overlay network with a degree of 25, while each of the 5 overlays used by BOLTS has a degree of 5. Furthermore, we have configured Thicket

to have a maximum forwarding load per node (parameter *maxLoad*) of 7. The timeout employed by protocols when receiving an announcement was set to 2 s. All experiences start with a stabilization period of 10 cycles that was not taken into account when extracting results. Simulations progress in cycles (using the cycle-based engine of PeerSim). Each simulation cycle corresponds to 20 s.

Additionally, we configured the streaming service so that, in each cycle the source disseminates T messages simultaneously, one message through each of the existing trees (in the case of Plumtree, which only embeds a single tree, all T messages are routed through that tree).

As stated before we assume perfect links however, messages are not delivered to nodes instantly, instead we consider the following delays when routing messages between peers (these delays are implemented by using the event based engine of the simulator which resolution is 1 ms):

- *Sender delay*: We assume that each peer in the system has a bounded uplink bandwidth. This allows to simulate uplink congestion when participants are required to send several messages consecutively. In particular we assume that each node can transmit 200Kbytes/s. Furthermore we assume that the payload of data messages is of 1,250 bytes, while SUMMARY messages have a payload of 100 bytes.

- *Network delay*: We assume that the core of the network introduces additional delays. In detail, in the simulations a message that is transmitted suffers an additional random delay selected uniformly between 100 and 300 ms. These values were selected by taking into consideration round trip time measurements that were performed using the PlanetLab infrastructure, and were already used in the evaluation of X-BOT in chapter 4.

5.5.1.2 PlanetLab Setting

In order to perform a pilot deployment in a real world setting, we have developed a prototype of Thicket written in Java as previously described in section 5.4. We have deployed the prototype over 400 PlanetLab nodes scattered throughout the world. In the experiments reported in this chapter, the values of parameters T , f , and *maxLoad* were the same employed in our simulation work discussed above. The prototype was enriched to include an implementation of a MP3 streaming application which resorts to the P2P streaming service described earlier.

Remember that the P2P streaming service that we developed resorts to a FEC library to disseminate controlled amounts of redundant information.

In all the experiments, we had the streaming application to disseminate a MP3 file of size 7,311,327 bytes at a bit rate of 256Kbit/s. The size of the payload messages was defined to 10 Kbytes. The FEC Library was used to send data segments composed of 4 chunks in 5 encoded messages. This prevents a single tree disconnection from affecting the reception of the original data segment at a peer. This results in a redundancy factor of $5/4 = 1.25$. The bit-rate of the original data 256 Kbit/s corresponds to 32 Kbytes/s. The redundancy factor introduced by FEC encoding increases the transmission rate to $32,000 * 1.25 = 40 \text{ Kbytes/s}$. This is equivalent to $40,000/10,000 = 4 \text{ chunks/s}$, which is within the rates suggested by Hegde et al. (2010).

After being received, a message is buffered at the Thicket layer for at least 30 s. All presented results are an average of 5 independent executions of each experiment.

5.5.2 Thicket Performance

In this section we report experimental results obtained both through simulation and prototype deployment in the PlanetLab testbed. The interested reader can find the original results for the experimental evaluation of Plumtree against a eager-push gossip protocol in previous publications (Leitão et al., 2007a; Leitão, 2007).

5.5.2.1 Simulation Results

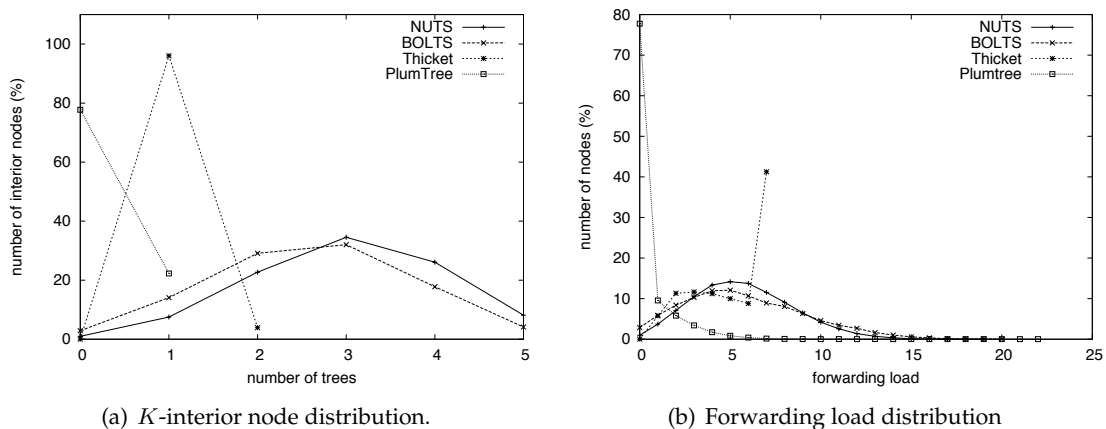


Figure 5.2: Results concerning the properties of Thicket in a stable environment.

5.5.2.1.1 Stable Environment First, we analyze the relevant performance metrics for Thicket in steady state. We start by evaluating the distribution of peers accordingly to the number of spanning trees in which they are interior. A value of 0 trees means that such peers are not interior in any of the trees, *i.e.*, they act as leaves in all trees. The results are depicted in Figure 5.2(a). Plumtree is plotted in the figure as a baseline for the single tree scenario. Note that, with a single tree, only 21% of all participants are interior nodes, and 79% are leaf nodes.

When using both the NUTS and BOLTS strategies, only a small fraction (below 20%) of participants are interior in a single tree (we repeat here the plot from Section 5.3.1 for the convenience of the reader). Also, for both approaches, there is a small number of participants that are interior in all 5 trees. As noted before, this motivates the need for some sort of coordination for the construction and maintenance of trees.

In sharp contrast, Thicket has almost all peers in the system acting as interior nodes in a single tree. A very small fraction (around 1%) serve as interior in 2 trees. This is a side effect of our localized tree repair mechanism, which ensures full coverage of all spanning trees. Still, no participant (with the exception of the source) acts as interior for more than 2 trees. This validates the design of Thicket. Interestingly, almost no participant occupies a leaf position in all trees; this contributes to the reliability of the dissemination process (see results in the next section) and ensures a uniform load distribution among participants. Furthermore, it allows us to use a much larger fraction of available system resources.

Figure 5.2(b) depicts the distribution of forwarding load in our system *i.e.*, the distribution of participants accordingly to the number of messages they must forward across all trees (assuming that one message is disseminated through each tree simultaneously). Because Thicket leverages on its integrated tree construction and maintenance strategy to limit the maximum load imposed on each peer, no participant is required to forward more than 7 messages across all trees where it is interior (usually 1 as we explained earlier). Additionally, more than 40% of peers are forwarding the maximum amount of messages, with more than 55% of nodes forwarding a smaller amount. Alternative solutions however have much more variable loads, with several peers forwarding more than 10 messages and some with loads above 15 messages. Notice that Thicket is the only protocol where almost no participant has a forwarding load of 0. This is a clear demonstration of the superior resource usage and load distribution that characterizes Thicket.

5.5.2.1.2 Fault-Tolerance We now present experimental results on the performance of Thicket in a catastrophic scenario where 40% of all participants in the system fail simultaneously. For such a high number of concurrent failures, all trees maintained by Thicket are affected. Therefore, there are no significant advantages of ensuring that participants are only interior in a single tree. Thus, we do not expect advantages from a reliability point of view. However, it is worth evaluating if Thicket is able to recover from this amount of failures and if, after recovery, the trees preserve their original properties, namely in terms of peers that are interior in a single tree and in terms of load distribution. Failures are induced after 100 cycles of message dissemination, to ensure that the spanning trees were already stabilized. Figure 5.3 summarizes our results. Once again we compare the performance of Thicket with that of the NUTS and BOLTS solutions discussed previously.

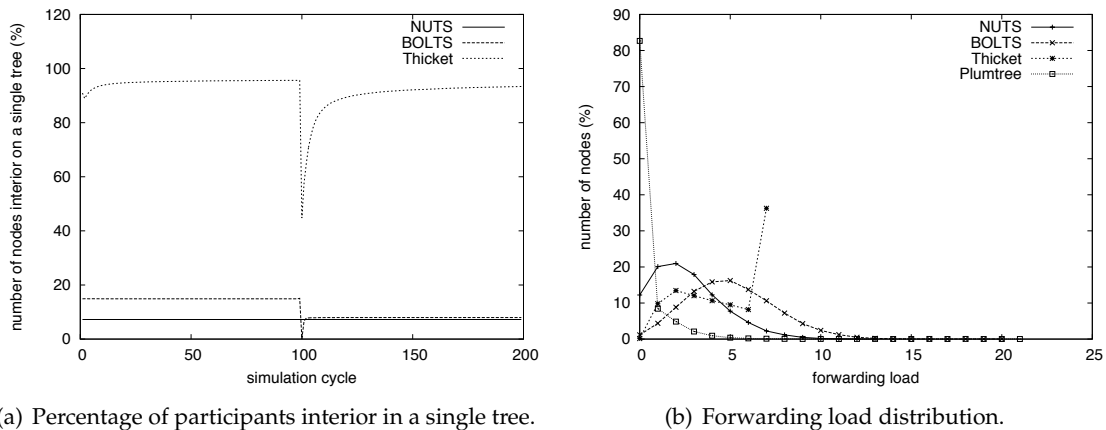


Figure 5.3: Performance results for Thicket properties in a catastrophic scenario.

Figure 5.3(a) shows the variation, for each of the tested protocols, of the percentage of peers that are interior in a single tree. Before the occurrence of the catastrophic failure, all solutions exhibit results consistent with the ones presented above in a stable environment, where no node joins or leaves the system. After the induction of failures the percentage of participants that act as an interior node in a single tree drops in BOLTS as result of its recovery procedure, that increases the percentage of participants acting as interior nodes in multiple trees. NUTS remains unaffected, as the percentage of peers in this condition is only 10% in steady state. Thicket drops to values in the order of 40% after failures. However the protocol is able to reconfigure itself in only a few simulation cycles.

Figure 5.3(b) depicts the forwarding load distribution for each considered solution. The

relevant aspect of this plot is that Thicket is able to regain a similar configuration to the one exhibited in a stable environment. The configuration of the remaining protocols remains the same, with participants exhibiting a wide range of forwarding loads. This is a clear indication that Thicket can regain its properties despite a large number of concurrent failures.

5.5.2.2 PlanetLab Deployment Results

This section presents experimental results concerning the performance of Thicket on the prototype deployment over PlanetLab.

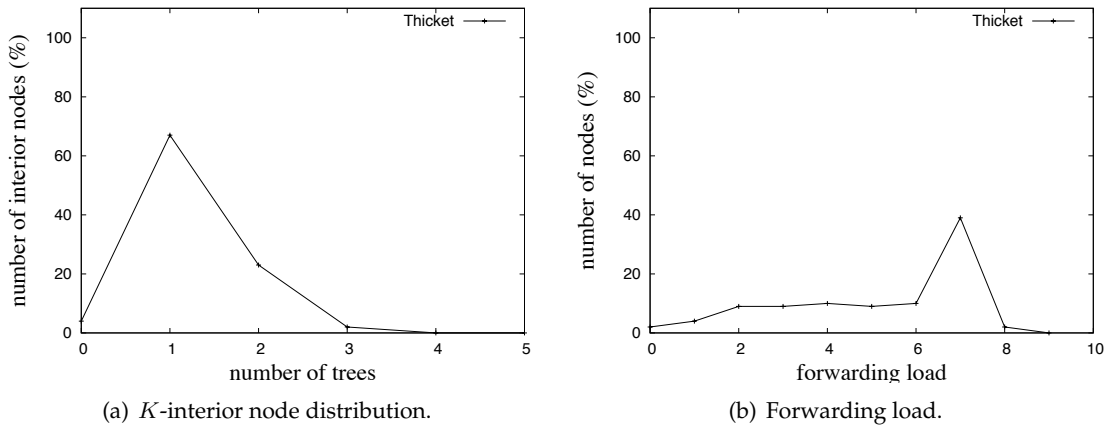


Figure 5.4: PlanetLab experimental results for the Thicket protocol.

5.5.2.2.1 Stable Environment Figure 5.4(a) depicts the distribution of participants while executing Thicket according to the number of trees in which they act as interior nodes. In a consistent way with the previously presented results, it is expected that most peers become interior in a single tree (limiting the forwarding load of all participants promoting the usage of all the available resources in the system). It can be observed that our prototype achieves the target desirable properties in the tree construction process: most peers (68 %) remain interior in a single tree while a small fraction, about 24 %, are interior in two trees ensuring the coverage of the trees across all participants. Furthermore, only 4 % of all participants are leaves in all trees, meaning that most peers actively contribute to the message dissemination process. Due to the constant changes in the network link throughput during the experiments, there is a small fraction (close to 2%) of the peer that are interior in more than two trees. This occurs because the repairing mechanism is triggered more often than in the controlled environment of the PeerSim simulations.

