Self-Organizing Topology Adaptation in Peer-to-Peer Networks

Atul Singh

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy

June 2007

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this or any other University, it is entirely my own work and I agree that the Library may lend or copy the thesis upon request.

Atul Singh

Dated: September 26, 2007

This thesis is dedicated to my parents whose love, sacrifice and generosity shaped me.

Acknowledgements

I would like to thank my wife for her motivation and all the tasty meals that fuelled this thesis. I am grateful to my supervisor for his support, advise and patience without which this thesis would not have been possible. This research work has been funded by the Irish Research Council for Science, Engineering and Technology (IRCSET) under the Basic Research Grant scheme. I am grateful to them for their support and confidence.

I would like to thank Dr. Thomas C. Schelling whose visionary work is the keel of this thesis. I would also like to thank the wonderful people in my research group for their cooperation and entertaining emails and the gracious staff of the college for their support.

Abstract

The peers in a Peer-to-Peer (P2P) system arrange themselves in a virtual network called the overlay network. The overlay network sits above the underlying physical network and is used to search for resources and peers, and to route messages between peers. The topology of the overlay network is the graph whose vertices are the peers and whose edges are the connections between the peers. The choice of topology affects application performance and the correct choice depends on the application. The literature contains examples of topologies that can be used to implement various types of applications.

Topology adaptation algorithms are used to maintain an overlay network's topology as per application requirements. Early P2P applications either did not use topology adaptation algorithms or used simple solutions based on a centralized component that results in poor fault-tolerance and scalability. Since 2002, decentralized algorithms have been proposed that may be used to create and maintain specific topologies. Although this is certainly an improvement, these algorithms are typically designed to create a specific topology for a specific application and are therefore not generally applicable or easily reusable. As a result, topology adaptation algorithms have to be designed from scratch and an application that has been developed using an existing algorithm cannot be altered easily to create and maintain a new topology.

This thesis presents a general approach to decentralized topology adaptation in unstructured P2P overlay networks. The approach is based on a single abstract algorithm called the PEer-to-peer Self-organizing TOpology (PESTO), which can be used to develop a family of self-organizing topology adaptation algorithms for unstructured decentralized P2P overlay networks. PESTO is inspired by an idea from sociology, proposed by economist Thomas Schelling, to explain the existence of segregated neighborhoods in urban areas. An instantiation of PESTO requires formulation of rules that use a peer's local awareness of the overlay network topology and result in overall emergent behavior of the network.

To validate PESTO, we present and evaluate three concrete realizations. Two of these can be used to cluster peers with a given property, such as geographical location. The third realization can be used to create a backbone network of powerful computers called hubs, which can be used for search and for routing of messages between peers. Simulations are used to demonstrate that the concrete realizations can create self-organizing overlay network topologies with desirable properties. The three concrete realizations show that PESTO is sufficiently general to be applicable across a range of applications.

The topology adaptation algorithms are placed in the context of a general-purpose P2P reference architecture, called the P2P reference Architecture that Supports Topology Adaptation (PASTA). A distinguishing feature of PASTA is that it contains a component that allows topology adaptation using the novel approach presented in the thesis. In addition, PASTA serves as a template, which can be used as a starting point for designing new P2P applications and middlewares. PASTA is sufficiently general to allow the description and comparison of the structure of existing P2P applications which is shown through structural description of a number of existing P2P applications. PASTA's ability to describe and compare P2P applications can be used to enhance the understanding of the P2P domain and the research done on it.

Contents

Acknowledgements i

Summary i

List of Figures ix

List of Tables xiv

Chapter 1 Introduction 1

1.1 Trends in Computing 2

1.2 Motivation 3

1.2.1 P2P Topology Adaptation 4

1.2.2 Reference Architecture 6

1.3 Defining Peer-to-Peer 7

1.3.1 First Category 8

1.3.2 Second Category 9

1.4 Terminology and Background 13

1.4.1 Self-Organization 14

1.4.2 Reference Architecture 14

1.4.3 Overlay Network 15

1.4.4 Topology Adaptation 16

1.4.5 Types of P2P Networks 17
1.4.6 Schelling's Model 18
1.5 The Thesis 19
1.6 Thesis Roadmap 21

Chapter 2 State of the Art **22**
2.1 Topology 22
2.1.1 Common Topology Metrics 23
2.1.2 Simple Topologies 25
2.1.3 Random Topology 27
2.1.4 Small World Topology 27
2.1.5 Power Law Topology 29
2.1.6 Super Peer and Hub Topologies 32
2.1.7 Clustered Topology 33
2.1.8 Summary 35
2.2 Topology Adaptation Algorithms 36
2.2.1 Improving Breadth First Search (BFS) 37
2.2.2 Building a Low-Diameter Connected Topology 39
2.2.3 Topology Adaptation For Load Balancing 41
2.2.4 Load Balancing in Biased Random Walk Search 43
2.2.5 Changing Topology Type for Load Balancing 46
2.2.6 Topology Adaptation for Distance-Based Clustering 48
2.2.7 Topology Adaptation for Content-Based Clustering 48
2.2.8 T-MAN 50
2.2.9 Summary 53
2.3 Reference Architecture 56
2.3.1 The Essence of P2P 56
2.3.2 JXTA 59

2.3.3 Summary 64
2.4 Summary 65

Chapter 3 Topology Adaptation Using Schelling's Model 66

3.1 Agent-Based Models 67
3.2 Schelling's Model 68
3.3 PESTO 72
3.3.1 Description 73
3.3.2 Messaging Cost 76
3.3.3 Designing Algorithms Using PESTO 77
3.3.4 Comparison with T-Man 79
3.4 Simulator 81
3.4.1 Description 81
3.4.2 Using the Simulator 82
3.5 Summary 87

Chapter 4 Clustered Topology 88

4.1 Clustering Algorithms 88
4.1.1 Messaging Cost 92
4.1.2 Time Delay 93
4.1.3 Analysis 93
4.2 Simulations 95
4.2.1 Metrics 96
4.2.2 Static Simulations 99
4.2.3 Dynamic Simulations 119
4.2.4 Discussion 124
4.3 Improving the Effective Bandwidth Between Peers 125
4.3.1 Percentage Increase in BBN 125

4.3.2 Bandwidth Distribution 126
4.3.3 Static Simulations 127
4.3.4 Dynamic Simulations 129
4.3.5 Summary 130
4.4 Related Work 131
4.5 Summary 133

Chapter 5 Hub Topology 134

5.1 HubAlgorithm 134
5.1.1 Messaging Cost of the HubAlgorithm 138
5.1.2 Time Delay 138
5.2 Simulations 139
5.2.1 Metrics 140
5.2.2 Static Simulations 142
5.2.3 Dynamic Simulations 143
5.2.4 Discussion 145
5.3 Related Work 146
5.4 Summary 147

Chapter 6 Reference Architecture 148

6.1 Service-Oriented Architecture 149
6.2 Documenting Software Architectures 150
6.3 Core Concerns for a P2P Application 152
6.3.1 Naming 153
6.3.2 Overlay Network Management 154
6.3.3 Service and Resource Management 155
6.3.4 Security 156
6.4 P2P Reference Architecture 156

6.4.1 Core Servents 158
6.4.2 Common Runtime 162
6.4.3 Topology Adaptation 165
6.4.4 Generic Servent 166
6.4.5 Interoperability 169
6.5 Validation of PASTA 170
6.5.1 Gnutella 171
6.5.2 Freenet 173
6.5.3 JXTA 176
6.5.4 Jabber 178
6.5.5 Conclusion 180
6.6 Comparison with Essence of P2P 180
6.7 Summary 181

Chapter 7 Conclusion **183**
7.1 Achievements 183
7.2 Future Work 185
7.3 Closing Remarks 186

Bibliography **187**

List of Figures

1.1 A plot of the improvement in CPU processing power, hard disk capacity and available memory of laptop computers [116]. 3
1.2 A plot of the number of computers connected to the Internet [18]. . . . 4
1.3 A P2P overlay network. An overlay network is the graph whose vertices are the peers and the edges are the connections between them. 16
1.4 A taxonomy of P2P systems based on overlay architecture. 18

2.1 Examples of simple topologies. 26
2.2 Examples of a super peer topology. 30
2.3 XML schema of a peer advertisement. 60

3.1 Schelling's world before (left) and after (right) the turtles are satisfied. 69
3.2 Plot of PTASN vs PSND. 71
3.3 UML Class diagram of the TopologyAdapter abstract class. The manageTopology method presents the structure of PESTO. The methods calculateSatisfaction, executeAdaptation and delayBeforeNextAdaptation are virtual abstract methods implemented in subclasses. 72
3.4 UML Class diagram of the TopologyGenerator Interface. 83
3.5 UML Class diagram of the SimulatorInterceptor class. 86

4.1 UML Class diagram depicting the TopologyAdapter interface and two concrete realizations. 90
4.2 A plot of relative decrease in the number of clusters after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 100
4.3 A plot of relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 100
4.4 A plot of the maximum, mean and minimum values of the relative decrease in the number of clusters after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four different power-law topologies with 5,000 peers. 101
4.5 A plot of standard deviation of RDNC, TC and MC against $PNSP_{desired}$ after applying SelflessClustering algorithm on four different power-law topologies with 5,000 peers. 101
4.6 A plot of PPTC after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 102
4.7 A plot of PPTC after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 102
4.8 Trend curves that show the relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 106
4.9 Trend curves that show the PPTC after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 106

4.10 Trend curves that show the maximum, mean and minimum values of the relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four different power-law topologies with 5,000 peers. 107
4.11 A plot of standard deviation of RDNC, TC and MC against $PNSP_{desired}$ after applying SelfishClustering algorithm on four different power-law topologies with 5,000 peers. 107
4.12 A plot of the cluster Coverage (C) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 109
4.13 A plot of the cluster Coverage (C) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers. 110
4.14 A plot of the Time to Converge (TC) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 111
4.15 A plot of the Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers. 112
4.16 Trend curves that show the Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers. 112
4.17 A plot of the maximum, mean and minimum Time to Converge (TC) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers. 113

4.18 The maximum, mean and minimum Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers. 113

4.19 A plot of the number of messages exchanged to perform search after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 115

4.20 A plot of the number of messages exchanged to reconnect disconnected peers after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers. 115

4.21 A plot of the maximum, mean and minimum number of messages exchanged to perform search after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers. 116

4.22 The maximum, mean and minimum values of the trend curves for the messages exchanged to reconnect disconnected peers after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers. 117

4.23 A plot of number of clusters (NC) against simulator iterations when the SelflessClustering and SelfishClustering Algorithm are applied. 120

4.24 The top figure shows a plot of PPTC against simulator iterations when the SelflessClustering Algorithm is applied. The bottom figure shows a plot of PPTC against simulator iterations when the SelfishClustering Algorithm is applied. 121

4.25 The top figure shows a plot of Coverage (C) against simulator iterations when the SelflessClustering Algorithm is applied. The bottom figure shows a plot of Coverage (C) against simulator iterations when SelfishClustering Algorithm is applied. 122

4.26 A plot of PIBBN after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500 and 1,000 peers. . . 126
4.27 A plot of PIBBN after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500 and 1,000 peers. . . 127
4.28 A plot of PIBBN against simulator iterations after applying the Selfless-Clustering and the SelfishClustering algorithms. 130

5.1 The HubAlgoritm is a concrete concrete realization of PESTO. 135
5.2 A plot of total number of messages exchanged against simulator iterations for dynamic simulations. 143
5.3 A plot of hub-hub and peer-hub degree against simulator iterations, for dynamic simulations. 144

6.1 A UML diagram providing a high level overview of the P2P reference architecture. 157
6.2 A UML diagram depicting the conceptual view of the core servents of the reference architecture. The core servents handle most of the concerns which need to be addressed by a P2P application. 158
6.3 A UML diagram presenting a conceptual view of the runtime component.162
6.4 A UML diagram presenting a conceptual view of the ServentCache component. 164
6.5 A UML diagram of the conceptual view of a generic servent. All the connectors depict local method invocation. The Routing component may use the Client component. 168

List of Tables

1.1 Defining features of a P2P application according to different P2P definitions. Bullet means that the definition suggests that the feature is a necessary criterion for calling an application P2P. 13

2.1 Components defined in JXTA Core Specification and Standard Services. 60

3.1 The operations that can be executed by the peers. The operations are used in this thesis to describe the concrete realizations of PESTO. . . . 74

3.2 A satisfaction criterion and topology adaptation steps that can be used to bring together peers with similar properties (e.g., bandwidth). 74

5.1 Satisfaction criteria and topology adaptation steps that will be executed by the hubs and the ordinary peers to create a backbone network of hubs within the overlay network. 137

5.2 $H_{maxCritical}$ and TC and MC at $H_{maxCritical}$ 142

6.1 The UML 2.0 elements used to draw component-connector diagrams. . 152

6.2 The core concerns that a P2P application needs to address. 153

6.3 The core servents of the reference architecture and the concerns they handle. The access concern is handled by the client and server components of the common runtime. The routing concern is handled by the individual servents. 159

6.4 The protocol obeyed by the ports of the core-servents in the reference architecture. 160
6.5 The protocol obeyed by the ports of the runtime components. 163
6.6 The protocol obeyed by the ports of the ServentCache component. . . . 164
6.7 Describing Gnutella using PASTA. 172
6.8 The ServentCache and the Routing Policy used in Freenet. 174
6.9 Describing Freenet Using PASTA. 175
6.10 A mapping between the JXTA protocols and PASTA. 177
6.11 The application servents provided by Jabber and the policies used by Jabber to implement the different core servents. The servents in Jabber do not use a ServentCache. The servent implementations in Jabber do not have routing capability. 179

Chapter 1

Introduction

> I'm going to show them a world without you, a world without rules and controls, without borders or boundaries, a world where anything is possible. Where we go from there is a choice I leave to you. The Matrix [123]

According to the Merriam-Webster dictionary the word peer means, "one that is of equal standing with another" [10]. Literally translated, P2P means equals communicating with equals [66]. As the literal translation suggests, P2P in the distributed systems domain deals with distributed applications in which instances of the application, called peers, running on different computers are equal in capability. Since all the peers in a P2P application can provide services, the P2P design paradigm can be used to utilize the resources of the computers that were previously relegated to being consumers of services as clients in the client-server model.

This thesis addresses the general area of P2P computing. The thesis presents a novel approach for designing P2P overlay network topology adaptation algorithms for use in unstructured P2P overlay networks. The thesis presents a family of algorithms that can be used to create and maintain the overlay network's topology. The topology adaptation algorithms are placed in the context of a reference architecture, which contains a topology adaptation component that can use the topology adaptation algorithms. The architecture is sufficiently general to allow the description and comparison of the structure of existing P2P applications and middlewares.

This chapter presents the background information required to understand the work in this thesis. The chapter first reviews the trends in computing that are contributing towards making P2P a popular choice for designing distributed applications and then presents the motivation behind the work in this thesis. We also define a number of key terms related to the thesis, before we present the thesis's key contributions.

1.1 Trends in Computing

Moore's Law [76], an empirical observation by Gordon Moore who cofounded Intel Corporation, suggests that the number of transistors on an integrated circuit (a measure of computer processing power) doubles every eighteen months [94, pg. 69]. Since porposed in 1965, this prediction has generally proved to hold. The processing power of desktop and mobile computers have been increasing rapidly with time as suggested by Moore's Law. In addition the memory, disk space and network connectivity of computers are also continuing to improve. Figure 1.1 shows the improvement in CPU processing power, hard disk capacity and available RAM in laptop computers [116].

At the same time, the number of computers with Internet connectivity is continually growing. Figure 1.2 shows the results of a study conducted twice per year by the Internet Systems Consortium (ISC). Their most recent report observed a 24% increase in the number of hosts online from January 2005 to the same month in 2006 [18].

One effect of these developments is that an increasing number of the computational resources reside on peoples desks in their homes and offices. These computers are connected to intranets and the internet. These computers are generally designed to satisfy the peak demand of individuals (in particular, human impatience regarding response time), but the typical personal and office use (web surfing, emailing, spreadsheets and

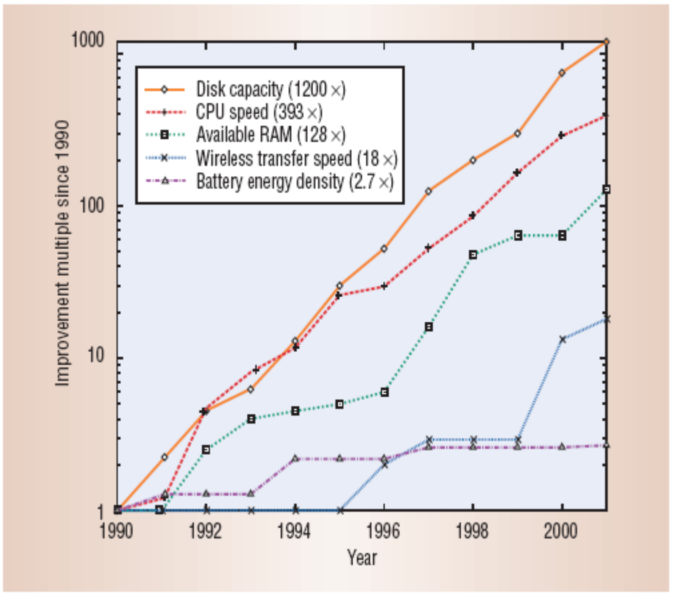


Fig. 1.1: A plot of the improvement in CPU processing power, hard disk capacity and available memory of laptop computers [116].

word processing) only consumes a fraction of a computers resources when measured over time. As a result, an increasing portion of the planet's computational resources are idle much of the time. P2P applications utilize the resources of these computers by allowing all the computers running the P2P software to become both consumers and suppliers of information and services.

1.2 Motivation

This section presents the motivation behind the work presented in this thesis. The motivation behind P2P topology adaptation, the primary contribution of this thesis, is discussed first and is followed by the motivation for the reference architecture which forms the secondary contribution of the thesis.

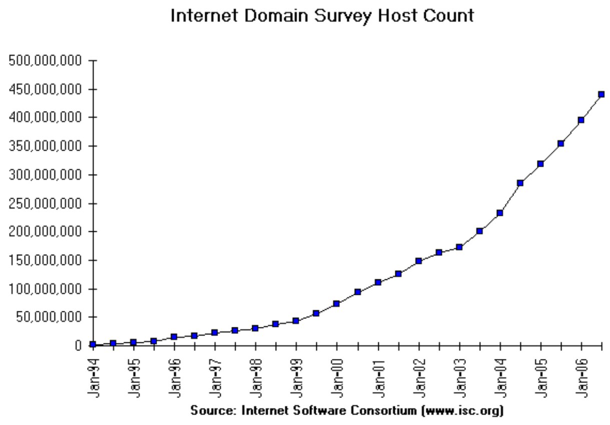


Fig. 1.2: A plot of the number of computers connected to the Internet [18].

1.2.1 P2P Topology Adaptation

The peers in a P2P system arrange themselves in a virtual network called the overlay network. The overlay network uses the underlying physical network for communication. The overlay network is used to perform tasks like searching for resources and services available on the overlay network and for routing messages between peers. The topology of the overlay network is the graph whose vertices are the peers and whose edges are the connections between the peers.

The overlay network topology affects the efficiency of tasks like search and routing of messages on the network. For example, P2P applications such as Gnutella [92], use Breadth First Search (BFS) to search for an entity on the overlay network. In BFS, the search request is forwarded to all the neighbors of a peer. Each search request has a time-to-live value that is decremented by one at each peer that receives the search request. The search request is forwarded by the peers until its time-to-live reaches zero. The coverage of a BFS search can be improved by reducing the overlay network's diameter so that a search request can reach a large number of peers on the overlay network.

Another popular way of arranging the peers is the super peer topology used in popular P2P applications like KaZaA [9] and Skype [13, 26]. If the high bandwidth consumed by the large number of messages exchanged to perform Gnutella-style BFS search is an issue for the application, then it may be desirable to create a backbone network of powerful peers that maintain a directory of resources on the network. A search request can then be directed to one of these powerful peers, which can collaborate with the other powerful peers to process the search request by using the distributed directory that they maintain. Such a backbone network may also be used as a shortcut for routing messages to remote peers on the overlay network.

Regardless of topology, the overlay network is used by the P2P application to implement its services. However, the most suitable overlay network topology depends on the P2P application, and the range of possible topologies is considerable. For example, a distributed game application can be made more challenging by bringing together peers with similar competency levels as neighbors. A P2P content distribution application or a P2P media streaming application would benefit from an overlay network topology in which geographically close peers are clustered together as neighbors, so that the content is distributed through neighbors that are physically close to each other and therefore avail of good bandwidth and latency characteristics.

It is desirable to have algorithms (called topology adaptation algorithms) that can create and maintain a topology as per application requirements. Such algorithms would allow an application to have a desired topology so that it can function efficiently. P2P applications typically lack a central component to improve scalability and fault-tolerance. Autonomous peers that act using their limited awareness of the overlay network, and the lack of an omniscient central component makes it challenging to design algorithms that can create a required topology. The flux of peers on an overlay network makes it difficult to maintain the topology as required by an application.

Initial P2P applications either did not use topology adaptation algorithms or used simple solutions based on a centralized component that results in poor fault-tolerance. The centralized component also acts as a bottleneck resulting in poor scalability. Since 2002, decentralized topology adaptation algorithms have been proposed in the literature (e.g., [105, 85, 70, 34, 93, 89, 118, 22, 122, 69, 79, 115] and [27]) that may be used to create specific topologies. Although this is certainly an improvement, these algorithms are typically designed to create a specific topology for a specific application and are therefore not generally applicable or easily reusable. For example, a topology adaptation algorithm designed for clustering can not be easily used for load balancing. As a result, topology adaptation algorithms have to be designed from scratch and an application that has been developed using an existing algorithm can not be altered easily to create and maintain a new topology. It is desirable to have an approach that facilitates the development of topology adaptation algorithms.

1.2.2 Reference Architecture

Since the dramatic success of Napster [11] in 1999, many new P2P applications have been developed. At the time of writing, Wikipedia lists more than one hundred such applications [17]. A recent survey show that P2P applications generate a high percentage of traffic on the Internet [97]. The high popularity of P2P applications among Internet users is because of their ability to allow users to share information and services without the need of servers managed by third parties. A forthcoming challenge for the developers of P2P applications is to ensure that these numerous P2P applications can interoperate with each other. This is referred to as the interoperability problem in this thesis. Despite the large number of P2P applications and middlewares, there also remains a lack of a common vocabulary to accurately describe and compare the structure of these middlewares and applications. A vocabulary to describe the structure of a P2P application can be used to classify the research done on the P2P domain on the basis of the application components it is intended for, and in this way enhance the understanding of the P2P domain and the research done on it.

Software architectures are used to communicate the high-level design of a system to a diverse audience, which may include programmers, managers and customers. The software architecture design phase is used to make design decisions that affect a system's development, deployment and maintenance phases. The growing popularity of P2P applications has given rise to numerous P2P middlewares (e.g., JXTA [80], MSN P2P [72]) that facilitate P2P application development. However, despite the popularity of P2P, the software architecture for any new P2P system is typically designed from scratch. The designers spend time in identifying the concerns that the application needs to address and produce a software architecture that addresses these concerns. It is therefore desirable to reduce the time spent on designing applications by developing a generally applicable software architecture of P2P applications.

1.3 Defining Peer-to-Peer

P2P applications are distributed systems. Distributed systems are those in which components, located at networked computers, communicate and coordinate their actions only by passing messages [38]. Distributed systems can be broadly classified into those based on the client-server model and those based on the P2P model. In a client-server model the client makes a request for a service to a server which provides the service and replies with a response. A program can play both the roles (i.e., client and server) but for a different purpose. The idea behind P2P computing is that each peer can act both as a client and as a server (a trait that is referred as *equivalent peers* in this thesis) in the context of some application [25]. Because a peer can act as a server, it can share resources like processing power, memory and bandwidth with other peers on the overlay network. Since each peer can act as a server, P2P applications typically do not require a well-known central server for providing services. The lack of requirement of a central server is referred as serverless design in this thesis.

In the literature, the term P2P is used to describe a wide variety of software applications. The applications that have been classified as P2P come from a diverse range of domains such as file-sharing, distributed computing, instant messaging and content distribution. In the literature, there is a lack of agreement on the set of criteria that should be used to call an application P2P [25, 73]. Computers that are connected to the Internet, but have variable connectivity and temporary network addresses are often called "computers on the edge of the Internet" [81]. For the purpose of this review, we discuss the definitions in two groups, depending on whether they emphasize the ability of P2P to utilize computers at the edge of the Internet, as the key defining characteristics of P2P. The definitions in the first group do emphasize the ability of P2P to utilize computers at the edge of the Internet, as the key defining characteristics of P2P, whereas those in the second category do not. The discussion below presents and analyzes definitions in both the categories of P2P and uses them to produce the criteria that will be adopted in this thesis to classify an application as P2P.

1.3.1 First Category

Definition 1

Published in 2001, in one of the earliest books on P2P computing the definition below was proposed for P2P computing:

> An application can be called P2P if it satisfies the following two litmus tests:
>
> 1. Allows for variable connectivity and temporary network addresses?
> 2. Gives the nodes at the <u>edge of the network</u> significant autonomy?
>
> [81, pg. 22]

Definition 2

A white paper from Microsoft defines P2P as follows:

> P2P networking is the utilization of the relatively powerful computers (personal computers) that exist at the edge of the Internet for more than just client based computing tasks [72].

Both definitions [1] emphasize the ability to utilize the computers on the edge of the Internet as a defining trait of P2P applications. These computers are inaccessible from other machines on the Internet because they have dynamic IP addresses and use Network Address Translation (NAT). According to the above definitions, the potential of P2P applications lie in their ability to utilize these computers. However making this the defining characteristics for P2P applications excludes P2P applications used in Intranets, where P2P applications are extremely useful for tasks such as collaboration. P2P applications deployed in an Intranet do not require the capability to utilize the computers at the edge of the Internet. While the ability to utilize computers at the edge of the Internet is a benefit of P2P applications we will not consider it a defining characteristic in this thesis.

1.3.2 Second Category

This section examines definitions of P2P in the second category, in order to identify the characteristics used in the existing literature to classify an application as P2P. Some existing P2P definitions listed in no particular order are:

[1]In both the definitions, underline is used to emphasize the key points of the definition and is not part of the original text.

Definition 3

In 2003, Detlef Schoder and Kai Fischbach proposed the following definition for P2P in CACM:

> P2P refers to technology that enables two or more peers to collaborate spontaneously in a network of equals by using appropriate information and communication systems without the necessity for central coordination [102].

This definition identifies three key aspects of a P2P system: no necessity for central coordination, participation in a network of equals and spontaneous collaboration. The former of these traits is equivalent to the serverless design mentioned earlier. In a P2P system, peers can collaborate and communicate with each other without a central authority. The network of equals trait is similar to the equivalent peers trait mentioned earlier. A peer in P2P computing can act both as a server and a client for same application. To achieve this, each peer is equivalent in functionality.

P2P systems are good at spontaneous collaboration because they do not need a centralized component for peers to communicate with each other. Spontaneous collaboration is also facilitated by the fact that the installation of special server software is not required while using P2P, because the software running on all the peers is equivalent in functionality. Spontaneous collaboration is an advantage and a direct consequence of the serverless design and equivalent peers, and so it is not considered a key defining feature of P2P systems in this thesis.

The serverless design and equivalent peers features discussed above are not completely displayed by all P2P systems. This is especially true for hybrid P2P systems such as Napster (1999) [11]. Hybrid systems have a central server that exercises centralized control over certain aspects of the operation of a P2P system. However the remaining functionality of the system operates without any central control, so such systems can still be classified as P2P systems. The central server has more capabilities and responsibilities when compared with other peers in the system, but the rest of the peers are equivalent in functionality. For example in Napster (1999) the central server is used for resource discovery. In Napster (1999) the location of a file is found through the central server only, but files are exchanged in a P2P fashion by direct communication between the peers.

Definition 4

Barkai proposes the following definition for P2P in his book:

> P2P computing is a network-based computing model for applications where computers share resources via direct exchange between participating computers [25].

Definition 5

In 2001, at the IEEE P2P conference the following definition was presented for P2P:

> A distributed network architecture may be called Peer-to-Peer (P-to-P, P2P,...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers,...). These shared resources are necessary to provide the Service and content offered by the network (e.g., file sharing or shared workspaces for collaboration). They are accessible by other peers directly without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requester (Servent-concept) [103].

Definition 6

A whitepaper from Intel proposes the definition below for P2P:

P2P can be defined as any application or network solution that supports the direct exchange of resources between computers without relying entirely on a central server [57].

The definitions 4 and 6 above emphasize the serverless design of P2P applications as a defining characteristic. All the definitions above emphasize the ability to share and exchange resources by direct communication between peers as a defining trait of P2P systems. While applications where users can share resources (e.g., files) have made P2P applications popular, P2P is not limited to applications that deal with the exchange of resources. P2P is also used for tasks like content distribution [37] and knowledge management [57]. The direct communication between peers is a consequence of the serverless design. P2P is not limited to exchange of resources and the direct communication between peers is a consequence of the serverless design. For this reason, the ability to share and exchange resources by direct communication between peers is not used as a defining characteristics of P2P systems in this thesis.

To summarize, according to the definitions above, the defining characteristics of a P2P systems are:

A Lack of central control (serverless design),

B Peers equivalent in functionality (equivalent peers),

C Utilizing the computers at the edge of the Internet,

D Sharing of resources,

E Spontaneous collaboration.

Table 1.1 shows the defining characteristic of P2P as suggested by each of the six definitions. Our discussion found that C, D and E are not key defining features of P2P systems. This leads to the definition of P2P used in this thesis which is:

	Features				
Definition	Serverless Design (A)	Equivalent Peers (B)	Use Computers At Edge (C)	Sharing Resources (D)	Spontaneous Collaboration (E)
1			●		
2			●		
3	●	●			●
4	●			●	
5	●			●	
6	●	●		●	

Table 1.1: Defining features of a P2P application according to different P2P definitions. Bullet means that the definition suggests that the feature is a necessary criterion for calling an application P2P.

> P2P are distributed systems without central control, where all the peers are equivalent in functionality.

This definition constraints the research in this thesis. This thesis concentrates on P2P systems without central control in which all the peers are equivalent in functionality.

1.4 Terminology and Background

This section defines the key terms that are required to understand the work presented in this thesis. This section defines the terms *self-organization*, *reference architecture*, *overlay network* and *topology adaptation* as used in the thesis. The section also describes the *types of P2P networks* and *Schelling's model.*

1.4.1 Self-Organization

Self-organization refers to the ability of a system to organize itself without any the influence of an external or an internal dedicated central entity. Camazine et al. define self-organization as:

> a process in which pattern at the global-level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern [33, pg. 8].

The pattern observed in a self-organizing system is an emergent property caused by the interaction between its components. Self-organizing systems are interesting because of their robustness and scalability [88]. They are robust against failure because there is no central point of failure. The lack of central control also means that self-organizing systems can typically scale easily to accommodate a large number of entities in the system. For these reasons, self-organization is a desirable property for any P2P overlay network.

1.4.2 Reference Architecture

A *software architecture* describes the structure of a software system. It describes and defines the software elements that comprise the system, "the externally visible properties of these elements, and the relationships among them" [65, pg. 21]. Software architectures are used to communicate the high-level design of a system to a diverse audience, which may include programmers, managers and customers. The software architecture design phase is used to make design decisions that affect a system's remaining development, deployment and maintenance life.

This thesis presents a *reference architecture* for the P2P domain. A reference architecture defines the software elements, their functional responsibilities and the allowed interactions between them for software systems belonging to a particular domain [55, 65, 47]. A reference architecture provides a template that can be used as a starting point for designing the software architecture for software systems within that domain. A reference architecture saves time and effort in the areas of requirement analysis and functional partitioning of the system [63]. Reference architectures are often used in mature domains like compilers and databases. For example, a compiler is composed of a scanner parser, semantic analyzer and a code generator subsystem [51].

1.4.3 Overlay Network

P2P systems organise the peers in a virtual network, called an *overlay network* (see figure 1.3). The overlay network utilizes the underlying physical network for communication. In an overlay network, a peer is connected to a small subset of available peers. This is so because if every peer on a P2P system is connected to all the other peers on the network, then the network would scale poorly. The overlay network's topology is the graph in which the peers are the vertices and the connections between them are the edges. The overlay network is self-organizing in nature. The P2P system takes care of tasks associated with new peers joining and old peer leaving the system.