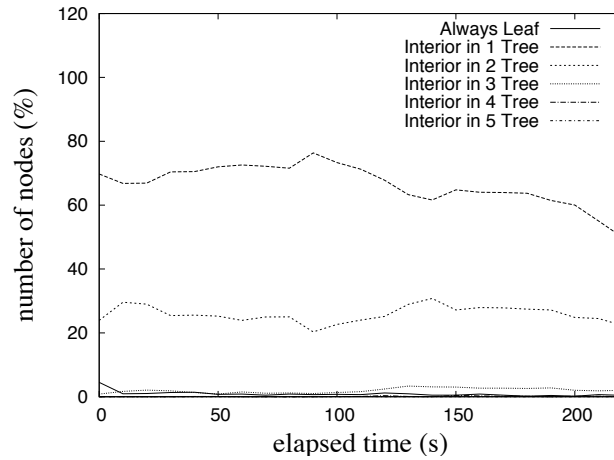


Figure 5.5: K -interior node distribution obtained by Thicket in a catastrophic scenario.

We have then measured the forwarding load distribution of all participants. It was expected that no node in the system would exceed $maxLoad$; additionally, the percentage of nodes with a forwarding load of 0 was expected to be low. Figure 5.4(b) presents the results obtained in the experiments. As expected, the results show that most participants (about 40%) exhibit a forwarding load equal to the $maxLoad$ parameter and consequently, most peers contribute equally to the dissemination process. Furthermore, only a small fraction (about 2.5%) of all participants present a forwarding load of 0. There are, however, some peers that exceed the forwarding load limit. This happens because nodes create tree branches during the repairing mechanism without checking the load limit (otherwise, one could make it impossible to ensure the reception of all data segments).

5.5.2.2.2 Faulty Environment Now, we evaluate the performance of Thicket prototype in a scenario where a large percentage of participants fail simultaneously. During these tests, 40% of all peers in the system fail after receiving 100 data segments. Failures happen approximately in the middle of the execution period of each experience. The presented results are extracted from the 60% of the participants that remain correct after the concurrent failure event.

Figure 5.5 presents the percentage of peers that are interior in 1, 2, 3, 4 and all trees as time elapses. After a large number of participants have failed, the percentage of participants that are interior in a single tree decreases due to the operation of the repair mechanism. Still, the protocol is able to maintain a configuration in which most peers are interior in a single tree and minimizing the number of participants that are interior in more than two trees. The observed

decrease is lower than the observed in simulations due to the fact that, in this experiment, participants do not fail exactly at the same time (the failure is triggered by the reception of the hundredth data segment), allowing Thicket to take recovery measures between participant failures.

The forwarding load distribution of all participants has also been measured in this faulty scenario. Figure 5.6 summarizes the results. From the figure, it is clear that even a large percentage of failures does not affect the forwarding load of the participants, which often exhibit values equal and below the *maxLoad*, as reported for the stable scenario.

5.5.3 Thicket Support for Streaming

In this section experimental results that evaluate the performance of the case study service are presented. We first introduce results obtained through simulation obtained with the PeerSim simulator. We then report obtained results over the PlanetLab testbed using the Java prototype previously described.

5.5.3.1 Simulation Results

5.5.3.1.1 Stable Environment: We start by evaluating the effect of Thicket in the dissemination of payload messages by the P2P streaming service. In particular we have evaluated the maximum number of hops required to deliver a message to all participants, and the maximum

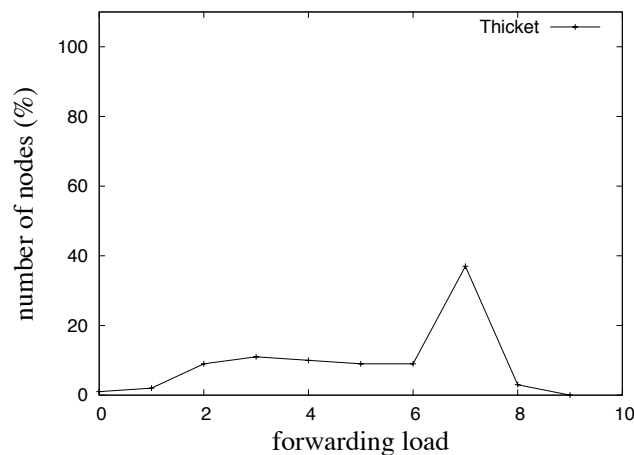


Figure 5.6: Forwarding load obtained by Thicket in a catastrophic scenario.

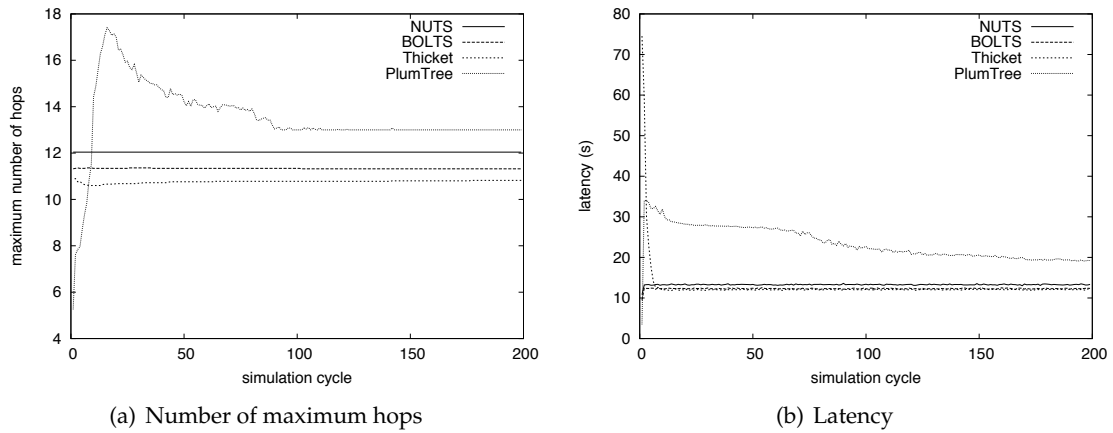


Figure 5.7: Performance results of the case study when leveraging Thicket in a stable environment.

latency between the source node and a receiver.

Figure 5.7(a) depicts the number of messages hops required to deliver a data message to all participants. Plumtree exhibits the highest value. This happens because Plumtree has some difficulties in dealing with variable network latency. This leads to situations where Plumtree triggers message recoveries too early, which increases the number of hops required to deliver a single message to all participants. Plumtree keeps on adjusting the structure of the embedded tree during the entire simulation, with the effect of slightly reducing the number of hops, stabilizing at 13 hops.

Thicket presents the best values (11 hops), as the trees embedded by the protocol are adapted, using the reconfiguration mechanism, to promote lower heights. The BOLTS approach presents a similar result. This happens because the use of several independent overlay networks forces the embedded spanning trees in each of the unstructured overlay networks to use the shortest paths between the source node and all receivers. NUTS has a higher value due to the use of a gossip-based tree construction scheme, that does not guarantee the use of all shortest paths available in the overlay.

Figure 5.7(b) presents the maximum latency for all protocols. These values are consistent with the last delivery hop values observed. One interesting aspect is that, contrary to all remaining solutions, Thicket presents higher initial values of latency, but these drop quickly in just 5 simulation cycles. This is due to the operation of the tree reconfiguration mechanism.

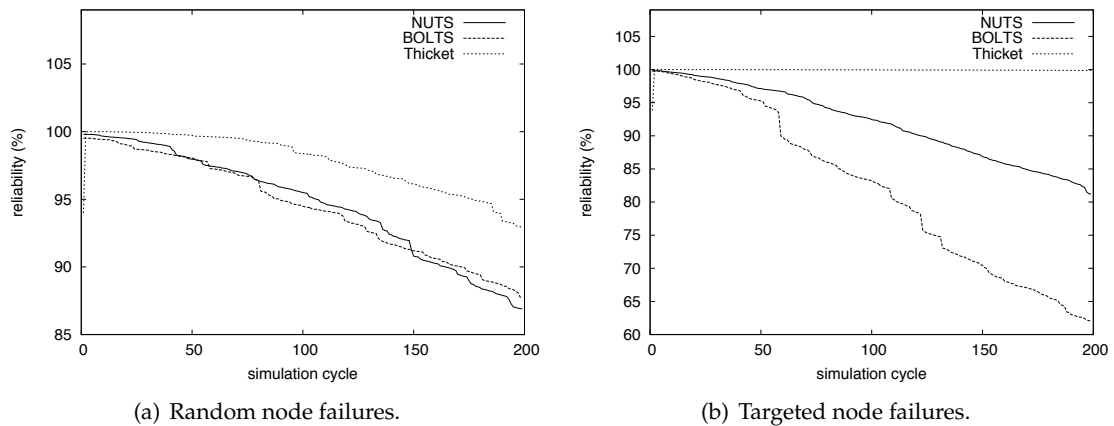


Figure 5.8: Reliability of a P2P streaming application leveraging the design of Thicket in a faulty scenario.

5.5.3.1.2 Fault-Tolerance: In this section we evaluate the performance of Thicket in a faulty scenario. In particular, we study the impact of sequential node failures in the dissemination reliability when using Thicket, NUTS, and BOLTS. In the experiments reported here the source node and the nodes that serve as root for trees in NUTS never fail.

As we assume that the P2P streaming service relies in FEC techniques when disseminating data, for each segment of data 5 messages are disseminated, one through each spanning tree, such that if a peer is able to receive at least 4 of these messages it is able to reconstruct the data segment, otherwise we consider that the participant misses the reception of this segment. We define reliability here as the percentage of correct nodes that are able to reconstruct disseminated data segments.

After an additional stabilization period (5 cycles) we configure the source to disseminate a data segment per cycle. In each cycle we also force a single peer to fail. We measure the reliability of the dissemination process at the end of each simulation cycle. Furthermore, we select the participant that fails in each cycle using two distinct policies: *i*) random; *ii*) random among the participants that are interior at a larger number of trees. We do not allow protocols to execute the repair mechanism during these simulations, to better capture the resilience of the embedded spanning trees. The results for all solutions using the repairing mechanism in this scenario would depict reliability measures close to 100%.

Figure 5.8 depicts the results for both scenarios. When we select random participants to fail (Figure 5.8(a)) the reliability of Thicket drops slowly. This happens because most peers are

interior in a single tree. So each failure, affects only participants bellow the failed one in a single tree, because peers can reconstruct the data segment even if they miss messages conveyed by one of the trees, most of them are still able to rebuild data segments as they remain connected to (at least) 4 trees. The reliability drops in a more visible way for both NUTS and BOLTS. This happens because a large majority of participants act as an interior node in more than a single tree, which results in the ability of a single failure to affect the flow of data in more than a tree.

Note that failing participants at random may not provide the best metric for reliability. For instance, failing random peers in a star network only has a noticeable effect in the reliability when the central peer fails (this is a single but also the only point of failure). The second experiment is more interesting, as it assesses what happens when “key” participants crash.

Interestingly, Thicket is extremely robust in face of such a targeted adversary (Figure 5.8(b)), and its reliability remains constant at 100%. This happens due to the following phenomena: as we limit the forwarding load imposed to each Thicket participant, when a peer acts as interior in more than a tree it is only responsible for forwarding messages to a smaller number of participants in each tree. Consequently, the effective number of peers that are affected in each tree is small. Furthermore, because links are never used for embedding more than a tree, these groups of participants are disjoint and therefore, can still receive messages sent through the remaining 4 trees. On the other hand, NUTS and BOLTS are severely affected by this scenario due to the fact that some peers are interior in all trees, which failure disrupts the flow of data in all embedded spanning trees.

5.5.3.2 PlanetLab Deployment Results

We now provide experimental results concerning the performance of the case study service in our prototype deployment over the PlanetLab test bed.

5.5.3.2.1 Stable Environment Figure 5.9 presents the average, and maximum delays between the reception of each data segments and the first data segment received. This metric captures the ability of the P2P streaming service to deliver messages with a constant delay, ensuring that data segments are delivered in time to be consumed by the application. The results show that the average data delivery time increases linearly with the data segment number (note that the inverse of the derivative of the line indicates the rate at which data segments are delivered) even when the maximum values observed can be as high as tree times the average.

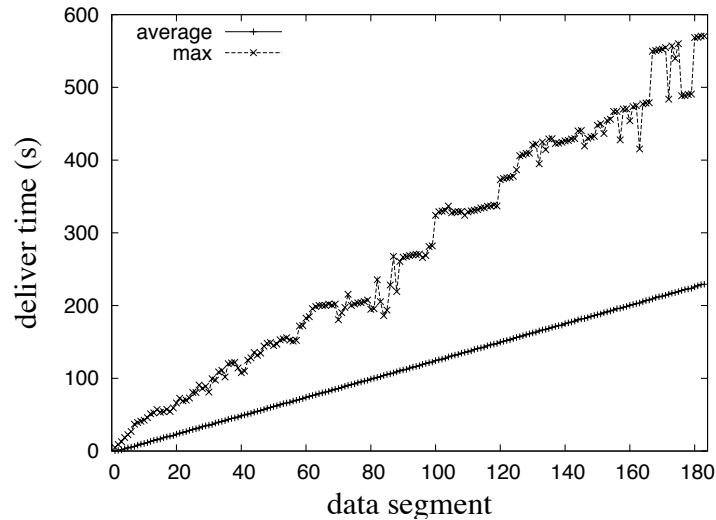


Figure 5.9: Delivery time of data segments for the Thicket prototype in a stable environment.

This means that peers which observe these delays may experience playback breaks. The most probable cause for this behavior is that some PlanetLab nodes, may temporarily lack the resources required to receive the stream without delays, due to the shared nature of the platform, without impacting the system as a whole. However, the results show that in general Thicket is able to efficiently support a stable streaming application.

5.5.3.2.2 Faulty Scenario Figure 5.10 presents the impact of the failures of a large percentage of participants on the delivery time of data segments. The results show that the protocol is able maintain almost constant delays in the average of all peers similar to the results obtained for the stable scenario. The worst case analysis indicates that failures do not affect negatively the delays on the correct participants since the delays obtained in this scenario are lower than during the experiments on the stable environment.