The overlay network is used by applications to perform tasks like searching for available peers, resources and services and for routing of messages between peers, especially to peers behind firewalls. The services available on a P2P overlay network may utilize the overlay network's topology to perform their tasks. For example, a distributed game service might choose the connected peers as the opponents against whom a competitive game may be played. A P2P grid service may delegate tasks that can be executed in parallel to the connected peers.

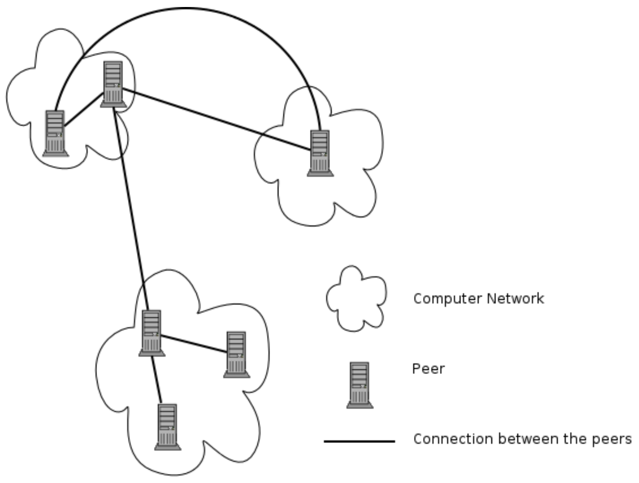


Fig. 1.3: A P2P overlay network. An overlay network is the graph whose vertices are the peers and the edges are the connections between them.

1.4.4 Topology Adaptation

The overlay network's topology affects the performance of tasks like routing and searching. For example, the effective bandwidth for routing messages between any two peers on the overlay network is the lowest hop bandwidth on the route between the peers. The messaging efficiency of routing can be improved by each peer having peers with similar bandwidths to itself as neighbors. The messaging cost of a search on the overlay network can be improved by arranging the topology to create a backbone network of peers that can maintain a directory of resources, services and peers available on the overlay network. Search requests can be processed by using the directory on the backbone network instead of flooding the network with search messages.

The best topology for services varies with their application. For example, in a distributed game service, peers hosting players with similar ability may be grouped together as neighbors to better challenge the players. In a P2P grid, it might be useful if peers with similar bandwidth are joined together as neighbor so that information can be efficiently exchanged between them.

Topology Adaptation involves maintaining an overlay network's topology as per application requirements. There is a messaging cost associated with maintaining a topology. The most suitable topology for an application depends upon the application requirements and the messaging costs that the application can accept.

1.4.5 Types of P2P Networks

On the basis of the overlay network architecture, P2P applications can be divided into three major categories (see figure 1.4): centralized, decentralized structured and decentralized unstructured [73]. Centralized architectures, also known as hybrid P2P, use a well known central server for performing organizational tasks to facilitate P2P communication. Napster (1999) [11] is a prominent example of a centralized architecture. According to the definition of P2P presented in section 1.3, a P2P system consists of equivalent peers. In existing literature (e.g., [73]) hybrid systems are often classified as a special case of P2P applications even though the central server has more capabilities and responsibilities when compared with other peers in the system because the rest of the peers are equivalent in functionality.

In decentralized architectures, also known as pure P2P, peers collaborate without a central server. In structured decentralized networks, the location of resources and peers in the network is controlled, typically using a distributed hash table (DHT) of resources available on the network. Chord [117] and Pastry [95] are prominent examples of systems based on this architecture. Unstructured decentralized P2P networks do not use a DHT to control the location of resources and peers on the overlay network. Gnutella [4] and KazaA [9] are popular P2P applications which use unstructured P2P networks. The position of peers in a structured decentralized network is controlled by the DHT algorithm that makes topology adaptation difficult. The presence of a central component makes it comparatively easy and uninteresting to perform topology adaptation in hybrid networks and so this thesis is concerned only with topology adaptation

$$\begin{tikzcd}
 & P2P \arrow[dl] \arrow[dr] & & \\
Centralized & & Decentralized \arrow[dl] \arrow[dr] & \\
 & Structured & & Unstructured
\end{tikzcd}$$

Fig. 1.4: A taxonomy of P2P systems based on overlay architecture.

in unstructured decentralized networks.

1.4.6 Schelling's Model

Decentralized systems abound in our world. The existence of life is best explained by Darwin's theory of evolution by natural selection, a process that lacks a central authority. In the eighteenth century Adam Smith, the father of modern economics, proposed that decentralized economies are a better and more efficient alternative to centralized control [91, pg. 7]. His ideas have since been validated by a general shift towards decentralized market-based economies by many nations. Even the human brain is a collaboration of relatively simple neurons that work together in a decentralized fashion.

The decentralized systems found in nature are typically self-organizing and self-managing in nature, i.e., they can organize and manage themselves even though no central coordinator exists. This makes them a candidate approach for designing solutions (e.g., topology adaptation algorithms) for P2P applications which also lack a central coordinator. A solution for P2P modelled on decentralized systems found in nature can also benefit from their ability to recover from perturbations which will make the solution ideal for dynamic environment of P2P systems.

An agent-based model is a tool that can be used to study the emergence of complex behavior from simple rules in decentralized systems. An agent-based model consists of large number of agents that change their properties and their environment by using their knowledge of the local neighborhood. In the 1966, economist Thomas Schelling [119, 100] proposed an agent-based model that can be used to explain the existence of segregated neighborhoods in urban areas.

This thesis presents a family of topology adaptation algorithms inspired by Schelling's model. In Schelling's model, people act using their awareness of the local neighborhood, which makes it especially attractive for P2P systems in which the peers lack a global picture of the network topology. In the model, grouping is maintained even when people join or leave the system (self-organizing), which makes the model ideal for the dynamic environments of P2P networks.

1.5 The Thesis

While the popularity of P2P applications is increasing, there is a lack of re-use while designing the software architecture for new P2P applications. At the same time, there is no common vocabulary to accurately describe and compare the structure of P2P middlewares and applications. This thesis addresses both these problems by proposing a reference architecture for the P2P domain. We first identify the concerns (use-cases) that are common across a wide variety of P2P applications, and then present the reference architecture, called the P2P reference Architecture that Supports Topology Adaptation (PASTA), which is based on these concerns. PASTA resolves the lack of re-use by providing a template that can be used as a starting point for developing the software architecture for new applications. PASTA identifies the components of a P2P application. The components and the policies used to implement the different components in an applications provide a vocabulary for the P2P domain. The description and comparison of P2P applications can be done on the basis of the policies used to implement the different components.

PASTA consists of modular components that are responsible for handling the core concerns that a P2P application needs to handle. PASTA is abstract and does not specify the policies that can be used to handle a core concern. PASTA can be used as a guide to develop a P2P application or middleware that use a plugin-based approach for implementing the different components. By changing the plugins for the different components, a P2P application could be made to easily interoperate with other P2P applications, and hence address the interoperability problem mentioned in section 1.2. PASTA is validated by describing the structure of four existing P2P applications and middleware. PASTA forms the secondary contribution of this thesis.

The primary contribution of the thesis is a novel approach for designing topology adaptation algorithms. The approach is based on an abstract algorithm inspired by Schelling's model called the PEer-to-peer Self-organizing TOpology (PESTO) that can be used to create a family of self-organizing topology adaptation algorithms for unstructured decentralized P2P overlay networks. This approach is captured as a key component of PASTA in the form of the topology adaptation layer, which is responsible for maintaining a topology that satisfies application requirements. To validate this approach, we present concrete algorithms that instantiate PESTO. Two of these are clustering algorithms, which can be used to bring together peers with similar characteristics as neighbors. Clustering algorithms are useful for a wide variety of P2P applications, for example in file-sharing applications where clusters can improve the performance of messaging on an overlay network by bringing together peers in the same geographical location as neighbors. The thesis also presents a hub algorithm based on PESTO that can be used to create a backbone network of powerful computers called hubs in an overlay network. The hub network can be used for tasks like maintaining a directory of resources on the overlay network that can be used to search for resources.

1.6 Thesis Roadmap

The remainder of the thesis is organized as follows:

Chapter 2 reviews the state of the art in the area of P2P topology adaptation and reference architecture.

Chapter 3 presents PESTO, an abstract algorithm that can be used to create a family of topology adaptation algorithms. The chapter also describes the simulator that has been used to evaluate the different topology adaptation algorithms.

Chapter 4 presents and evaluates two algorithms based on PESTO that can be used to cluster peers with similar properties.

Chapter 5 presents and evaluates a concrete realization of PESTO that can be used to create a backbone network of hubs.

Chapter 6 presents PASTA and validates it by using it to describe four existing P2P applications.

Chapter 7 concludes the thesis.

Chapter 2

State of the Art

The artist is the creator of beautiful things. To reveal art and conceal the artist is art's aim. The critic is he who can translate into another manner or a new material his impression of beautiful things [125].

This chapter presents the state of the art in the area of P2P topology adaptation and P2P reference architectures. The topology adaptation review takes the form of a review of the existing desirable topologies for applications and the existing algorithms that can be used for topology adaptation, including the only other approach in the literature for designing topology adaptation algorithms. The reference architecture review takes the form of a review of a contemporary reference architecture and JXTA which is an influential P2P middleware with particular attention to their support for topology adaptation. The chapter concludes with a summary of the state of the art study.

2.1 Topology

As defined in section 1.4, the topology of a P2P application is the graph in which the peers are the vertices and the edges are the connections between them. The topology is important in P2P applications because it affects the performance of tasks such as routing of messages on the network. This section reviews a range of interesting topologies for a P2P application. As discussed in section 1.4.6 systems found in nature are particularly interesting because of their decentralized characteristics.

In this section, we first present common metrics that are used to discuss the topologies presented later in this section. We then discuss simple topologies, such as the star, that may be used in P2P applications. This is followed by a discussion of the earliest topology that has been proposed to study complex topologies in nature. The next two topologies are found in social networks and in the world around us. Both have performance and fault-tolerance characteristics that makes them relevent for P2P applications. Super-peer topology is the next topology discussed in this section. It is interesting because it is used in two influential P2P applications, namely KaZaA [9] and Skype [13]. The super-peer topology has poor tolerance to the failure of super peers. Hub topology is a variation of the super-peer topology which rectifies this flaw. The last topology discussed in this section is the Clustered topology. It is also found in the world around us and is useful for a wide array of application scenarios.

The discussion on the topologies is structured as follows: We first describe the topologies and the metrics associated with them. This is followed by examples of possible and existing usage of these topologies in P2P networks.

2.1.1 Common Topology Metrics

This section presents common metrics used to evaluate the topologies presented in this section. These metrics provide valuable information about a P2P topology and can also be used to evaluate other P2P topologies.

Characteristic Path Length

The characteristic path length (CPL) measures the average number of hops in the shortest path connecting any two vertices [124]. Let V be the set of vertices on the graph, then:

$$CPL = \frac{2 \times \sum_{v_i, v_j \epsilon V, v_i \neq v_j} min\_routing\_distance(v_i, v_j)}{|V| \times (|V| - 1)} \tag{2.1}$$

where $min\_routing\_distance(v_i, v_j)$ returns the minimum routing distance between vertex v_i and v_j.

The CPL value for a given network is one of the determining factors on the performance of message routing in the network. A low CPL value would imply that messages can be routed quickly between peers on the network. A low CPL value also means that a Breadth First Search (BFS) message will reach a large number of peers on the topology.

Clustering Coefficient

An overlay network's Clustering Coefficient (CC) gives an indication of how well the vertices in the graph are connected to each other. The function $neighbors(v_i)$ returns the set containing the neighbors of a vertex v_i and function $connections(U)$ returns the number of connections between the vertices in the set U where $U \subseteq V$. The clustering coefficient of a vertex v_i is defined as:

$$CC(v_i) = \frac{2 \times connections(neighbors(v_i))}{|neighbors(v_i)|(|neighbors(v_i)| - 1)} \tag{2.2}$$

A low value of $CC(v_i)$ means that the failure of the vertex (v_i) could adversely affect the connectivity among $v_i's$ neighbors because its neighbors are not well connected with each other. The CC for the whole network gives an indication of the cliquishness of the topology [124]. It is defined as:

$$CC = \frac{\sum_{v_i \epsilon V} CC(v_i)}{|V|} \tag{2.3}$$

A high CC value means that the vertices in the topology are well connected to each other. In a topology with well-connected vertices there is a significant wastage of bandwidth because of redundant messages if BFS is used.

Degree Distribution

The degree of a vertex is the number of other vertices to which it is connected. The degree distribution is studied using a function $p(k)$, which gives the probability that a vertex on the topology will have a degree k. A plot of the degree distribution function with $p(k)$ on the vertical axis and k on the horizontal axis, can be used to visualize the presence of vertices with a given degree in the topology. The degree distribution can be used to identify the presence of peers with a high connectivity, whose failure may adversely affect the connectivity of the topology and the routing distance between the peers.

2.1.2 Simple Topologies

Figure 2.1 illustrates four simple topologies: a star, a ring, a lattice and a tree that may be used in a P2P application. In a star topology all the peers maintain a connection to a central peer. A P2P messaging application with a small number of peers is an example of an application that may use this topology. The application can use the central peer for tasks such as maintaining a directory of information about other peers, authenticating the peers using the information in the directory and routing messages between peers. The short routing distance of two hops between all the non-central peers and one hop between the central and all the non-central peers is an advantage of this topology. However, this topology is not interesting to this study because the central peer is a single point of failure that can cause the entire application to stop

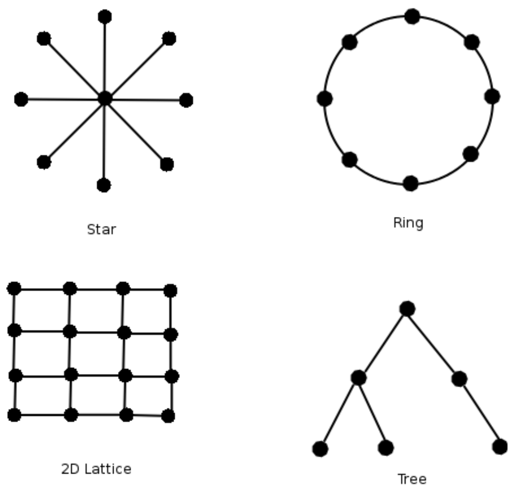


Fig. 2.1: Examples of simple topologies.

functioning and it would become a performance bottleneck once the number of peers on the network rises.

Ring, lattice and tree topologies may be used in a P2P application to support application-level multicasting. These topologies are typically accompanied by a routing algorithm to prevent redundant traffic. For example, in the lattice topology in [28] the four neighbors of a peer are called the north (N), south (S), east (E) and west (W) neighbors. A peer initiating a multicast request sends the request to all its four neighbors. A peer receiving a multicast request from its S neighbor only sends it to its N, E and W neighbors. A peer receiving a multicast request from its N neighbor only sends it to its S, E and W neighbors. A peer receiving a multicast request from its W neighbor only sends it to its E neighbor and a peer receiving a multicast request from its E neighbor only sends it to its W neighbor. The authors show in [28] that the routing rules prevent redundant traffic. The author of this thesis is not aware of decentralized approaches to create and maintain these topologies under a flux of peers and so these topologies are not discussed in detail in this review.

2.1.3 Random Topology

Random Topology refers to the topology proposed by Solomonoff and Rapport [112] and independently by Erdös and Reńyi [45] to study the complex topologies in the world around us. In a random topology n vertices are connected to each other with a probability p. This means that each edge can exist with an equal probability p.

Let λ be equal to $p(n-1)$. The CPL of the random topology is small and is equal to $log(n)/log(\lambda)$. The CC of the random topology is equal to p/n and tends to zero as n becomes large [pg. 21][77]. The degree distribution function of random topology is a Poisson distribution which means that [77]:

$$p(k) = \frac{e^{-\lambda}\lambda^{k}}{k!} \tag{2.4}$$

The author is not aware of an algorithm or an application from the P2P domain that tries to create and maintain a random topology. This topology is presented here primarily because of its historical significance.

2.1.4 Small World Topology

In a small world topology, all the pairs of vertices are connected to each other through a path consisting of a small number of other vertices. The connectivity between the vertices, through a small number of intermediate vertices, implies that the routing distance (measured as the number of vertices in the path minus one) for exchanging messages between the vertices on the topology will be small. The short routing distance makes small world a desirable topology for P2P networks. The value of CPL in small world topology is proportional to log of the number of vertices on the network [pg.

11][77].

A human social network is a prominent example of a network with a small world topology. In a human social network, people are connected to each other through a small number of mutual acquaintances. American economist Stanley Milgram did an experiment to demonstrate this. He asked participants all over America to send packages to a particular address through their acquaintances. The packages that reached the destination were routed there through five to six acquaintances on an average. The small world phenomenon is also called *six degrees of separation* because in Milgram's experiment on an average six acquaintances were used to send the letter to the destination. Another experiment used data from Internet Movie DataBase (imdb) [5] to show that the collaboration graph of actors is a small world with 3.65 being the average number of vertices that connect one vertex to another [124].

In 1998, Watt and Strogatz presented a model (WS model) to study small world topology [124]. In their model, a two-dimensional lattice topology is rearranged to create a small-world topology. In the lattice topology N vertices are arranged around a circle and each vertex is connected to its immediate and next-to immediate neighbors. Rearranging the lattice involves going through the vertices on the circle and rewiring the edges with a probability p. The rewiring involves removing the existing edge and connecting the vertex v to another vertex chosen at random with the constraint that an edge connecting the randomly chosen vertex and v does not already exist.

When p is 0, the generated topology is the two-dimensional lattice. The CC is high and it is approximately 3/4 for the two dimensional lattice for a high, average degree of the vertices (k). The CPL is large and is approximately equal to $N/4k$ for large N. When p is 1, a random topology is generated. The CPL of the random topology is small and is proportional to $log(N)$, however the CC is also very low and is approximately equal to $2 \times k/N$. When p is between 0 and 0.1, the generated topologies display small-world behavior with a low CPL and a high CC when compared to the CC of the random topology. The degree distribution of the small world topologies in the WS model is centered around k and the topology lacks vertices with a high degree, whose loss could adversely affect the connectivity of the topology and the routing distance between the vertices [24]. This makes this topology robust against targeted attacks.

A variation of the WS model has been proposed independently by Monasson [74] and by Newman and Watts [78] in which instead of rewiring edges, shortcut edges joining randomly chosen vertex pairs are added to create the small-world effect. A similar idea has been used by Zhang [130] to create small-world effect in Freenet. Freenet is a P2P file storage system that provides anonymity to its users. In Freenet a key is used to obtain a file stored on the network. Peers in Freenet maintain a cache for serving files and routing file requests. Research done by Zhang suggests that ability of Freenet to successfully locate a file under load can be improved by using a cache replacement algorithm that creates a small world effect. The work is based on the observation that a small world topology can be constructed if each peer in the network knows its own neighbors and a small number of randomly chosen distant peers. The later connections are shortcuts that help in forwarding a file request to distant peers and help in improving the chances of locating a file.

2.1.5 Power Law Topology

A power law topology is another popular topology that is found in the world around us. In a power law topology some vertices have a very high degree while the rest of the peers have a low degree. The vertices with a high degree act as hubs. These high degree vertices provide short cut paths between vertices on the network. Examples of power law topology in the world around us include the graph of the world wide web in which the vertices are documents and the edges are the links pointing from one document to another and the graph formed by citation patterns in which the vertices are the papers and the edges are the links to the articles cited in a paper.

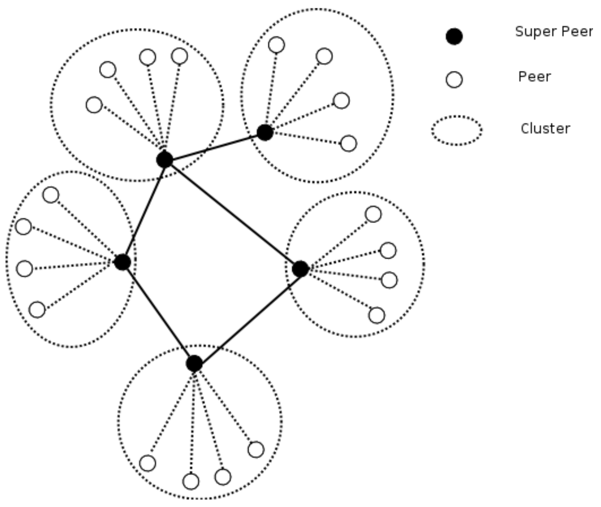


Fig. 2.2: Examples of a super peer topology.

In a power law network the degree distribution function $p(k)$ is related to k by a power law relation. This implies that:

$$p(k) = a * k^{\gamma} \tag{2.5}$$

where a, and γ are constants. On taking log of both sides the above equation becomes:

$$log(p(k)) = log(a) + \gamma * log(k) \tag{2.6}$$

This means that when plotted on a log-log graph the power law relation will appear as a straight line where the slope of the line is γ. The value of γ is 2.45 [24] for the graph of the world wide web and 3 for the graph formed by citation patterns [90]. The CPL of a power law network is proportional to $log(N)/log(log(N))$ where N is the number of vertices in the network [29]. For topologies with the same number of vertices, the CPL value of a power law topology is lower than the CPL value of a small world topology. The low CPL value is a useful characteristic for P2P applications because messages can be routed at a faster rate between peers. While both small world and power law topologies have low CPL, they differ in the degree distribution function. In the former topology the degree distribution function is centered around k (the average degree of the peers) while in the later it follows a power law. Small world topology lacks vertices with a very high degree when compared to the other vertices in the topology.

Simulations have been done to compare the performance of power law and small world topologies under failure of vertices [23]. In the simulations, vertices were removed and the change in the diameter of the graph was observed. The study suggests that power law topology is robust against random failure when compared to small world topology. Small world topology performs well against directed removal, however the power law topology is severely affected by a directed removal of highly connected vertices. Similar results were observed when the performance of power law and random topologies under failure of vertices was compared. This low resistance to targetted attacks makes power law especially suitable for applications that are not under fear of targetted attacks. For example, this topology might be suitable for applications running in a secure intranet.

A popular model [24] used to explain the existence of power law topology suggests that power law topology is created because of a higher preference of new vertices to attach themselves to a vertex with a high connectivity. Power law networks are referred as scale-free networks in this model because the value of γ is independent of time and the number of vertices in the network. In this model, the new peers have global knowledge of connectivity of all the vertices in the topology. In a P2P network peers lack global knowledge. However, work done in [48] suggests that a power law topology can be created even when the new vertices lack a global knowledge and use their local awareness to search for a highly connected peer to which to attach.

Research done by Saroiu [98] suggests that there is diversity in the bandwidth of the peers that participate in a P2P network. The next section presents three algorithms from the literature that can be used to create a power law topology in a P2P application to utilize this diversity among the peers by making the peers on high bandwidth connections high degree vertices that are responsible for a major share of the routing.

2.1.6 Super Peer and Hub Topologies

KaZaA [9] and Skype [13] are successful P2P applications that use a super peer topology [26]. A super peer topology can be used to improve the performance of a pure unstructured P2P application by introducing a degree of structure into the network. In a super peer topology, the vertices are divided into two categories: super peers and ordinary peers. Each ordinary peer is connected to exactly one super peer. The super peers are connected to each other to form an overlay backbone network of super peers. A super peer and the ordinary peers connected to it form a cluster. The size of the cluster is the number of peers in the cluster including the super peer [127]. Figure 2.2 presents an example of a super peer topology.

The CPL of the super peer topology depends on the topology of the overlay backbone network of super peers. In the worst case, when the overlay backbone network of super peers is arranged in a line, the CPL will be of the order of the square of number of super peers (nsp) in the topology. The degree distribution has a spike at 1 and at nsp with the spike at 1 being much higher than the spike at nsp. The spike at nsp is caused by the super peers and at 1 by the ordinary peers.

The existing pure unstructured P2P applications such as Gnutella propogate search messages using flooding. A super peer network can be used to maintain a directory of resources on the overlay network. The ordinary peers can send the search request to their super peer which can collaborate with other super peers on the backbone to return a response. This reduces the need for expensive broadcasts on the overlay network. The super peer network can also be used to route messages to other peers. The ordinary peers can send a message to its super peer which can collaborate with other peers on the super peer network to route the message to its destination. As mentioned at the beginning of this section, both KaZaA and Skype use a super peer topology. However, both are proprietary products and the details of the algorithms used in them to create the super peer topology are unfortunately not available.

In a super peer system, the failure of a super peer can have a catastrophic consequences causing a complete communication failure for the cluster of nodes attached to it. To resolve the impediment of catastrophic failure, some P2P systems (e.g., JXTA [121]) use a variation of the super-peer topology in which the ordinary peers are connected with each other and can be connected to more than one super peer (called a *hub* in this thesis). The hubs are connected to each other to form a network of hubs in a similar fashion to the backbone in super peer networks. In this topology the failure of a hub is not a catastrophic failure, because the connections between the peers can be used for communication in case of a hub failure. A hub topology could be considered as a variation of power law topology in which the vertices with a high degree are connected to each other.

2.1.7 Clustered Topology

Clustered topology is another topology that is found in the world around us. Examples of such clustered topologies include residential segregation in urban areas [100] and clustering of slime molds in adverse conditions to form one big organism [62, pg. 13].

The number of clusters NC in the graph G is a metric that can be used to measure clustering. The topology is the graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Let $P = \{p_1, p_2, \dots p_n\}$ be the set that enumerates the types of vertices in the graph G. Let $SG_{p_i} = (V(p_i), E(p_i))$ be a subgraph of graph G that contains all the vertices from the original graph with property p_i and the connections between them, so that:

$$V(p_i) = \{v \epsilon V : v.property = p_i\}$$

$$E(p_i) = \{\{v, w\} \epsilon E : v, w \epsilon V(p_i)\}$$

A connected component of a graph is a sub-graph in which all the vertices are connected to each other. We use the term *cluster* to refer to a connected component of the graph SG_{p_i}. Let $CC_{p_i} = \{CC_1, CC_2, \ldots CC_n\}$ be the set of connected components of the graph SG_{p_i}. The number of clusters (NC) in the graph G is :

$$NC = \sum_{i=1}^{n} |CC_{p_i}|$$

Clustering algorithms are used to rearrange the overlay network so that peers with similar characteristics are brought together as neighbors. Clustering algorithms change the overlay network's topology so that the number of clusters (NC) is minimized.

The type and effect of clustering depends on application concerns but the topology is useful in a variety of settings. For example, the performance of messaging on an overlay network can be improved by clustering peers that are close to each other on the underlying physical network [46, 133]. In a file sharing application, peers sharing similar files may be clustered together, which can be used to improve search performance because search requests can be routed to an appropriate cluster and then a deep search can be performed within the cluster [46, 69, 79]. The performance of messaging on a P2P overlay network can be improved by clustering peers in the same geographical location. Clusters with low intra-cluster network latencies can be used to provide coarse-grained parallelism in which parts of a parallel application are distributed across hosts in the cluster [22]. In a distributed game, clustering can be used to bring together players with similar levels of competency so that their gaming experience can be improved.

2.1.8 Summary

The optimal topology for any given application depends on the application's requirements. The first three topologies presented in this section have a small CPL which makes them useful for applications that desire a small distance between the peers. The first two topologies have fault-tolerance characteristics that are useful for applications that expect targeted attack on peers, while power law topology is useful for applications that expect only random failure or attack. The power law, super peer and hub topologies are good choices for applications that desire to optimize the search operation on the overlay network by creating a directory of information about other peers on the network. The hub topology is a better choice when compared to super peer topology for an application that desires high fault-tolerance.

The first three and the last topology presented in this section are inspired by nature. These topologies offer desirable properties such as small average distance between the peers and fault-tolerance to applications. The topologies from nature are interesting because typically they are created and maintained in a decentralized, self-organizing and self-maintaining manner. The topology review presented here is not exhaustive and it would be interesting to discover other topologies in nature that have interesting and desirable traits for applications. Another challenge is to modify these topologies so that they are more useful or provide other uses for applications. For example, an interesting topology could be designed by merging the power law and clustered topology, in which the clusters of peers have peers with high degrees that can maintain directory of resources available in their clusters.

The peers in the first three topologies inspired from nature presented in this section have a small average distance (CPL) between them. The average distance is typically of the order of $log(N)$ and is similar to distance between peers on structured overlay networks. However there is a lack of routing algorithms that can effectively utilize the short average distance between the peers on the topologies presented in this section to achieve routing distance of the order of $log(N)$.

A challenge after identifying a topology is to design algorithms that can create and maintain the topology in a self-organizing and self-maintaining manner. The next section reviews existing topology adaptation algorithms.

2.2 Topology Adaptation Algorithms

For the purpose of review, the existing algorithms for topology adaptation have been divided into two major groups: centralized and decentralized. The decentralized algorithms do not require a central server unlike the centralized algorithms that require a well-known central server to create and maintain the topology. This section presents and evaluates existing topology adaptation algorithms that are relevant to this thesis.

While our main interest is in decentralized algorithms, the first two algorithms reviewed in this section are centralized algorithms. The rest of the topology adaptation algorithms discussed in this section are decentralized topology adaptation algorithms. The first algorithm is significant because it is one of the earliest that recognizes topology adaptation as a means for improving the performance of P2P applications. The second algorithm is interesting because it presents a mathematical analysis that guarantees the resulting topology to be connected and have a fixed diameter. It is difficult to provide such guarantees for decentralized algorithms.

The next three algorithms discussed in this section utilize the diversity in the capability (e.g., bandwidth, processing power) of the peers on the overlay network by creating and maintaining a topology, similar to the power-law topology, in which the powerful peers have a higher degree. The last algorithm among these three is particularly interesting because it changes the type of the topology from power law to a random mesh and vice versa with a change in the load on the peers.

For the purpose of review, the existing decentralized algorithms for clustering peers can be divided into two major categories based on the criteria used for clustering: distance-based clustering and content-based clustering. We first review distance-based clustering algorithms. This is followed by the review of a content-based clustering algorithm, which is one of the earliest in which a peer uses its local awareness of the network to establish connections to peers with a similar property. Finally, the section examines a contemporary (published in 2006) approach for designing decentralized topology adaptation algorithms.

The review of each algorithm is divided into two parts, the description and the discussion. The discussion is structured as follows: We first analyze the topology created by the algorithm and then the algorithm itself. This is followed by a discussion of the analysis techniques used to evaluate the algorithm.

2.2.1 Improving Breadth First Search (BFS)

In 2004, Silvey et al. proposed two topology adaptation techniques [105], that can be used to optimize the BFS used in Gnutella. These techniques have been included in this review because to our knowledge [105] constitute the earliest work in the literature that uses the term *topology adaptation* and recognizes it as a technique to improve the performance of P2P applications.

In [105] the authors argue that the search results obtained by BFS in pure unstructured networks like Gnutella are exhaustive because they contain response from multiple peers who may interpret the search request in different ways. The authors aim to optimize the topology of pure unstructured P2P networks for flooding-based BFS search by creating a topology with a small diameter and low message redundancy in flooding. The small diameter ensures that a message can reach all the peers in a short time. A redundant message is one that is received by a peer from more than one connection. A lower message redundancy will help in reducing the overall bandwidth consumption in the network.

For reducing message redundancy, the authors propose the technique of removing redundant neighbors. An algorithm for assigning a score to all the neighbors is presented in which a neighbor is assigned a score of $1/M$ for each message, where M is the number of neighbors that send a given message. The peers remove neighbors that have a low score. The authors assume that the topology adaptation layer has a global knowledge of the overlay network topology. To prevent disconnection, the authors suggest that the redundant links not be removed once the topology is disconnected and the redundant link whose removal caused the topology to be disconnected be reintroduced. For reducing the diameter of the overlay network, the authors suggest adding shortcut links between peer pairs whose minimum path length is equal to the diameter of the graph. The authors do not suggest how the peer pair whose minimum path length is equal to the diameter of the graph should be selected.

Simulations have been done in which the technique for reducing the message redundancy and the diameter of the network are applied to an overlay network in which there is no flux of peers. The simulation results show that on an average there is a 7% reduction in the average number of messages passed for a query and a 15% reduction in the diameter of the network.