Finally, we have performed an evaluation on the reliability of the protocol under the previously described catastrophic scenario (the setup is similar to the one described previously). Figure 5.11 presents the impact of a large percentage of failures in the reception of data segments. As the results show, reliability decreases on the segments close to segment 100 (the time at which this segment is sent corresponds approximately to the time where the failure event occurs). However as the plot shows, the effect of the failures is contained during the period required to disseminate approximately 10 messages. This corroborates the previous observation that Thicket is able to reconfigure itself in face of failures in a timely fashion. Also, note that

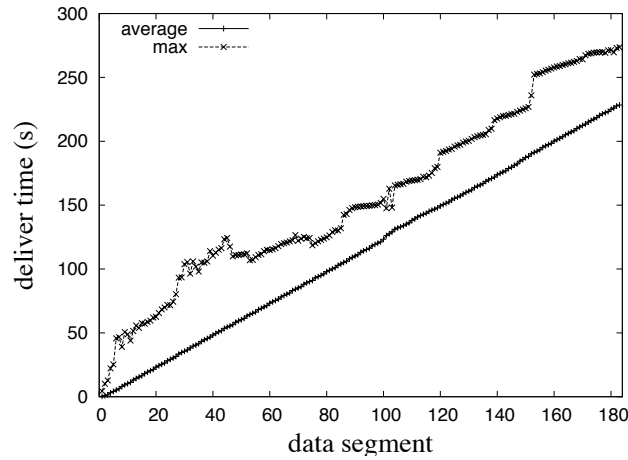


Figure 5.10: Delivery time of data segments for the Thicket prototype in a catastrophic scenario.

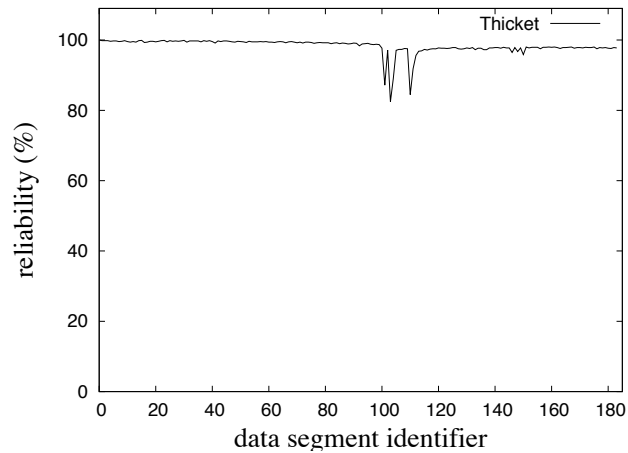


Figure 5.11: Reliability of a P2P streaming application leveraging the design of Thicket in a catastrophic scenario.

when nodes fail, the reliability of the dissemination process never falls below 80%.

5.6 Related Work

Previous research efforts have addressed the use of the embedding technique to achieve embedded spanning trees on overlay networks to support efficient and robust dissemination over P2P infrastructures. The motivation for this, is that embedded trees allows to combine the best features of pure gossip solutions and the pure tree approaches.

Note that the embedded tree approach can be applied to both structured and unstruc-

tered overlays. An example of the former is Scribe (Rowstron et al., 2001), that builds trees on top of the Pastry DHT (Rowstron & Druschel, 2001); examples of the later can be found in MON (Liang et al., 2005) and GoCast (Tang & Ward, 2005). Solutions based on unstructured overlays are more appealing as they have the *potential* to be more robust in face of system dynamics: since unstructured overlays pose less constraints on the topology, they can be repaired faster than structured overlays, as discussed previously in the thesis.

Existing solutions can be classified in single-tree or multiple-tree. Single tree solutions are naturally simpler but have two main problems: they promote an unbalanced resource usage among peers (interior nodes consume resources to forward data while leaf nodes only receive data); they also suffer from temporary disruptions when one interior node fails and the tree needs to be repaired. Multiple-tree solutions, as the name implies, rely on several trees connecting the same set of participants. Trees are embedded in such a way that a peer only acts as an interior node in one, or a small subset of, all embedded trees, and a leaf node in the remaining. This approach provides load-balancing, as all participants contribute with their resources (*e.g.*, bandwidth) to forward data. Furthermore, by sending redundant information in some trees (for instance by using network coding techniques (Frey et al., 2010; Chou & Wu, 2007)), it is possible to achieve higher fault-tolerance: since the failure of a node only disrupts the tree where it acts as an interior node, receivers are still able to operate using the data received from the remaining trees.

In the context of the approaches that embed multiple trees, they can be further classified according to the type of algorithm that is used to build and maintain trees. Centralized algorithms rely on some specialized nodes, that have a global knowledge of the topology, to deploy the trees. Note that even a centralized solution is not trivial, as the problem of optimal tree construction is NP-hard (Johnson et al., 1978). Centralized approaches have little practical interest for very-large scale systems, as they are not scalable and it is complex to make them fault-tolerant.

This contribution focus on decentralized approaches. In this context the most relevant examples are SplitStream (Castro et al., 2003) and Chunkyspread (Venkataraman et al., 2006).

SplitStream leverages on a variant of Scribe to build multiple interior-node-disjoint spanning trees over the Pastry (Rowstron & Druschel, 2001) DHT. Similar to Thicket, the authors strive to build trees in which a node is interior in a single tree. Additionally the authors pro-

pose a scheme that allows participants to control their degree in the tree where they are interior (*i.e.*, controlling the forwarding load of each peer) according to their capacity. Unlike Thicket, SplitStream relies on a DHT; peers are interior in a single tree by design, as each tree is rooted in peers with DHT identifiers that have distinct prefixes. Notice that the overhead of maintaining a DHT is superior to that of maintaining an unstructured overlay network. Additionally, the unstructured overlay can potentially recover from failures faster than Pastry: in Pastry a crashed overlay neighbor cannot be replaced by any given peer, only participants with an adequate identifier (accordingly to the DHT organization logic) can be used. Moreover, the scheme employed by the authors to enforce maximum degree on peers that act as interior nodes in a tree may result in several peers becoming disconnected from the tree with a negative impact on the reliability of the data dissemination mechanism. SplitStream also requires additional links between peers in addition to the ones provided by Pastry, which results in additional overhead.

Chunkyspread (Venkataraman et al., 2006) is a protocol that embeds several spanning trees on top of an unstructured overlay network, while trying to limit the load and degree of peers accordingly to their capacities. However, the mechanism at the core of Chunkyspread does not attempt to control the number of trees where a peer is interior. This results in trees that are not interior-node-disjoint, *i.e.*, where peers can act as an interior node in several trees. This is clearly an undesirable property from the reliability point of view. In fact, we have demonstrated in Section 5.5.3 that independent trees are extremely relevant in scenarios where participants can fail.

Summary

In this chapter we have addressed the problem of disseminating information in a P2P environment through the use of highly efficient spanning trees which are embedded over a low-cost robust unstructured overlay. To that end we have designed Thicket, a protocol that explores a topology management mechanism that operates at the P2P service layer to embed and maintain multiple and *interior-node-disjoint* spanning trees over a single unstructured overlay network. The design of Thicket leverages previous work performed to develop the Plumtree protocol. In Thicket most participants in the system act as an interior node in a single embedded spanning tree. This allows to significantly improve the load balancing of participants in tree-based multicast systems, as long as each tree is used to transmit a similar amount of data.

We have extensively evaluated the performance of Thicket through simulation. Furthermore, we have implemented a prototype of Thicket and evaluated its performance in a real world setting, through a deployment over 400 PlanetLab nodes scattered throughout the world. Our results show that Thicket is efficient and highly resilient to peer failures, being able to quickly recover from scenarios where large fractions of participants fail simultaneously. Additionally, Thicket is able to ensure that the load is shared evenly among participants, and offers the opportunity for using forward error correction for supporting a reliable streaming infrastructure at the planetary scale.

In the following chapter we explore another topology management technique which operates at the P2P service layer. That technique, which we named *enrich* allows one to take into consideration the operation of a P2P service to enrich an unstructured overlay network with additional (and eventually temporary) overlay links to improve the overall operation of the system.

Publications

The work presented in this chapter has been published through the following publications:

Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay. M. Ferreira, J. Leitão, and L. Rodrigues. Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS), New Delhi, India, 31 October-3 November 2010.

Gossip-based Interior-Node-Disjoint Trees. J. Leitão, M. Ferreira, J. Pereira, and L. Rodrigues. IEEE Transactions on Parallel and Distributed Systems (**Submitted**).

6 Enrich the Topology: OpenFire

In this chapter we introduce the OpenFire protocol which exploits the *enrich* approach, which operates at the P2P service layer, to temporarily add overlay links to an unstructured overlay network taking into consideration feedback from the operation of the P2P service. In particular in this chapter we aim at addressing the problem of limited communication patterns that exist in network where firewalls and NAT boxes operate. More precisely, we focus on balancing the behavior of a rumor mongering service in such networks.

This chapter is organized as follows: Section 6.1 motivates and define the goals of OpenFire. Our solution is presented in Section 6.2 while the case study used to evaluate the impact of the solution is introduced in Section 6.3. Experimental results for the impact of our solution over a rumor mongering service are presented in Section 6.4. Finally, related work is discusses in Section 6.5.

6.1 *Motivation and Goals*

6.1.1 **Motivation**

Most rumor mongering protocols have inherent load-balancing properties as long as peers are deployed over a network with a “flat” topology, that is, a topology where any pair of participants may engage in a rumor mongering exchange. This is due to the fact that, when a participant selects another with whom it exchanges information, that peer is selected uniformly at random from the entire population. This can be easily achieved if each participant in the system knows the complete membership (Birman et al., 1999). However, even if nodes only have a partial view of the entire membership, as long as neighbor sets offer an uniform sample of the entire system (Ganesh et al., 2003), this can be achieved if the protocol operates over an unstructured overlay network where each peer only has access to a partial view of the system (in the form of a local neighbor set). When these conditions are met, on average, participants tend

to engage in the same number of rumor mongering exchanges (in any time period sufficiently large to have statistical relevance).

Unfortunately, in many realistic settings, not all pair-wise interactions are possible. For instance, in the Internet, a large portion of peers sit behind firewalls or boxes that execute Network Address Translation (so-called NAT boxes). If two peers are behind the same box, we say that they belong to the same *confinement* domain. Also, we denote a participant that can be directly accessed from any point in the Internet, without any sort of restrictions, as an *unconfined* peer. In practice, a peer in a confinement domain can only initiate communication with other peers in the same confinement domain or with unconfined participants. Unconfined peers can only contact other unconfined peers. This results in an unbalanced participation of nodes in the rumor mongering process. As we shall see, such an imbalance can be problematic.

Rumor mongering protocols typically implement mechanisms that allow to use network bandwidth efficiently. The most significant bandwidth consumption may happen when two peers are exchanging messages to reconcile their respective states. To mitigate these costs, in the original Clearinghouse paper (Demers et al., 1987), the authors propose an iterative reconciliation technique, where participants compare their internal states using hash functions, and exchange the most recent updated they performed until their states become reconciled. Byers, Considine, and Mitzenmacher (Byers et al., 2002) improve on this design by combining Bloom filters, Merckle trees, and Patricia tries. Minsky et al. (2003) propose a method based on characteristic polynomials.

While effective in reducing bandwidth, these techniques require significant CPU resources to support the state reconciliation operation. In addition to the computations involved in state reconciliation itself, there is often a non-negligible amount of work required to serialize and deserialize objects that are transmitted, as well as signing and/or encryption tasks when needed. For example, in a commercial Java-based deployment of Astrolabe (Renesse et al., 2003), a gossip-based aggregation service that uses Bloom filters and Merckle trees for reconciliation, nodes spend approximately 3% of all their available CPU time on all these operations. In a PlanetLab deployment of Fireflies (Johansen et al., 2006), a secure gossip-based overlay network that uses the reconciliation technique of Minsky et al. (2003), as well as public key cryptography, participants use approximately 10% CPU time.

It is therefore worth considering what happens in scenarios where there are a relatively

small number of unconfined participants in the presence of a large number of “islands” with confined peers behind a firewall. The unconfined participants have to act as gateways for all information that is exchanged between all participants in the system. In doing so, either unconfined peers will experience a much larger networking and CPU load than ordinary peers (likely becoming CPU saturated), or by equalizing loads, dissemination times would be significantly increased compared to a “flat” network in which no nodes are confined.

6.1.2 Goals

Considering the motivation presented above, in this chapter we propose a technique, that we named OpenFire, which allows to balance the exchanges and processing of rumor mongering protocols on network that are unbalanced due to the existence of Firewalls and NAT boxes in the underlay. The main properties of OpenFire are as follows:

- The solution is completely decentralized.
- It allows participants to have a similar processing load during the execution of a rumor mongering P2P service.
- It does not require peers to be aware if they are behind a firewall/Nat box.
- It does not require any coordination among participants.

6.2 The Open Fire Protocol

In this section the rationale for the design of OpenFire is presented. We also discuss the assumptions made concerning the underlying unstructured overlay network and finally, we provide a full description for the operation of OpenFire.

6.2.1 Rationale

Our approach stems from the observation that the only way to convey information from a peer p (in a confinement domain) to another peer q (in a different confinement domain) is via an unconfined participant u . That is, since p and q cannot communicate directly, it is unavoidable that some unconfined node u acts as a mediator. One way for u to act as a mediator is to engage

in two full rumor mongering exchanges. In detail, if participant p first engages in a exchange with u and, subsequently, participant q engages in another rumor mongering exchange with u , u is able to convey information from p to q . However, as discussed previously, if u is required to engage in a full state reconciliation step with each of the confined peers, it might incur in excessive processing overhead. It is possible to alleviate the extra load on u if it merely serves as a router of messages exchanged between p and q , instead of executing the full rumor mongering exchange.

Therefore, the key idea of OpenFire is that peers in the system should have a dual operation mode: they sometimes participate in complete rumor mongering exchanges (to update and propagate their own state) and sometimes they participate only as routers, forwarding messages exchanged between two other peers that would otherwise be unable to exchange information directly. There are several challenges in the implementation of this strategy:

i) Participants should not be required to be aware if they are confined or unconfined. Ideally, all peers in the system would simply execute the same algorithm, and the emergent behavior of the system would ensure that a balanced participation in full rumor mongering exchanges would happen.

ii) Participants should use a localized algorithm to decide when to engage in a full rumor mongering exchange or when to merely serve as a router.

iii) An unconfined peer u should be able to route messages between confined nodes p and q despite the fact that it cannot be the initiator of communication to either p or q .

These challenges are addressed by the proposed solution using the following two complementary techniques:

i) Peers executing the rumor mongering service keep track of how many exchanges they initiated and how many gossip exchanges they have accepted *i.e.*, how many rumor mongering exchange they have participated that were initiated by other peers). In a balanced network, on average, every peer participates in the same number of rumor mongering exchanges initiated by itself and by other peers. Therefore, we define a quota that limits the number of exchanges initiated by other peers, in excess of the rumor mongering exchanges initiated by itself, in which a node participates. The quota is increased when the node initiates a rumor mongering exchange and decreased when it accepts an exchange initiated by another peer. When a par-

ticipant runs out of quota it no longer accepts gossip exchanges and simply acts as a router for requests received from other peers.

ii) Participants that accept a connection from a peer, keep the connection to that node open¹. This is achieved by enriching the underlying unstructured overlay network with a temporary new link. For instance, if an unconfined peer u receives a rumor mongering request from node q , it creates a temporary overlay link to q effectively keeping the connection to q open. This overlay link is kept until it receives another gossip request directly from some other peer. In this way, if peer u is later contacted by another participant p and its quota has been exhausted, u can route p 's request to q using that temporary overlay link (and the underlying connection that materializes that link).

6.2.2 Underlying Unstructured Overlay Network

As discussed previously in the thesis, in order to guarantee the scalability of a rumor mongering service in a large-scale system with potentially a (highly) dynamic filiation, and unstructured overlay network should offer a membership service. We now discuss the assumptions we make concerning the properties of the underlying unstructured overlay network.

We assume that the neighbor set exported to the P2P service layer should implicitly encode the communication restrictions imposed by the existence of Firewalls in the execution environment of the system. This translates in the following properties:

- Unconfined peers can only contain the identifiers of unconfined participants with whom they can take the initiative to start a rumor mongering exchange.
- Confined peers can only contain the identifiers of unconfined participants or of other peers that belong to their confinement domain, as these are the participants with whom they can engage in rumor mongering exchanges directly.

From a practical point of view this can easily be achieved by exploiting two possible strategies at the overlay layer: *i)* similar to what is proposed in unstructured overlay network management protocols such as HyParView (Leitão et al., 2007b) or CellFarm, which was described

¹The mechanism to maintain this connection open depends on the transport protocol being used.

in Chapter 3, overlay links can be materialized through TCP connections. In this case, peers will be unable to establish TCP connections with participants with whom they cannot engage in the exchange of rumors directly; alternatively, *ii*) when adding the identifier of a peer to the local neighbors set, the protocol that is responsible for managing the unstructured overlay network may send a probe, that should be acknowledged by its peer, to verify if direct communication with that potential new overlay neighbor is possible.

Also, in the context of OpenFire, we do not assume the underlying overlay network to be connected (taking into consideration the definition provided previously in Chapter 2), as such a strict property is hard to ensure in an environment populated by firewalls, where participants are not required to be aware if they are sitting behind a firewall, and where participants are also not expected to take proactive measures to circumvent these firewalls (for instance, by relying on hole punching techniques). Instead, we take into consideration a relaxed form of connectivity which explicitly takes into consideration the existence of firewalls as follows:

Firewall-aware connectivity: This property requires that the following constraints are met: *i*) all peers in a confinement domain must belong to a single connected component where there exists at least one overlay path between every pair of peers in that domain; *ii*) all unconfined peers must belong to a single connected component; and finally, *iii*) there must be at least one overlay link departing from each confinement domain to an unconfined peer.

Notice that, contrary to the definition of overlay connectivity previously considered, firewall-aware connectivity does not require the existence of an overlay path connecting peers in different confinement domains, nor the existence of overlay paths between unconfined and confined peers.

6.2.3 Protocol

We now provide the full description of the OpenFire protocol. As discussed previously it is assumed that each peer in OpenFire has access to a neighbor set that is populated with a sample of other participants identifiers with whom it can engage in direct communication (*i.e.*, unconfined nodes or nodes within the same confinement domain). Algorithm 14 presents pseudocode for our protocol.

Algorithm 14: OpenFire Protocol

```

Internal data:
  cache  $\leftarrow \perp$ 
  quota  $\leftarrow 1$ 
  neighborSet //Managed by an external membership protocol

1: every  $\Delta T$  do
2:    $p \leftarrow \text{random}(\text{neighborSet})$ 
3:   Send(RUMORMONGERINGREQUEST( $\emptyset \cup me, 1, p$ ))
4:   quota  $\leftarrow$  quota +1

5: upon Receive(RUMORMONGERINGREQUEST(path,hop)) do
6:   if quota > 0 or hop = TTL or cache =  $\perp$  then
7:     quota  $\leftarrow$  quota -1
8:     trigger Deliver(RUMORMONGERINGREQUEST)
9:      $n \leftarrow \text{last}(\text{path})$ 
10:    Send(RUMORMONGERINGREPLY(path \  $n$ ),  $n$ )
11:   else
12:     path  $\leftarrow$  path  $\cup$   $me$ 
13:     Send(RUMORMONGERINGREQUEST(path,hop+1), cache)
14:   if hop = 1 then
15:     cache  $\leftarrow$  last(path)

16: upon Receive(RUMORMONGERINGREPLY(path)) do
17:   if path =  $\emptyset$  then
18:     trigger Deliver(RUMORMONGERINGREPLY)
19:   else
20:      $n \leftarrow \text{last}(\text{path})$ 
21:     Send(RUMORMONGERINGREPLY(path \  $n$ ),  $n$ )

```

As noted before, each peer maintains a quota value, initially set to 1, that encodes the number of rumor mongering exchanges it can accept from other peers. Additionally, each participant keeps a single-entry *cache* that serves to enrich the unstructured overlay network topology with an additional, and temporary, overlay link to the last peer from which it received a rumor mongering request directly; this overlay link maintains active the connection for that peer allowing the node to contact it regardless of other connectivity constraints (*e.g.*, the existence of a firewall).

When out of quota, OpenFire forwards rumor mongering requests. We limit the maximum number of times that a message can be forwarded using a protocol parameter denoted TTL. This avoids network congestion scenarios due to the accumulation of messages in the system whose processing keeps being postponed.

Periodically (Alg. 14, line 1) every peer tries to initiate a rumor mongering exchange with a peer selected at random from its local neighbor set (Alg. 14, lines 2 – 3) by sending a RUMORMONGERINGREQUEST message. That peer also increases its quota (Alg. 14, line 4), which will enable it to engage in an additional rumor mongering exchange initiated by another peer.

Upon receiving a RUMORMONGERINGREQUEST message from a peer (Alg. 14, line 5) a participant engages in the gossip exchange if at least one of the following conditions is true: *i*) it has available quota (i.e. its quota value is above zero); *ii*) the RUMORMONGERINGREQUEST message has been already forwarded TTL times; or *iii*) the cache of the participant does not contain a connection that can be used to route the RUMORMONGERINGREQUEST message to another peer.

If none of the above conditions is met, the node simply routes the RUMORMONGERINGREQUEST message using the connection in its cache (i.e., to the last peer from which it received a rumor mongering request directly). Notice that the peer adds its own identifier to the path associated with this message. This is required to allow the bi-directional rumor mongering exchange between participants in distinct confinement domains, as the RUMORMONGERINGREPLY message has to traverse the inverse path in the network (Alg. 14, lines 16 – 21)².

Whenever a node receives a RUMORMONGERINGREQUEST message directly from its source (i.e, a RUMORMONGERINGREQUEST message that has not been routed; Alg. 14, line 14), it updates its local cache to establish a new overlay link to that peer. This means that the next time the peer needs to route a RUMORMONGERINGREQUEST message it will send it to a different participant. RUMORMONGERINGREQUEST messages that have been routed and RUMORMONGERINGREPLY messages do not lead a peer to update its local cache therefore, no temporary overlay link is created.

We note that there is an interesting symbiosis between the cache and the quota mechanisms that helps in having routed RUMORMONGERINGREQUEST messages quickly accepted and processed. When peer u adds to its cache a connection to p , p 's quota is known to be greater than 0, as it has just initiated a rumor mongering exchange; therefore, p is likely to still have a positive quota value when a RUMORMONGERINGREQUEST is routed to it by a quota exhausted participant.

²This requires peers to keep these connections open for some time, by using an additional cache outside the protocol's scope.

6.3 Case Study

In the context of this contribution we take into consideration a simple case study application, where nodes coordinate among themselves to maintain a distributed state in a consistent fashion. We assume that the internal state of nodes only have one writer, and that the state has a monotonic increasing version number associated with it.

To this end, each participant in the system executes a rumor mongering protocol where periodically it selects a random peer from its local neighbor set (provided by the underlying unstructured overlay network) and sends a rumor mongering request which encode the current value and version number of its internal state. When a peer accepts a rumor mongering request it replies with a rumor mongering reply which encodes a similar information. After this exchange peers update their local internal state, if the version number contained in the message received by its peer is above the version number currently owned by it.

Our aim is to verify that our approach allows nodes to disseminate updates executed over their internal state (by a particular participant in the system) in an efficient and timely manner, while at the same time balancing the number of rumor mongering exchanges in which each participant is required to participate.

6.4 Evaluation

In this section we evaluate the efficacy of our approach. In particular we want to validate that the number of rumor mongering exchanges in which peers engage is balanced across all peers even in an environment which is highly unbalanced due to the existence of firewalls and NAT boxes. The results presented in this section also assert the costs in terms of dissemination latency and the message forwarding overhead of OpenFire due to the necessity of forwarding messages when peers become quota exhausted.

6.4.1 Experimental Setting

We start by describing the experimental setup employed, and provide motivation for the network model used in the simulations. We conducted extensive simulations in the PeerSim simulator (Montresor & Jelasity, 2009), using its event driven engine. In our experiments we simu-

lated 12800 nodes distributed in a variable number of distinct confinement domains that ranges from 1 (the equivalent of a flat network topology) to 12100. For simplicity, we model all unconfined nodes as belonging to domain 1. In each experiment, we ensure that each confinement domain has at least one node, and then we distribute the remaining nodes at random among all domains.

Because we are not concerned in evaluation the performance of OpenFire in environments where nodes fail, we run our experiments on top of a static unstructured overlay network, where each peer in the system has overlay links to all other participants in its own confinement domain plus all peers in domain 1 (i.e., all unconfined participants). We have evaluated our approach configuring the TTL parameter with different values in order to assert the practical impact of this parameter over the performance of the rumor mongering service. In particular, we performed simulations for TTL values of 1, 2, 5 and 10. Notice that a TTL value of 1 prevents a message from ever being routed, and corresponds to a classic rumor mongering protocol, which is used as a baseline.

The internal state maintained by the case-study application, for simplicity, is modeled by a single bit, initially set to 0. When an experiment begins, a random peer sets its state to 1 (simulating an update to the internal state of that participant). Then, every participant periodically engages in a rumor mongering step to exchange this value. When executing rumor mongering, peers execute a simple reconciliation protocol, in which they set their value to the largest value, between their own and the value received from its peer. This process models, in an abstract manner, the propagation of information over the population of the system.

In each simulation each participant initiates 500 gossip exchanges. Every peer gossips its value every 10 time units, therefore a simulation takes 5000 time units. Each link has a random latency between 2 and 7 simulator time units. All results reported in this chapter are an average of 100 independent simulations. Confidence intervals reported in figures were calculated to a confidence of 95%.

6.4.2 Experimental Results

Figure 6.1 depicts the maximum number of gossip interactions in which a peer participates for the scenarios described above. Notice that with a flat network (i.e., a single confinement domain) both the baseline (TTL = 1) and our approach behave similarly (for all tested TTL values).

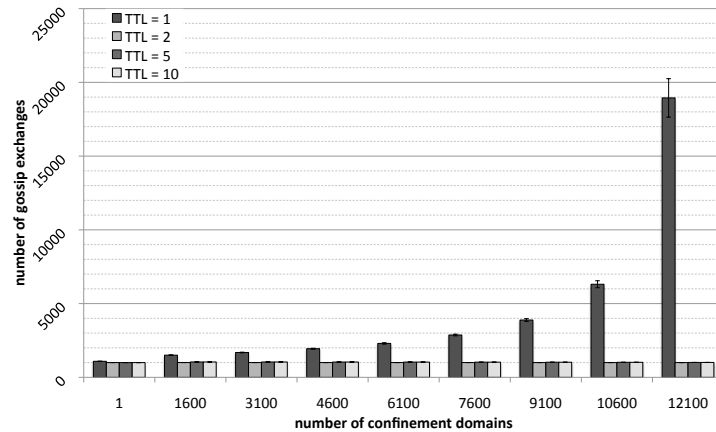


Figure 6.1: Max rumor mongering exchanges / peer with OpenFire.