Discussion

The authors suggest creating a topology with a small diameter and fewer redundant links to optimize the performance of BFS. The suggestion to create a small diameter is good and is similar to the approach presented in [130] (discussed in 2.1.4) to create a small world effect by creating shortcut links. The suggestion to remove redundant links will however reduce the fault-tolerance of the topology. Removing the redundant links will lower the CC of the topology and will increase the risk of disconnecting the topology under churn of peers.

An interesting point of this algorithm is that it identifies that peers can use local information (e.g., the neighbors with a low score) to take actions that can change the overlay network's topology. The authors present approaches such as removing neighbors with low score that can be used to remove redundant links and reduce the diameter of an overlay network. However, the authors do not present concrete ideas on how the approaches for managing the topology could be executed without relying on a central server with a global knowledge of the topology.

Another weak point of this algorithm is that it does not consider overlay networks with a flux of peers, a scenario that is common in real life overlay networks. To conclude, this algorithm was an interesting early attempt that recognized topology adaptation as a general approach for improving the performance of P2P applications, however it does not represent any viable solution for decentralized topology adaptation.

2.2.2 Building a Low-Diameter Connected Topology

This algorithm has been proposed by Pandurangan et al. in 2001 [85]. It utilizes a central server, which has a limited knowledge of the peers on the overlay network, to create and maintain a connected topology with a constant degree and a logarithmic diameter. This algorithm is interesting because a mathematical model that predicts the diameter of the generated topology has also been presented by the authors. Existing decentralized topology adaptation algorithms typically lack a mathematical analysis that can be used to predict the properties of the generated topology, and so this approach is potentially interesting for applications that require a guarantee on the quality of the topology generated.

The algorithm requires a cache running on a well-known server called the *host server*, which can maintain information about a constant number (say K) of peers on the overlay network. When a new peer P joins the overlay network, it is connected to D nodes chosen at random by the host server from the cache. When the number of neighbors of a peer P in the cache reaches C, then P is replaced by another peer from the overlay network. The peer leaving the cache creates a connection to the peer that replaces it if they are not already connected. The peer entering the cache is the preferred peer of the peer leaving the cache. If a peer loses its preferred peer, then it chooses a random peer from the cache to be its preferred peer.

When a peer has to leave the cache, its neighbors are searched for a peer with degree less than C that can replace it. If the search is unsuccessful then a similar search is performed on the neighbors of the peers that were in the cache at that position till a peer with degree less than C is found to replace the leaving peer. This search is performed by the host server. When a peer loses a neighbor, it chooses a random node in the cache with the probability D/d where d is the degree of the peer before its neighbor left.

The authors have presented a stochastic analysis to show that as a result of this protocol, the resulting overlay network will be connected with diameter $log(N)$ where N is the number of peers on the overlay network.

Discussion

This algorithm can be used to create and maintain an overlay network with diameter $log(N)$ where N is the number of peers on the overlay network. The diameter of the graph is the largest possible distance between vertex pairs and so the CPL of the graph will be also equal to or less than $log(N)$ which is the same as in a small-world topology. A small network helps in routing by reducing the number of hops a message has to take between any two peers on the overlay network. As mentioned in section 2.2.1, a small diameter can also improve the performance of a search that uses BFS because the message reaches a large number of peers on the overlay network. The algorithm ensures that the degree of a peer is maintained between D and C. This is helpful for applications that do not desire a power law distribution in peer degree, which makes the overlay network prone to targeted attacks.

This topology adaptation algorithm is not decentralized and requires a central server. The server is used by new peers to join the overlay network and for finding the replacement peer for a peer leaving the cache. Finding a replacement peer for the cache would require the server to maintain the information about all the peers that where ever a part of the cache. The central server could become a bottleneck if there is a high flux of peers or if the peers are from physically distant locations.

It is difficult to provide guarantees on the characteristics of the topology created by decentralized algorithms because of the large number of variables (e.g., the state of each peer executing the topology adaptation algorithm, initial topology etc.) involved in the system. This algorithm is specifically useful for applications that desire a guarantee on the properties of the overlay network.

2.2.3 Topology Adaptation For Load Balancing

The aim of this distributed topology adaptation algorithm first presented in 2002 [70] is to adapt the topology to control the flow of the traffic on the network so that peers do not get overloaded with messages. The algorithm is decentralized and self-organizing in nature. This algorithm adapts the topology so that more requests are directed towards peers with a high capacity. Capacity (C) is defined as the number of messages a peer can process in a given time interval T.

In this algorithm the peers maintain information about the number of incoming messages from each of their neighbors. The peers periodically check if they are overloaded, i.e., whether they are receiving more messages than they can process in a given time interval. If a peer is overloaded, then it picks one of its neighbors with a high incoming message rate and redirects it to a peer with a high spare capacity. The overloaded peer finds a peer with a high capacity from a list of peers on the overlay network and their spare capacity that it maintains. The list is populated with data obtained by periodic exchange of information with neighbors. The simulations suggest that the degree distribution in the evolved topology matches the capacity distributions, i.e., higher capacity peers have significantly higher degrees.

Discussion

This algorithm adjusts the topology so that the peers with a higher capacity have a higher degree and consequentially higher responsibility. While the peers with a higher degree help in load balancing, they reduce the fault-tolerance of the topology because their failure would adversely affect the diameter and connectivity. The topology generated by this algorithm will be a power law topology if the capacity distribution follows a power law. A similar attempt to utilize the power of peers with a higher capacity is also done in the topology adaptation algorithms presented in sections 2.2.4 and 2.2.5 discussed below.

In this algorithm an overloaded peer searches for a peer to direct the search request, from a list of peers and their spare capacities that it maintains by periodic exchange of information between peers. The information in the list may be temporally inaccurate because of the flux of peers on the network and the change in the load on the peers in the list. It might happen that the peer to which a message has been redirected has left the network. The authors do not propose how the information about the peers that have left the overlay network would be removed from the list.

In this algorithm a peer uses the number of messages that it receives from another peer (say s) in a given time unit to decide whether the peer s is a constrain on its resources. However, the number of messages that a peer is sending in a given time unit does not give an accurate indication of the load it causes on the recipient peer. The type of message also affects the load on the peer. For example, the messages to obtain information about the peer can be processed with less load than the messages to search for a resource, which might require going through all the records that the peer has.

The simulations do not evaluate the scenario where the capacity C of a peer in the topology changes with time. The capacity C of a peer might change with time because with time the user of the peer might like to change the amount of resources available to the application. Simulations to study the effect of churn, especially the departure of higher degree peers, on the topology have also not been done. The algorithm seems to be capable of handling the first scenario, while the effect of the second scenario needs to be studied.

While the algorithm suffers from minor drawbacks such as temporally inaccurate cached entries, it does successfully utilize a peer's limited awareness of the overlay network to create an effective topology for load balancing in which the peers with a higher capacity have a higher degree. The contribution of this algorithm would have risen tremendously if it had studied the effect of churn on the topology, especially the departure of peers with a higher degree.

2.2.4 Load Balancing in Biased Random Walk Search

GIA [34] was presented in 2003, and two contributors of the previous algorithm are part of the team which presented it. GIA stands for GIAnduia, which is another name for the hazlenut spread, Nutella after which Gnutella is named. The authors of GIA propose that the peers maintain a directory of resources available on their neighbors. The directory is utilized while performing a search operation using random walks that are biased towards peers with a high degree that might be in a better position to answer a query because of replication. The authors present a topology adaptation algorithm to create a topology in which the peers with a high capacity are the peers with a high degree, which ensures that the load is divided among peers on the basis of their ability to handle it. The capacity of a peer is the estimated number of messages that the peer can handle per second and is directly related to the bandwidth of the peer.

In GIA each peer maintains a *host cache* that is used by the topology adaptation algorithm. The host cache is a list of peers and associated information such as IP address, port number and capacity. The host cache is populated through a well-known server or by exchanging host information with neighbors.

In the topology adaptation algorithm, each peer periodically calculates its satisfaction level (S) which is a value between 0 and 1. The satisfaction level indicates the satisfaction of a peer with its neighbors. When S is 1, the peer is fully satisfied and the algorithm stops. When S is not 1 the peer is dissatisfied. The available capacity of a peer can be defined as its capacity divided by its neighbors.

The satisfaction level of a peer is 0 if the number of its neighbors is less than the minimum possible number of neighbors that it can have and 1 if it is equal to the maximum possible number of neighbors. The minimum and maximum possible number of neighbors that a peer can have is a predefined constant. If the number of neighbors of a peer is between the minimum and maximum possible number of neighbors, then the satisfaction level is the ratio of the sum of the available capacity of all its neighbors divided by its own capacity. If the ratio is greater than 1 then satisfaction level is 1 and the peer is satisfied. The criterion for satisfaction level means that a peer with a high capacity will be satisfied when it is connected to a large number of peers with a lower capacity.

A dissatisfied peer searches the host cache to find a peer with a capacity greater than its own capacity and tries to connect to it as a neighbor. If a peer with capacity greater than its own capacity is not found then the dissatisfied peer chooses a random peer from the host cache and tries to connect to it as a neighbor. If the peer with which the dissatisfied peer is trying to establish a connection can not accommodate a new neighbor, then it drops an existing neighbor if the new peer improves their satisfaction level.

The desire of the peers to be connected to a peer with a higher capacity leads to a topology in which the peers with a high capacity have a high degree. An implementation of GIA has been developed using C++ and a simulation has been done on

PlanetLab testbed [12] using 83 peers. The simulation results demonstrate that the peers with a higher capacity have a high degree.

Discussion

This algorithm is similar to the topology proposed in section 2.2.3 as it also attempts to generate a topology in which the peers with a high capacity are the peers with a high degree. The generated topology in this algorithm shares its advantages and disadvantages with the topology in the previous section. The key advantage of the generated topology is load balancing, and the key disadvantage is reduced resistance to targeted attacks because of the presence of peers with a high degree.

In the previous algorithm (section 2.2.3) the topology is adapted with a change in a dynamic criterion (the incoming messaging rate of a peer), while in this algorithm the topology is adapted to account for the capacity of the peers, which is a static value. This algorithm uses the static value for topology adaptation because the search technique used in GIA relies on a directory of resources maintained by a peer which will not be exhaustive for a peer with a lower degree. This choice of criteria for topology adaptation however does not provide dynamic load balancing depending on the load on the peers and could lead to overloading on peers with a high degree.

In GIA each peer maintains a directory of resources available on its neighbors that is used to respond to a search request. The authors do not suggest how this directory should be updated when peers leave the overlay network. The topology adaptation algorithm searches for a peer from a cache that it maintains. The cache might have temporally inaccurate data because of peers leaving the system and the authors do not suggest how to remove these peers that have left the network from the cache of other peers.

The topology adaptation algorithms are evaluated using a very small overlay network of 83 peers. The authors study the generated topology through the degree of a very small number of nodes. The evaluation should have included better metrics to evaluate the generated topology. The performance of the algorithm under a flux of peers has not been evaluated.

This algorithm is similar in essence to the earlier algorithm as both of them utilize a peer's limited awareness of the network to create a topology in which the powerful peers are the peers with the high degree. The techniques discussed in section 2.2.1 show how topology adaptation can be used to optimize BFS. This algorithm adapts the topology for biased random walks search and thus reinforces the applicability of topology adaptation as a general technique for enhancing unstructured P2P applications.

2.2.5 Changing Topology Type for Load Balancing

DANTE is another algorithm presented in 2006 that extends the idea of using a peer's local awareness to create and maintain a desired topology [93]. This algorithm changes the type of the topology based on the load on the peers. The algorithm creates a power-law topology when there is less congestion and a random graph when there is congestion on the network. This is based on the assumption that on a congested network, hubs might not be able to bear the load and on Guimera's suggestion in [49] that a random topology is a better choice in a congested scenario.

In this algorithm a peer is congested if the number of queries processed by it in the last minute is greater than its processing rate, which is a known constant. In this algorithm the connections that a peer initiates are called native connections and the connections that it receives are called foreign connections. The topology adaptation algorithm is executed periodically by the peers and is used to change the native connections. The peers send random walk messages with TTL to get a list of traversed peers that become the candidate peers for being used as native connections. A selection function is applied on the candidate peers to select n peers for the new native connections. The present native connections are terminated and connections with these n

peers is established.

The selection function selects a candidate peer to be the new native connection with a probability that is the peer's attachment kernel normalized by the sum of the attachment kernel of all the candidate peers. The attachment kernel of a candidate peer is 1 if it is congested and is the degree of the peer raised to the power of 2 if it is not congested. This means that an uncongested peer has a higher chance of being selected as a neighbor than a congested peer and among congested peers, the peer with a higher degree has a higher chance of being selected as a neighbor. The authors present a visualization of the overlay network to demonstrate that the topology changes from a power law topology to a random graph under load.

Discussion

An interesting aspect of this algorithm is that it creates a random graph topology under load and a power-law like topology when not under load. Another interesting idea is the selection function which the peers periodically apply on a list of peers in the topology that they are aware of, to decide their neighbors. This is unlike the idea of updating a few neighbors that has been used in the previous two algorithms. The selection function selects an uncongested peer to be its neighbor with a probability that is proportional to its degree. This is similar to the preferential selection criterion suggested by Barbasi in [24] (discussed in section 2.1.5) to create a power-law topology.

In the algorithms a peer decides its state on the basis of the peers returned by a selection function on the results of a random walk. The selection function itself uses chance to decide the peers that should be used as neighbors. Because of the element of chance, the active connections that are selected in one iteration of the algorithm might not be selected in the next, leading to an unnecessary churn in connections, which is expensive. Further, this implies that the overlay network's topology is never going to stabilize and a part of the bandwidth will be continuously used to maintain it.

The algorithm creates an interesting distinction between the connections that a peer has. A peer can not refuse or remove foreign connections and this might lead to the undesired effect of a peer being overloaded. The authors do not present an explanation as to why a peer can not refuse or remove foreign connections.

The stochastic nature of the selection function is the main drawback of this algorithm. This algorithm is unique because it uses a peer's local awareness to change the type of topology of the overlay network.

2.2.6 Topology Adaptation for Distance-Based Clustering

Algorithms for clustering peers that are close to each other on the underlying network have been presented in [89, 118] and [22]. These three algorithms use a similar approach which involves using selected landmark peers called originators. The peers measure their underlying network distance to the originators and connect themselves to the originator that is closest to them. This approach results in clusters that are based around the originators.

To allow the ordinary peers to measure their distance from the originators, [89] proposes that originators broadcast messages with a weight parameter that is reduced at each peer. The peers join the cluster headed by the originator whose messages had the highest weight (implies closest distance) when they received it. The clustering provided by these algorithms depend on the choice of originators. The algorithms require separate approaches for handling the arrival and departure of peers. These algorithms are not discussed in detail in this review because of their dependency on the choice of originators which are selected statically.

2.2.7 Topology Adaptation for Content-Based Clustering

Algorithms presented in [122, 69, 79, 115, 27] are designed to create clusters of peers that have a similar property. For file-sharing applications this property is typically the type of content that a peer is sharing. In all of these algorithms the peers use their local awareness of the network to establish connection to peers with similar property. These algorithms are typically accompanied by a routing algorithm that directs search requests to the most suitable cluster for handling the request. Unlike the rest of the algorithms that are executed periodically, the algorithm in [27] is executed only when a peer joins the overlay network. This review concentrates on the earliest algorithm in this category proposed by Hang et al. in 2002 [79].

In this algorithm, every peer P maintains a host cache that contains information about other peers on the network. The host cache is populated by exchanging information with other peers on the network. Each peer P periodically broadcasts a message. The peers that receive this message exchange their information with the peer P.

To perform clustering, each peer P periodically executes the topology adaptation algorithm in which it computes the similarity of the content that it is sharing with the content that the peers in the host cache are sharing. The similarity is computed using a function that depends on the application. The peer P selects as its neighbors the peers whose content is most similar to the content that it is sharing.

A routing algorithm has been proposed to utilize the clustering that is created by the topology adaptation algorithm. The peer that receives a request computes the relevance of the query to the content that it is sharing. If the content is not relevant, then it is forwarded to randomly chosen peers. If the content is relevant, then it means that the request has reached its target cluster and it is forwarded to all the neighbors.

The simulation are done on overlay networks with up to twenty thousand peers. The simulations present the ratio of the the percentage of desired results retrieved (R) to the percentage of peers visited (V) for a query. The aim of this algorithm is to increase R and decrease V. The simulations show that the routing algorithm coupled with the topology adaptation algorithm outperforms search using BFS on an overlay network.

Discussion

This algorithm creates a clustered topology of the type discussed in detail in section 2.1.7. This algorithm is similar to the previous four algorithms as both in this algorithm and in the algorithms described earlier the peers utilize their local awareness to create the desired topology. The simulations for this algorithm concentrate on demonstrating the utility of the topology adaptation algorithm and do not explore the properties of the created topology. The simulations also do not study the effect of churn on the network. For example, the routing algorithm utilizes the edges (inter-cluster edges) that connect peers from different clusters to route the search query to the appropriate cluster. The authors do not explore the ratio of the inter-cluster edges to the total number of edges and how they affect the performance of the routing.

This topology adaptation algorithm is similar to the algorithm that has been proposed by Voulgaris et al. in 2004 for semantic clustering [122]. The only difference is that the host cache in Voulgaris's algorithm is populated by gossip between randomly chosen peers on the overlay network to share information about other peers on the overlay network. The author feels that the presence of two algorithms that share the same structure but differ in the way certain operations such as populating the host cache make the case for an abstract topology adaptation algorithm that can be used to describe and compare the existing topology adaptation algorithms.

2.2.8 T-MAN

First presented in 2004 as a technical report by Jelasity et al. from the University of Bologna, T-Man [59] presents a general approach for developing topology adaptation algorithms. An extended version of the research on T-Man was later published in [60]. This review concentrates on [60] as it contains the latest research on T-Man. T-Man suggests developing rules that the peers can execute using their local awareness to create and maintain a desired topology. Like GIA, the peers in T-Man maintain a cache of other peers on the network called the "random view" that is populated by "gossip" between the peers about other peers on the network. Each peer also maintains a list (called the "view") of peers with whom it maintains active connections.

In the algorithm, each peer P uses a ranking function to order a list of peers according to its ($P's$) preference of having them as a neighbor. The ranking function of a peer P takes the list of peers as input and return a sorted list of peers arranged in a descending order of preference. The challenge in the T-Man algorithm is to design an appropriate ranking function that can select an individual peer's neighbors using only limited knowledge of the overlay network, so that a desired topology can be created.

In this approach, each peer P executes the topology adaptation algorithm once at a randomly chosen time in every consecutive time interval of duration T. In the algorithm, the peer selects the best neighbor BN, obtained by applying the ranking function on the view. The peer merges the information about itself with the information about other peers in the "view" and the "random view" to create a merged list. The peer then exchanges its merged list with the merged list of the peer BN. Both P and BN merge the information in the merged lists and then apply the ranking function on it to select the top n peers that will form the view. Apart from executing the topology adaptation algorithms, the peers periodically exchange their random views and use the exchanged information to enlarge their local random views.

The authors present ranking functions for creating topologies such as line, ring, torus and binary tree. The ranking functions presented by the authors rank the peers on the basis of their distance from the peer which is executing the algorithm. For some of the topologies like ring and torus the ranking function required a global knowledge of the overlay network. For the binary tree topology the ranking function uses an identifier assigned to the peers on the basis of their position in the tree.

Simulations

The authors use simulations to demonstrate that the algorithms developed using T-Man converge. Simulations have been done on networks with size $2^{14}, 2^{17}$ and 2^{20}. The simulations suggest that within 30 simulator iterations, the algorithms converge which means that no new edges from the desired topology are introduced. There is a logarithmic relation between network size and convergence speed. It may happen that even after the algorithms converge, some peers are not placed at their appropriate location in the generated topology. The authors claim that the simulations suggest that this is not common and even if it happens then the misplaced peers eventually reach their appropriate position.

The views in this algorithm cache information about peers on the network. This information may become obsolete because of peers that leave the overlay network. To remove entries for peers that have left the network, the authors propose that the peers also maintain age information about the peers in their view and purge the H oldest entries while merging the merged lists described above. The authors present simulation results that suggest that 1 is the best choice for H.

Discussion

A host cache that contains a peer's view or knowledge of the topology has been used in earlier algorithms (e.g., the algorithms discussed in sections 2.2.4, 2.2.5 and 2.2.7). The idea of using a selection function (referred as ranking function) on the host cache to create a desired topology has also been used before (e.g., in the algorithms discussed in sections 2.2.5 and 2.2.7). The authors have identified the general applicability of this idea in T-Man.

The authors suggest that gossip between the peers should be used to populate this list. Since the authors present a general approach for designing topology adaptation algorithms, it would have been better if the operation used for populating the list was left abstract and the gossip-based approach was presented as a possible option for implementing it. The gossip based approach might not be desirable in some applications as it requires a continuous bandwidth usage even when there is no flux of peers on the network and the desired topology has been achieved. The gossip-based search is also not helpful if there is an unusually high churn (e.g., more than 10%) of peers on the network.

In this algorithm the peers keep applying the selection function to the peers in the host-cache even when they have a desired neighbor set. The authors present T-Man as an approach for constructing topologies. It would reduce resource consumption (e.g., computing, network resources consumed to establish a new connection) if a delay was introduced between successive applications of the ranking function if a desired topology has been achieved or the peers that would be best suited as neighbors are not found in the cache in successive applications of the selection function, resulting in a change in the active connections.

The authors evaluates only simple topologies such as ring and mesh that can be generated without the need of a global knowledge of the overlay network. The evaluation does not include any interesting topologies that can be generated without the knowledge of the global network topology. The authors claim that there is a logarithmic relationship between the network size and the convergence speed. However, an inaccurate and confusing definition of convergence is used in the simulations. In the simulations the algorithm is said to have converged when no new edge from the desired topology is created. The converged state however does not guarantee that the desired topology has been achieved nor that the algorithm is not consuming resources.

2.2.9 Summary

The algorithms in this section demonstrate the utility of topology adaptation as a general technique for improving P2P applications. The section presents algorithms which demonstrate that topology adaptation can be used for load balancing, latency improvement and to improve the performance of search.

The majority of the existing topology adaptation algorithms concentrate on creating and maintaining a desired topology. The algorithm presented in section 2.2.5 is novel in its proposal to completely change the type of the overlay network with change in conditions on the overlay network. This could be an interesting direction for future research in the area of P2P topology adaptation algorithms.

The search operation for new neighbors is a major cost in the topology adaptation algorithms. The algorithms discussed in this section typically maintain a cache that contains information about other peers on the network and can be used for finding peers with desired characteristics. This approach suffers from the drawback of temporally inaccurate data in the cache because of peers leaving the system. Only one possible solution has been proposed in T-Man to address this drawback. It would be interesting to investigate other possible solutions for handling this problem.

A lack of mathematical analysis is another common problem with the existing decentralized topology adaptation algorithms. An exception is the centralized algorithm presented in section 2.2.2 which presents an analytical approach to predict the diameter of the generated topology. In general it is difficult to use analytical models to predict the characteristics of topology generated by decentralized topology adaptation algorithms because of the large number of variables in the system. None of the decentralized algorithms discussed in this section present a mathematical analysis of their algorithms but rely on simulations to study their behavior. This is because the existing mathematical techniques are inadequate to analyze the decentralized topology adaptation algorithms and more research is required in this direction. In section 4.1 we will show how the existing mathematical techniques are inadequate to analyze the decentralized topology adaptation algorithm presented in this thesis.

Instead of topology-related metrics (e.g., whether generated topology is connected or disconnected, the average degree of all the peers on the network) the algorithms discussed in this section are evaluated using the metrics that measure the increase in performance because of using the topology. Not using the topology-related metrics would result in a scenario where the evaluation would not be able to capture problems such as a disconnected topology because of applying the decentralized topology adaptation algorithm. None of the decentralized topology adaptation algorithms presented in this section (except T-Man) evaluate the effect of peers leaving and joining the overlay network.

The existing algorithms are application-specific and are not generally applicable or easily reusable. As a result, application developers have to design topology adaptation algorithms from scratch and an application that has been developed using an existing algorithm can not be altered easily to create and maintain a new topology. While the decentralized topology adaptation algorithms presented in this section are application specific they do have a common structure. All the decentralized algorithms discussed in this section explore the idea of peers using their limited local knowledge to create the desired topology. In all of these algorithms the peers periodically check their neighbors to determine if they are satisfied with them and if they are not satisfied, then they rearrange their neighbors. In the algorithms presented in sections 2.2.4 and 2.2.3, a peer uses a satisfaction criterion to decide if it is satisfied with its neighbors. In the algorithms presented in sections 2.2.5 and 2.2.7, a peer applies a selection function on a list of peers to determine the desired neighbors and if the existing neighbors are not the same as the desired neighbors then the peer is not satisfied. The general applicability of the later approach has been recognized in T-Man, while that of the former has not been studied.

2.3 Reference Architecture

As discussed in section 1.4.2, a reference architecture for a domain can be used to compare and describe the structure of existing P2P applications and as a template for designing applications. To the author's knowledge only one other reference architecture has been proposed for the P2P domain that aims to develop a vocabulary of P2P application components and facilitate P2P application development. This section presents and evaluates this reference architecture. JXTA is a general purpose P2P middleware. The JXTA specification contains guidelines for developing P2P applications and just like a reference architecture it can be used as a starting point for developing P2P applications. This section presents JXTA and evaluates its use as a reference architecture. The section examines the suitability of JXTA and the "The Essence of P2P" for topology adaptation.

2.3.1 The Essence of P2P

In 2005, Aberer et al. presented a reference architecture for the P2P domain [19]. The authors claim that the overlay network with its resource location service is the core component which allows advanced P2P services and applications to be developed. The authors first present a conceptual model for P2P overlay networks. As a part of the model, the authors present the key steps that are required in designing an overlay network and the desired attributes in the different steps. The key steps in designing the overlay network are enumerated below:

1. Deciding on the identifier space that will be used to assign unique identifiers to peers and resources on the overlay network. The identifier space should be scalable so that it can support large systems. It should provide identifiers that are independent of the entities' actual physical location.

2. Deciding on the mapping that will be used to assign an identifier to the peers

on the overlay network. The mapping might be one-to-one or if the system uses replication, it is one-to-many.

3. Identify the mapping that will be used to assign resource identifiers to peers. Typically, all the identifiers in the resource identifier space will be allocated to a peer. To provide fault-tolerance, a set of peers rather than just one might be made responsible for an identifier.

4. Decide on the neighborhood relationship that determines the set of peers to which a given peer maintains connections. It is desirable to have a relationship that creates a connected topology with a small diameter.

5. Design the routing strategy that will be used to decide to which peer from the neighbor set an incoming message will be routed.

6. Decide on the strategy to handle the flux of peers. This typically involves repairing the routing tables.

The authors propose a three-layered reference architecture in which the lower layer is responsible for providing the functionality associated with the overlay network. The middle layer consists of P2P services that utilize the services offered by the lower layer. The authors describe a storage service as an example. The top layer is composed of P2P applications that utilize the services offered by the middle and lower layer. API's for the lower layer and the storage services have been proposed. The API for the lower layer includes functions to allow a peer to join and leave the overlay network, find a peer whose identifier is provided, route a message to one or multiple peers and get information about the local peer and its neighbors. The API for the storage service includes functions to insert, delete, update and query data.

Discussion

The reference architecture and the conceptual model do not capture how services and resources can advertise their presence to the overlay network. In some applications, resources and services might like to advertise their presence to other peers so that other peers can use these advertisements to locate the resources and services present in the network. The authors do recognize that creating and maintaining the topology would be the key steps in designing the overlay network, however they do not define and describe components in the reference architecture for this. The authors talk about services as a layer in the reference architecture and present the APIs for a service that can handle storage on a P2P network. However, the authors do not describe how to structure services or how the overlay network APIs can be used by a generic service. Also no validation of the reference architecture is presented.

The authors have presented a conceptual model that enumerates the various steps in designing an overlay network. However it is hard to consider this a comprehensive reference architecture. "The Essence of P2P" cannot considered to be a reference architecture as per the definition presented in section 1.4.2 because it does not provide any details of the software components that may be used to implement this high level component. The "Essence of P2P" only defines the APIs for a very high level component (the lower layer in the discussion) that provides the services associated with the overlay network. The API's are however useful as they capture the common across a wide variety of applications and can be used to design a more comprehensive reference architecture.

The conceptual model recognizes that creating a desired topology (step 3) and maintaining this topology under a flux of peers (step 5) are key steps in designing an overlay network. The reference architecture however does not describe or define the software components that will be responsible for creating and maintaining the topology. This reduces the utility of this reference architecture for topology adaptation.

2.3.2 JXTA

JXTA was conceived by Sun Microsystems in 2001 and has been developed as an open source project. JXTA defines a set of six protocols for P2P applications [8, 80]. The JXTA community provides a reference implementation of JXTA protocol specification. The reference implementation can be used for P2P application development.

The JXTA protocols work together to perform services such as discovery, organization, monitoring and communication between peers. The JXTA protocols can be used for creating a new service and accessing existing services. The protocols use XML schemas to describe the format of the messages exchanged between peers to perform a service. JXTA is policy agnostic and does not specify how the service provided by a protocol will be implemented. The standardization of protocols helps in interoperability between peers.

The JXTA protocol specification is divided into three sections: JXTA Core Specification, JXTA Standard Services and JXTA Reference Implementations. JXTA Core Specification must be followed by an implementation to be JXTA compliant. Implementation of core specification do not guarantee interoperability with other implementations. JXTA Standard Services define optional components and behavior for a JXTA implementation. Implementing these components improve the interoperability of JXTA components. JXTA Reference Implementations contains information about components implemented by various JXTA bindings which are not required and specified in either of the sections above. Table 2.3.2 specifies the components defined in JXTA Core Specification and Standard Services.

The JXTA id is a location independent, unique identifier for entities in a JXTA network. Resources, services and peers in a JXTA network are described using XML documents called *advertisements*. JXTA specifies XML schemas which define the structure of the advertisements. Figure 2.3 shows the XML schema of a peer advertisement. JXTA defines a socket like abstraction called *Pipe* that can be used as a virtual commu-

Core Specification	Standard Services
Id	Peer Discovery Protocol(PDP)
Advertisements	Rendezvous Protocol(RVP)
Message	Peer Information protocol(PIP)
Service	Pipe Binding Protocol(PBP)
Peer Resolver Protocol(PRP)	Message Transport Bindings
Endpoint Routing Protocol(ERP)	Wire Transport

Table 2.1: Components defined in JXTA Core Specification and Standard Services.

```
<xs:element name="PA" type="jxta:PA"/>
<xs:complexType name="PA" />
<xs:sequence/>
<xs:element name="PID" type="JXTAID" />
<xs:element name="GID" type="JXTAID" />
<xs:element name="Name" type="xs:string" minOccurs="0" />
<xs:element name="Desc" type="xs:anyType" minOccurs="0" />
<xs:element name="Svc" type="jxta:serviceParams" minOccurs="0"
 maxOccurs="unbounded" />
</xs:sequence />
</xs:complexType />
```

Fig. 2.3: XML schema of a peer advertisement.

nication channel to send and receive messages between peers on the overlay network.

A JXTA enabled service is described using a module specification advertisement. The module specification advertisement of a native JXTA service contains a pipe advertisement which can be used to invoke the service. A client interested in invoking the service uses the *Pipe Binding Protocol* (PBP) to resolve the Pipe Advertisement to a pipe on the overlay network. PBP defines the structure of the query and result XML documents which will be exchanged to locate the pipe.

The PBP query and result are embedded in a *Peer Resolver Protocol* (PRP) message. PRP uses XML schema to define the structure of a XML request and response messages which are exchanged between peers to resolve a query. The request message contains the id of the peer from which the query originates. On a peer the request is directed to a handler, which is generally a service on a peer. For example in this case the service will be PBP.

JXTA does not puts a restriction on how the PRP message is propagated to peers in the network. The client may use *Rendezvous Protocol* (RVP) to propagator the message on the network. Once the input pipe is located the client needs to determine whether a route to the service provider exists or not. If a direct connection to the peer providing the service cannot be found then *Endpoint Routing Protocol* (ERP) is used to find intermediate hosts which can route the information to the destination peer. The route information to the destination peer is stored at the source peer. Once the connection is established the client utilizes the service by exchange *messages* through the pipe.