The maximum number of rumor mongering interactions in which a single peer participates is approximately 1000, which reflects a perfect balance: 500 interactions initiated by the peer itself and another 500 that are initiated by other peers.

However, in scenarios with several confinement domains, the maximum number of rumor mongering exchanges in which a single peer may participate starts to rise with the baseline protocol. This is expected, as participants that are in the Internet domain (unconfined peers) are selected for more rumor mongering exchanges due to the existing lack of balancing over the in-degree of the underlying unstructured overlay network due to the existence of firewalls. As the number of domains increases, this effect becomes more visible. This happens because the number of participants in the Internet domain decreases (given that we maintain the total number of peers constant in our experiments). In a scenario with 12100 domains, the number of interactions in which a single node may be requested to participate approximates 20000. In sharp contrast, when using OpenFire, in all tested scenarios, results show a constant value which is very close to 1000. This shows that our approach effectively succeeds in balancing rumor mongering interactions in networks with firewalls.

Figure 6.2 presents the amount of simulation time units that are required to infect the entire population. With a flat network our approach presents an increase in latency of approximately 10 time units, that is, a single rumor mongering round. This is observed for all TTL values. As the number of confinement domains increases, the latency increases slightly. This happens because more requests need to be routed among peers, resulting in additional latency in the processing of rumor mongering requests. Notice however, that the maximum increase in la-

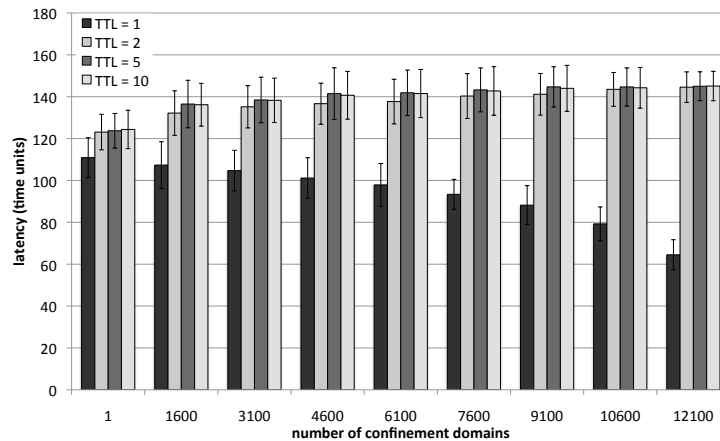


Figure 6.2: Maximum latency for the rumor mongering when relying on OpenFire

tency observed is only of 30 time units (roughly 3 rumor mongering rounds).

Interestingly, as the number of confinement domains increases, the latency of the classic rumor mongering approach decreases. This happens because the number of participants in the Internet domain also decreases, which leads the system to behave like a centralized architecture. Therefore, the dissemination becomes very fast, by first contaminating the central Internet domain and then, having peers in all other domains pulling the value from that domain. This is achieved at the cost of overloading unconfined peers.

Figure 6.3 depicts results for the maximum number of messages forwarded by a single peer. This is a measure of the communication overhead that is imposed by our approach. In the flat network topology our approach presents a negligible overhead, given that there are

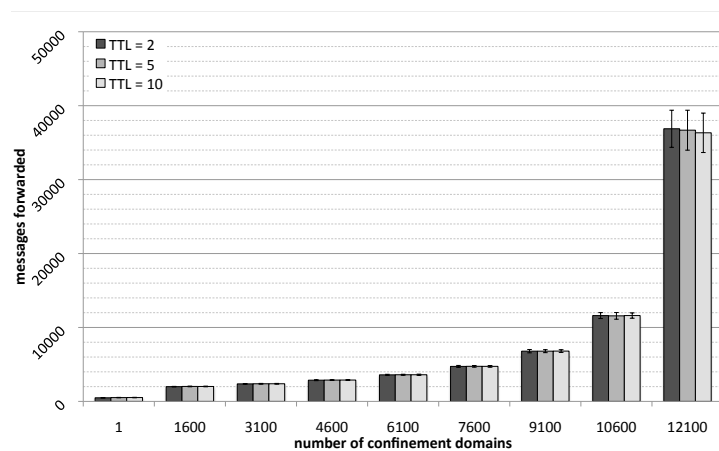


Figure 6.3: Max forwarded messages / peer for the operation of OpenFire.

few requests that need to be routed. As expected, when the number of confinement domains increases, the maximum number of forwarded messages by a single peer also increases, as participants in the Internet domain are forced to route more requests to avoid being overloaded. Considering that the CPU overhead imposed by forwarding a message is low (the peer does not need to deserialize the payload, check signatures, and so on), we believe this overhead to be acceptable.

In our experiments the efficacy is mostly unaffected by the TTL configuration parameter. This is because with high probability most requests are accepted in their second hop as we previously discussed in this chapter.

6.5 Related Work

In this section we discuss prior works that addressed the problem resulting from the fact that the Internet is not flat. There are essentially two approaches. One approach is to exploit the structure of the Internet, while the other tries to find ways to hide it. In the first approach, many overlay networks, structured and unstructured, have introduced the notion of *superpeers*. Superpeers are nodes that have static, globally addressable IP addresses (*i.e.*, are unconfined), are well-connected and exhibit little churn, and are altruistic, generously providing their resources for the good of the entire system. The popular file sharing service Kazaa³ is a good example of an unstructured P2P network that uses superpeers. Yang e Garcia-Molina (2003) explores how best to use superpeers in an unstructured network. Various others explored the use of superpeers in structured networks. These works can be subdivided into structured overlay networks that exploit heterogeneity but hide it to users, such as the work of Mizrak et al. (2003), and those overlays that expose the heterogeneity in the network, for instance following the approach taken in the design of Astrolabe (Renesse et al., 2003). Unfortunately, systems based on superpeers require nodes acting as a superpeer to do significantly more work than regular peers.

The other approach is to try to hide the structure of the Internet, so that all peers can directly communicate with one another. Some firewalls support explicit protocols for tunneling (Leech et al., 1996). Since this is not widely supported, another option is hole-punching through NAT

³www.kazaa.com

boxes (Kegel, 1999). Ford et al. (2005) finds that hole-punching works for UDP in about 80% of cases, and for TCP in about 65% of cases. Based on hole-punching, Nylon is a gossip-based service that provides each peer with a random sample of nodes with whom it can communicate with (Kermarrec et al., 2009). Unfortunately, and contrary to OpenFire, Nylon requires participants in the system not only to be aware of their confined/unconfined status, but also to know which type of NAT box is between them and the Internet.

The work of Dowling e Payberah (2012) proposes a peer sampling service protocol (which implicitly establishes an unstructured overlay network among participants in a P2P system) that is aware to the presence of NAT boxes in the underlay. Unfortunately, and in sharp contrast with OpenFire, the solution proposed by the authors require peers to be aware if they are behind a NAT box, and also to calculate an estimate of the unconfined to confined peer ratio.

OpenFire can be thought of a way to combine advantages of both approaches. We use a superpeer approach that does not require any special features of firewalls, but the only extra work that the superpeers do is forwarding traffic instead of being cumbered with processing the large majority of messages exchanged among peers. Otherwise, all peers are equal participants. This hybrid approach does not prevent the overlay protocol from exploiting heterogeneity or proximity. Thus protocols that try to exploit heterogeneity such as HEAP (Frey et al., 2009) can take advantage of our approach to overcome the presence of firewalls, while allowing peers with high capacity, even confined ones, to do more work than others.

Finally, previous works (such as the work of Jelasity et al. (2004)) have evaluated the performance of rumor mongering protocols over random graphs. Typically these works assume either a regular graph, or that every pair of peers can exchange messages directly. In our work we are studying the behavior of rumor mongering protocols in scenarios that impose limitations to which peers can interact directly, particularly in the context of network populated by firewalls and NAT boxes which result in a clear unbalance over the rumor mongering exchange patterns exhibited among participants.

Summary

In this chapter the OpenFire protocol was presented. OpenFire explores a new approach that operates at the P2P service layer that enriches the topology of unstructured overlay network

with temporary overlay links which promote a more efficient operation of the P2P service. In particular in this chapter the goal was to balance rumor mongering exchanges in networks populated by firewalls and NAT boxes. When compared with classic rumor mongering protocols, our approach is able to ensure that all peers in the system participate in a similar number of rumor mongering exchanges independently of the network topology. Moreover, we have presented experimental results showing that the increase in latency imposed by OpenFire is acceptably low, and that the communication overhead is acceptable for rumor mongering-based applications which are CPU intensive.


This was the last of the four main contributions of the thesis. The next chapter discusses additional contributions that emerged from the work previously presented in the thesis, and that were achieved through collaborations and concludes the thesis.

Publications

The work presented in this chapter has been published through the following publication:

Balancing Gossip Exchanges in Networks with Firewalls. J. Leitão, R. van Renesse and L. Rodrigues. Proceedings of the 9th International Workshop on Peer-to-Peer Systems (IPTPS '10), San Jose, CA, USA, 27 April, 2010.

Conclusions and Future Work



This chapter closes the thesis. Section 7.1 summarizes the main results presented in the thesis and briefly presents additional results that emerged from the work previously presented. It discusses aspects of the management of unstructured overlay network's topology that were not explicitly addressed in the thesis and closes by providing an answer to the questions introduced in Chapter 1. Finally, Section 7.2 concludes the thesis by discussing future research directions.

7.1 Conclusions

The thesis has proposed, developed, and evaluated solutions based on four distinct approaches to manage the topology of unstructured overlay networks. The benefits of each of these strategies in the context of P2P systems was illustrated through a different case study.

The considered approaches can be grouped in two families: *i*) approaches that operate at the overlay network layer. This family of solutions directly manipulate the selection of neighboring relations established among peers. This directly affects the links that form the overlay network; and *ii*) approaches that operate at the P2P service layer. This family of solutions take into consideration feedback from the execution of a particular P2P service. That information is then employed to either select different communication strategies accordingly to the properties of each overlay link, or to enrich the underlying overlay network with additional temporary links (outside the logic of the underlying overlay management protocol).

In particular, for the family of solutions that operate at the overlay network layer we studied two distinct approaches:

CellFarm explores the control approach, where soft constraints are enforced over the overlay topology by directly manipulating the protocol that manages the overlay. CellFarm has demonstrated that in fact, by leveraging soft topological constraints one can build highly flexible overlay topologies which can offer additional functionality to P2P services and applications

executed on top of them. CellFarm illustrated this by providing a P2P infrastructures that can be used by P2P services to replicate data among peers and distribute the load of accessing that data across the peers that replicate it. This was shown through the design of a P2P resource location service that employed a limited (*i.e.*, structured) one-hop replication mechanism. CellFarm has demonstrated that the design space of overlay networks can be further explored to fill the existing gap between unstructured and structured solutions.

X-BOT uses the bias approach, that enables it to exchange overlay links over time by other links that better match any given criteria X , encoded in a companion oracle. *X-BOT* has demonstrated that solutions to tackle the topology mismatch problem can be generalized to also take into consideration other underlay properties or application/service specific requirements. Furthermore, *X-BOT* was designed to protect relevant properties of the original unstructured overlay during its operation. The benefits of *X-BOT* were demonstrated through an application-level broadcast service, whose performance was significantly boosted by taking advantage of the improved overlay topology generated by *X-BOT*. This work has shown that specialized overlays can be easily tailored for specific applications by taking advantage of the bias approach. This can not only increase the spectrum of P2P services that can be efficiently deployed over the Internet, but also highly simplify their design, as specific requirements of services can be handled directly at the overlay layer.

We note however, that when the requirements of the P2P service layer are dynamic, they cannot be easily translated into an independent oracle, as the oracle would be required to continually monitor the internal state and protocol transitions at the P2P service layer. Although devising such oracles would be possible, an alternative approach is to develop topology management schemes that operate directly at the P2P service layer. Such approaches offer the potential to easily react and adapt to the dynamic requirements of the service. The thesis discusses two approaches that fit in this family of solutions:

Thicket is a novel protocol that relies on the embed approach to deploy multiple interior-node-disjoint spanning trees over a single unstructured overlay network. The structure of these spanning trees can easily adapt to changes in the execution environment. For instance, if a link starts omitting messages due to congestion, the protocol is able to remove such a link from currently deployed spanning trees and replace it with other available links. This operation can be performed while ensuring that the majority of peers contribute with a similar amount of re-

sources. Thicket illustrates the benefits of an embed approach, by offering the necessary mechanisms to easily build a reliable P2P streaming system, which exploits the additional resources made available by the use of multiple trees, which combines the strengths of both tree-based and unstructured overlays.

OpenFire illustrated how an enrich approach can be employed to deal with the existence of Firewalls and NAT boxes in the underlay, allowing peers to coordinate among them to ensure a balanced CPU consumption for a rumor mongering protocol, without exposing the complexity of the underlay to the P2P architecture. Enriching the overlay network topology at the P2P service level allows the service to have an improved behavior independently of the overlay topology provided by the underlying layer.

Finally, and considering the work presented in the thesis, an answer can be provided to the fundamental question addressed by the thesis: *What kind of techniques can be devised to provide some degree of structure to overlay networks, such that the performance of the P2P applications can be improved without compromising the robustness and low cost of random overlays?*

The results presented throughout the thesis, and discussed above, have studied four distinct techniques that manage the topology of unstructured overlays by imbuing some form of structure over it. As discussed in previous chapters, all approaches were able to improve the performance of P2P services without compromising the reliability of random overlay, or incurring in high maintenance overheads. In summary, the thesis proposed the following approaches: control, bias, embed, and enrich. All these approaches have been shown to be able to imbue some form of structure on the topology of unstructured overlay networks, without compromising their relevant properties. Notice that these results are not enough to assure that no other technique can be developed which exhibit similar properties, by exploring other areas of the design space.

Combining Approaches

Although the thesis has provided examples for the combination of each of the four considered approaches, the problem of combining all four addressed approaches in a single system was not explicitly addressed. Note however that some of the approaches were demonstrated to have a high synergy between them. In particular:

- The control and bias approaches have been shown to be easily combined. This happens because the bias approach can take into consideration the topology constraints imposed by the control approach during its operation. This was demonstrated by the *X-BOT* protocol, which was effectively executed over the HyParView protocol, which resorts to the control approach to enforce a topology constraints in particular, it enforces overlay link symmetry. *X-BOT* operation allows to bias the overlay topology while ensuring that the overlay links encoded in the active views of nodes remain symmetric.
- The control and embed approaches are also known to present a good synergy. Both previous work (Plumtree) as well as the Thicket protocol validate this as they employ the embed approach to deploy spanning trees over an unstructured overlay. In particular we tested these solutions over the HyParView overlay which, as discussed above, relies on the control approach.

Ramifications and Collaborations

The research work conducted in the context of the thesis lead to additional research activities, which were pursued through collaborations. These works have applied the four discussed approaches to different scenarios and applications. These results were not discussed in the thesis for sake of conciseness. In the following, a brief summary of these results is presented and the relation of these works to the main contributions of the thesis is briefly discussed:

Large-Scale P2P Autonomic Monitoring A large-scale P2P autonomic monitoring infrastructure has been designed based on two of the approaches discussed in the thesis (Leitão et al., 2008). In particular, we exploited the control approach, through the HyParView protocol (Leitão et al., 2007b) to establish balanced and robust monitoring relations among components of a large-scale data center. Additionally, the embed approach was proposed as a building block for devising a mechanism to, efficiently and reliably, disseminate alarms issued by the monitoring infrastructure to specialized monitoring consoles.

RASM RASM (Allani et al., 2010) was an initial effort to improve the embedding approach for achieving robust and efficient spanning trees embedded over unstructured overlay networks. RASM enriched the original Plumtree design (Leitão et al., 2007a) by taking into consideration the expected reliability of both links and peers in the system. Additionally,

we modeled the capacity of peers through a quota value (a model that would be later be considered in the design of Thicket). The design of RASM also includes an heuristic, that is used by each peer individually to optimize the use of its available quota. Notice that the quota of each peer is used to transmit redundant messages to compensate for the (predicted) unreliability of some overlay links, contributing to improve the reliability of the dissemination service on unreliable environments.

Curiata Curiata (Alveirinho et al., 2010) is a resource location infrastructure tailored for assisting in resource management on cloud computing data centers. Curiata aims at improving the location, and allocation, of computational resources to support elastic requirements of applications deployed in the cloud. Curiata leveraged the design of the *X-BOT* protocol which used the bias approach at its core. *X-BOT* was used to establish a (dynamic) unstructured overlay network connecting nodes owning similar resources (*e.g.*, CPU, memory, disk space). An additional DHT was combined with the design of *X-BOT* to assist in routing queries to zones of the unstructured overlay network populated by nodes whose locally available resources allows them to satisfy the query.

Rollerchain Rollerchain (Paiva et al., 2011b, 2011a) is a robust and well balanced DHT which combines features from a structured overlay network (particularly, Chord by Stoica et al. (2001)) and the CellFarm unstructured overlay network, which exploits the control approach for managing its overlay topology. In Rollerchain, peers use CellFarm to form clusters of nodes (*i.e.*, Cells) with a similar size. These Cells then join a DHT acting as virtual nodes. Rollerchain includes mechanisms to ensure that peers have a balanced number of incoming and outgoing DHT links, and also that the division procedure of CellFarm allows the system to match potential non-uniform distributions of the stored data keys over the identifier space. Virtual peers allow the Rollerchain DHT to be more robust to churn, as the virtual peer abstraction provided by CellFarm allows to mask churn effects, and also promotes load distribution for both communication and storage over the DHT layer.

In the following section, future research directions, that emerged from the work presented in the thesis, are discussed.

7.2 Future Work

As discussed previously, additional research efforts are necessary to fully understand the strengths and limitations that arise from the combination of the four individual techniques to manage the topology of unstructured overlays that were discussed in the thesis. By combining these techniques one can aim at designing more complex P2P systems. This topic shall be addressed in future work. In the following two possible research vector that can be pursued considering the contributions of thesis, and the combination of the discussed approaches, are motivated and briefly presented.

Improving volunteer parallel computing platforms

The Boinc infrastructure (Anderson, 2004) is a clear example of the extreme computing power that one can harness from hosts scattered across the Internet. Initially motivated by the need of high computational power at low costs for the Seti@Home project¹, the BOINC initiative is currently able to harness an average of 5.857 PetaFLOPS per day. This computational power is being currently used to support hundreds of scientific research projects that require significative computational power to process huge amounts of data.

Despite the notorious success of Boinc, the current platform follows a client-server architecture and is limited to support what is commonly known as *embarrassingly parallel applications*. This class of parallel applications resort to a simple strategy to achieve parallelization. This form of parallelization requires the input data to be partitioned into (small) independent fragments (*i.e.*, a work unit). Each work unit is then processed independently by a computational (volunteer) node. When the work unit is fully processed by that node, it returns the computed result to a centralized component. Although this programming model is enough to support several research projects, it is fundamentally limited, as several computational problems cannot easily be tackled by partitioning the input data, consider for instance the problem of predicting the whether.

An alternative to current volunteer cycle sharing infrastructures such as Boinc, is to leverage some of the results presented in the thesis to design a new decentralized P2P volunteer

¹<http://setiathome.berkeley.edu/>.

cycle sharing architecture. In particular, such an architecture could rely on the control and bias approaches to build an unstructured overlay network composed of cliques of peers that are both geographically close and have similar computational and storage capacity. This would allow such groups to efficiently collaborate on storing data and perform collaborative parallel computations (*e.g.*, by exchanging partial results among them periodically). In particular, this would enable such infrastructure to store input data for computational tasks submitted to the infrastructure by a user, and results from computations already processed by volunteer computing nodes. Additionally, by enabling direct communication among nodes, this P2P architecture can be extended to support more complex parallel programming models, such as MapReduce.

Such an architecture will also benefit from the availability of efficient application level routing that is capable of operating correctly in high churn environments. This can be achieved by resorting to the bias approach, embodied by the *X-BOT* protocol, to design a new family of distributed hash tables management protocols. This new class of DHTs would, instead of relying on the correctness of the routing mechanism to manage the topology of the overlay, rely on a fully distributed biasing iterative mechanism. To better capture the frequent communication patterns that arise across peers during the execution of specific tasks, the enrich approach can be used to establish temporarily overlay links, improving the overall performance of the system, in a decoupled fashion of the overlay management protocols logic.

Such research endeavor will be instrumental to assist research projects that require high amount of processing power, by providing a platform with a immense aggregated computational power, that supports more rich parallel programming models and that at the same time, allows research groups without access to highly available and powerful servers to leverage the platform.

User-centric platform for social applications

The Internet is becoming mostly an user-centric environment, in opposition to the more classical server-centered environment. There are several signals of this phenomenon in the way users interact with the Internet now-a-days when compared with typical interactions a decade or two ago, where the large fraction of users consumed contents made available by a limited group of entities. To provide an exhaustive list is clearly outside the scope of this discussion,

however one can consider the following three observations: *i*) end users are now responsible for the production of most content available in the Internet. Big contributors to this paradigm shift were web services such as YouTube², Blogger³, Tweeter⁴, and social networks in general, just to name a few; *ii*) social networks popularity have grown significantly in recent years. The success of Facebook⁵ is a clear example of this phenomenon; and finally, *iii*) with the growth in the popularity of social networks, games and other applications that operate over the social graph inherent to the operation of the social network has also found a significative amount of success.

Unfortunately, despite the fact that the larger portion of content in the Internet is now produced by the end-user, that information becomes owned and controlled by large corporations that own the large data centers that are essential to support these applications in a centralized fashion. Although the user produces the data, that data is no longer owned, or even controlled, by that user. To address this mismatch, one can rely in the work presented in the thesis to procure further advances in world-scale P2P architectures. In particular, to devise and deploy an user-centric platform that can support fully distributed social network and applications without resorting to the support of large centralized data centers.

A first step in this direction, is to support a persistent and fully decentralized P2P social network, where users can establish and manage social relations among them in a decentralized fashion. This can be achieved by establishing a peer-to-peer network connecting the end-users devices, that continually adapts and evolves to match users relations. To ensure persistency, one can rely on schemes similar to the CellFarm protocol, combined with robust X-BOT like DHTs (discussed above) to replicate user-essential and public data to other nodes in the system. Such an approach guarantees that the user still owns a virtual presence in the system, even if all her devices are temporarily disconnected or unavailable. To support such a complex infrastructure in the worldwide Internet, one would also have to rely on mechanisms such as the ones proposed in the OpenFire protocol, to mask the complex nature of the underlying topology from such service.

Such endeavor however, requires additional research, not only in the area of distributed

²<http://www.youtube.com>

³<http://www.blogger.com>

⁴<http://www.tweeter.com>

⁵<http://www.facebook.com>

systems in general, and P2P protocols in particular, but also in the context of cryptography and security, to devise novel solutions that empower the user to have a tighter control over who can access her own data, once it has been published through the P2P infrastructure.

Building such a social persistent P2P network would allow to further develop social applications on top of it. This can include, but is not limited to, social games, resource sharing applications, streaming applications, content dissemination, among others. Solutions to embed sub-networks on top of the social driven overlay network, for instance based on the Thicket approach, can be instrumental to allow the efficient operation of these new social applications. The immediate benefit of the approach presented here is that, when compared with current centralized solutions, the data produced by the user, is still controlled and owned by that user. This may be a fundamental step in distributed systems research in a time where Internet censorship is a topic currently in debate in several countries. Additionally, this would open the market of social applications to small companies which do not own large data-centers currently required to support these applications.

Publications

Below, the reader can find a list of relevant publications that have resulted from the collaborative work discussed in this chapter.

Large-Scale Peer-to-Peer Autonomic Monitoring. J. Leitão, L. Rosa and L. Rodrigues. Proceedings of the Distributed Autonomous Network Management Systems Workshop (DANMS), New Orleans, USA, Oct, 2008.

RASM: A Reliable Algorithm for Scalable Multicast. M. Allani, J. Leitão, B. Garbinato and L. Rodrigues. Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP), Italy, Pisa, Feb, 2010.

Flexible and Efficient Resource Location in Large-Scale Systems. J. Alveirinho, J. G. Paiva, J. Leitão, and L. Rodrigues. Proceedings of the 4th ACM SIGOPS/SIGACT International Workshop on Large-Scale Distributed Systems and Middleware (LADIS), Zürich, Switzerland, 28-29 July 2010.

Rollerchain: a DHT for High Availability. J. Paiva, J. Leitão and L. Rodrigues. INESC-ID Tec. Rep. 20/2011, May 2011 (Available in: <http://www.inesc-id.pt/ficheiros/publicacoes/7240.pdf>).

Rollerchain: a DHT for High Availability (Poster). J. Paiva, J. Leitão and L. Rodrigues.

11th ACM/IFIP/Usenix Middleware Conference (Poster Session) Lisbon, Portugal, December 2011.

Bibliography

- Aberer, K., Alima, L., Ghodsi, A., Girdzijauskas, S., Haridi, S., & Hauswirth, M. (2005, August). The essence of P2P: a reference architecture for overlay networks. In *Proceedings of the 5th IEEE international conference on peer-to-peer computing (P2P'05)* (pp. 11 – 20). Konstanz, Germany.
- Allani, M., Leitão, J., Garbinato, B., & Rodrigues, L. (2010, February). RASM: A reliable algorithm for scalable multicast. In *Proceedings of the 18th Euromicro international conference on parallel, distributed and network-based computing (PDP'10)* (pp. 137 – 144). Pisa, Italy.
- Alveirinho, J., Paiva, J., Leitão, J., & Rodrigues, L. (2010, July). Flexible and efficient resource location in large-scale systems. In *Proceedings of the 4th international workshop on large scale distributed systems and middleware (LADIS'10)* (pp. 55–60). Zurich, Switzerland.
- Anderson, D. (2004, November). BOINC: A system for public computing and storage. In *Proceedings of the 5th IEEE/ACM international workshop on grid computing (GRID'04)* (pp. 4 – 10). Pittsburgh, USA.
- Arak, V. (2007, August). *Skype Blog: what happened on august 16*. http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html.
- Araújo, F., Rodrigues, L., Kaiser, J., Liu, C., & Mitidieri, C. (2005, June). CHR: A distributed hash table for wireless ad hoc networks. In *Proceedings of the 4th international workshop on distributed event-based systems (DEBS'05)* (pp. 407 – 413). Columbus, Ohio, USA.
- Balakrishnan, H., Kaashoek, M., Karger, D., Morris, R., & Stoica, I. (2003, February). Looking up data in P2P systems. *Communications of ACM*, 46(2), 43 – 48.
- Baldoni, R., Bonomi, S., Cerocchi, A., & Querzoni, L. (2010, July). Improving validity of query answering in dynamic systems. In *Proceedings of the 3rd international workshop on reliability, availability, and security (WRAS'10)* (pp. 4:1 – 4:6). Zurich, Switzerland.