A *Message* is the basic unit of data exchanged between peers. A JXTA application can exchanges messages by using the pipe abstraction. Messages are transported on the network using existing network protocols. JXTA specification contains *Message Transport Bindings* which specify the rules for exchanging messages using HTTP, TCP/IP and TLS. JXTA also specifies the *wire representation* of messages. Messages can be represented as XML or binary data. Message may contain an arbitrary number of named sub-sections which can hold any form of data.

Advertisements of resources (e.g., pipes) and services in a peer group (overlay network) can be published and discovered using a Peer Discovery Protocol (PDP). PDP defines the format of the XML messages that will be exchanged to discover the advertisements.

Some JXTA peer-groups may have special peers called rendezvous peer, which cache advertisements. Rendezvous Protocol (RVP) allows peers to subscribe to the rendezvous peer. It also allows messages to be send to all the peers who have subscribed to the rendezvous peer. RVP can be used to propagate messages or to discover an advertisement. Peer Information protocol (PIP), can be used to inquire the status of a peer.

The JXTA reference implementation [121] uses a hub topology to perform resolution operations and for connecting peers that do not have a direct physical connectivity because of firewall, NAT etc. The rendezvous peers are hubs that organise themselves into a loosely-coupled network. In JXTA name resolution involves finding one or more appropriate advertisement. Rendezvous peers maintain a index of advertisements published by the ordinary peers. The index maintained by the rendezvous peers is used for name resolution. Each rendezvous peer maintains a list of other rendezvous peers it is aware of. At regular time intervals, the rendezvous peers pass this information to all known other rendezvous peers. The information in the list is used by the rendezvous peers to maintain the loosely-coupled network.

JXTA as a Middleware

While JXTA specification provides mechanisms to advertise, discover and access services it does not specifies how services should be described. There is no JXTA specified way to describe a service which might be a drawback for a programmer interested in using the rest of the JXTA stack for his P2P services.

JXTA is a middleware for P2P applications designed and developed from scratch. It reinvents and implements considerable amount of work around messaging which has been done for conventional middleware. For example, JXTA defines the format of the messages exchanged between the peers and it also specifies how these messages could be transported across peers using an existing transport like HTTP or TCP. JXTA could have build upon existing middleware solutions, like CORBA, RMI or SOAP to provide this functionality. If JXTA was built upon an existing middleware it could have benefitted from the support for type safe invocations provided in these middlewares. The current JXTA specification lacks support for type safe invocations, which can lead to difficult to track bugs in a JXTA application. Bugs caused by mismatch in data type are very comman and difficult to diagnose. In JXTA application programmer has to do all the programming for a type safe invocation.

In JXTA, the specification for exchanging messages between peers is interleaved with the specification for the core functionality (e.g., finding routes, identifying and describing resources) required by a P2P system. For example the Core Specification contains the rules for ids and advertisements, along with the structure of a message. While reading the JXTA specification, considerable effort is required to map the protocols to the layer (application or messaging) at which it operates. This makes JXTA overly complex and difficult to understand.

There are two ways to implement a service in JXTA: pipe and an existing middleware. There are two ways to publish a service in a JXTA network: using a pipe advertisement and a module specification advertisement. These approaches provides a high level of control to application developers, however complexity like this makes the learning cure for JXTA very steep.

JXTA as a Reference Architecture

JXTA provides protocol standards for a P2P middleware and a reference implementation. As per the definition used in this thesis (see section 1.4.2) a reference architecture describes and defines the software elements that comprise the system. While JXTA specification provides protocols for many different components of a P2P application, it does not makes an attempt to describe and define the core components of a P2P application and so it can not be classified as a reference architecture. A reference architecture provides the vocabulary that can be used to discuss the structure of existing P2P applications. The complexity of JXTA specifications and the mixture of P2P specific concerns with messaging concerns makes it difficult to develop a vocabulary for P2P that is based on JXTA. JXTA provides an easy to use API that can be used to develop P2P applications. However, it does not provide a design that can be used to understand or develop the software architecture of the P2P application.

JXTA and Topology Adaptation

JXTA specification is topology agnostic and does not provide any direct support for topology adaptation. However, the protocols from JXTA specification may be used for implementing topology adaptation algorithms. For example, the request and response messages between peers can be exchanged using PRP. The JXTA reference implementation uses a hub topology for implementing rendezvous service. The JXTA reference implementation however does not provide control over the underlying topology that the applications built using the JXTA reference implementation use.

2.3.3 Summary

Neither JXTA nor "The Essence of P2P" provide support for topology adaptation. "The Essence of P2P" does not describes how topology adaptation can be implemented in an application. The JXTA protocols may be used for topology adaptation, however it does not recognizes topology adaptation as a core service for applications and provides no direct support for topology adaptation.

According to the definition of reference architecture used in this thesis neither JXTA nor "The Essence of P2P" can be classified as a reference architecture. JXTA provides limited support for describing and comparing the structure of existing applications and can not be classified as a reference architecture as per the definition used in this thesis. "The Essence of P2P" does not presents the details of the software components that may be used to implement an application. This makes it of limited use while designing a P2P application. However, it does present the steps in designing the overlay network and the choices used in these steps can be used to describe and compare existing applications. It can therefore be seen as a first step towards a general-purpose reference architecture.

2.4 Summary

This chapter reviewed topologies, topology adaptation algorithms and reference architectures from literature. The chapter reviewed topologies from nature and found them to be particularly interesting because of their self-organizing nature and desirable characteristics such as low average distance between peers. The decentralized topology adaptation algorithms share a common structure, but were developed from scratch. The existing mathematical techniques are inadequate for analyzing the decentralized algorithms and more research is required in this direction. As per the definition of reference architecture used in this thesis neither JXTA nor "The Essence of P2P" discussed in the review can be classified as a reference architecture. Further both provide limited support for topology adaptation reducing their utility for designing applications that require topology adaptation.

Chapter 3

Topology Adaptation Using Schelling's Model

God does not play dice with the universe [35].

In 1969, American economist Thomas Schelling proposed an agent-based model that can be used to describe the existence of residential segregation in urban areas. This chapter presents an abstract algorithm inspired by Schelling's model that can be used to create a family of decentralized and self-organizing topology adaptation algorithms. A concrete realization of the abstract algorithm can be executed by the peers to create and maintain an overlay network with a particular topology.

This chapter gives a brief introduction to agent-based models and then describes Schelling's model as proposed in 1969. The chapter subsequently presents the abstract algorithm called the PEer-to-peer Self-organizing TOpology (PESTO). This is followed by a discussion of the messaging cost associated with PESTO, a description of the steps involved in instantiating PESTO, and a comparison of PESTO with T-Man discussed in section 2.2.8. The chapter then presents the design and usage details of a simulator that can be used for evaluating the concrete realizations of PESTO.

3.1 Agent-Based Models

An agent-based model [44] is a tool that can be used to study the emergence of complex behavior from simple rules in decentralized systems. An agent-based model consists of large number of agents that act to change their properties and their environment by using simple rules based on the agent's limited awareness of the entire system. Schelling's Model described below is a putative example of an agent-based model. Agent-based models are difficult to analyze mathematically and software frameworks like Jade [6] and StarLogo [15] are typically used to simulate and understand their behavior.

P2P applications share many similarities with agent-based models:

1. Both peers and agents lack a complete picture of the environment and act exclusively on the basis of local information.
2. Both P2P applications and agent-based models lack central control.

The characteristics of agent-based models that makes them a promising approach for designing P2P applications are:

1. The accumulated actions of the individual agents that act autonomously on simple rules using their limited awareness of their environment result in a far more complex and interesting behavior (called *emergent behavior*) than the behavior of individual agents. Peers can be programmed in a similar fashion to act on simple rules using their limited awareness of the environment to create a desired global behavior. The challenge is to design the simple rules that lead to the desired global behavior.
2. Agent-based systems are typically good at adapting to changing environmental conditions and are resilient to deviant behavior in a small number of agents. The

adaptive and resilience characteristics are promising because P2P applications work in a dynamic environment with a constant flux of peers.

3.2 Schelling's Model

Residential segregation is a common phenomenon in urban areas and has been observed as early as in 1842 by Friedrich Engels, one of the founding fathers of communism. He observed an inexplicable pattern in the residential plan of the city of Manchester in the UK. In his book *The Conditions of the Working Class in England*, Engels observed that the design of the 19th century Manchester city segregated people belonging to different classes, even though the city was not built to any plan [62, pg. 36]. Engels was unsuccessful in finding a central cause of this segregation and failed to recognize that the segregation could occur in the absence of a central figure directing the segregation. In 1969, American economist Thomas Schelling proposed an agent-based model to explain the existence of segregated neighborhoods in urban areas [99, 119, 100]. While the model is well-known for explaining residential segregation, it can also be used to explain segregation between any two classes in a variety of scenarios such as boys and girls in a party or students and faculty.

Description

In Schelling's Model, the world is considered an $m \times n$ grid. The world is populated by two types of agents, called blue and red turtles, which are analogous to people in the real world. Approximately two thirds of the cells in the grid are populated by blue or red turtles. The remaining cells are empty. Each cell can host a maximum of one turtle. In the beginning, a random number of blue and red turtles are randomly distributed on the grid.

All the turtles act using a simple rule which states that a turtle is satisfied if at

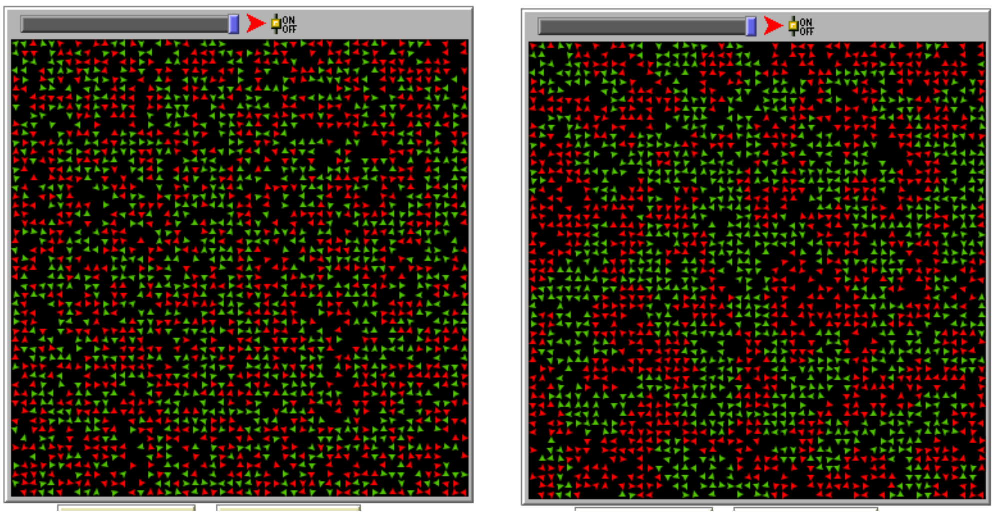


Fig. 3.1: Schelling's world before (left) and after (right) the turtles are satisfied.

least a certain percentage of its neighbors are of the same color as itself and that if a turtle is not satisfied with its neighbors, then it moves to an adjacent empty cell (if available) chosen randomly. The simulation goes on until all the turtles are satisfied with their neighbors. As the simulation progresses, segregation can be observed on the grid. Such segregation is an emergent behavior caused by the desire of the turtles to ensure that a certain minimum percentage of their neighbors are the same color as themselves. Figure 3.1 shows snapshots of a Schelling's world before and after the turtles are satisfied, when the turtles desire 30 percentage of their neighbors to be similar. The diagrams were drawn with NetLogo [126], an agent-based modelling software.

In Schelling's Model, the turtles act using their awareness of the local network topology. This makes the model interesting for P2P systems, because the peers do indeed lack a global picture of the network topology. In Schelling's Model, grouping is maintained even when turtles join or leave the system, which makes it attractive for the dynamic environments of P2P networks.

Schelling's work is often misclassified as an example of a cellular automaton [128, 53]. A cellular automaton is a model that can be used to study the emergence of complex behavior from simple rules. A cellular automaton is a lattice of sites in which each site can have k possible values. The value of each site is updated by using a rule that depends on the value of sites in the neighborhood around it [83]. Schelling's Model is concerned with agents (turtles), not sites, that change their location property using simple rules based on the state of their neighborhood.

Variations of Schelling's Model have been mathematically analyzed by Zhang [131, 132] and Young [129], as a game played between people. In the game, each player has a strategy and a payoff that is determined by the status of the player's neighborhood. By using theories from stochastic dynamical systems, Zhang has shown that the stable state for the system is a segregated state.

Simulations

NetLogo comes with a sample model that can be used to simulate Schelling's Model. The sample model has two parameters: the total number of turtles (N) and the percentage of similar neighbors ($PSND$) that each turtle desires. In the NetLogo model, segregation can be observed visually in a graphical display of the turtle world (see figure 3.1). The model tracks the percentage of turtles having all similar neighbors ($PTASN$) over time. This metric gives a numerical idea of segregation.

The simulations are done for a grid of 50×50, which has 1500 turtles; half of which are blue and half red. Figure 3.2 plots $PTASN$ after the simulations are complete versus $PSND$. The parameter $PSND$, has a critical value $PSND_{unstable}$, beyond which the simulation does not converge to a state where all the turtles are happy with their neighbors. The $PTASN$ value does not stabilize when $PSND$ is beyond $PSND_{unstable}$. The value of $PSND_{unstable}$ observed in the simulations was 75% and so $PTASN$ values for $PSND$ after 75% are not plotted. When $PSND$ is beyond $PSND_{unstable}$ the

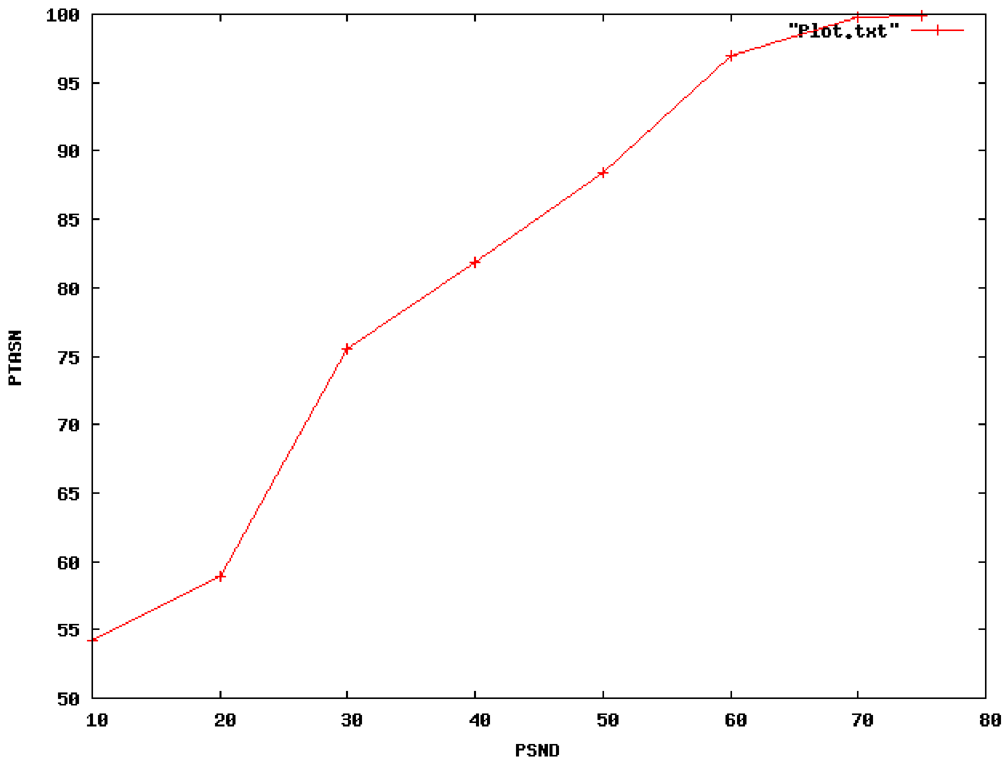
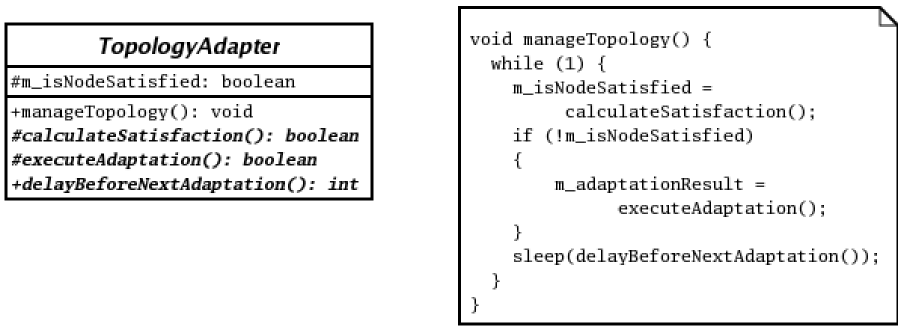


Fig. 3.2: Plot of PTASN vs PSND.


TopologyAdapter
#m_isNodeSatisfied: boolean
+manageTopology(): void
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int

```
void manageTopology() {
  while (1) {
    m_isNodeSatisfied =
         calculateSatisfaction();
    if (!m_isNodeSatisfied)
    {
        m_adaptationResult =
             executeAdaptation();
    }
    sleep(delayBeforeNextAdaptation());
  }
}
```

Fig. 3.3: UML Class diagram of the TopologyAdapter abstract class. The manage-Topology method presents the structure of PESTO. The methods calculateSatisfaction, executeAdaptation and delayBeforeNextAdaptation are virtual abstract methods implemented in subclasses.

percentage of dissatisfied turtles fluctuates around a high value (e.g., for $PSND$ 75 the percentage of dissatisfied turtles fluctuates around 85%) and movement of turtles is observed in the grid. This suggests that when $PSND$ is beyond $PSND_{unstable}$ the movement of some of the dissatisfied turtles to an adjacent empty cell on the grid does not changes their satisfaction with their neighbors and because of this the simulation does not converges.

Even a small value for $PSND$, leads to the emergent behavior of a very high percentage of turtles having all similar neighbors ($PTASN$). Simulations suggest that segregation does not depend on the number or number of types of turtles. The sample NetLogo model was modified so that the world is populated with more than two types of turtles. An emergent behavior similar to the old model was observed.

3.3 PESTO

Schelling's algorithm is the algorithm executed by turtles in the Schelling's Model. In Schelling's Algorithm, the turtles periodically calculate their satisfaction with their neighborhood and if they are not satisfied then they execute steps to change their neighborhood. Variations of Schelling's algorithm can be created by changing the satisfaction criteria, the frequency with which the satisfaction criteria are calculated and the steps taken to change the neighborhood. The self-organizing and decentralized nature of Schelling's algorithm makes it an attractive approach for creating and maintaining a desired topology in P2P overlay network. This section presents an abstract algorithm called the PEer-to-peer Self-organizing TOpology (PESTO) that can be used to create a family of topology adaptation algorithms for decentralized unstructured P2P overlay networks. PESTO is based on Schelling's algorithm. While PESTO may be used for topology adaptation in hybrid P2P networks, however as discussed in section 1.4.5 the presence of a central component makes it comparatively easy and uninteresting to perform topology adaptation in hybrid networks and so this thesis concentrates only on utilizing PESTO for topology adaptation in unstructured decentralized networks

3.3.1 Description

In the Template Method [41] design pattern, the skeleton of an algorithm is defined in an operation, deferring the steps that may change to a subclass. The subclasses implement the steps that vary. The skeleton of the algorithm provides a template that can be used to create a family of algorithms. In this work, the Template Method design pattern is used to create PESTO, an abstract algorithm based on Schelling's algorithm, which provides a template for P2P topology adaptation algorithms. The pseudo-code for the manageTopology method in figure 3.3 encapsulates PESTO. Just like Schelling's algorithm, the steps that may vary in PESTO are the satisfaction criteria, the actions to be performed if a peer is not satisfied and the frequency with which the satisfaction state should be checked. A peer calculates its satisfaction state at pre-defined intervals and if it is not satisfied, it executes its topology adaptation steps (TAS).

Satisfaction state is a boolean value indicating whether a peer is satisfied with

Operation	**Details**
count(*property*)	The number of neighbors of a given node matching the given property.
add(*peers*)	Add the given peer or peers as a neighbor
drop(*peer*)	Drop the given peer as a neighbor
neighbor(*property*)	Returns a neighbor with the given property.
search(*property*)	Search for peers on the overlay network with the given property.

Table 3.1: The operations that can be executed by the peers. The operations are used in this thesis to describe the concrete realizations of PESTO.

Satisfaction Criteria	**Topology Adaptation Steps**
$\frac{\textbf{count}(\textit{same property})*100}{\textbf{count}(\textit{all})} > PNSP$ where PNSP is the desired Percentage of Neighbors with Similar Property	*step 1:* **drop**(**neighbor**(*different property*)) *step 2:* **add**(**search**(*same property*))

Table 3.2: A satisfaction criterion and topology adaptation steps that can be used to bring together peers with similar properties (e.g., bandwidth).

its local view of the overlay network's topology. The satisfaction state of a peer is calculated using the calculateSatisfaction method. If a peer is not satisfied with its neighbors then the *topology adaptation steps* are performed by calling the executeAdaptation method. The topology management algorithm, specified in the manageTopology method, is executed repeatedly. The time delay between successive executions of the topology management algorithm is determined by the return value of the method delayBeforeNextAdaptation. The satisfaction criteria (SC), the topology adaptation steps and the time delay will vary with the application and the topology desired.

Table 3.1 presents the operations that can be used to describe SC and TAS. Table 3.2 presents a simple example of SC and TAS inspired by Schelling's Model that can be used for clustering peers with similar properties. The SC state that a peer is satisfied if it has at least a small percentage of similar peers as neighbors. The associated TAS is that the peers drop a dissimilar peer, find similar peers and add them as neighbors.

PESTO is evaluated by presenting concrete realizations that can be used to create a clustered and a hub topology. Section 2.1 presented a review of a range of interesting topologies for P2P applications. The simple topologies such as ring, lattice and tree are not interesting because to the author's knowledge it is difficult to create and maintain them using a decentralized approach. While both small-world and power law topologies are interesting because of their low CPL, the power law topology is more useful for P2P applications as it utilizes the diversity among peers on the overlay network. Section 2.2.3 and 2.2.4 presents two existing topology adaptation algorithms that may be used to create a power law topology and so we have not explored in detail the option of creating a power law topology using PESTO. We present examples of SC and TAS in section 3.3.3 that can be used to create a power law topology but we do not evaluate these SC and TAS. PESTO is based on Schelling's model that is used to explain the existence of segregated neighborhoods in urban areas. In a clustered topology peers with similar characteristics are grouped together as neighbors. The clustered topology is an obvious first choice that has been used in this thesis for evaluating PESTO, because of its similarity with segregated neighborhoods. Chapter 4 explores the approach to clustering, presented in table 3.2 in more detail and also presents another example for SC and TAS that can be used for clustering in pure unstructured P2P networks. The hub topology is the other topology that has been used in this thesis to evaluate PESTO because of a lack of topology adaptation algorithms to create and maintain a hub topology. To the author's knowledge, the JXTA reference implementation (discussed in section 2.3.2) uses the only other existing topology adaptation algorithm for creating a hub topology. Chapter 5 presents a concrete realization of PESTO that can be used to create a network of hubs in a pure unstructured P2P network.

PESTO is advantageous because it can be used as a basis for designing decentralized, self-organizing topology adaptation algorithms. Another advantage of this approach is that an application can dynamically change its topology adaptation algorithm. For example, an application would be coded against the TopologyAdapter abstract class and the concrete implementation of the abstract algorithm could be changed at runtime by using the Component Configured design pattern that allows “applications to link and unlink its component implementations at runtime without having to modify, recompile or statically relink the application” [32, pg. 75].

3.3.2 Messaging Cost

There is a messaging cost associated with utilizing the topology adaptation algorithms based on PESTO. A topology adaptation algorithm based on PESTO may consume bandwidth to calculate the satisfaction state and to execute the topology adaptation steps. The messages exchanged to execute the topology adaptation steps depend on the choice of the topology adaptation step. In a topology adaptation algorithm based on PESTO, a peer may take a proactive approach and ping its neighbors periodically to find information about them which it can use to decide its satisfaction state. If k is the average number of peers to which each peer on the overlay network is connected and n is the number of peers on the overlay network, then a topology adaptation algorithm that takes a proactive approach requires the peers to exchange kn messages every iteration. This is a common messaging cost that will be incurred by all the topology adaptation algorithms based on PESTO that take a proactive approach.

If this common messaging cost is a concern for an application then a topology adaptation algorithm may use a reactive approach in which a peer P uses cached information about its neighbors to decide its satisfaction state. The peer P may change the cached information about a neighbor and recompute the satisfaction state when an event such as the ones described below occur:

1. the peer P receives a notification from its neighbors about a change in their status,
2. the underlying network connection between P and its neighbor N breaks. In this scenario P may try to reestablish connection to N and if the reconnection attempt fails then P may remove N as its neighbor.
3. P is unable to deliver a message to a neighbor.

For a satisfied peer using the reactive approach, the topology adaptation algorithm will wait in the $delayBeforeNextAdaptation(\ldots)$ method till an event such as the one described above occurs. Compared to the proactive approach, a small number of messages are exchanged to compute the satisfaction state when the reactive approach is used. A drawback of this approach is that a neighbor might not notify P about a change in its status. Another drawback of the reactive approach is that it may take time before a peer realizes that a neighbor is not available.

3.3.3 Designing Algorithms Using PESTO

This section enumerates the steps in designing a topology adaptation algorithm based on PESTO.

Step 1: Identify the Desired Topology

The first obvious step in designing a topology adaptation algorithm is to identify the desired topology for the application. Different applications require different topologies. Section 2.1 presents several topologies that may be utilized in an application. For example, a messaging application might desire a small routing distance between the peers and a high tolerance to failure of peers. For such an application, the power-law topology discussed in section 2.1.5 would be a good choice.

Step 2: Identify the Topology Metrics

The next step is to decide on the metrics that can be used to evaluate the topology created by the topology adaptation algorithm. For the messaging application presented in step 1, the generated topology can be evaluated using the metrics characteristic path length (CPL) and degree distribution function ($p(k)$) discussed in section 2.1.1. The desired topology will be achieved when the value of CPL is of the order of $log(N)$ where N is the number of peers on the network and pk has a power law relation with the degree (k) of a peer.

Step 3: Define the SC and TAS

The key challenge in designing topology adaptation algorithms using PESTO involves creating satisfaction criteria (SC) and topology adaptation steps (TAS) that can be executed autonomously by the peers, without requiring a global knowledge of the network, to create the desired topology.

Here we outline possible SC and TAS that can be used to create a power law topology. Let C be the ordered list of peers that a peer is aware of. The list C is arranged in a descending ordered of the degree of the peers so that the peer with the highest degree is at the beginning of the list. The list could be populated by sending random walk messages to explore the network as done in DANTE (described in section 2.2.5). For creating a power law topology, a possible SC could be that a peer P is connected to at least n peers from the m peers at the beginning of the list C. We discussed in section 2.1.5 that power law topology is created because of a preference of a peer to connect to other peers with a high degree and we expect that this SC would result in a preferential attachment to peers with a high degree. In the TAS the peer would try to establish connections to the m peers at the beginning of the list C so that it (P) is connected to at least n of them. If the peer P is unsuccessful in establishing the n connections then it may send search messages in its neighborhood to determine other peers with a high degree with which it may establish a connection.

Step 4: Verify the Algorithm

The topology adaptation algorithm is verified through simulations using simulators such as the one presented in section 3.4. The algorithms are evaluated using the metrics identified in step 2. If the results are as decided in step 2, then the topology adaptation algorithm can be used. If the results are not as expected then the designer has to go back to step 3 and refine the SC and TAS. The simulations can also be used to decide on the values of the variables in SC. For the topology example in this section, simulations may be used to decide on the values of m and n.

3.3.4 Comparison with T-Man

PESTO was first presented in 2004 at a workshop in Parallel Problem Solving in Nature VIII (PPSN-VIII) [107]. Presented in the same year as PESTO and discussed in section 2.2.8, T-Man is another recent approach that has been proposed for designing topology adaptation algorithms. The aim of both PESTO and T-Man is to provide a template for designing decentralized topology adaptation algorithms. It was observed in section 2.2.9 that typically a decentralized topology adaptation algorithms can be described using a common structure. Typically in existing decentralized topology adaptation algorithms the peers periodically check their neighbors to determine if they are satisfied with them and if they are not satisfied, then they rearrange their neighbors. In the algorithms, a peer may use a satisfaction criterion to decide if it is satisfied with its neighbors or it may apply a selection function on a list of peers to determine the desired neighbors, and if the existing neighbors are not the same as the desired neighbors then the peer is not satisfied. The general applicability of the later approach for deciding satisfaction has been recognized in T-Man, while that of the former is used in PESTO.

In both algorithms, the challenge is to design rules that peers can execute using their limited awareness to create the desired topology. PESTO provides the SC and TAS abstractions and T-Man provides the ranking function abstraction that can be used to design the rules. In the opinion of this author, the abstractions for designing the simple rules are equally intuitive for both the algorithms. The rest of this section presents the advantages of PESTO when compared to T-Man.

T-Man relies on a cache populated by gossip between peers to search for the desired neighbors. Such a gossip-based search suffers from the disadvantage that bandwidth is continuously consumed even when there is no flux of peers on the network, which might not be desirable for an application in which flux of peers is not expected after a certain time. As described in section 2.2.8, gossip-based search also suffers from the disadvantage of temporally inaccurate information in the cache because of peers leaving and joining the system. In contrast, PESTO allows more flexibility to the designers as it does not enforce a particular search method. In fact, in chapter 4 we present a topology adaptation algorithm based on PESTO that does not require a search operation.

In the algorithms designed using T-Man, the peers execute the topology adaptation algorithm periodically. This is unlike PESTO that gives the designers the flexibility to control the interval between successive execution of the topology adaptation algorithm through the $delayBeforeNextAdaptation(...)$ method.

PESTO is more flexible when compared to T-Man and does not suffer from the drawback of a high churn of socket connections. In algorithms designed using T-Man, the peers apply a ranking function on the peers in the cache to choose a neighbor. The information in the cache populated by a gossip-based search evolves with time, and a peer will have to change the socket connections to its neighbors several times before its neighbor set stabilizes. Establishing a socket connection is a time-consuming process and the churn in the socket connections is another weak point of algorithms designed using T-Man. The algorithms designed using PESTO and studied in this thesis do not suffer from this drawback.

3.4 Simulator

A simulator has been developed as a part of this thesis to evaluate concrete realizations of PESTO. In this section we first present a general description of the simulator and then the instructions on how to utilize it.

3.4.1 Description

In this simulator, all the peers are within one simulator process. The simulator is single threaded, which means that the peers execute their algorithm sequentially. Each peer has an implementation of PESTO associated with it. Each peer can be associated with a different implementation of the abstract algorithm if desired. In each iteration (also referred as a *time unit*) the simulator iterates over all the peers so that they can execute their topology adaptation algorithms. In the simulator, each peer is assigned an identifier which determines the order in which the simulator goes through the peers. The simulator goes through the peers in the ascending order of their identifier. The simulator allows the users to customize the conditions when the simulations should stop. By default, the simulations stop when all the peers are satisfied or when a user-defined number of iterations is reached.

The approach of sequentially executing the topology adaptation algorithms of the peers in a simulator iteration is a simplification that does not captures the concurrency issues and the stochastic nature of the real network. For example, in a real network in which the peers are using the topology adaptation steps specified in table 3.2, two peers having the same property executing the step 2 of the topology adaptation steps might have only one third peer with a similar property available that can accept only one other neighbor. In the simulator the peer with the lower identifier will be able to successfully execute its topology adaptation step and add the third peer as its neighbor. In a real network the peer from whom the third peer first receives the add message will successfully complete its topology adaptation step. While the sequential execution is a drawback it has been chosen because it is simple to implement and scales well for a large number of peers. Further, the sequential execution is an approach that has been used for P2P simulations in the existing literature. For example, QueryCycle simulator [101], [130] and PeerSim [61] use sequential execution in simulations. While PeerSim also supports an event driven approach the authors do recognize in [61] that in their experience the sequential execution should work fine for most scenarios.