- Baset, S., & Schulzrinne, H. (2004, March). An analysis of the skype peer-to-peer internet telephony protocol. In *Proceedings of the 23rd conference of the IEEE communications society (INFOCOM'04)* (pp. 1 – 11). Hong Kong.
- Birman, K. (2007, October). The promise, and limitations, of gossip protocols. *SIGOPS Operative Systems Review*, 41(5), 8 – 13.
- Birman, K., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., & Minsky, Y. (1999, May). Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2), 41 – 88.
- Birman, K., & Renesse, R. van. (1994). *Reliable distributed computing with the ISIS toolkit* (R. van Renesse, Ed.). Los Alamitos, CA, USA: IEEE Computer Society Press.
- Blake, C., & Rodrigues, R. (2003, May). High availability, scalable storage, dynamic peer networks: Pick two. In *Proceeding of the 9th workshop on hot topics in operating systems (HotOS-IX)* (pp. 1 – 6). Lihue, Hawaii.
- Bloom, B. (1970, July). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 422 – 426.
- Bollobás, B. (1981). Random graphs. *Combinatorics (ED. H. Temperley) London Mathematical Society Lecture Note Series(52)*, 80 – 102.
- Broder, A., & Mitzenmacher, M. (2004, November). Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4), 485 – 509.
- Byers, J., Considine, J., & Mitzenmacher, M. (2002, July). *Fast approximate reconciliation of set differences* (Tech. Rep. No. 2002-019). <http://hdl.handle.net/2144/1665>.
- Cardellini, V., Casalicchio, E., Colajanni, M., & Yu, P. (2002, June). The state of the art in locally distributed web-server systems. *ACM Computer Surveys*, 34, 263 – 311.
- Carvalho, N., Pereira, J., Oliveira, R., & Rodrigues, L. (2007, June). Emergent structure in unstructured epidemic multicast. In *Proceedings of the 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)* (pp. 481 – 490). Edinburgh, Scotland, UK.
- Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., & Singh, A. (2003, October). SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of the*

- nineteenth ACM symposium on operating systems principles (SOSP'03)* (pp. 298 – 313). Bolton Landing, New York, USA.
- Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., & Shenker, S. (2003, August). Making gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'03)* (pp. 407 – 418). Karlsruhe, Germany.
- Chou, P., & Wu, Y. (2007, September). Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5), 77 – 85.
- Chu, Y.-H., Rao, S., Seshan, S., & Zhang, H. (2002, October). A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8), 1456 – 1471.
- Cohen, B. (2008). *The bittorrent protocol specification*. http://www.bittorrent.org/beps/bep_0003.html.
- Crespo, A., & Garcia-Molina, H. (2002, July). Routing indices for peer-to-peer systems. In *Proceedings of the 22nd international conference on distributed computing systems (ICDCS'02)* (pp. 23 – 23). Vienna, Austria.
- Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M., & Morris, R. (2004, March). Designing a DHT for low latency and high throughput. In *Proceedings of the 1st symposium on networked systems design and implementation (NSDI'04)* (pp. 7 – 20). San Francisco, California: USENIX Association.
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007, October). Dynamo: Amazon's highly available key-value store. In *Proceedings of 21st ACM SIGOPS symposium on operating systems principles (SOSP'07)* (pp. 205 – 220). Stevenson, Washington, USA.
- Deering, S., & Cheriton, D. (1990, May). Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems (TOCS)*, 8(2), 85 – 110.
- Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., et al. (1987, August). Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th annual acm symposium on principles of distributed computing (PODC'87)* (pp. 1 – 12). Vancouver, British Columbia, Canada.

- Deshpande, M., Xing, B., Lazardis, I., Hore, B., Venkatasubramanian, N., & Mehrotra, S. (2006, July). CREW: A gossip-based flash-dissemination system. In *Proceedings of the 26th IEEE international conference on distributed computing systems (ICDCS'06)* (pp. 45 – 52). Lisboa, Portugal.
- Dingledine, R., Mathewson, N., & Syverson, P. (2004, August). Tor: The second-generation onion router. In *Proceedings of the 13th USENIX security symposium* (pp. 303 – 320). San Diego, California, USA.
- Diot, C., Levine, B., Lyles, B., Kassem, H., & Balensiefen, D. (2000, January). Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1), 78 – 88.
- Dowling, J., & Payberah, A. (2012, June). Shuffling with a Croupier: Nat-aware peer-sampling. In *Proceedings of the 32th IEEE international conference on distributed computing systems (ICDCS'12)* (p. (to appear)). Macau, China.
- Druschel, P., & Rowstron, A. (2001, May). Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th workshop on hot topics in operating systems, 2001 (HotOS'01)* (pp. 75 – 80). Elmau/Oberbayern, Germany.
- Eugster, P., & Guerraoui, R. (2002, June). Probabilistic multicast. In *Proceedings of the 32th annual IEEE/IFIP international conference on dependable systems and networks (DSN'02)* (pp. 313 – 322). Bethesda, Maryland, USA.
- Eugster, P., Guerraoui, R., Handurukande, S., Kouznetsov, P., & Kermarrec, A.-M. (2003, November). Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems (TOCS)*, 21(4), 341 – 374.
- Eugster, P., Guerraoui, R., Kermarrec, A.-M., & Massoulié, L. (2004, May). From epidemics to distributed computing. *IEEE Computer*, 37(5), 60 – 67.
- Fonseca, P., & Miranda, H. (2008, September). Improving scalability of autonomic systems: the frequency-aware search approach. In *Proceedings of the 2nd international conference on autonomic computing and communication systems (Autonomics'08)* (pp. 36:1 – 36:10). Turin, Italy.
- Ford, B., Srisuresh, P., & Kegel, D. (2005, June). Peer-to-peer communication across network

- address translators. In *Proceedings of the USENIX annual technical conference (ATEC'05)* (pp. 13 – 26). Anaheim, California, USA.
- Frey, D., Guerraoui, R., Kermarrec, A.-M., Koldehofe, B., Mogensen, M., Monod, M., et al. (2009, November). Heterogeneous gossip. In *Proceedings of the 10th ACM/IFIP/USENIX international conference on middleware (Middleware'09)* (pp. 3:1 – 3:20). Urbana Champaign, Illinois, USA: Springer-Verlag New York, Inc.
- Frey, D., Guerraoui, R., Kermarrec, A.-M., & Monod, M. (2010, August). Boosting gossip for live streaming. In *Proceedings of the IEEE 10th international conference on peer-to-peer computing (P2P'10)* (pp. 1 – 10). Delft, The Netherlands.
- Ganesh, A., Kermarrec, A.-M., & Massoulié, L. (2002, July). HiScamp: Self-organizing hierarchical membership protocol. In *Proceedings of the 10th ACM SIGOPS european workshop (EW'10)* (pp. 133 – 139). Saint-Emilion, France.
- Ganesh, A., Kermarrec, A.-M., & Massoulié, L. (2003, February). Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), 139 – 149.
- Garbacki, P., Epema, D., & Steen, M. van. (2007, June). Optimizing peer relationships in a super-peer network. In *Proceedings of the 27th international conference on distributed computing systems (ICDCS'07)* (pp. 31 – 40). Toronto, Canada.
- Ghodsi, A., Haridi, S., & Weatherspoon, H. (2007, October). Exploiting the synergy between gossiping and structured overlays. *SIGOPS Operative Systems Review*, 41(5), 61 – 66.
- Glendenning, L., Beschastnikh, I., Krishnamurthy, A., & Anderson, T. (2011, October). Scalable consistency in scatter. In *Proceedings of the 23rd ACM symposium on operating systems principles (SOSP'11)* (pp. 15 – 28). Cascais, Portugal.
- Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., & Stoica, I. (2003, August). The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'03)* (pp. 381 – 394). Karlsruhe, Germany.
- Gupta, A., Liskov, B., & Rodrigues, R. (2004, March). Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st symposium on networked systems design and implementation (NSDI'04)* (pp. 9 – 23). San Francisco, California, USA.

- Gupta, I., Birman, K., Linga, P., Demers, A., & Renesse, R. van. (2003, March). Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd international workshop on peer-to-peer systems (IPTPS'03)*. Cambridge, Massachusetts, USA: Springer Berlin / Heidelberg.
- Gupta, I., Kermarrec, A.-M., & Ganesh, A. (2006, July). Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Transaction on Parallel Distributed Systems (TPDS)*, 17(7), 593 – 605.
- Harvey, N., Jones, M., Saroiu, S., Theimer, M., & Wolman, A. (2003, March). SkipNet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th USENIX symposium on internet technologies and systems (USITS'03)* (pp. 9 – 25). Seattle, WA, USA.
- Hayden, M., & Birman, K. (1996). *Probabilistic broadcast* (Tech. Rep.). Ithaca, NY, USA. <http://ecommons.library.cornell.edu/bitstream/1813/7261/1/96-1606.pdf>.
- Hegde, N., Mathieu, F., & Perino, D. (2010, May). On optimizing for epidemic live streaming. In *Proceedings of the 2010 IEEE international conference on communications (ICC'10)* (pp. 1 – 5). Cape Town, South Africa.
- Hei, X., Liang, C., Liang, J., Liu, Y., & Ross, K. (2007, November). A measurement study of a large-scale P2P IPTV system. *IEEE Transactions on Multimedia*, 9(8), 1672 – 1687.
- Hsiao, H.-C., Liao, H., & Huang, C.-C. (2009, November). Resolving the topology mismatch problem in unstructured peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(11), 1668 – 1681.
- Iamnitchi, A., Foster, I., & Nurmi, D. (2002, July). A peer-to-peer approach to resource location in grid environments. In *Proceedings of the 11th IEEE international symposium on high performance distributed computing (HPDC'02)* (pp. 419 – 419). Edinburgh, Scotland, UK.
- Jelasity, M., Guerraoui, R., Kermarrec, A.-M., & Steen, M. van. (2004, October). The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on middleware (Middleware'04)* (pp. 79 – 98). Toronto, Canada.
- Jelasity, M., & Kermarrec, A.-M. (2006, September). Ordered slicing of very large-scale overlay

- networks. In *Proceedings of the 6th IEEE international conference on peer-to-peer computing (P2P'06)* (pp. 117 – 124). Cambridge, UK.
- Jelasey, M., & Montresor, A. (2004, March). Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of the 24th international conference on distributed computing systems (ICDCS'04)* (pp. 102 – 109). Tokyo, Japan.
- Jelasey, M., Montresor, A., & Babaoglu, O. (2009, August). T-Man: Gossip-based fast overlay topology construction. *Journal Computer Networks: The International Journal of Computer and Telecommunications Networking*, 53(13), 2321 – 2339.
- Jiang, S., & Zhang, X. (2003, December). FloodTrail: an efficient file search technique in unstructured peer-to-peer systems. In *Proceedings of the IEEE global telecommunications conference (GLOBECOM '03)* (Vol. 5, p. 2891 - 2895 vol.5). San Francisco, California.
- Johansen, H., Allavena, A., & Renesse, R. van. (2006, April). Fireflies: Scalable support for intrusion-tolerant network overlays. In *Proceedings of the 1st ACM SIGOPS/EuroSys european conference on computer systems (EuroSys'06)* (pp. 3 – 13). Leuven, Belgium.
- Johnson, D., Lenstra, J., & Kan, A. (1978, December). The complexity of the network design problem. *Networks*, 8(4), 279 – 285.
- Kaashoek, M., & Karger, D. (2003, March). Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd international workshop on peer-to-peer systems (IPTPS'03)*. Cambridge, Massachusetts, USA: Springer Berlin / Heidelberg.
- Karwaczyński, P. (2007, September). Fabric: Synergistic proximity neighbour selection method. In *Proceedings of the 7th international conference on peer-to-peer computing (P2P'07)* (pp. 229 – 230). Galway, Ireland.
- Karwaczyński, P., Konieczny, D., Moćnik, J., & Novak, M. (2007, March). Dual proximity neighbour selection method for peer-to-peer-based discovery service. In *Proceedings of the 22nd ACM symposium on applied computing (SAC'07)* (pp. 590 – 591). Seoul, Korea.
- Kegel, D. (1999, July). *NAT and peer-to-peer networking*. <http://www.alumni.caltech.edu/~dank/peer-nat.html>.

- Kempe, D., Dobra, A., & Gehrke, J. (2003, October). Gossip-based computation of aggregate information. In *Proceedings of the 44th annual IEEE symposium on foundations of computer science (FOCS'03)* (pp. 482 – 491). Washington, DC, USA.
- Kermarrec, A.-M., Massoulie, L., & Ganesh, A. (2003, March). Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(3), 248 – 258.
- Kermarrec, A.-M., Pace, A., Quema, V., & Schiavoni, V. (2009, June). NAT-resilient gossip peer sampling. In *Proceedings of the 29th IEEE international conference on distributed computing systems (ICDCS'09)* (p. 360 -367). Montreal, Quebec, Canada.
- Koldehove, B. (2003, October). Buffer management in probabilistic peer-to-peer communication protocols. In *Proceedings of the 22th IEEE symposium on reliable distributed systems (SRDS'03)* (p. 76-87). Florence, Italy.
- Kulbak, Y., & Bickson, D. (2005). *The eMule protocol specification*. <http://www.cs.huji.ac.il/labs/danss/presentations/emule.pdf>.
- Kun, Z., Zhendong, N., Yumin, Z., & Jun, Y. (2009, November). Group-based search in unstructured peer-to-peer networks. In *Proceedings of the IEEE global telecommunications conference (GLOBECOM'09)* (pp. 1 – 6). Honolulu, Hawaii, USA.
- Lakshman, A., & Malik, P. (2010, April). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35 – 40.
- Lamport, L. (1998, May). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2), 133 – 169.
- Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., & Jones, L. (1996, March). RFC 1928: SOCKS protocol version 5. <http://www.ietf.org/rfc/rfc1928.txt>.
- Leitão, J. (2007). *Gossip-based broadcast protocols*. Master's thesis, Faculdade de Ciências da Universidade de Lisboa. <http://docs.di.fc.ul.pt/jspui/bitstream/10455/3076/1/07-15.pdf>.
- Leitão, J., Pereira, J., & Rodrigues, L. (2007a, October). Epidemic broadcast trees. In *Proceedings of the 26th IEEE international symposium on reliable distributed systems (SRDS'07)* (pp. 301 – 310). Beijing, China.

- Leitão, J., Pereira, J., & Rodrigues, L. (2007b, June). HyParView: a membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)* (pp. 419 – 429). Edinburgh, Scotland, UK.
- Leitão, J., Rosa, L., & Rodrigues, L. (2008, November). Large-scale peer-to-peer autonomic monitoring. In *Proceedings of the IEEE GLOBECOM workshops* (pp. 1 – 5). New Orleans, USA.
- Leong, B., & Li, J. (2004, November). Achieving one-hop DHT lookup and strong stabilization by passing tokens. In *Proceedings of the 12th international conference on networks (ICON'04)* (pp. 344 – 350). Singapore.
- Li, H., Clement, A., Wong, E., Napper, J., Roy, I., Alvisi, L., et al. (2006, November). BAR gossip. In *Proceedings of the 7th symposium on operating systems design and implementation (OSDI'06)* (pp. 191 – 204). Seattle, Washington, USA.
- Li, J., Stribling, J., Morris, R., Kaashoek, M., & Gil, T. (2005, March). A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proceedings of the 24th annual joint conference of the IEEE computer and communications societies (INFOCOM'05)* (pp. 225 – 236). Miami, Florida, USA.
- Liang, J., Ko, S., Gupta, I., & Nahrstedt, K. (2005, December). MON: on-demand overlays for distributed system management. In *Proceedings of the 2nd conference on real, large distributed systems - volume 2 (WORLDS'05)* (pp. 13 – 18). San Francisco, CA.
- Liben-Nowell, D., Balakrishnan, H., & Karger, D. (2002, March). Observations on the dynamic evolution of peer-to-peer networks. In *Proceedings of the 1st international workshop on peer-to-peer systems (IPTPS '02) - revised papers* (pp. 22 – 33). Cambridge, Massachusetts, USA: Springer-Verlag.
- Liu, S., Chen, M., Sengupta, S., Chiang, M., Li, J., & Chou, P. (2010, June). P2P streaming capacity under node degree bound. In *Proceedings of the IEEE 30th international conference on distributed computing systems (ICDCS'10)* (pp. 587 – 598). Genoa, Italy.
- Lv, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002, June). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on supercomputing*

- (ICS'02) (pp. 84 – 95). New York, New York, USA.
- Maniyamaran, B., Bertier, M., & Kermarrec, A.-M. (2007, June). Build one, get one free: Leveraging the coexistence of multiple P2P overlay networks. In *Proceedings of the IEEE 27th international conference on the distributed computing systems (ICDCS'07)* (pp. 33 – 40). Toronto, Canada.
- Massoulie, L., Kermarrec, A.-M., & Ganesh, A. (2003, October). Network awareness and failure resilience in self-organising overlay networks. In *Proceedings of the 22nd symposium on reliable distributed systems (SRDS'03)* (p. 47-55). Florence, Italy.
- Maymounkov, P., & Mazières, D. (2002, March). Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st international workshop on peer-to-peer systems (IPTPS '02) - revised papers* (pp. 53 – 65). Cambridge, Massachusetts, USA: Springer-Verlag.
- Melamed, R., & Keidar, I. (2004, August). Araneola: A scalable reliable multicast system for dynamic environments. In *Proceedings of the 3rd IEEE international symposium on network computing and applications (NCA'04)* (pp. 5 – 14). Cambridge, Massachusetts, USA.
- Minsky, Y., Trachtenberg, A., & Zippel, R. (2003, September). Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9), 2212 – 2218.
- Mizrak, A., Cheng, Y., Kumar, V., & Savage, S. (2003, June). Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proceedings of the 3rd IEEE workshop on internet applications. (WIAPP'03)* (pp. 104 – 111). San Jose, California, USA.
- Montresor, A., & Jelasity, M. (2009, September). PeerSim: A scalable P2P simulator. In *Proceedings of the 9th international conference on peer-to-peer (P2P'09)* (pp. 99 – 100). Seattle, WA.
- Moura, M. de, & Endler, M. (2003). Managing adaptive fault tolerant CORBA applications. In R. de Lemos, T. Weber, & J. Camargo (Eds.), *Dependable computing* (Vol. 2847, pp. 160 – 180). Springer Berlin/Heidelberg.
- Mullender, S. (Ed.). (1993). *Distributed systems, 2nd edition*. Addison-Wesley.

- Padmanabhan, V., Wang, H., Chou, P., & Sripanidkulchai, K. (2002, May). Distributing streaming media content using cooperative networking. In *Proceedings of the 12th international workshop on network and operating systems support for digital audio and video (NOSSDAV'02)* (pp. 177 – 186). Miami, Florida, USA.
- Paiva, J., Leitão, J., & Rodrigues, L. (2011a, May). *Rollerchain: a DHT for high availability* (Tech. Rep.). Lisbon, Portugal. <http://www.inesc-id.pt/ficheiros/publicacoes/7240.pdf>.
- Paiva, J., Leitão, J., & Rodrigues, L. (2011b, December). Rollerchain: a DHT for high availability (Poster). In *Proceedings of the workshop on posters and demos track of the ACM/IFIP/USENIX 12th international middleware conference (Middleware'11)* (pp. 17:1 – 17:2). Lisbon, Portugal.
- Papadakis, H., Roussopoulos, M., Fragopoulou, P., & Markatos, E. (2009, September). Imbuing unstructured P2P systems with non-intrusive topology awareness. In *Proceedings of the IEEE 9th international conference on peer-to-peer computing (P2P'09)* (pp. 51 – 60). Seattle, Washington, USA.
- Pereira, J., Rodrigues, L., Monteiro, M., Oliveira, R., & Kermarrec, A.-M. (2003, October). NEEM: Network-friendly epidemic multicast. In *Proceedings of the 22th IEEE symposium on reliable distributed systems (SRDS'03)* (pp. 15 – 24). Florence, Italy.
- Pereira, J., Rodrigues, L., Pinto, A., & Oliveira, R. (2004, October). Low-latency probabilistic broadcast in wide area networks. In *Proceedings of the 23th IEEE symposium on reliable distributed systems (SRDS'04)* (pp. 299 – 308). Florianopolis, Brazil.
- Powell, D. (1994, February). Distributed fault tolerance: lessons from Delta-4. *IEEE Micro*, 14(1), 36 – 47.
- Præsthholm, S., Schwefel, H.-P., & Andersen, S. (2007, June). A comparative study of forward error correction and frame accumulation for VoIP over congested networks. In *Proceedings of the 20th international teletraffic conference on managing traffic performance in converged networks (ITC20'07)* (pp. 374 – 385). Ottawa, Canada: Springer-Verlag.
- Rabbe, L. (2010, December). *CIO update: Post-mortem on the skype outage*. http://blogs.skype.com/en/2010/12/cio_update.html. The Big Blog: Skype Development Blog.

- Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001, September). A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)* (pp. 161 – 172). San Diego, California, USA.
- Ratnasamy, S., Handley, M., Karp, R. M., & Shenker, S. (2001, November). Application-level multicast using content-addressable networks. In *Networked group communication: Proceedings of the 3rd international COST264 workshop, (NGC'01)* (pp. 14 – 29). London, UK: Springer Berlin / Heidelberg.
- Renesse, R. van, Birman, K. P., & Vogels, W. (2003, May). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems (TOCS)*, 21(2), 164 – 206.
- Renesse, R. van, Minsky, Y., & Hayden, M. (1998). *A gossip-style failure detection service* (Tech. Rep.). Ithaca, NY, USA. <http://ecommons.cornell.edu/handle/1813/7341>.
- Reynolds, P., & Vahdat, A. (2003, June). Efficient peer-to-peer keyword searching. In *Proceedings of the ACM/IFIP/USENIX 2003 international conference on middleware (Middleware'03)* (pp. 21 – 40). Rio de Janeiro, Brazil.
- Rhea, S., Geels, D., Roscoe, T., & Kubiatowicz, J. (2004, June). Handling churn in a DHT. In *Proceedings of the USENIX annual technical conference (ATEC'04)* (pp. 10 – 20). Boston, Massachusetts, USA.
- Risson, J., Harwood, A., & Moors, T. (2006, June). Stable high-capacity one-hop distributed hash tables. In *Proceedings of the 11th IEEE symposium on computers and communications (ISCC'06)* (p. 687-694). Cagliari, Italy.
- Rostami, H., & Habibi, J. (2007, May). Topology awareness of overlay P2P networks. *Concurrency and Computation: Practice & Experience - Autonomous Grid*, 19, 999 – 1021.
- Rowstron, A., & Druschel, P. (2001, November). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM international conference on distributed systems platforms heidelberg (Middleware'01)* (pp. 329 – 350). Heidelberg, Germany.

- Rowstron, A., Kermarrec, A.-M., Castro, M., & Druschel, P. (2001, November). Scribe: The design of a large-scale event notification infrastructure. In *Networked group communication: Proceedings of the 3rd international COST264 workshop, (NGC'01)* (p. 30-43). London, UK: Springer Berlin / Heidelberg.
- Sahota, V., Li, M., Baker, M., & Antonopoulos, N. (2009, March). A grouped P2P network for scalable grid information services. *Peer-to-Peer Networking and Applications*, 2(1), 3 – 12.
- Schneider, F. (1990, December). Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4), 299 – 319.
- Shafaat, T., Ghodsi, A., & Haridi, S. (2007, September). Handling network partitions and mergers in structured overlay networks. In *Proceedings of the 7th IEEE international conference on peer-to-peer computing (P2P'07)* (pp. 132 – 139). Galway, Ireland.
- Stoica, I., Morris, R., Karger, D., Kaashoek, M., & Balakrishnan, H. (2001, September). Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)* (pp. 149 – 160). San Diego, California, USA.
- Stutzbach, D., & Rejaie, R. (2006, October). Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on internet measurement (IMC'06)* (pp. 189 – 202). Rio de Janeiro, Brazil.
- Tang, C., & Ward, C. (2005, June). GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *Proceedings of the 35th annual IEEE/IFIP international conference on dependable systems and networks (DSN'05)* (pp. 140 – 149). Yokohama, Japan.
- Tsoumakos, D., & Rousopoulos, N. (2003, September). Adaptive probabilistic search for peer-to-peer networks. In *Proceedings of the 3rd international conference on peer-to-peer computing (P2P'03)* (pp. 102 – 109). Linköping, Sweden.
- Tsoumakos, D., & Rousopoulos, N. (2006, May). Analysis and comparison of P2P search methods. In *Proceedings of the 1st international conference on scalable information systems (InfoScale'06)* (pp. 25 – 34). Hong Kong.
- Venkataraman, V., Yoshida, K., & Francis, P. (2006, November). Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Proceedings of the 14th IEEE international*

- conference on network protocols*, 2006. (ICNP'06) (pp. 2 – 11). Santa Barbara, California, USA.
- Voulgaris, S., Gavidia, D., & Steen, M. (2005, June). CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2), 197 – 217.
- Winick, J., & Jamin, S. (2002, May). *Inet-3.0: Internet topology generator* (Tech. Rep. No. UM-CSE-TR-456-02). <http://www.eecs.umich.edu/techreports/cse/02/CSE-TR-456-02.pdf>.
- Wong, B., & Guha, S. (2008, February). Quasar: A probabilistic publish-subscribe system for social networks. In *Proceedings of the 7th international conference on peer-to-peer systems (IPTPS'08)* (pp. 2 – 7). Tampa Bay, FL, USA.
- Yang, B., & Garcia-Molina, H. (2002, July). Improving search in peer-to-peer networks. In *Proceedings of the 22nd international conference on distributed computing systems (ICDCS'02)* (pp. 5 – 14). Vienna, Austria.
- Yang, B., & Garcia-Molina, H. (2003, March). Designing a super-peer network. In *Proceedings of the 19th international conference on data engineering* (pp. 49 – 60). Bangalore, India.
- Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., & Kubiawicz, J. (2004, January). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 41 – 53.
- Zhuang, S., Zhao, B., Joseph, A., Katz, R., & Kubiawicz, J. (2001, June). Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on network and operating systems support for digital audio and video (NOSSDAV'01)* (pp. 11 – 20). Port Jefferson, New York, USA.