The simulator software provides an implementation of a peer class. It expects the users to provide implementations for the TopologyAdapter interface that will be associated with the peers. The simulator uses the Interceptor [32, pg 109] design pattern to allow users to execute their own code before and after each simulator iteration and when the simulation starts and ends. The simulator allows users to customize how peers are connected to the overlay network. It provides default implementations that can be used to create a random topology and a power-law topology using the algorithms in [24].

The simulator provides a Reporter class that can calculate the value of metrics such as the clustering coefficient and the characteristic path length of the network. The Reporter class uses the boost C++ graph library [104] for graph-related algorithms such as Dijkstra's algorithm [50] used to find the shortest path between vertices in a graph.

3.4.2 Using the Simulator

The simulator is a C++ library and a user will have to write a C++ application to utilize it. The library is enclosed within the namespace SchellingSimulator. This

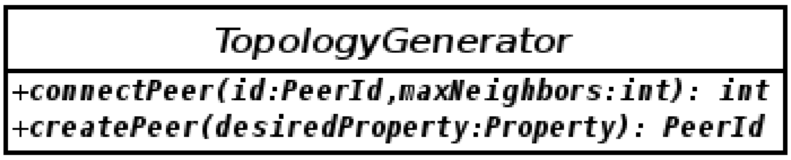

TopologyGenerator
+connectPeer(id:PeerId,maxNeighbors:int): int
+createPeer(desiredProperty:Property): PeerId

Fig. 3.4: UML Class diagram of the TopologyGenerator Interface.

section describes the key steps in writing an application to perform a simple experiment using the simulator.

Step 1: Implement the Topology Adaptation Algorithm

The Adapter [41, pg. 139] design pattern is used by the Simulator to interact with the topology adaptation algorithm. The simulator expects the topology adaptation algorithm to be encapsulated in a class that implements the TopologyAdapter abstract class shown in figure 3.3. The simulator provides implementations for the methods in Table 3.1 that are used in the concrete realizations of PESTO evaluated in this thesis. As discussed in section 2.2.9 the search(*property*) operation used to find peers with the given *property* is an integral part of topology adaptation algorithms. The simulator provides implementations for doing search(...) using Breadth First Search (discussed in section 2.2.1) and biased random walk search with replication (discussed in section 2.2.4).

Step 2: Implement the Topology Generator

The simulator expects an implementation of the TopologyGenerator interface (shown in figure 3.4) that is used by it to create new peers using the implementation of the method *createPeer*(...) and to connect them to the existing peers in the topology using the implementation of the method *connectPeer*(...). The TopologyGenerator implementation allows users to customize the topology generation functionality of the simulator.

The simulator provides two partial implementations of the TopologyGenerator interface that provide implementations for the *connectNode*(...) method. These implementations can be used to create a random topology using the algorithm described in section 2.1.3 and a power-law topology using the algorithm discussed in section 2.1.5.

Step 3: Write the Main Application

The application first obtains an instance of the Simulator class that has been implemented using the Singleton design pattern [41, pg. 127], as shown below:

```
Simulator *s = Simulator::getSimulator();
```

Create an instance of the TopologyGenerator interface implemented above and pass the instance to the Simulator as shown below. The instance will be used by the Simulator to create new peers and to connect them to the topology.

```
MyTopologyGenerator m;
s->setTopologyGenerator(m);
```

Create the desired number of peers and connect them to the topology using the code below. In this code, the simulator is used to create a peer with a property p and an implementation of the abstract Schelling's algorithm provided in the class MyTopologyAdapter. The simulator is also used to connect the peer to the topology. The property of the peer is an integer value and depends on the experiment.

```
MyTopologyAdapter mta;
PeerId p = s->createPeer(p, mta);
s->connectPeer(p);
```

Once the desired number of peers and a topology have been created, the experiment can be run by calling the *run*(...) method of the Simulator class as shown below.

```
s->run(0);
```

In the *run*(...) method, the simulator executes a loop. In every iteration (also referred as simulator iteration) of the loop, the simulator goes through all the peers in the Topology, executing their topology adaptation algorithm specified in the $manageTopology(...)$ method of their implementation of the TopologyAdapter abstract class.

The input to the *run*(..) method is a pointer to a function that is used to return a boolean value and expects no input. At the beginning of the loop, a *true* return value is expected from this function for the experiment to be terminated. Users can customize the termination of the experiment by providing their implementation of the *run*(...) method. When the input to the *run*(...) method is 0, the default function is used in which the experiment terminates when all the peers are satisfied or 1,500 simulator iterations are reached.

Step 4: Analyzing Experiment Results

The methods to analyze the results of an experiment are encapsulated in the Reporter class. The methods in Reporter use the current topology stored in the singleton Simulator object to calculate metrics such as percentage of satisfied peers, clustering coefficient, characteristic path length and degree distribution. The last three metrics are discussed in detail in section 2.1.1.

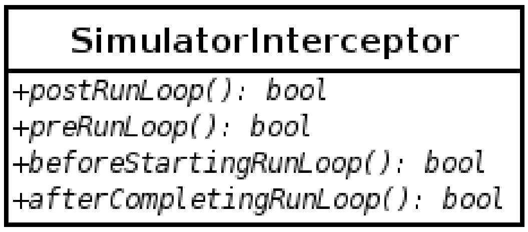

SimulatorInterceptor
+postRunLoop(): bool
+preRunLoop(): bool
+beforeStartingRunLoop(): bool
+afterCompletingRunLoop(): bool

Fig. 3.5: UML Class diagram of the SimulatorInterceptor class.

Customizing the Simulator

The users can execute their own code at different points in the *run*(...) method of the Simulator class to perform tasks such as adding new peers to the topology at the beginning of an iteration. To utilize this functionality, a user has to extend the SimulatorInterceptor class (shown in figure 3.5), add their code to the appropriate hook method and then create an instance of the extended class in their application. When an instance of the extended class is created, the default constructor of the base class registers the user's simulator interceptor with the Simulator. Users can create as many simulator interceptors as they desire.

The simulator executes the code from the simulator interceptor methods, before, after and inside the loop in the *run*(...) method of the Simulator class. The code in the *beforeStartingRunLoop*() method is executed before starting the loop, the code in the *afterCompletingRunLoop*() method is executed after completing the loop, the code in the *preRunLoop*() method is executed before starting the code in the loop and the code in the *postRunLoop*() method is executed after completing the code in the loop. The default implementation of these simulator interceptor methods in the base class do nothing and simply return the boolean value *true*.

3.5 Summary

This chapter has described the agent-based Schelling's Model that can be used to describe residential segregation in urban areas and is the inspiration behind the abstract algorithm for topology adaptation presented in this thesis. The chapter presented the abstract algorithm called PESTO that provides a template for designing topology adaptation algorithms. PESTO was compared with T-Man which provides an alternative approach for designing topology adaptation algorithms. The comparison suggests that PESTO is more flexible when compared to T-Man and does not suffers from the drawback of a high churn of connections. Finally we presented a simulator that can be used for evaluating concrete realizations of PESTO. The next two chapters present concrete realizations of PESTO that can be used for topology adaptation.

Chapter 4

Clustered Topology

No man is an island, entire of itself; every man is a piece of the continent, a part of the main . . . [40]

As described in section 2.1.7, clustering algorithms rearrange the overlay network's topology to create clusters of peers that have similar characteristics. This chapter presents two clustering algorithms that are concrete realizations of PESTO and the simulation results of applying these algorithms to a series of overlay networks. The chapter then presents a case study to demonstrate the utility of the clustering algorithms. The results show that the performance of messaging on an overlay network can be improved by applying the clustering algorithms to bring together peers with similar bandwidth connections. The algorithms have been published in [111] and the case study in [107]. The chapter finally compares the clustering algorithms with the state-of-the-art algorithms for clustering discussed in section 2.2.

4.1 Clustering Algorithms

In section 2.1.7, a formal definition of clustering was presented and the advantages of clustering in overlay networks discussed. In the simplest form, clustering involves

Algorithm 1 SelflessClustering Algorithm

```
$PNSP_{desired} \leftarrow$ % of neighbors with similar property desired
while $true$ do
    $PNSP_{actual} \leftarrow \frac{\textbf{count}(\textit{same property}) * 100}{\textbf{count}(\textit{all})}$
    if $PNSP_{actual} < PNSP_{desired}$ then
        if count($all$) $> 1$ then
            drop(neighbor(n: n.property $=$ $different\ property$ and n.count($all$) $> 1$))
        end if
        add(search($same\ property$))
    end if
    sleep($delay$)
end while
```

Algorithm 2 SelfishClustering Algorithm

```
$PNSP_{desired} \leftarrow$ % of neighbors with similar property desired
while $true$ do
    $PNSP_{actual} \leftarrow \frac{\textbf{count}(\textit{same property}) * 100}{\textbf{count}(\textit{all})}$
    if $PNSP_{actual} < PNSP_{desired}$ then
        drop(neighbor(n: n.property $=$ $different\ property$))
    end if
    sleep($delay$)
end while
```

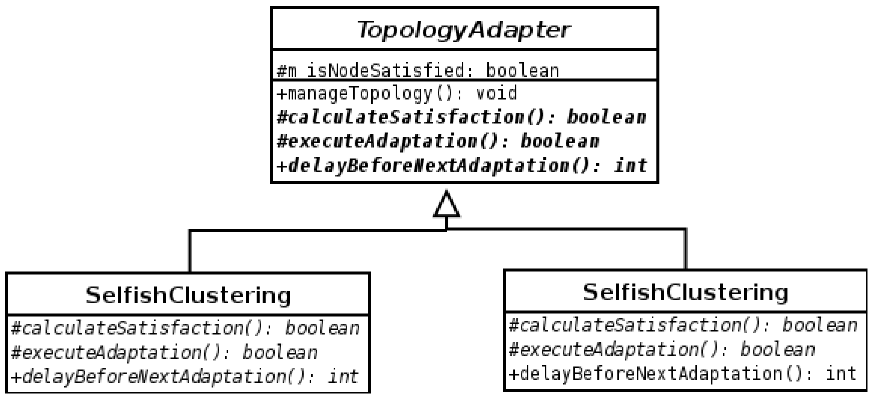

TopologyAdapter
#m_isNodeSatisfied: boolean
+manageTopology(): void
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int
SelfishClustering
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int
SelfishClustering
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int

Fig. 4.1: UML Class diagram depicting the TopologyAdapter interface and two concrete realizations.

rearranging the overlay network's topology to minimize the number of clusters (NC). In the literature, the term *clustering* is often used to refer to *identification* of clusters in a network. However, in this work the term refers to the *creation* of clusters. Cluster identification can be done by using algorithms proposed in [118, 22, 89, 133].

This section presents two clustering algorithms, called the SelflessClustering Algorithm and the SelfishClustering Algorithm, which can be executed by the peers on an overlay network. The clustering algorithms are concrete realizations of PESTO as shown in figure 4.1. In both algorithms, a peer examines its neighbors at regular intervals to see if it is satisfied with them, and if it is not satisfied then it executes a series of topology adaptation steps. The clustering algorithms use the same satisfaction criteria but differ with regards to the topology adaptation steps.

The clustering algorithms are self-organizing which means that they organize the peers using their local awareness of the overlay network's topology and without any central authority. The clustering algorithms maintain the reorganized topology under a flux of peers by calculating a peer's satisfaction state at regular intervals and taking topology adaptation steps if the peer is not satisfied with its neighborhood.

Algorithms 1 and 2 present the pseudo-code for the clustering algorithms. The operations that have been used in the pseudo-code are described in table 3.1. The clustering algorithms are described below in detail:

- **SelflessClustering Algorithm:** The satisfaction criteria states that a peer is satisfied if at least a certain percentage of its existing neighbors are similar to it. This percentage is called the desired percentage of neighbors with similar property ($PNSP_{desired}$). In this algorithm, a dissatisfied peer performs the steps below:

  - **step 1** - Drop a dissimilar neighbor N if it is not the only neighbor and the neighbor N is connected to at least one other peer (to ensure that the topology remains connected).
  - **step 2** - Search for a similar peer that has a free connection slot. If a suitable peer is found, then add it as a neighbor.

- **SelfishClustering Algorithm:** This algorithm uses the same satisfaction criteria that is used in the SelflessClustering Algorithm. In this algorithm a dissatisfied peer drops one randomly chosen dissimilar neighbor. This algorithm is called SelfishClustering because in this algorithm a dissatisfied peer drops a dissimilar neighbor even if the dissatisfied peer is the only link connecting the dissimilar neighbor to the overlay network. The selfish dropping of a dissimilar neighbors can result in peers that are disconnected from the overlay network.

The topology adaptation steps taken by the peers are intended to create a feedback effect that results in an overlay network with a $PNSP$ value that is higher than the $PNSP_{desired}$. When an unsatisfied peer adds another similar peer or drops a dissimilar peer as its neighbor, it increases its $PNSP$ value as well as the $PNSP$ value of the peer it is interacting with.

4.1.1 Messaging Cost

The clustering algorithms incur a common messaging cost that is associated with calculating the satisfaction state. As discussed in section 3.3.2, a topology adaptation algorithm may take a proactive or a reactive approach to calculating the satisfaction state. The best choice of approach to calculating the satisfaction state depends on the application. An application that can incur the high messaging cost may choose a proactive approach for better accuracy. Alternatively an application may choose a reactive approach. The cost associated with calculating the satisfaction state will be incurred throughout the life of the overlay network and is therefore not very useful in evaluating the algorithms. Apart from this common messaging cost, the SelflessClustering Algorithm incurs an additional messaging cost of searching for similar peers on the network.

Search Cost

The number of messages exchanged to perform the search operation for similar peers is a major cost of utilizing the SelflessClustering algorithm. The cost depends on how the search operation is implemented, which can be done in a variety of ways. For example, the search can be performed by using a Gnutella-like breadth first search (BFS) that floods the overlay network with a search request [92]. The number of messages exchanged to perform BFS is high, but it performs a thorough search of the peer's neighborhood and is very likely to find a similar peer if one exists. If the high traffic overhead caused by BFS is an issue, then the search can be performed by random walks [20], which is cheaper but will not search a peer's neighborhood as thoroughly and is therefore less likely to find a similar peer. Another alternative is biased random walk proposed by Chawathe et al. [34], which performs a more exhaustive search than random walk. In this approach, each peer maintains a directory of resources available on its neighbors, and the random walk is biased towards peers with a high degree because they are assumed to have have more information about resources on the overlay network.

A fourth possibility is to implement the search operation as a gossip-based search. In a gossip-based search, a peer uses a list of peers, populated with information gleaned from messages routed through it on the overlay network and periodic exchange of information about other peers on the overlay network with neighbors, to find a similar peer [75]. The performance of a gossip-based search is stochastic, and the chance of finding a suitable peer will depend upon the diversity (in terms of information about other peers on the overlay network) of the messages that are routed through the peer looking for a similar peer. Other search algorithms are of course also possible, and the most suitable approach depends on application requirements.

4.1.2 Time Delay

The time delay between successive estimations of the satisfaction state is another important factor that affects the cost of utilizing the clustering algorithms. A satisfied peer need not estimate its satisfaction state repeatedly. Similarly, a peer that is unable to successfully execute its topology adaptation steps need not estimate its satisfaction state repeatedly. They can instead use an exponentially increasing time delay between successive estimation of the satisfaction state.

4.1.3 Analysis

This section explains why only a specific case of the clustering algorithms can be mathematically analyzed using the existing techniques for analyzing Schelling's model. The specific case that can be analyzed is when $PNSP_{desired}$ is 100, which unfortunately will not be relevant in a real life implementation because, as will be shown in section 4.2, it generates a disconnected overlay network topology. Zhang [131, 132] has mathematically analyzed variations of Schelling's model. In his analysis, the models are treated as a game played between two players chosen at random from the turtles on the grid. A potential function is then obtained for the game. The potential function can be used to analyze the overall behavior of the game instead of concentrating on the activity of an individual turtle. Zhang defines the potential function as follows:

> Let Γ be an n-person game with finite strategy sets $Z_1, Z_2, \ldots Z_n$. The payoff function of player i is $u^i : Z \rightarrow R$ where $Z = Z_1 \times Z_2, \ldots Z_n$ is the set of strategy profiles. A game Γ is a *potential game* if there exists a function $\lambda : Z \rightarrow R$ such that for every i and for every $z_{-i} \epsilon Z_{-i}, u^i(x, z_{-i}) - u^i(y, z_{-i}) = \lambda(x, z_{-i}) - \lambda(y, z_{-i})$, for every $x, y \epsilon Z_i$. λ is called the *potential function* of this game [131, p.151].

The change in a player's utility is equal to the change in the potential function. Let S be the set of all the states that maximize the potential ρ:

$$S = \{x | \rho(x) \geq \rho(y), \forall y, x \epsilon X\}$$

where X is the set of all the possible states in which the system can exist. It can be shown that:

> S is stochastically stable under the perturbed process. That is, in the long run, we will see a state in S almost all the time [131, p. 153].

It is not possible to obtain a potential function for the clustering algorithms, if $PNSP_{desired}$ is not 100. In the clustering algorithms, the utility function for the players is $min\{PNSP_{actual}, PNSP_{desired}\}$. In the clustering algorithms, a peer can choose from one of the following strategies: add a similar neighbor, drop a dissimilar neighbor or do both. The change in the utility of a peer when it chooses one of these strategies depends upon its state (satisfied or unsatisfied), when $PNSP_{desired}$ is less than 100. When an unsatisfied peer drops a dissimilar neighbor or adds a similar neighbor or does both then there is an increase in the utility of the unsatisfied peer. However, when a satisfied peer loses a dissimilar neighbor or gains a similar neighbor or both, then there is no change in its utility. It is not possible to obtain a potential function when $PNSP_{desired}$ is less than 100 because the change in the utility of a peer depends on its state.

When $PNSP_{desired}$ is 100, then the change in the utility of a peer is not dependent on its state. A satisfied peer does not have the choice of dropping a dissimilar neighbor because a connection with a dissimilar peer can not exist if it is satisfied. When $PNSP_{desired}$ is 100, the addition of a similar neighbor will always increase the utility. Let US be the utility sum of all the peers. When a peer adds a similar peer or drops a dissimilar peer, then the change in its utility is $\frac{\Delta US}{2}$ because the utility of the peer and the peer being dropped or added change by a similar amount. $\frac{US}{2}$ is the potential function for this system as the change in the utility of a peer on choosing a strategy is $\frac{\Delta US}{2}$. Theoretically, the maximum possible value of the potential function is $50 \times$ number of peers. This will happen when there are no connections between dissimilar peers and the similar peers are connected together in clusters. Dropping a dissimilar peer or adding a similar peer maximizes the potential function, increasing the stability of the system and bringing it close to a clustered state.

4.2 Simulations

This section presents the simulation results of applying the clustering algorithms to a series of overlay networks. The simulations were done using the simulator described in section 3.4. Two types of simulations were performed: *static simulations* which use an overlay network that has no flux of peers and *dynamic simulations* in which new peers are added to the overlay network at regular intervals. The static simulations were performed because it is easy to simulate and analyze the results when there is no flux of peers on the overlay network. The static simulations also demonstrate the ability of the clustering algorithms to create a desired topology in a special scenario when there is no flux of peers on the overlay network. The dynamic simulations are used to show that the clustering algorithms can maintain the desired topology even when there is a continuous arrival of peers on the overlay network.

The rest of this section is structured as follows: we first define the metrics that are used to study the result of applying the clustering algorithms on the overlay networks. This is followed by a description of the experimental setup. Finally we present and analyze the results of the static and dynamic simulations.

4.2.1 Metrics

The first metric presented in this section is used to measure the relative decrease in the number of clusters caused by the clustering algorithms. The next two metrics are used to examine the clusters created by the clustering algorithms. Then we present a metric that is used to measure the time (measured in simulator iterations) taken by the clustering algorithms to converge. Finally, we present the metric that is used to measure the messaging cost of the clustering algorithms.

Relative Decrease in Number of Clusters ($RDNC$)

The Number of Clusters (NC) metric was formally defined in section 2.1.7. The metric Relative Decrease in Number of Clusters ($RDNC$) is used to measure the decrease in the number of clusters because of applying the clustering algorithm. Let NC_{orig} be the number of clusters in the original overlay network. Let NC_{curr} be the number of clusters in the overlay network after applying the clustering algorithms. $RDNC$ can then be expressed as:

$$RDNC = \frac{(NC_{orig} - NC_{curr}) \times 100.00}{NC_{orig}} \tag{4.1}$$

A high value of $RDNC$ indicates that a large number of groups of similar peers on the overlay network are merged to form a small number of large sized groups. The desired value of $RDNC$ will vary with application. An application will typically desire to have a high value of $RDNC$, but the application might have to do a trade-off between $RDNC$ and Coverage described below.

Coverage (C)

Coverage (C) gives an indication of the cost of clustering. It is the cost of clustering in terms of the number of edges used to maintain the clustering. Let m be the number of intra-cluster edges and $\overline{m}$ be the number of inter-cluster edges. Then C [31] is defined as:

$$C = \frac{\overline{m}}{m + \overline{m}} \tag{4.2}$$

C can have a minimum value of 0, when none of the clusters are connected to each other. The minimum value of C is undesirable because it will lead to an unacceptable situation of a disconnected overlay network topology.

The desired value of C will vary with the application. For example, an application that desires to cluster peers with a similar property, such as bandwidth to improve the performance of messaging on the overlay network, will desire to have a low value of C. As discussed in section 2.1.7 in a file sharing application, peers sharing similar files may be clustered together, which can be used to improve search performance because search requests can be routed to an appropriate cluster and then a deep search can be performed within the cluster. Such an application will desire to have a high value of C so that the inter-cluster edges could be used to quickly route search requests to the appropriate cluster.

Percentage of Peers in Top Clusters ($PPTC$)

The metric Percentage of Peers in Top Clusters ($PPTC$) is used to examine the distribution of peers in the clusters of different types of peers. As defined in section 2.1.7, let $P = \{p_1, p_2 \ldots p_n\}$ be the set that enumerates the types of peers on the overlay network and $CC_{p_i} = \{CC_1, CC_2 \ldots CC_n\}$ be the set of clusters with peers of type p_i. Let $max(CC_{p_i})$ be a function that returns the cluster CC_i with the maximum number of peers from the set CC_{p_i}. $PPTC$ is the count of the peers in the clusters with the maximum size for each type of peer.

$$PPTC = \sum_{p_i \in P} |max(CC_{p_i})| \tag{4.3}$$

A high $PPTC$ implies that most of the peers are a part of the largest clusters of each category. The clustering algorithms merge the existing clusters creating clusters with a large size and a high value of $PPTC$ for the overlay network. $PPTC$ is used to examine the distribution of peers in the clusters of different types of peers, and the value of this metric will vary with application.

Time to Converge (TC)

A topology adaptation algorithm converges when all the peers are satisfied with their neighbors. The Time to Converge (TC) for a topology adaptation algorithm, is the total number of simulator iterations in which all the peers are satisfied with their neighbors. This metric is a measure of the time taken by a topology adaptation algorithm to converge and is used in both the static and the dynamic simulations. A good clustering algorithm will have a low TC value.

Messaging Cost (MC)

As discussed in section 3.3.2, there is a common messaging cost associated with utilizing the topology adaptation algorithms based on PESTO. The common messaging cost is used by the peers to determine the status of their neighbors. The status information is used to calculate the satisfaction state. For the common messaging cost, the clustering algorithms may either use a proactive approach if the application can incur the large extra messaging cost or a reactive approach in which a small number of messages are exchanged by a peer to notify its neighbors about a change in its state. This common messaging cost will be incurred throughout the life of the overlay network and is therefore not very useful in evaluating the algorithms.

The number of messages exchanged to perform the search operation for similar peers is the other messaging cost in utilizing the SelflessClustering algorithm. The number of messages exchanged to reconnect the disconnected peers to the topology is the extra messaging cost for the SelfishClustering algorithm. The Messaging Cost (MC) metric measures the number of messages exchanged, other than the common messaging cost to determine the status of neighbors, by a PESTO based topology adaptation algorithm. In the simulations MC measures the number of messages exchanged till all the peers are satisfied with their neighbors or a pre-defined number of simulator iterations is reached. A small value of MC is desired for the clustering algorithms.

4.2.2 Static Simulations

Simulation Setup

In the static simulations, the effects of applying the clustering algorithms on power-law topologies is studied. The clustering algorithms are applied on the topologies with $PNSP_{desired}$ varying from 10 to 100 in increments of 1, and the effect of the algorithms is studied using the metrics described in section 4.2.1. The topologies were

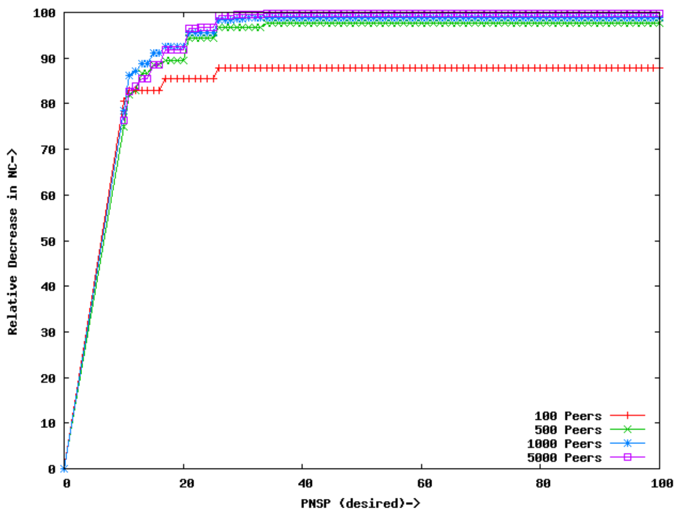


Fig. 4.2: A plot of relative decrease in the number of clusters after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

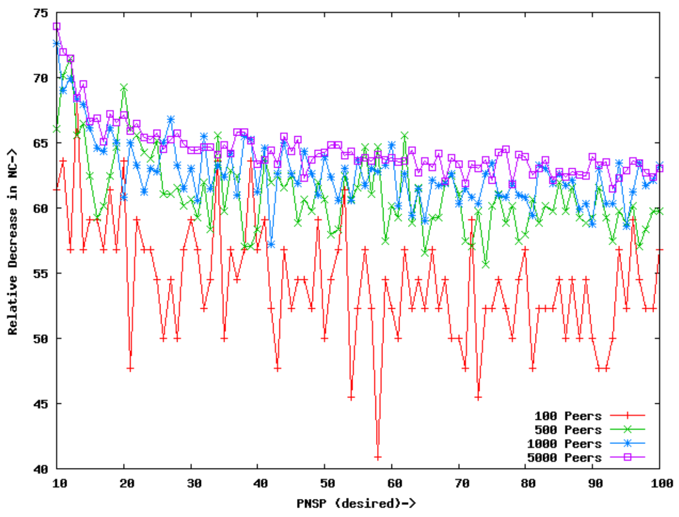


Fig. 4.3: A plot of relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

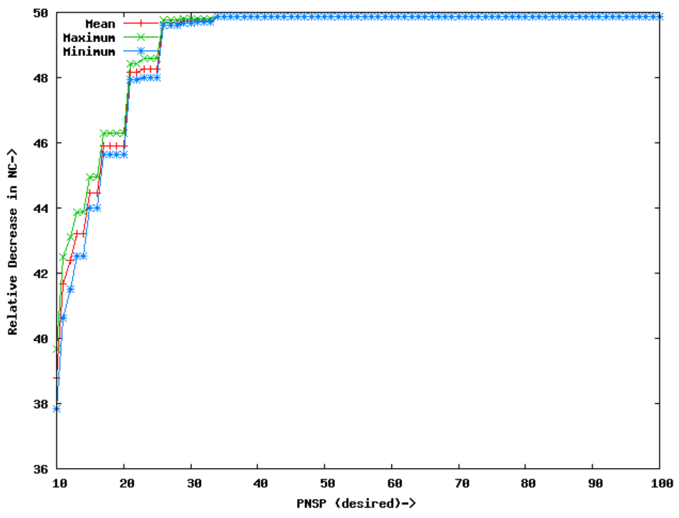


Fig. 4.4: A plot of the maximum, mean and minimum values of the relative decrease in the number of clusters after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four different power-law topologies with 5,000 peers.

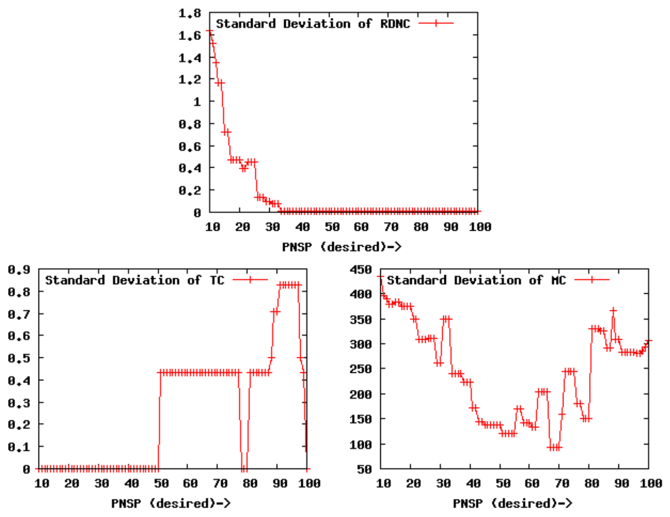


Fig. 4.5: A plot of standard deviation of RDNC, TC and MC against $PNSP_{desired}$ after applying SelflessClustering algorithm on four different power-law topologies with 5,000 peers.

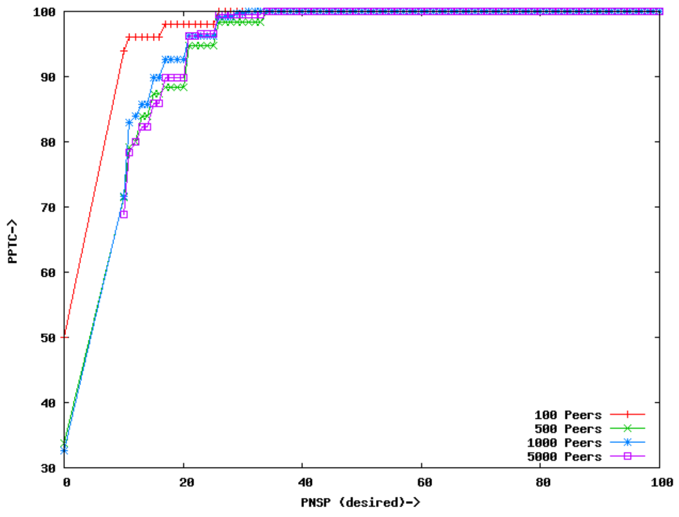


Fig. 4.6: A plot of PPTC after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

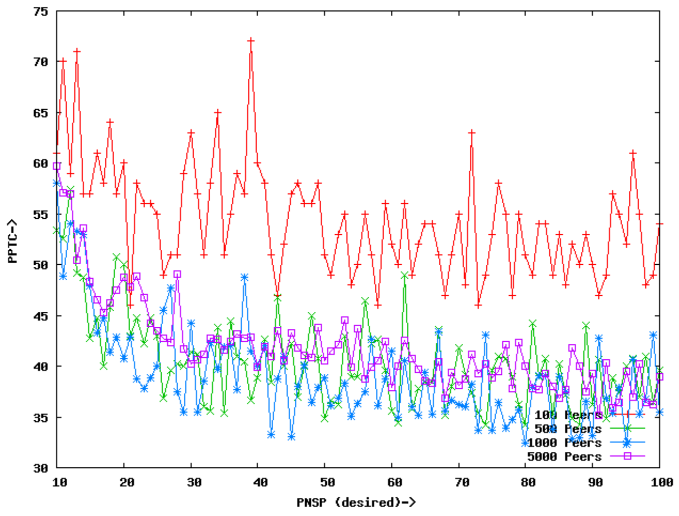


Fig. 4.7: A plot of PPTC after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

generated using the simulator described in section 3.4. A study of initial Gnutella topologies by Ripeanu [92] suggests that the degree distribution followed a power law (see section 2.1.5 for power-law topology). Simulations have been done using four different power-law topologies of 100, 500, 1,000 and 5,000 peers created by using the algorithm discussed in section 2.1.5. In these topologies, each peer can have a maximum of 10 neighbors and a new peer is connected to 3 peers chosen randomly with a bias towards peers with a high degree. It is ensured that the initial topologies are connected before applying the clustering algorithms on them.

In the simulations, the overlay networks have equal proportions (selected randomly) of five different types of peers. Each type of peer has a unique property that distinguishes it from other types of peers. In a real-life scenario, this unique property could be the geographical location of the peer or the content type that is being shared.

In the SelflessClustering Algorithm, the unsatisfied peers search for peers similar to themselves on the overlay network. In these simulations, a variation of GIA's [34] search algorithm, described in section 2.2.4 is used. In GIA, each peer maintains a directory of resources available on its neighbors, and searches are performed using random walk. The random walk is biased towards peers with a higher degree that are in a better position to handle a search request because their directory contains information about a larger number of peers on the network than a peer with a low degree. In the simulations, each peer maintains a directory that contains information about its neighbors and the neighbors of its neighbor instead of resources. This directory is used to handle search requests.

In the clustering algorithms, a peer has to periodically check the status of its neighbors to ensure that it is satisfied. The information about the neighbors of neighbors can be piggy-backed on these requests. This is an advantage when compared to gossip-based search, because gossip-based search will use separate messages to interact with randomly chosen peers on the overlay network for populating a similar directory of information about peers. In section 2.2.9, we observed that temporally inaccurate data in directories is a problem. In the search approach taken here, the data in the directories can be periodically validated at no extra message cost when the status of the neighbors is obtained for topology adaptation.

RDNC and PPTC: SelflessClustering

Figure 4.2 plots $RDNC$ against $PNSP_{desired}$ after applying the SelflessClustering Algorithm on power-law topologies with 100, 500, 1,000 and 5,000 peers. Figure 4.6 plots $PPTC$ against $PNSP_{desired}$ after applying the SelflessClustering Algorithm on the same topologies. As figure 4.2 shows, $RDNC$ increases with a rise in $PNSP_{desired}$. The increase in $RDNC$ becomes constant after a certain $PNSP_{desired}$ value that varies with the topology and is under 35 in the simulations. Figure 4.6 shows that, at this point, the value of $PPTC$ is 100 which means that all the peers of a type are part of the same cluster.

Figure 4.2 shows that the $RDNC$ is higher for topologies with a large number of peers than topologies with a small number of peers. This is because the topologies with a large number of peers have large number of clusters to start with, and for all the topologies the SelflessClustering Algorithm reduces the number of clusters to 5, the total number of types of peers on the overlay network. This happens consistently after $PNSP_{desired}$ reaches 35. Figure 4.4 plots the average, minimum and maximum $RDNC$ against the $PNSP_{desired}$ after applying the SelflessClustering Algorithm on four different power-law topologies with 5,000 peers. The figure shows that there is a small difference between the maximum, minimum and mean values. Figure 4.5 shows the standard deviation of $RDNC$ against $PNSP_{desired}$ for the four different topologies. The standard deviation is approximately 0, when $PNSP_{desired}$ is above 32 which suggests that the SelflessClustering algorithm produces similar $RDNC$ in all the four topologies.

RDNC and PPTC: SelfishClustering

Figure 4.3 plots $RDNC$ against the $PNSP_{desired}$ after applying the SelfishClustering Algorithm on power-law topologies with 100, 500, 1,000 and 5,000 peers. Figure 4.7 plots $PPTC$ against $PNSP_{desired}$ after applying the SelfishClustering Algorithm on the same topologies. The decrease in the number of clusters after applying the SelfishClustering Algorithm varies between 40% and 75% and $PPTC$ varies between 30% and 75%. Just like the SelflessClustering Algorithm, in the SelfishClustering Algorithm also overlay networks with a higher number of peers display a higher value of $RDNC$ because they have a large number of small clusters to start with, which are merged together by the SelflessClustering Algorithm to create a small number of large clusters.

In the SelfishClustering Algorithm peers drop dissimilar neighbors without checking if the dissimilar peer has other neighbors. This selfish dropping of peers results in peers being disconnected from the overlay network. The decrease in clustering and $PPTC$ for the SelfishClustering Algorithm is erratic when compared to the SelflessClustering Algorithm. This is because the disconnected peers are reconnected to three randomly chosen peers on the overlay network.

The erratic variations in the $RDNC$ and $PPTC$ results makes it difficult to estimate the trend in variation of $RDNC$ and $PPTC$ with $PNSP_{desired}$. There are four commonly used techniques that can be used in such a scenario to estimate the trend [114, pg. 437]. They are: the freehand method, the moving-average method, the method of semiaverages and the least-square method. These methods can be used to draw a trend line or a trend curve that represents the simulation results. In the freehand method visual inspection is used to draw the trend line or the trend curve. The method suffers from the disadvantage of individual judgment and is not very interesting. In the moving-average method the simulation results are averaged out to estimate the trend. In this method n consecutive data values are taken at a time starting from the first data value in the simulation results. The average of the n consecutive data

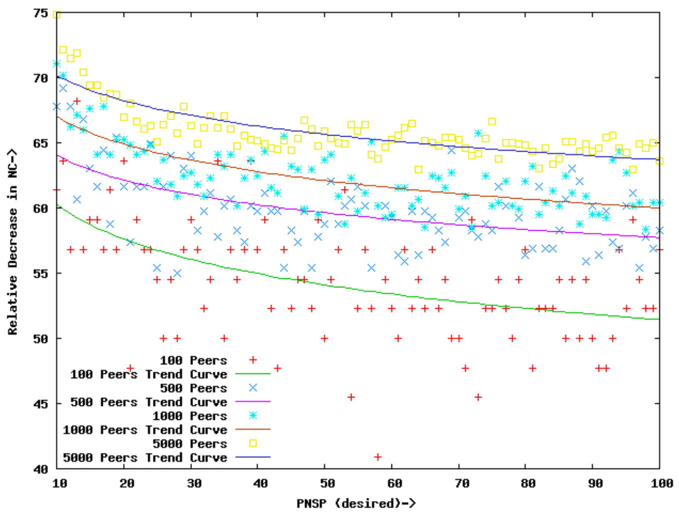


Fig. 4.8: Trend curves that show the relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

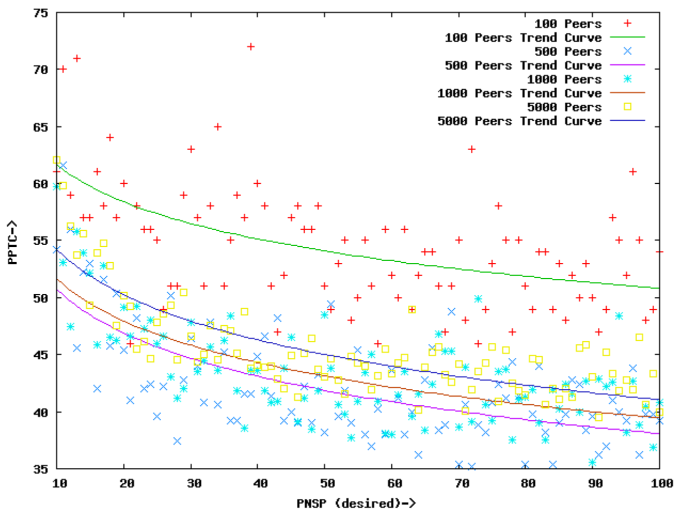


Fig. 4.9: Trend curves that show the PPTC after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

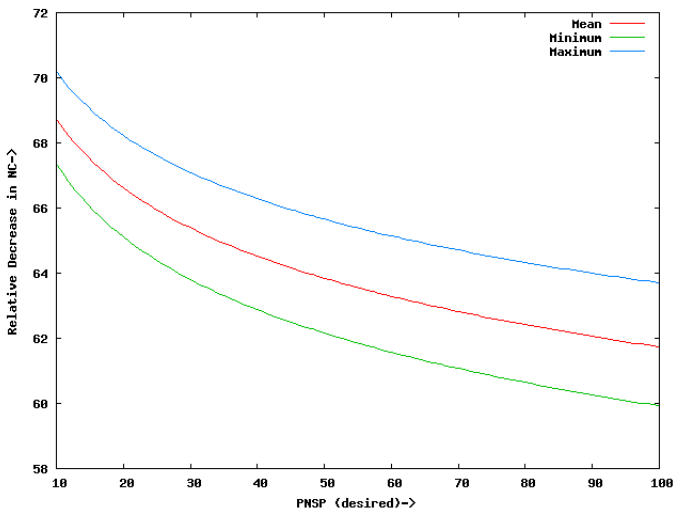


Fig. 4.10: Trend curves that show the maximum, mean and minimum values of the relative decrease in the number of clusters after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four different power-law topologies with 5,000 peers.

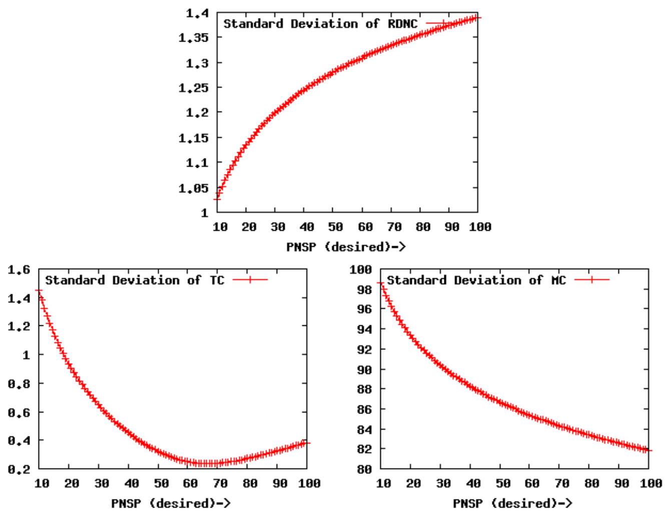


Fig. 4.11: A plot of standard deviation of RDNC, TC and MC against $PNSP_{desired}$ after applying SelfishClustering algorithm on four different power-law topologies with 5,000 peers.

values becomes the new simulation result that is used to plot the trend. This methods suffers from the disadvantage that the data items at the start and the end are lost. Further, extreme data values may adversely affect the estimated trend and so this method has not been used. In the semiaverages method the data is divided into two parts. the trend line is drawn through the average of the data in both the parts. This method is simple to implement, however the trend line depends on the choice of the size of the parts. In the least-square method a trend line or a curve is drawn through the simulation results that minimizes the sum of the square of the distance between the simulation results and the trend line or the curve [114, pg. 283]. In this thesis least-square method is used for predicting trend because it minimizes the error measured using distance between the data points and trend line or curve. The least-square method implementation in Gnuplot [3] is used for predicting trends.

A visual inspection of figure 4.3 suggests that $RDNC$ first drops sharply and then decreases normally with an increase in $PNSP_{desired}$. A similar pattern is observed in the $PPTC$ against $PNSP_{desired}$ graph in figure 4.7. Curves of the form $A*ln(B*x)+C$ where A, B and C are constants that can be obtained using least-square method and $ln(x)$ is the natural logarithm of x, show a similar pattern. The least-square method is used to fit a trend curve of the form $A*ln(B*x) + C$ in the results to study the variation of $RDNC$ and $PPTC$ with a change in $PNSP_{desired}$ (see figure 4.3 and 4.7 for raw data). Figure 4.8 plots the trend curves that show the variation of $RDNC$ with $PNSP_{desired}$ after applying the SelfishClustering Algorithm on power-law topologies with 100, 500, 1,000 and 5,000 peers. Figure 4.9 plots the trend curves that show the variation of $PPTC$ with $PNSP_{desired}$ after applying the SelfishClustering Algorithm on the same topologies. The plots show that the $RDNC$ and $PPTC$ is highest when $PNSP_{desired}$ is 10 an decreases with an increase in $PNSP_{desired}$. As described in section 2.1.5 in a power-law network some vertices have a very high degree while the rest of the peers have a low degree. The peers with a low degree have a high chance

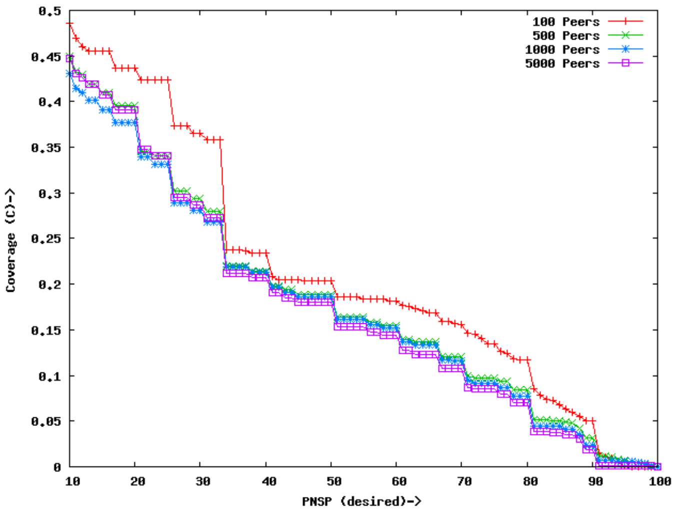


Fig. 4.12: A plot of the cluster Coverage (C) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

of being disconnected from the topology as $PNSP_{desired}$ increases because of their low degree. The random reconnections of a large number of disconnected peers decreases $RDNC$ and $PPTC$.

The static simulations generate results for variation of $RDNC$ with a change in $PNSP_{desired}$ after applying the SelfishClustering Algorithm on four different power-law topologies with 5,000 peers. These results also suffer from erratic variations because of random reconnections. The least-square method is used to fit a trend curve of the form $A * ln(B * x) + C$ to study the variation of $RDNC$ with a change in $PNSP_{desired}$ on each topology. Figure 4.10 plots the minimum, maximum and mean values of the $RDNC$ against $PNSP_{desired}$ for the four topologies. The $RDNC$ decreases with an increase in $PNSP_{desired}$. The figure shows that there is a small difference between the minimum, maximum and mean values. Figure 4.11 plots the standard deviation in the $RDNC$ against $PNSP_{desired}$.

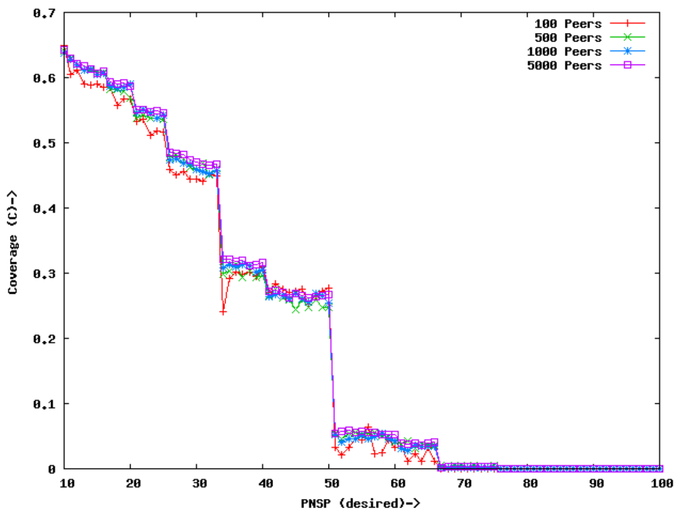


Fig. 4.13: A plot of the cluster Coverage (C) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers.

Cluster Coverage: SelflessClustering

Figure 4.12 shows a plot of Cluster Coverage (C) against $PNSP_{desired}$ for the Selfless-Clustering Algorithms when applied on overlay networks with 100, 500, 1,000 and 5,000 peers. The cluster coverage decreases with an increase in $PNSP_{desired}$ and reaches 0 when $PNSP_{desired}$ reaches 90%. As mentioned in section 4.2.2, the number of clusters reaches 5, one for each type of peer on the networks, when $PNSP_{desired}$ is 35 for the different topologies. Applying the clustering algorithms beyond that reduces the inter-cluster edges and increases the intra-cluster edges.

Cluster Coverage: SelfishClustering

Figure 4.13 shows a plot of Cluster Coverage C against $PNSP_{desired}$ for the Selfish-Clustering Algorithm when applied on overlay networks with 100, 500, 1,000 and 5,000 peers. The cluster coverage decreases with an increase in $PNSP_{desired}$ and reaches 0

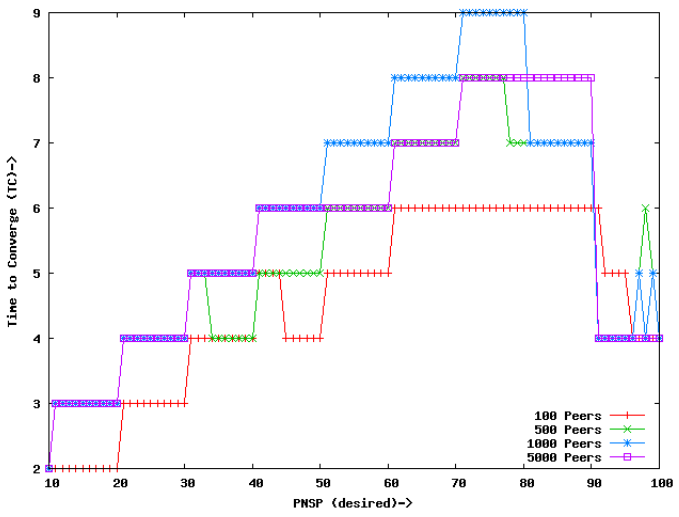


Fig. 4.14: A plot of the Time to Converge (TC) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

when $PNSP_{desired}$ is between 60% and 70%. When cluster coverage reaches 0, the topology is disconnected and this might not be good for applications.

Time to Converge: SelflessClustering

Figure 4.14 shows a plot of TC against $PNSP_{desired}$ for the SelflessClustering Algorithm when applied on overlay networks with 100, 500, 1,000 and 5,000 peers. The TC is between 2 and 10. The graphs show that TC varies slightly with the number of peers (N) on the overlay network.

TC changes in steps with a rise in $PNSP_{desired}$ by 10. This is because in simulations a peer can have a maximum of 10 neighbors, and a rise only by 10 in $PNSP_{desired}$ causes a peer's satisfaction criterion to change. The TC increases with an increase in $PNSP_{desired}$ and then decreases when $PNSP_{desired}$ reaches 70. Figure 4.17 shows a plot of the maximum, mean and minimum Time to Converge (TC) against $PNSP_{desired}$ for the SelflessClustering Algorithm when applied on four different power-law topologies

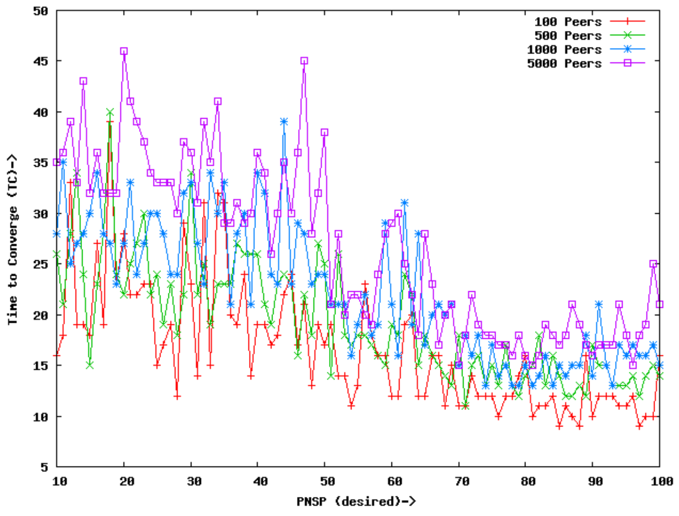


Fig. 4.15: A plot of the Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers.

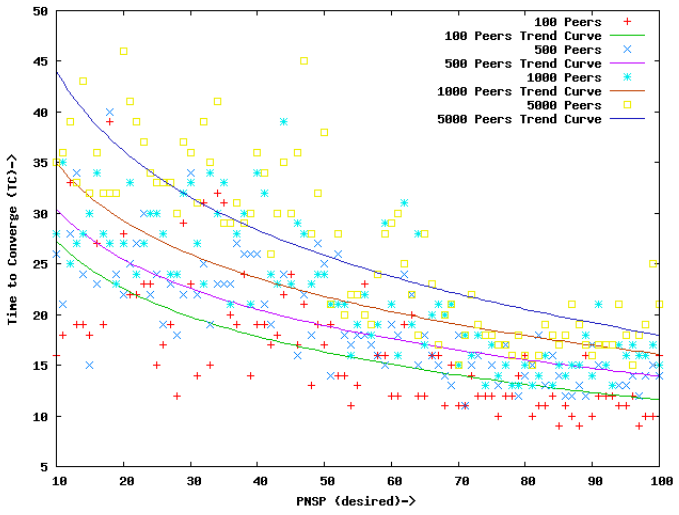


Fig. 4.16: Trend curves that show the Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 500, 100, 1,000 and 5,000 peers.

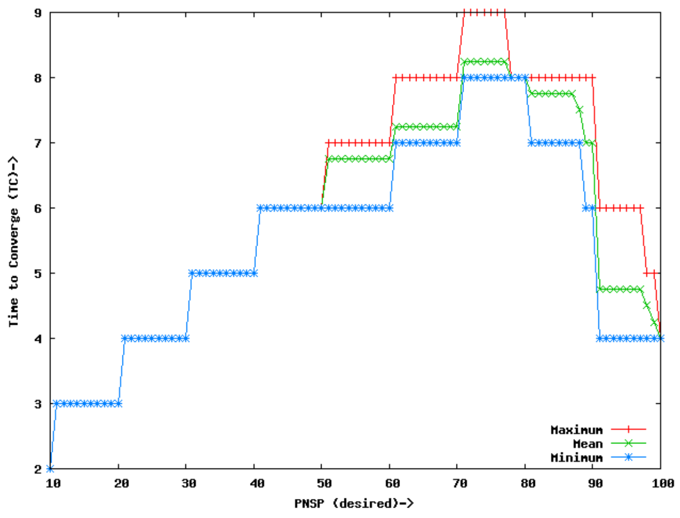


Fig. 4.17: A plot of the maximum, mean and minimum Time to Converge (TC) after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers.

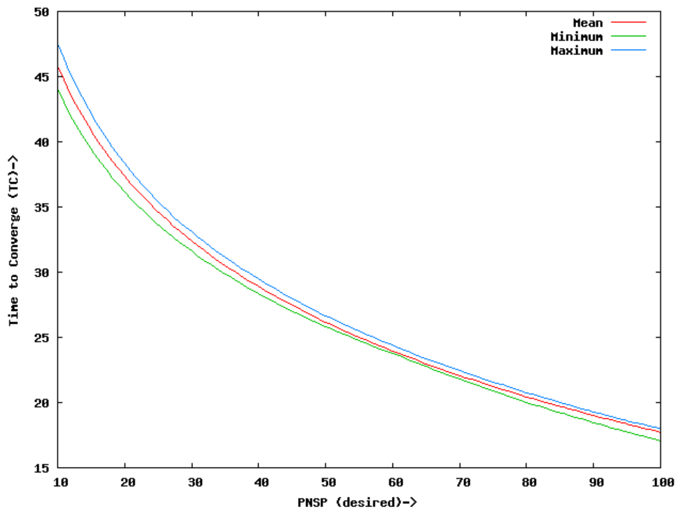


Fig. 4.18: The maximum, mean and minimum Time to Converge (TC) after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers.

with 5,000 peers. The TC is between 2 and 10. The graphs show that that there is a small difference between the maximum, mean and minimum values. Figure 4.5 shows the standard deviation of TC against $PNSP_{desired}$ for the four different topologies.

Time to Converge: SelfishClustering

Figure 4.15 shows a plot of Time to Converge (TC) against $PNSP_{desired}$ and figure 4.16 show the Time to Converge (TC) trend curves against $PNSP_{desired}$ for the SelfishClustering Algorithm when applied on overlay networks with 100, 500, 1,000 and 5,000 peers. Just like the $RDNC$ values for the SelfishClustering algorithm, the least-square method is used to fit trend curves of the form $A * ln(B * x) + C$ on the raw TC data to obtain the trend curves. For the SelfishClustering Algorithm, the TC is between 5 and 50. Just like the SelflessClustering Algorithm, the TC varies slightly with the number of peers (N) on the overlay network.

Like the SelflessClustering Algorithm, the TC is low for high $PNSP_{desired}$. TC is low for high $PNSP_{desired}$ because when $PNSP_{desired}$ is high, the topology adaptation steps of a larger number of peers contribute towards satisfying dissatisfied peers. When $PNSP_{desired}$ is low, a smaller number of peers are dissatisfied that execute topology adaptation steps to satisfy themselves. Figure 4.18 shows the maximum, mean and minimum values of the trend curves for Time to Converge (TC) against $PNSP_{desired}$ for the SelfishClustering Algorithm when applied on four different power-law networks with 5,000 peers. There is a small difference between the maximum, mean and minimum values. Figure 4.11 plots the standard deviation in the TC against $PNSP_{desired}$.

Messaging Cost: SelflessClustering

In the SelflessClustering Algorithm, apart from the mandatory messaging cost for the clustering algorithms, the peers generate messages to search the overlay network for a peer with a desired property if they are not satisfied with their neighbors. In every

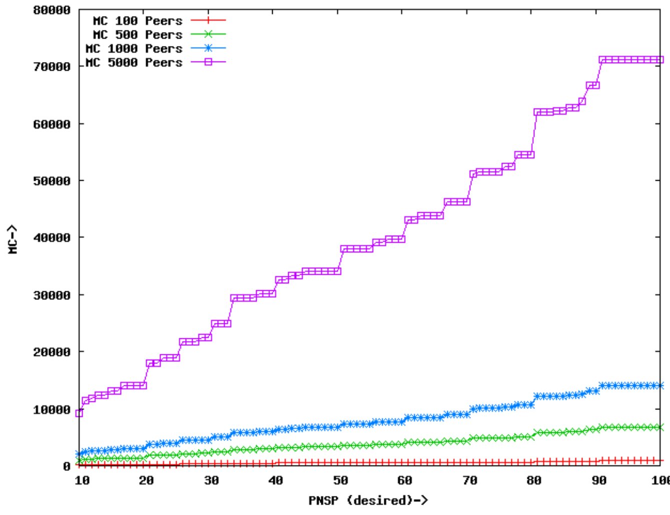


Fig. 4.19: A plot of the number of messages exchanged to perform search after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

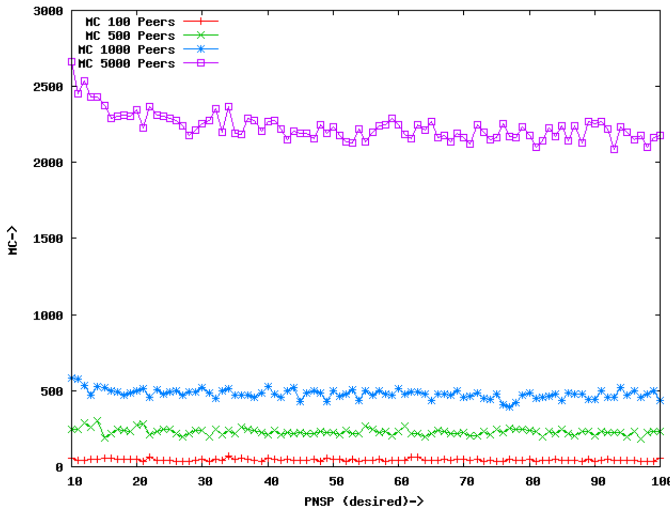


Fig. 4.20: A plot of the number of messages exchanged to reconnect disconnected peers after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500, 1,000 and 5,000 peers.

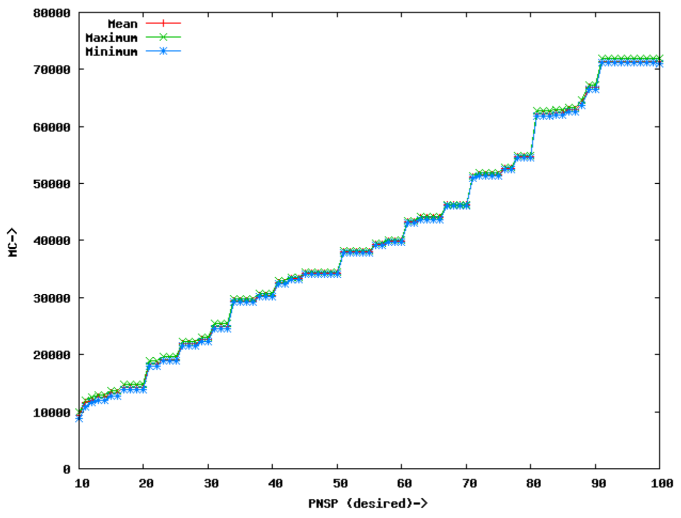


Fig. 4.21: A plot of the maximum, mean and minimum number of messages exchanged to perform search after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers.

iteration, the messages exchanged to perform the search will be proportional to the number of unsatisfied peers $\times$ $O(search)$. In the simulations, GIA's search algorithm [34] described in section 2.2.4 is used to implement the search operation. The messages exchanged to maintain the directory on each peer can be coupled with the mandatory messages exchanged to determine the status of a neighbor and are therefore not included in the search cost in this algorithm. The search operation is of $O(h)$ where h the horizon of search and is set to 6 in the simulations.

Figure 4.19 shows a plot of MC against $PNSP_{desired}$ for the SelflessClustering Algorithms when applied on power-law topologies with 100, 500, 1,000 and 5,000 peers. The number of messages exchanged to perform search is approximately 70,000 when $PNSP_{desired}$ is 100. As $PNSP_{desired}$ increases, the number of messages exchanged to perform search increases because the number of peers that are dissatisfied with their neighbors increases. The number of messages exchanged to perform search increases with increase in number of peers in the topology because the number of dissatisfied

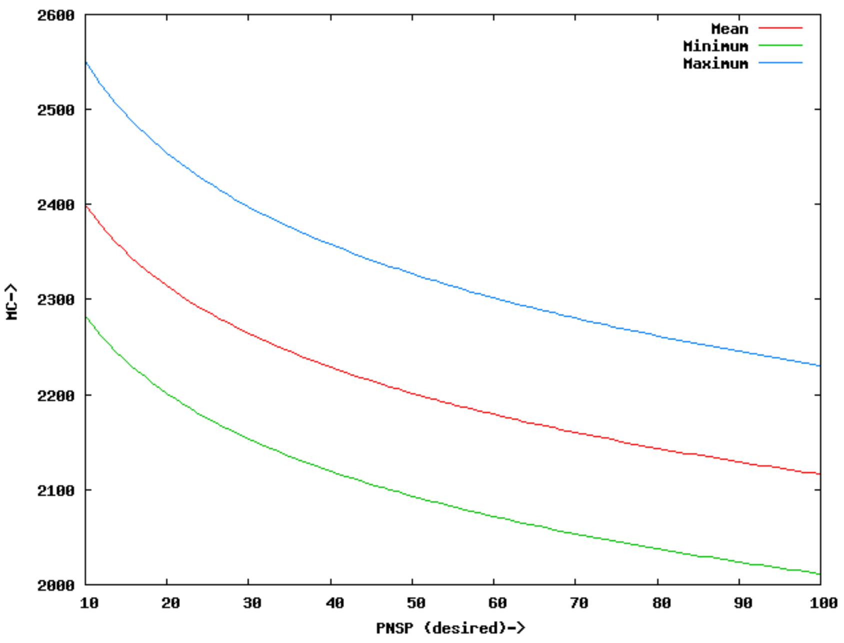


Fig. 4.22: The maximum, mean and minimum values of the trend curves for the messages exchanged to reconnect disconnected peers after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on four power-law topologies with 5,000 peers.

peers that need to perform search increases.

Figure 4.21 shows a plot of the maximum, average and mean of the messages exchanged to perform search against $PNSP_{desired}$ for the SelflessClustering Algorithm when applied on four different power-law topologies. The MC varies between $10,000$ and $70,000$. Figure 4.5 shows the standard deviation of TC against $PNSP_{desired}$ for the four different topologies.

Messaging Cost: SelfishClustering

In the SelfishClustering Algorithm, the peers do not search for other peers on the overlay network. In the SelfishClustering Algorithm there is an overhead of messages exchanged to reconnect the peers that are disconnected from the overlay network because of selfish dropping. However, this overhead is far less than the overhead of searching for a similar peer on the network. Figure 4.20 shows a plot of MC against $PNSP_{desired}$ for the SelfishClustering Algorithms when applied to power-law topologies with 100, 500, 1,000 and 5,000 peers. The total number of messages exchanged for reconnections is within 3,000 and constitutes only a fraction of the messages exchanged to perform the search for the SelflessClustering Algorithm. Figure 4.22 shows the maximum, mean and the minimum of the trend curves (of the form $A * ln(B * x) + C$ obtained by using least-square method on the raw data) for MC against $PNSP_{desired}$ for the SelfishClustering Algorithm when applied on four different power-law topologies with 5,000 peers. The MC values fall between $2,000$ and $2,600$. Figure 4.11 plots the standard deviation in the MC against $PNSP_{desired}$.

Disconnected Topology

A critical value of $PNSP_{desired}$ (called $PNSP_{critical}$) was observed in the simulation results above which the overlay network's topology was disconnected. The value of $PNSP_{critical}$ is different for different networks. We have not been able to find any correlation between the network and the $PNSP_{critical}$ value. For the SelflessClustering Algorithm, the $PNSP_{critical}$ value is close the value of $PNSP_{desired}$ when C reaches 0 and there are no inter-cluster edges. A typical value of $PNSP_{critical}$ is 90 for the SelflessClustering Algorithm and 30 for the SelfishClustering Algorithm.

Discussion

For both the clustering algorithms a low value of $PNSP_{desired}$ (e.g., 10 to 20) is sufficient to achieve a substantial decrease in the number of clusters. The Cluster Coverage (C) decreases with an increase in $PNSP_{desired}$. As mentioned in section 4.2.1, some applications might desire a high value of C whereas others might desire a low value of C. For the former applications a low value of $PNSP_{desired}$ may be used whereas for the latter a high value of $PNSP_{desired}$ may be used.

MC and $RDNC$ are higher for the SelflessClustering Algorithms when compared to the SelfishClustering Algorithm for the same $PNSP_{desired}$. We suggest that an application with a long life incur the high MC of SelflessClustering Algorithm to achieve a high $RDNC$, and an applications with a short life use the SelfishClustering Algorithm to achieve a low $RDNC$ at a low MC.

4.2.3 Dynamic Simulations

This section presents the results of the dynamic simulations in which there is a continuous arrival of peers on the overlay network. The overlay network topology has 100 peers initially. Five peers are added to the network in every simulator iteration till there are 5,000 peers on the network. The simulations go on till all the peers are satisfied or 1,500 simulation iterations are reached. The simulations have been done using $PNSP_{desired}$ value of 10 because it is far below the $PNSP_{critical}$ value for both SelflessClustering and SelfishClustering Algorithms. Just like the static simulations, the overlay network has equal proportions of five different peers of each type and a bi-

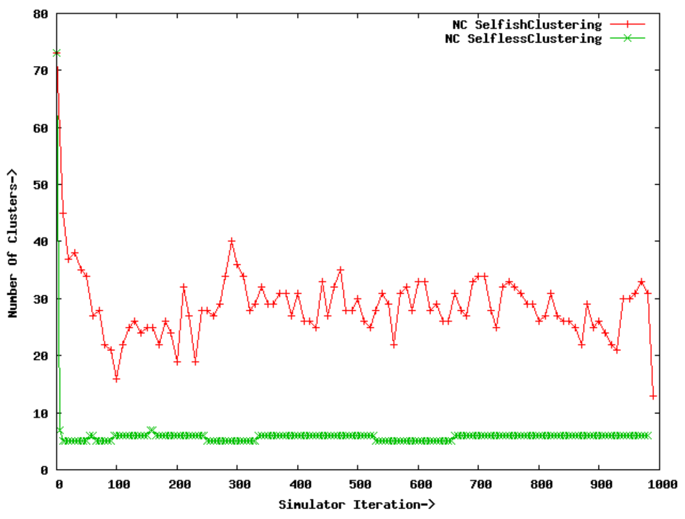
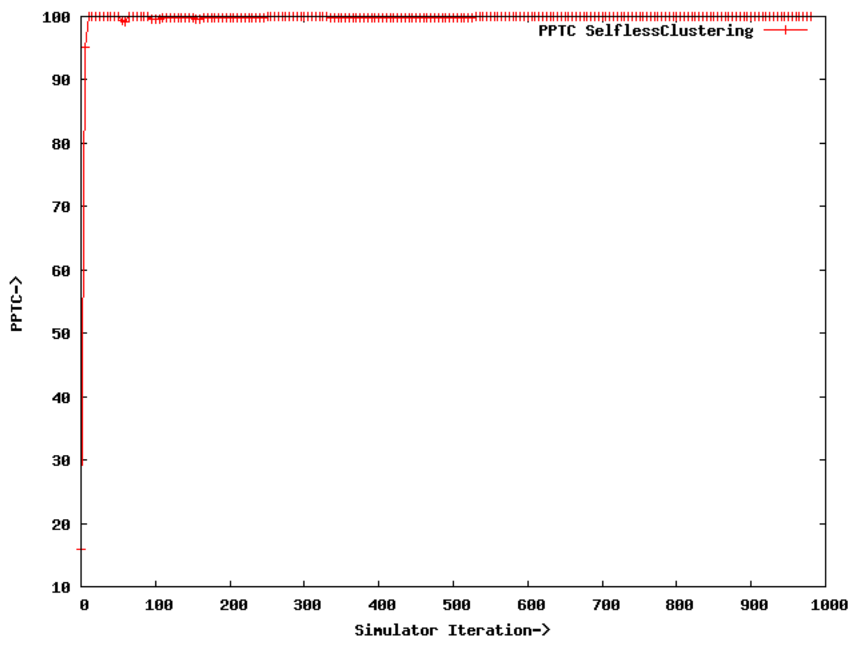


Fig. 4.23: A plot of number of clusters (NC) against simulator iterations when the SelflessClustering and SelfishClustering Algorithm are applied.

ased random walk algorithm with replication has been used to search for similar peers in the SelflessClustering Algorithm.

Results

For the SelflessClustering Algorithm the simulations stop in 980 iterations when the number of peers on the overlay network reaches 5,000 and for the SelfishClustering Algorithm the simulations stop in 990 iterations when the number of peers on the overlay network reaches 5,000 and all the peers are satisfied. Figure 4.23 shows a plot of the number of clusters on the overlay network against time (simulator iteration) when the SelflessClustering and SelfishClustering algorithms are applied on the overlay network of 100 peers described above. The number of clusters are calculated every fifth simulator iteration. The SelflessClustering Algorithm maintains the value of the number of clusters between 5 and 6 even when there is an arrival of peers on the network. This is higher than the number of clusters when using the SelfishClustering



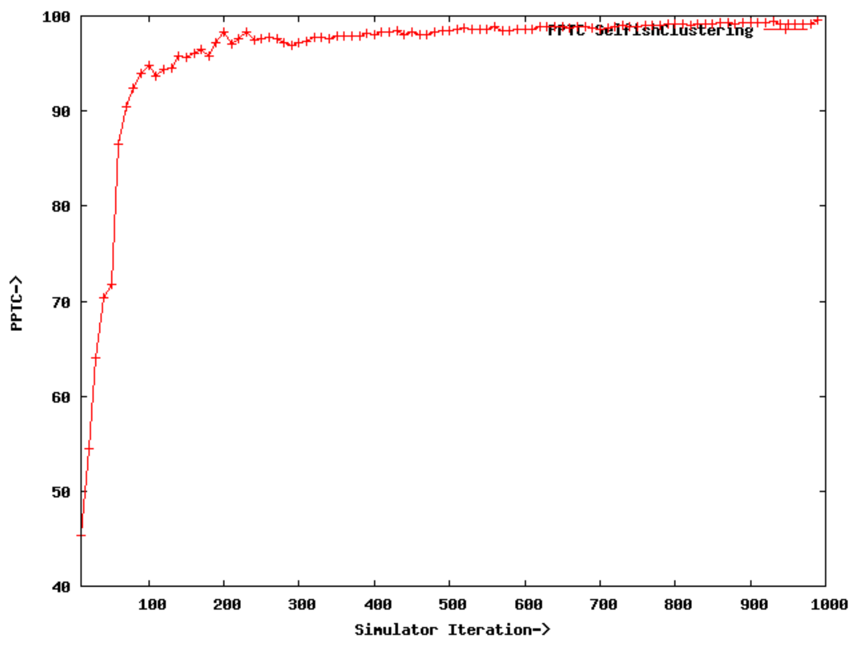
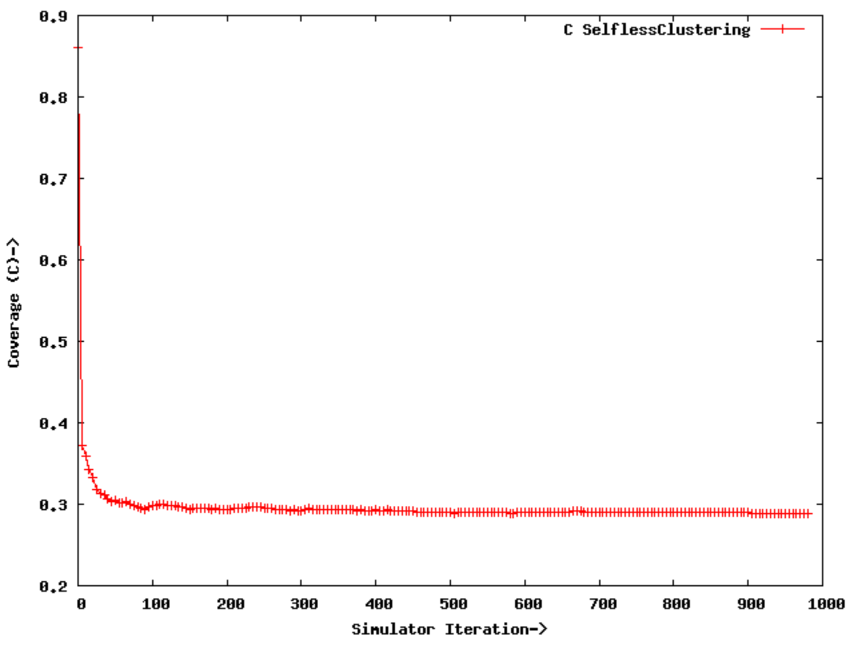


Fig. 4.24: The top figure shows a plot of PPTC against simulator iterations when the SelflessClustering Algorithm is applied. The bottom figure shows a plot of PPTC against simulator iterations when the SelfishClustering Algorithm is applied.



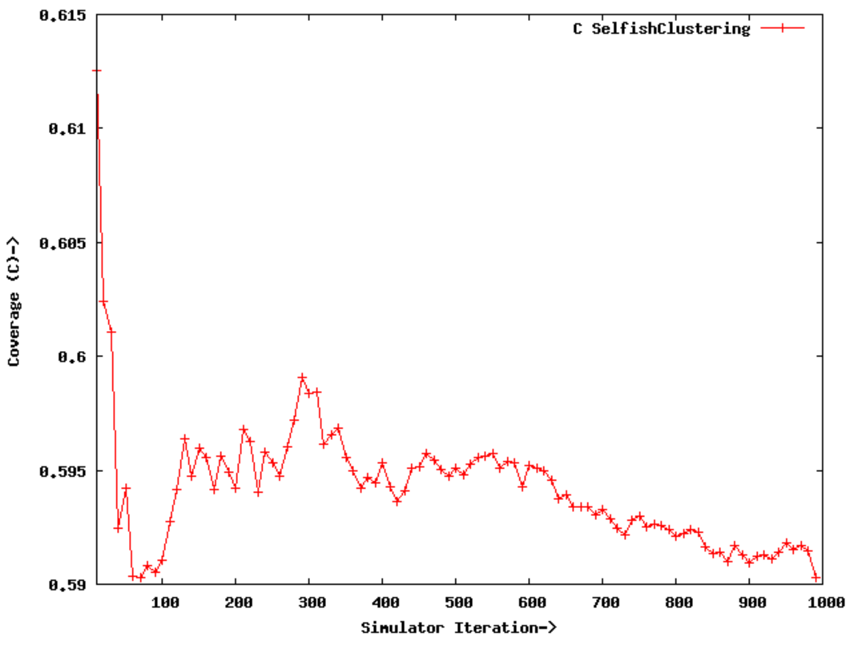


Fig. 4.25: The top figure shows a plot of Coverage (C) against simulator iterations when the SelflessClustering Algorithm is applied. The bottom figure shows a plot of Coverage (C) against simulator iterations when SelfishClustering Algorithm is applied.

Algorithm which vary because of the reconnection of the disconnected peers. The number of clusters remains between 15 and 40 for the SelfishClustering Algorithm.

Figure 4.24 shows a plot of $PPTC$ against time (simulator iteration) when the SelflessClustering and SelfishClustering algorithms are applied on the overlay network of 100 peers with continuous arrival of peers described above. In the SelflessClustering Algorithm more than 99% of the total number of peers are part of the clusters with the maximum number of peers for each category. For the SelfishClustering Algorithm this figure varies between 98% and 99%. For static simulations on an overlay network with 5,000 peers a $PNSP_{desired}$ value of 10 produces a low $PPTC$ value of 68.82 for SelflessClustering algorithm (see figure 4.6) and 59.76 for SelfishClustering algorithm (see figure 4.7). The $PPTC$ values for an overlay network with 5,000, are on the higher side because the influx of peers changes the satisfaction state of already satisfied peers on the network who engage in further topology adaptation. This does not happens in static simulations.

Figure 4.25 shows a plot of Coverage (C) against time (simulator iteration) for the dynamic simulations. The Coverage (C) is between 0.29 and 0.36 for the Selfless-Clustering Algorithm and 0.59 and 0.6 for SelfishClustering Algorithm. As shown in section 4.2.2 a lower value of C might be desired by some applications and it can be obtained by choosing a higher value of $PNSP_{desired}$.

The clustering algorithms display emergent behavior. When all the peers are satisfied and the number of peers on the overlay network reaches 5000, the $PNSP$ value is 55 for the overlay network on which the SelflessClustering Algorithm was applied and 47 for the overlay network on which the SelfishClustering Algorithm was applied. The $PNSP$ value is much higher than the $PNSP_{desired}$ value of 10.

4.2.4 Discussion

The simulations show that the clustering algorithms can create and maintain a clustered topology even when there is a continuous arrival of peers on the overlay network. A visual examination of figures 4.14 and 4.16 show that the TC varies slightly with the number of peers on the network and this makes the clustering algorithm good for overlay network with a large number of peers on the network. Both the algorithms have a mandatory messaging cost that is generated to check the status of the neighbors. An implementation of the clustering algorithms could piggyback on these messages the information required to maintain the directories on the peers.

The SelflessClustering Algorithm has a high MC and $RDNC$ when compared to the SelfishClustering Algorithm. An application that requires a high $RDNC$ (e.g., an application that desires to optimize the performance of messaging on the overlay network by bringing together peers in the same geographical location) could use the SelflessClustering Algorithm. However, if the large number of messages generated by the SelflessClustering Algorithm is an issue then an application may use the Selfish-Clustering algorithm. As discussed in section 4.2.2, the choice of $PNSP_{desired}$ depends on the $RDNC$ and C that an application requires and the MC that it can incur.

In the clustering algorithms, the value of the parameter $PNSP_{desired}$ is set using static simulations. This is a major drawback of the clustering algorithms. It would be useful if SC and TAS are designed that can dynamically decide the value of $PNSP_{desired}$. The overlay network is disconnected when $PNSP_{desired}$ reaches $PNSP_{critical}$. A disconnected topology is of limited use for an application. A major challenge in dynamically deciding the $PNSP_{desired}$ value is identifying when the overlay network will get disconnected.

In an overlay network that utilizes the clustering algorithms presented in this thesis, a peer needs to spend some time and resources before it can efficiently utilize the capability of the P2P system. The time and resource expenses might act as a deterrent for peers to leave the system after they have consumed the resource of their interest.

4.3 Improving the Effective Bandwidth Between Peers

This section presents the simulation results for a use-case that demonstrates the utility of the clustering algorithms. In an overlay network, a message is routed through a number of peers before it reaches its destination. The bottleneck bandwidth between the source and the destination is the smallest of the bandwidths of the hops in the route. The bottleneck bandwidth gives an accurate upper bound of the rate at which information will be exchanged between peers [98]. Typically, in a pure P2P network (e.g., Gnutella) the location of the peers is decided randomly, and therefore peers with high bandwidth may be adjacent to peers with low bandwidth, introducing undesired low bottleneck bandwidths in the network. This section shows how the clustering algorithms can be used to cluster peers on similar bandwidth connections, so that the effective bandwidth between the peers on the overlay network is high, which will help the peers to exchange information at a faster rate.

In this section, we first present the metric that is used to measure the effect of the clustering algorithms on the effective bandwidth of the network. This is followed by a description of the bandwidth distribution of the peers for the overlay networks used in the simulations. Finally we present the results of two types of simulations that have been done using both the clustering algorithms: *static simulations* which use an overlay network that has no flux of peers and *dynamic simulations* in which peers are added at regular intervals in the system.

4.3.1 Percentage Increase in BBN

The *bottleneck bandwidth for the network* (BBN) is the average of the bottleneck bandwidth between all possible pairs of peers in the network. In the simulations,

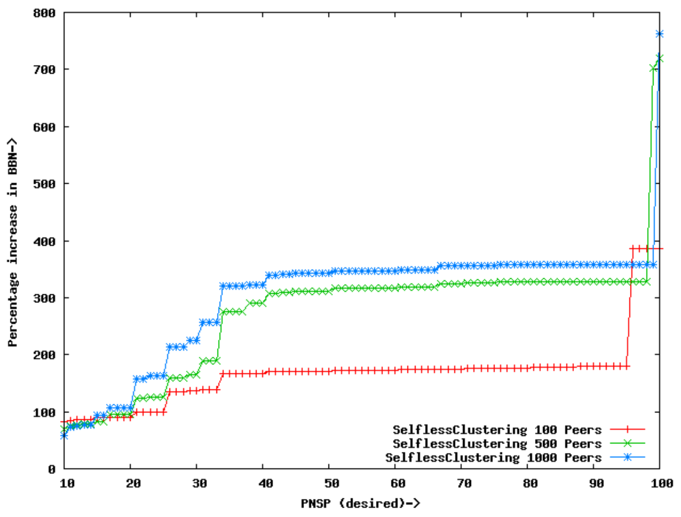


Fig. 4.26: A plot of PIBBN after applying the SelflessClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500 and 1,000 peers.

the metric Percentage Increase in BBN ($PIBBN$) is used to measure the rise in the effective bandwidth of the network. Percentage Increase in BBN is defined as :

$$PIBBN = \frac{(BBN_{current} - BBN_{original}) \times 100}{BBN_{original}} \tag{4.4}$$

4.3.2 Bandwidth Distribution

A 2003 study of the types of hosts that participate in a Gnutella network [98] suggests that 30% of the peers are on very high bandwidth connections of at least 3 Mbps and 30% of them are on bandwidth connections of at least 1 Mbps. Even though the study is four years old, we expect the bandwidth heterogenity to persist. The simulations have been done on overlay networks in which 30% of peers chosen at random are connected to the Internet on 1 Mbps and 3 Mbps lines, and the rest of them are connected to the Internet on 56 Kbps lines.

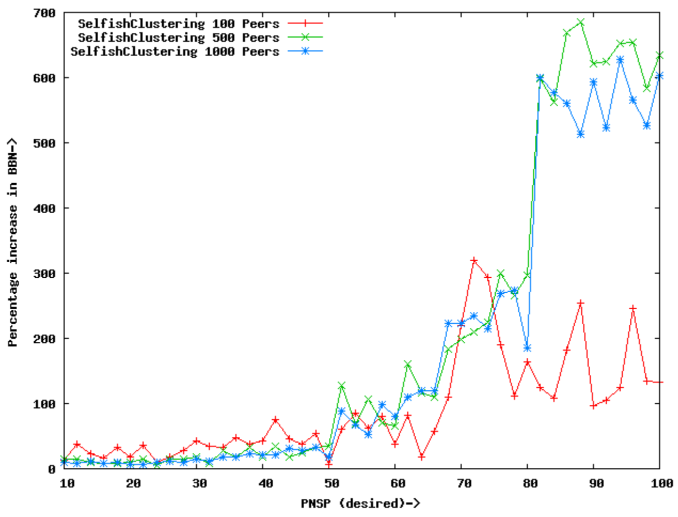


Fig. 4.27: A plot of PIBBN after applying the SelfishClustering Algorithm against $PNSP_{desired}$ on power-law topologies with 100, 500 and 1,000 peers.

4.3.3 Static Simulations

The static simulations the static simulations have been done on three different power-law networks of 100, 500 and 1,000 peers each using $PNSP_{desired}$ values from 10 to 100 in increments of 1. The power law topologies are similar to the once used in section 4.2.2. In each iteration, the simulator goes through all the peers, checking their satisfaction state and executing the topology adaptation steps for each peer that is not satisfied. The simulations go on till all the peers are satisfied or 1,000 simulator iterations are reached. The metric $PIBBN$ is used to measure the effect of the clustering algorithms. In the simulations in this section, the effects of time delay between successive estimation of satisfaction state has not been investigated and the peers estimate their satisfaction state at every simulator iteration.

In the SelflessClustering Algorithm, each unsatisfied peer searches for a similar peer on the overlay network. Just like the simulations in section 4.2.2, the search operation in the simulations has been implemented using random walk that is biased towards peers with a higher degree.

Improvement in BBN

Figures 4.26 and 4.27 show plots of $PIBBN$ against $PNSP_{desired}$ for the SelflessClustering and SelfishClustering algorithms respectively when applied on overlay networks with 100, 500 and 1,000 peers. $PIBBN$ is up to 800% for the SelflessClustering Algorithm and 700% for the SelfishClustering Algorithm. The $PIBBN$ rises when the clustering algorithms are applied because the connections between dissimilar peers are removed and similar peers are clustered together as neighbors. In the SelfishClustering Algorithm peers drop dissimilar neighbors without checking if they are the only neighbor of the dissimilar peer. This selfish dropping of peers results in peers disconnected from the overlay network. The change in BBN for the SelfishClustering Algorithm is slightly erratic compared to the SelflessClustering Algorithm because the disconnected peers are reconnected to three randomly chosen peers on the overlay network. Figure 4.13 shows that for the SelfishClustering algorithm the coverage C becomes 0 when $PNSP_{desired}$ is close to 80. In figure 4.27 a sharp rise in $PIBBN$ is observed when $PNSP_{desired}$ is 80 because there are no connections between peers on different bandwidths.

Discussion

Simulations in section 4.2.2 showed that the overlay network topology is typically disconnected when $PNSP_{desired}$ is above 30 for the SelfishClustering Algorithm and above 90 for the SelflessClustering Algorithm. A disconnected topology is not useful for an application because all the peers will not be able to communicate with each other. When $PNSP_{desired}$ is below 30, the $PIBBN$ is very low (approximately below 10) for the SelfishClustering Algorithm which makes it less attractive when compared to the SelflessClustering Algorithm.

Section 4.2.2 discusses the messaging cost involved with performing topology adaptation using the clustering algorithms. There is a trade-off between $PIBBN$ and the number of messages exchanged. The optimal choice of $PNSP_{desired}$ and the clustering algorithms depend on the application and its characteristics. For an overlay network with a long life, it may be advisable to use the SelflessClustering Algorithm to gain a high $PIBBN$ by exchanging many messages. However for an overlay network with a short life span, it may be better to choose a small $PNSP_{desired}$ and the SelfishClustering Algorithm, which does not provide a high $PIBBN$, but which is less costly in terms of the number of messages exchanged.

4.3.4 Dynamic Simulations

This section presents the results of the dynamic simulations in which there is a continuous arrival of peers. The network has 100 peers initially and five peers are added to the network in every simulator iteration till there are 5,000 peers on the network. The simulations go on till all the peers are satisfied or 1,500 simulation iterations are reached. The simulations have been done using $PNSP_{desired}$ value of 10 because it is far below the $PNSP_{critical}$ value for both the SelflessClustering and SelfishClustering algorithms. Just like the static simulations, the distribution of types of peers on the overlay network follow the pattern described in section 4.3.2 and a biased random walk with replication has been used to search for similar peers in the SelflessClustering Algorithm.

Results

For both algorithms, the simulations stop within 980 iterations when the number of peers on the overlay network reaches 5,000. Figure 4.28 shows a plot of the $PIBBN$ against time (simulator iteration) when the SelflessClustering and SelfishClustering algorithms are applied on the networks described above. $PIBBN$ is calculated every

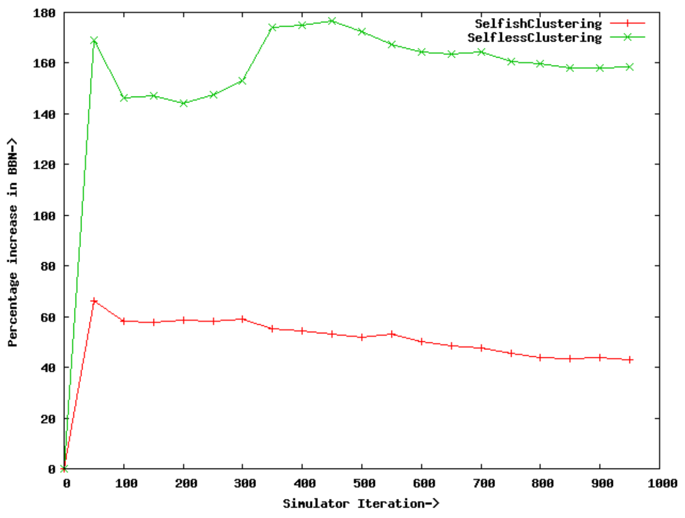


Fig. 4.28: A plot of PIBBN against simulator iterations after applying the SelflessClustering and the SelfishClustering algorithms.

twentieth simulator iteration. The SelflessClustering results in a percentage rise in BBN that varies between 140% and 180% and the SelfishClustering Algorithm results in a percentage rise in BBN that varies between 40% and 70%. $PIBBN$ is lower for SelfishClustering Algorithm when compared to the SelflessClustering Algorithm. This is similar to the behavior of $PIBBN$ in static simulations. The results show that the clustering algorithms can maintain a substantial rise in BBN even when there is an influx of peers on the overlay network.

4.3.5 Summary

The simulations in this section show that the clustering algorithms result in a rise in BBN and can maintain this rise even when there is a continuous arrival of peers on the network. The optimal choice of $PNSP_{desired}$ and clustering algorithm depends upon the application and its characteristics. Typically the $PNSP_{desired}$ is restricted to be less than $PNSP_{critical}$.

A drawback of the simulations in this section and the simulations in the previous section is that the effect of peers leaving the overlay network has not been studied. However we expect that the clustering algorithms should work even when peers leave the overlay network at regular intervals. In the clustering algorithm a peer periodically checks its neighbors to decide if it is satisfied with them and if some of the neighbors have left the overlay network then it will account for them in its satisfaction state. If the peer is unsatisfied because of a neighbor leaving, then it can execute topology adaptation steps to maintain the topology.

4.4 Related Work

As mentioned in section 2.2 for the purpose of review, the existing decentralized algorithms for clustering peers can be divided into two major categories based on the criteria used for clustering: content-based clustering and distance-based clustering. The distance based clustering algorithms are optimized to cluster peers that are close to each other on the underlying physical network. The content-based clustering algorithms discussed in section 2.2.7 are designed to create clusters of peers having a similar property. For file-sharing applications this property is typically the type of content that a peer is sharing. The content-based clustering algorithms are accompanied by a routing algorithm that directs search requests to the most suitable cluster for handling the request.

The distance-based clustering algorithms presented in section 2.2.6 use a similar approach which involves using selected landmark peers. The peers measure their underlying network distance to the landmark peers and connect themselves to the landmark peer that is closest to them. This approach results in clusters that are based around the landmark peers. If the landmark peers are not evenly distributed across the overlay network's topology then this algorithm would result in unevenly sized clusters. The major drawback of these algorithms when compared to the clustering algorithms presented in this chapter is their dependency on centralized components (landmark peers) which determine the clustering in the topology.

Hang et al. proposed one of the earliest algorithms for content-based clustering [79]. The algorithm is described in detail in section 2.2.7. In this algorithm, each peer P maintains a host-cache containing information about other peers on the network. The peer P obtains the information in the host-cache by flooding the overlay network with a request for information about other peers. The peer P then establishes connections to peers that share similar content. In this algorithm, the peers will keep executing the expensive operation of flooding the network for topology adaptation even when they already have the neighbors required for the desired topology. In comparison, the peers using the clustering algorithms presented in this chapter do not consume the network resources to search for new neighbors once the desired topology has been achieved. In the SelflessClustering algorithm the peers will only search for new neighbors if the topology desired by a peer is altered due to peers leaving or joining the overlay network.

Another example of content-based clustering is presented in [27], in which a new peer which is in the process of joining the overlay network obtains information about other peers that share similar content and then establishes connections to them. The system is in a clustered state before the peer joins, and the choice of neighbors assures that it remains in a clustered state. While this is a conceptually elegant model, it is not without problems. Typically, a joining peer has only minimal knowledge of the network's topology and this knowledge is often limited to just one bootstrap node. Unless the bootstrap node happens to share similar content to the new peer, it is unlikely that its position in the clusters will be very useful for the new node, and an expensive search operation may be required to find suitable neighbors.

4.5 Summary

The chapter has demonstrated that Schelling's model can be used effectively for clustering in P2P overlay networks. The chapter has presented two algorithms, SelfishClustering and SelflessClustering algorithms; based on PESTO. The chapter has reviewed the latest technique for mathematically analyzing Schelling's model and explained why only a specific case of the clustering algorithms can be analyzed with this technique. Further research is required on the mathematical analysis to explain the other cases of the clustering algorithms. Simulations are used to examine the clustering achieved by the two algorithms. The simulation results demonstrate that the algorithms can be used to bring together similar peers on the overlay network even when there is a continuous arrival of peers.

The chapter also presented a case study to demonstrate that the clustering algorithms can be used to increase the effective bandwidth between peers on an overlay network, so that they can exchange information at a faster rate. The rise in the bandwidth is caused by clustering of peers using similar bandwidth connections on the overlay network. A similar clustering can be expected when instead of bandwidth a different criterion (e.g., geographical location of the peers) is used.

Chapter 5

Hub Topology

All animals are equal, but some animals are more equal than others [82].

In section 2.1.6 we described the hub topology in which peers with high capacity and availability (called hubs) are connected together to form a backbone network in the overlay network. The backbone network can be used for tasks such as maintaining a directory of resources on the network and for routing messages between peers. This chapter presents a decentralized topology adaptation algorithm called the HubAlgorithm for creating and maintaining a hub topology. The algorithm has been published in [108] and [109]. The HubAlgorithm is a concrete realization of PESTO. The description of the algorithm is followed by simulation results that demonstrate that the HubAlgorithm can be used to create and maintain a hub topology. The chapter finally compares the HubAlgorithm with the algorithm used in the JXTA reference implementation which also creates a hub topology.

5.1 HubAlgorithm

In section 2.1.6, a formal definition of a hub topology was presented and the advantages of a hub topology were discussed. This section presents a decentralized topology

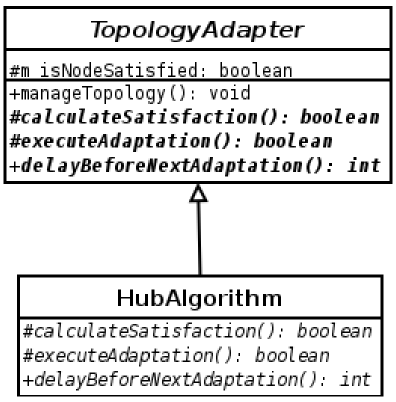

TopologyAdapter
#m_isNodeSatisfied: boolean
+manageTopology(): void
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int
HubAlgorithm
#calculateSatisfaction(): boolean
#executeAdaptation(): boolean
+delayBeforeNextAdaptation(): int

Fig. 5.1: The HubAlgoritm is a concrete concrete realization of PESTO.

adaptation algorithm, called the HubAlgoritm, which can be executed by the peers on an overlay network. The HubAlgoritm is a concrete realization of PESTO as shown in figure 5.1. In the HubAlgoritm a peer examines its neighbors at regular intervals to see if it is satisfied with them and if it is not satisfied, then it executes a series of topology adaptation steps.

The HubAlgoritm is self-organizing because it organizes the peers using their local awareness of the overlay network's topology and without any central authority. The HubAlgoritm maintains the reorganized topology under a flux of peers by calculating a peer's satisfaction state at regular intervals and taking topology adaptation steps if the peer is not satisfied with its neighborhood.

In a hub topology there are two categories of peers: ordinary peers and peers with high capacity and availability called the hubs. In the HubAlgoritm, the peers themselves choose the role of hub or peer. To create a hub topology, the ordinary peers and hubs execute different versions of the HubAlgoritm depending on their type. The HubAlgoritm executed by the ordinary peers is referred to as the HubAlgoritm(peer) and the HubAlgorithm executed by the hubs is referred as the HubAlgoritm(hub). Algorithm 3 shows the pseudocode for the HubAlgoritm(peer) and algorithm 4 shows the pseudocode for the HubAlgoritm(hub). The operations that have been used in the pseudo-code are described in table 3.1.

Algorithm 3 HubAlgorithm(peer)

```
maxNeighbors ← Maximum number of neighbors that a peer can have
while true do
  if count(hubs) == 0 then
    if count(all) == maxNeighbors then
      drop(neighbor(n: n.property == any and n.count(all) > 1))
    end if
    add(search(hub))
  end if
  sleep(delay)
end while
```

Algorithm 4 HubAlgorithm(hub)

```
$H_{max}$ ← Maximum umber of hubs desired as neighbors
while true do
  if count(hub) >= $H_{max}$ then
    drop(neighbor(hub))
  end if
  if count(hub) == 0 then
    add(search(hub))
  end if
  sleep(delay)
end while
```

Table 5.1 shows the satisfaction criteria and the topology adaptation steps that will be executed by the hubs and the ordinary peers. The SC and TAS are different for ordinary peers and hubs. The satisfaction criteria for $HubAlgoritm(peer)$ states that an ordinary peer is satisfied if it is connected to at least one hub. If an unsatisfied ordinary peer is connected to the maximum possible number of neighbors, then it drops a randomly chosen ordinary peer neighbor that is connected to more than one peer. The unsatisfied peer then searches for a hub and adds it as its neighbor. Irrespective of the success or the failure of the TAS, the SC is evaluated again after waiting for a

Peer	**SC**	**TAS**

Hub	$H_{max} >$ **count**(hub) and **count**$(hub)\ != 0$
	
	where, H_{max} is the maximum number of hubs desired as neighbors.
		step 1:
		if (**count**$(hub) > H_{max}$)
		    drop(**neighbor**(hub))
		step 2:
		if (**count**$(hub) == 0$)
		    add(**search**(hub))

Normal	**count**$(hubs) > 0$	*step 1:*
		if (**count**$(all) == maxNeighbors$)
		    drop(**neighbor**(n: n.property == *any* and n.**count**$(all) > 1$))
		step 2
		    add(**search**(hub))

Table 5.1: Satisfaction criteria and topology adaptation steps that will be executed by the hubs and the ordinary peers to create a backbone network of hubs within the overlay network.

time period specified by the method $delayBeforeNextAdaptation(\ldots)$.

The satisfaction criteria for the $HubAlgoritm(hub)$ states that a hub is satisfied if it is connected to at least one other hub and has less than H_{max} hubs connected to it. If an unsatisfied hub does not have any hubs as its neighbors then it searches for another hub HO and adds HO as its neighbor. If the unsatisfied hub has more than H_{max} hub neighbors, then it drops an existing hub neighbor that is connected to more than one hub. This is to ensure that a hub will not get disconnected from the hub network. The former topology adaptation step, taken by a hub that is not connected to any other hub, creates a feedback effect that strengthens the hub network. When an unsatisfied hub adds another hub HN as its neighbor, it increases the number of hubs to which it is connected as well as the hub neighbor count of HN.

5.1.1 Messaging Cost of the HubAlgorithm

A peer in an overlay network using the HubAlgoritm generates messages to obtain information about its neighbors so that it may decide its satisfaction state. As described in section 3.3.2, if the peers use a proactive approach then they generate a total of kn messages, where k is the average degree of the peers and n is the number of peers on the overlay network; every iteration to check the status of their neighbors. If this messaging cost is unacceptable, then a peer P may choose a reactive approach in which it may take time before P receives notifications about the change in status of its neighbors. The choice of the approach used by a peer for obtaining information about its neighbors is an implementation choice. As described in section 3.3.2, this is a common messaging cost that will be incurred throughout the life of the overlay network and is therefore not applicable in evaluating the specific topology adaptation algorithm.

Search Cost

Apart from the mandatory messaging cost associated with calculating the satisfaction state of a peer, the number of messages exchanged to perform the search operation for similar peers is the other major cost of utilizing the HubAlgoritm. The search operation can be implemented in a variety of ways such as breadth first search (BFS), random walk, random walk with a bias towards peers with a high degree as described in section 2.2.4 or as a gossip-based search. Other search algorithms are of course also possible, and the most suitable approach depends on application requirements.

5.1.2 Time Delay

As described in section 4.1, the time delay between successive estimation of the satisfaction state is another important factor that affects the cost of utilizing the HubAlgoritm. A satisfied peer or hub need not estimate its satisfaction state repeatedly. Similarly, a peer or hub that is unable to successfully execute its topology adaptation steps need not estimate its satisfaction state repeatedly. It can instead use an exponentially increasing time delay between successive estimation of the satisfaction state.

5.2 Simulations

This section presents the simulation results of applying the HubAlgorithm on overlay networks. The simulations have been done using the simulator described in section 3.4. Two types of simulations have been done using the HubAlgorithm: *static simulations* which use an overlay network that has no flux of peers and *dynamic simulations* in which new peers are added to the overlay network at regular intervals. As described in section 4.2 it is easy to simulate and analyze the results when there is no flux of peers. The dynamic simulations are used to show that the HubAlgorithm can maintain a hub topology even when there is a continuous arrival of new peers on the network.

All the simulations have been done on overlay networks in which 90 percent of the peers chosen randomly are assigned the role of ordinary peer and the rest are assigned the role of hub. The maximum number of connections is 5 for an ordinary peer and 20 for a hub. It is ensured that the initial overlay network topology is connected. The **search** operation is performed using a Depth First Search (DFS) on the overlay network.

The rest of this section is structured as follows: we first define the metrics that are used to study the result of applying the HubAlgorithm on the overlay networks. This is followed by a description of the setup, and presentation and discussion of the results of static and dynamic simulations.

5.2.1 Metrics

This section describes the metrics that are used to evaluate the HubAlgorithm. In this section **H** is the set of hubs and **P** is the set of peers on the overlay network. Let $\mathbf{E}_{H,H}$ be the set of connections connecting two hubs, $\mathbf{E}_{P,P}$ be the set of connections connecting two peers and $\mathbf{E}_{H,P}$ be the set of connections connecting a hub to a peer on the overlay network.

The simulation in this section use the five metrics defined below. Apart from these metrics, Time to Converge (TC) and Messaging Cost (MC) defined in section 4.2.1 are also used to evaluate the algorithm. As defined in section 2.1.6, the hub topology consists of hubs that are connected to each other to form a backbone network of hubs and ordinary peers that are connected to these hubs. The first two metric's presented below are used to examine the hub network. The next metric measures the average connectivity between the ordinary peers and the hubs. The last two metrics provide an average count of the remaining two types of connections in the topology.

Hub-Hub Degree

Hub-Hub (HH) degree is defined as:

$$\frac{|E_{H,H}|}{|H|} \tag{5.1}$$

HH shows how well the peers in the hub network are connected to each other. It is not desirable to have a high value of HH because then the number of slots available with the hubs for ordinary peers will decrease.

Percentage of Hubs in Largest Hub Component

The Hub Network HN is a graph in which the vertices are members of the set H and edges are members of the set $E_{H,H}$. The Percentage of Hubs in Largest Hub Component

($PHLHC$) is the percentage of the total number of hubs that are part of the largest component in HN. A large value of $PHLHC$ is desirable for the HN.

Peer-Hub Degree

Peer-Hub (PH) degree is defined as:

$$\frac{|E_{H,P}|}{|P|} \tag{5.2}$$

It is desirable to have PH close to 2 for better fault tolerance. If the PH is 2 then an ordinary peer can access the hub network through another hub if one of the hubs to which it is connected fails.

Hub-Peer Degree

Hub-Peer (HP) degree is defined as:

$$\frac{|E_{H,P}|}{|H|} \tag{5.3}$$

HP measures the average number of connections between a hub and an ordinary peer. The HP value can be high or low depending on the number of ordinary peers in the topology. Typically, HP will have a high value close to the maximum number of connections

Peer-Peer Degree

Peer-Peer (PP) degree is defined as:

$$\frac{|E_{P,P}|}{|P|} \tag{5.4}$$

It is useful to have a PP value of 1 or higher so that the ordinary peer is not disconnected from the topology, in case the hubs to which it is connected fail.

Overlay Network Size	$H_{maxCritical}$	TC	MC
100	3	4	70
500	4	5	860
1,000	3	5	2314

Table 5.2: $H_{maxCritical}$ and TC and MC at $H_{maxCritical}$

5.2.2 Static Simulations

The static simulations have been done on power law networks because a study of initial Gnutella topologies by Ripeanu [92] suggests that the degree distribution followed a power law. The simulations go on till all the peers are satisfied or 1,000 simulator iterations are reached. Simulations have been done on four different power law networks (created by using different seeds values for the random network generator on Linux) of 100, 500 and 1,000 peers each using H_{max} values from 1 to 10.

A critical value of H_{max} (called $H_{maxCritical}$) was observed below which all the peers were not satisfied even after 1,000 simulator iterations. The value of $H_{maxCritical}$ is different for different power law networks. A typical value of $H_{maxCritical}$ observed in the simulations is 5. When H_{max} is below $H_{maxCritical}$ the simulations do not converge because some of the hubs are not satisfied as they are not able to find another hub to which to establish a connection.

Table 5.2 shows the $H_{maxCritical}$ and TC and MC at $H_{maxCritical}$ for overlay networks with 100, 500 and 1,000 peers. When H_{max} is greater than or equal to $H_{maxCritical}$ all the peers are satisfied and the TC is 5 simulator iterations. Apart from the mandatory messaging cost to check the satisfaction state, the peers on the overlay network using the HubAlgorithm exchange messages to search for other peers. For a network of 100 peers, typically the MC is 100. For a network of 1,000 peers, typically the MC is less than 3,000. The simulations converge when all the peers are satisfied, which means that each hub on the overlay network is connected to at least one other hub and at

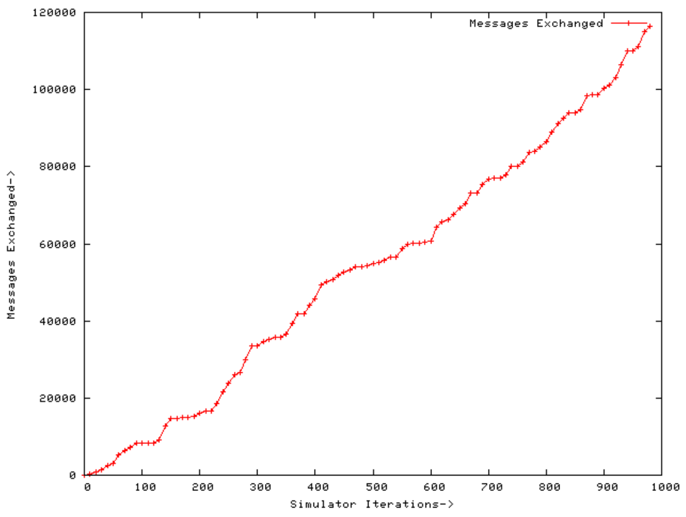


Fig. 5.2: A plot of total number of messages exchanged against simulator iterations for dynamic simulations.

most to H_{max} hubs, and all the ordinary peers are connected to at least one hub.

5.2.3 Dynamic Simulations

The second set of simulations have been performed on a power law network where new peers join the system every simulator iteration to demonstrate that the approach can be used in dynamic environments. The simulations start with a small network of 100 peers, and 5 new peers are added every iteration till the number of nodes reaches 5,000. The simulations have been done using a H_{max} value of 5, as this was a typical $H_{maxCritical}$ value. The simulations go on till there are 5,000 peers on the overlay network and all the peers are satisfied or 1,500 simulator iterations are reached. The simulations were repeated for five different power law networks generated by using different seed values for the random number generator on Linux.

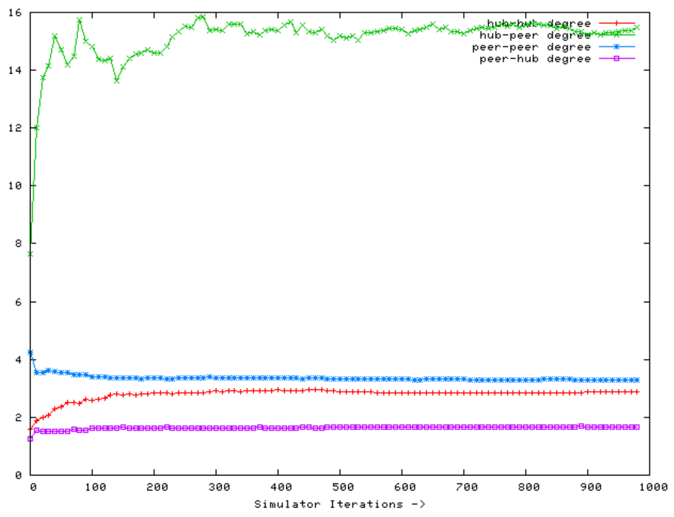


Fig. 5.3: A plot of hub-hub and peer-hub degree against simulator iterations, for dynamic simulations.

Results

The time to converge (TC) is 980 for the HubAlgorithm. Figure 5.3 shows a plot of HH and PH against time (simulator iterations) for simulations performed on one of the power law networks. Throughout the simulations, PH has the desired value of 2. When the simulations start, each hub is connected on average to approximately 2 other hubs. However, within 100 simulator iterations HH changes to 3 and it stays at that value throughout the simulations, because of the HubAlgorithm. In the simulations the hubs can have a maximum of 20 neighbors. The HP is close to a high value of 16 which means that the ordinary peers get a high number of slots to connect to the hubs. The PP is close to 4 throughout the simulations. The high value of PP will be useful for an ordinary peer to search for new hubs to connect to if it is disconnected from the hubs to which it is connected.

Figure 5.2 shows a plot of the total number of messages exchanged to perform topology adaptation against time (simulator iterations) for the power law network of

Figure 5.3. The total number of messages exchanged (120,000) to perform topology adaptation may seem to be on the high side. However, for an overlay network with a long life it may be advisable to create an adaptive network of hubs by exchanging lots of messages. Also, in the simulations the **search** operation was implemented without any caching. Caching can be used to improve the efficiency of the **search** operation, thereby reducing the total number of messages exchanged.

The simulations show that the HubAlgorithm does not ensure an unpartitioned HN graph. Only for fifty percent of the simulations is the HN graph unpartitioned. This is a small percentage. However for all the simulations $PHLHC$ has a very large value of more than 99% which means that a large number of hubs are members of the largest component of the HN graph. Whether this is acceptable depends on the application.

When the simulations converge, all the peers are connected to at least two hubs. The hubs are connected to at least one other hub and form a hub network with $PHLHC$ more than 99%. The simulations show that the HubAlgorithm can maintain an adaptive network of hubs even when there is a continuous arrival of new peers on the network.

5.2.4 Discussion

The previous sections presented simulation results to demonstrate that the HubAlgorithm can create and maintain a hub topology. The HubAlgorithm maintains the desired PH value of 2 and has a very high value of $PHLHC$ which is more than 99%. In the simulations, 10% of peers chosen randomly are assigned the role of hubs. This means that a very small number of approximately 5 hubs out of 500 are not connected to the hub network. Further effort is required to refine the HubAlgorithm so that $PHLHC$ can be increased further. For the dynamic simulations, it takes 980 iterations to add 4,900 peers to change the number of peers on the overlay network to 5,000. The TC for the dynamic simulations is 980, which means that all the peers are satisfied within 1 simulator iteration after adding the last batch of 5 peers.

In the HubAlgorithm, static simulations are used to decide the value of the parameter H_{max}. This is a drawback of the HubAlgorithm and further research is required to dynamically decide the value of H_{max} in an application. In the HubAlgorithm the peers themselves choose the role of hub or peer. This is a disadvantage because the peers lack global knowledge about other peers on the overlay network and might incorrectly classify themselves as hubs when peers with a higher capacity and availability are available or ordinary peers when they are the peers with a higher capacity and availability. A workaround to this problem could be that the application decides the specifications that will be used to categorize a peer as hub or ordinary peer. Another solution to this problem is a decentralized algorithm proposed in [96], that can be used by peers to determine the range of a characteristics and the number of peers with the given characteristics on the overlay network. This algorithm is a candidate solution to enhance the HubAlgoritm so that the peers can autonomously decide if the they should join the network as a hub or an ordinary peer. A peer may choose to join the overlay network as a hub if it has a higher value for the characteristics used to decide whether a peer should act as a hub or an ordinary peer.

5.3 Related Work

To the author's knowledge, the JXTA reference implementation (discussed in section 2.3.2) uses the only other existing topology adaptation algorithm for creating a hub topology. In the topology adaptation algorithm used in the JXTA reference implementation, each hub maintain a list of all the other hubs it is aware of. To create a hub topology, a hub periodically exchanges its list of hubs with its neighbors. The list of hubs received is appended to the hub's existing list. A hub will establish connection to the other hubs in the list to create the hub network. A drawback of the topology adaptation algorithm in the JXTA reference implementation is that it does not specify the number of hubs to which each hub maintains a connection. This could be a problem, because if the hubs maintain connections to a large number of other hubs then ordinary peers may not get free slots to connect to a hub. If the hubs maintain connections to a small number of other hubs, then other hubs, may not get slots to connect to hubs. Unfortunately, no evaluation of the topology adaptation algorithm in JXTA is presented, which makes meaningful comparison difficult.

5.4 Summary

Chapter 4 presented two clustering algorithms based on PESTO that can be used to create a clustered topology. This chapter presented the HubAlgoritm, the third concrete realization of PESTO that can be used to create a hub topology. Together, the three algorithms demonstrate the general applicability of PESTO as a template for creating topology adaptation algorithms. The chapter presented simulation results to demonstrate that a self-organizing network of hubs can be created by using the HubAlgoritm. In the HubAlgoritm a peer decides whether it should act as a hub or an ordinary peer. Future work could involve investigating the possibility of creating an algorithm based on PESTO, in which the peers mutually decide whether a peer should act as a hub or an ordinary peer.

Chapter 6

Reference Architecture

Science means simply the aggregate of all the recipes that are always successful. All the rest is literature [87].

This chapter presents a reference architecture for the P2P domain called P2P reference Architecture that Supports Topology Adaptation (PASTA). While the primary contribution of the thesis is an abstract algorithm for designing topology adaptation algorithms, PASTA complements this contribution by providing a template for developing applications that require topology adaptation. PASTA has been published in [110]. The chapter begins with a definition of Service Oriented Architecture (SOA) and a discussion on the rationale behind taking a service centric view in the reference architecture. Then we present the diagrammatic notation used to describe the reference architecture and a summary of a study of existing P2P applications and middlewares, which was used as the basis for deriving the reference architecture. Next we present PASTA itself and its validation through description of the structure of existing P2P applications and middlewares using the reference architecture. The validation demonstrates that PASTA is not limited to applications that require topology adaptation and is generally applicable. Finally, we present a comparison between PASTA and an existing reference architecture called "The Essence of P2P" that was discussed in section 2.3.1.

6.1 Service-Oriented Architecture

A service is a unit of work done by a service provider to achieve desired results for a service consumer [52]. Service-oriented architecture (SOA) is an approach that utilizes software services as fundamental elements for developing software applications [84, 52, 86]. In SOA, software resources are services available and discoverable on a network. Common Object Request Broker Architecture (CORBA) [54], Microsoft's Distributed Component Object Model (DCOM) [120] and SOAP [14] are examples of middlewares that provide support for SOA.

In SOA, the underlying implementation details of a service are hidden from the service consumer through encapsulation. There is a loose coupling between the service provider and the service consumer, which means that changes can be done in either one of them without affecting the other. The service provider and the service consumer must agree on an interface that does not change. The service consumer uses the service interface description and the service address to access the service. The service provider can easily change the implementation of the service without breaking the service consumer. Also, if required, the service consumer can use a service at an alternate location if it has same service interface as the original service. The loose coupling between the service provider and consumer facilitates the use of the same service across many applications, thereby reducing the development cost and errors in the application and ensuring faster development time [71, pg. 17-18].

PASTA uses the service-oriented approach because it lends itself well to modelling P2P applications and middlewares. P2P systems are typically used to share a wide variety of resources and services. A P2P system can be used to access shared services which perform tasks such as providing information about other peers on the network, numerical computations and routing of messages on the overlay network. A P2P system can be used to access a wide variety of shared resources such as files, videos, messages, CPU cycles and storage space. The shared resources can be accessed through services running on the peers. Since all the sharing in a P2P system can be conceptualized using services, a service-oriented approach has been used in the reference architecture. PASTA is designed to support the sharing and access of services on a P2P network. A service-centric view facilitates the development of applications using PASTA because the software implementation of the applications can use the support for service-oriented architecture provided by the existing middlewares like CORBA [54] and SOAP [14].

Peers join a P2P network to utilize services offered by other peers. The services can be divided into two types: *network service (NES)* and *node service (NOS)*. A network service is available on more than one peer in the network. Different instances of the network service work together to execute a task. A node service is offered by an individual peer. The service is specific to the peer offering the service.

6.2 Documenting Software Architectures

Software architectures are documented using *views*, each of which concentrates on a different aspect of the software system. Different researchers suggest using different types and numbers of views for documenting software architectures. For example Booch et al. [30] suggest using 4+1 views, Bass et al. [65] suggest using 3 views and Soni et al. [55, 113, 56] suggest using 4 views. This work follows the suggestion given by Soni et al. They suggest using four views called the *conceptual view, module view, execution view* and *code view* to document a software architecture. The views are enumerated in the order in which they should be designed. The code view describes the organization of the source code and the object code. The execution view defines the run-time entities of the software system and their associated attributes, such as the memory usage and hardware assignment. The module view describes the decomposition of the software and its organization into layers. The last three views deal with the actual implementation of the software system using programming languages, operating systems, communication mechanisms and so forth and so are not relevant for documenting a reference architecture.

The conceptual view is the only of the four that is suitable for documenting a reference architecture. The conceptual view describes the structure of the software system at a high level of abstraction, using architecture elements which can not be directly implemented by using software technology [55]. In this view, the functionality of the system is mapped to architecture elements called *conceptual components* (or just *components*), with coordination and data exchange handled by elements called *connectors*. Both components and connectors can be further decomposed into more components and connectors. The notion of building a system by interconnecting components is appealing because of the potential for reuse and for incorporating off-the-shelf components into implementations based on the architecture. The components interact with the outside world using elements called *ports*. The description of a port includes description of the messages (operations) that the component can process as well as the messages that it invokes. Connectors interact with the components using elements called *roles*. Both ports and roles obey *protocols*. A protocol is defined as a set of incoming message types, outgoing message types and the valid message exchange sequence. A port and a role can be connected together if the port's protocol is the conjugate of the role's protocol.

The diagram used to describe a conceptual view is called a component-connector (CC) diagram. A CC diagram shows the elements (e.g., components and connectors) of the conceptual view. This thesis uses the UML 2.0 standard to draw the CC diagrams. There is no direct mapping between the elements of the component-connector diagram and the UML notation [58]. Table 6.1 shows the UML elements that have been used

Component-Connector element	**UML 2.0 element**
Component	Component
Connector	Associations
Role	Not represented
Port	Port
Protocol	Note

Table 6.1: The UML 2.0 elements used to draw component-connector diagrams.

to draw the elements of CC diagrams. UML components can contain other UML elements, which makes them an ideal choice for representing CC components. The newly introduced concept of *ports* in UML 2.0 is similar to the ports in CC diagrams and has been used to represent the ports. UML associations have been used to represent connectors as suggested in [58]. Since roles and ports which are connected to each other must obey similar protocols, roles are not documented in the CC diagrams.

6.3 Core Concerns for a P2P Application

Current P2P applications can be divided into three major groups based on application domain: parallel computing, content management and collaborative applications [73]. We have studied middlewares and applications belonging to all the three categories and have identified the concerns that a P2P application needs to address [106]. The concerns can be divided into five groups. Table 6.2 presents the groups along with the concerns. The concerns are the use-cases that a P2P application needs to address. PASTA handles all these use-cases.

In the rest of this section, we discuss each group and the concerns that fall within it. We also give examples of how these concerns are handled in existing applications.

Group	Concern
Naming	Identification of entities in a P2P system (*identification*).
	Resolution of entity id to a physical address (*resolution*).
Overlay Management	Search for a connected peer on the overlay network (*search*).
	Handle join requests (*join*).
	Handle leave requests (*leave*).
	Maintain the topology (*topology*).
Service Management	Advertise services/resources to share (*advertise*).
	Discovery of shared services/resources (*discovery*).
	Access the service/resources (*access*).
	Routing of messages (*routing*).
Security	Authentication of peers (*authentication*).
	Authorization of peers to access a resource/service (*authorization*).
	Secure transmission of messages (*confidentiality*).

Table 6.2: The core concerns that a P2P application needs to address.

6.3.1 Naming

The concerns belonging to this group are: assigning an identifier to the entities (e.g., peers, resources, services, etc.) in the system (*identification*) and resolving an entity's identifier to its physical network address (*resolution*). P2P systems assign a network (underlying physical network) and location-independent identifier to the entities in a system. The identifiers allow P2P systems to manage entities whose network address (typically IP address) changes with time. The identifiers are dynamically resolved to determine the current network address of the entity.

Different applications take different approaches to naming. In the JXTA [80] reference implementation, the identifier is a 128 bit universally unique identifier (UUID) generated by each peer. The JXTA reference implementation uses a combination of a central server (called rendezvous server) and IP multicast to resolve identifiers to a network address. Jabber uses a central server for both identification and resolution. Pastry [95] maintains a Distributed Hash Table (DHT) which can be used to locate the entity with a given identifier.

6.3.2 Overlay Network Management

The concerns in this group are: searching for an existing overlay network to join (*search*), handling of requests to connect to the network (*join*), leaving the network (*leave*) and managing the topology (*topology*). Peers join an overlay network through another peer which is already connected to the overlay network. Connected peers' addresses may be a well-known information it can be obtained by other means such as an IP multicast. The peer accepting the join request may redirect the incoming peer to another peer. The connecting peer may also be supplied with information relevant for it such as the network address of other peers connected to the overlay network. As discussed in Section 2.1 and 2.2 the topology of an application affects its performance. Managing the connections of a peer so that the desired topology can be created and maintained is a major challenge for applications.

In hybrid solutions such as Jabber [21, 39], peers connect to a well-known server to join the overlay network. In JXTA the connecting peer uses the combination of a well-known server (rendezvous server) and IP multicast to obtain the connected peer's address that will be used by it to connect to the overlay network. In Pastry, the connected peer's address has to be obtained through out-of-band means. The connecting peer in Pastry receives information which it can use to populate its routing table. In Jabber the connecting peer only receive information about the connectivity status of the peers in which they are interested. The peers register their interest with the central server. Peers may inform one or more peers on the overlay network when they leave the overlay network. For example, in Jabber the peers send a presence message of type unavailable to the central server.

6.3.3 Service and Resource Management

The concerns in this category are: advertising the details of the services and resources that the peer offers to the overlay network (*advertise*), discovering services and resources to use on the network (*discovery*), accessing the services and resources (*access*) and routing the messages (*routing*). Peers use the overlay network to advertise their resources and services and to discover new resources and services which they can use.

P2P systems generally maintain a directory of services and resources. A peer can publish the details of the resources and services it is offering to the directory. The directory could be maintained on one peer (e.g., Jabber). Alternatively the directory can be distributed across some (e.g., JXTA) or all the peers (e.g., casca [43]) in the network. The peers maintaining the directory can be chosen through an algorithm or they can be special-purpose peers responsible for maintaining the directory. Peers can discover the details about the shared services and resources by sending a query message to the peers responsible for maintaining a directory. Peers may also send multicast messages to all the peers on the network to find the details about services and resources available on the network. The service/resources can be accessed by using mobile code (e.g., JXTA, SpeakEasy [42, 43]) or by doing a Remote Procedure Call (RPC).

A peer can directly send the message to the destination peer if the destination peer's address is available. If the destination peer is not accessible or its network address is not available, then the source peer may route the message to one or more known peers on the network, which can route the message to the destination peer. The intermediate peers can be chosen from the known peers randomly or by using an algorithm (e.g., Pastry), or they might be well-known specialized peers (e.g., the rendezvous peers in JXTA route messages to peers behind firewalls).

6.3.4 Security

The concerns related to security are: authentication of peers to ensure the identity of a peer (*authentication*), checking the authorization of a peer to access a resource or a service (*authorization*), and secure transmission of message (*confidentiality*). Authentication and authorization can be mutual between the interacting peers (e.g., SpeakEasy) or a central peer can be used (e.g., Jabber). The reference architecture presented in PASTA does not elaborate on the security-related concerns.

6.4 P2P Reference Architecture

This thesis uses the term *servent* created by combining the first three letters of the word *server* and the last three letters of the word *client*, to refer to components which provide a service and can be used to access a similar service provided by another peer. A servent can act both as a client and a server. As defined in section 1.3 all the peers in a P2P network are equivalent in functionality and can act both as a client and a server. A peer needs to act both as a consumer (client) and provider (server) for all the concerns mentioned in table 6.2 because of which PASTA handles most of these concerns using servents.

Figure 6.1 presents a high-level overview of PASTA. PASTA consists of three key components: common runtime, security and core servents. The common runtime provides the messaging infrastructure used to exchange messages between peers in order to invoke (*access*) a service. The common runtime is used by all types of servents to communicate with servents on other peers. The security component handles the security-related concerns such as authentication, authorization and the secure transmission of messages. We suggest using aspects [64, 1] to implement the security-related concerns. PASTA does not define the structure of the security components. Except for routing, the core servents address all the concerns that a P2P application needs to

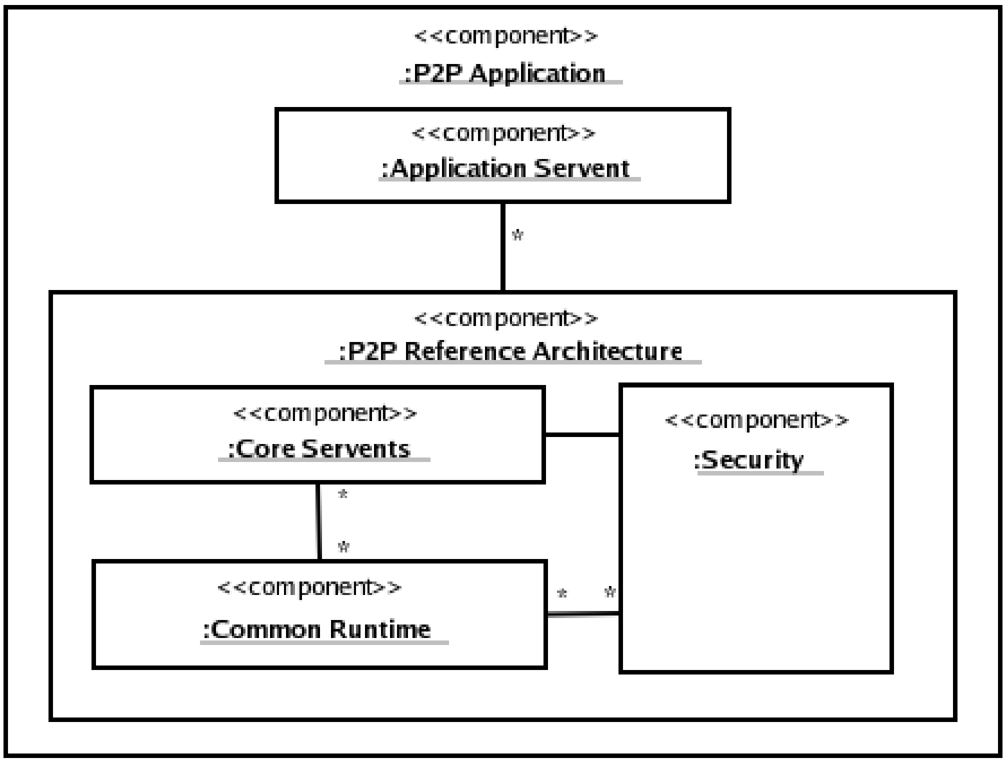


Fig. 6.1: A UML diagram providing a high level overview of the P2P reference architecture.

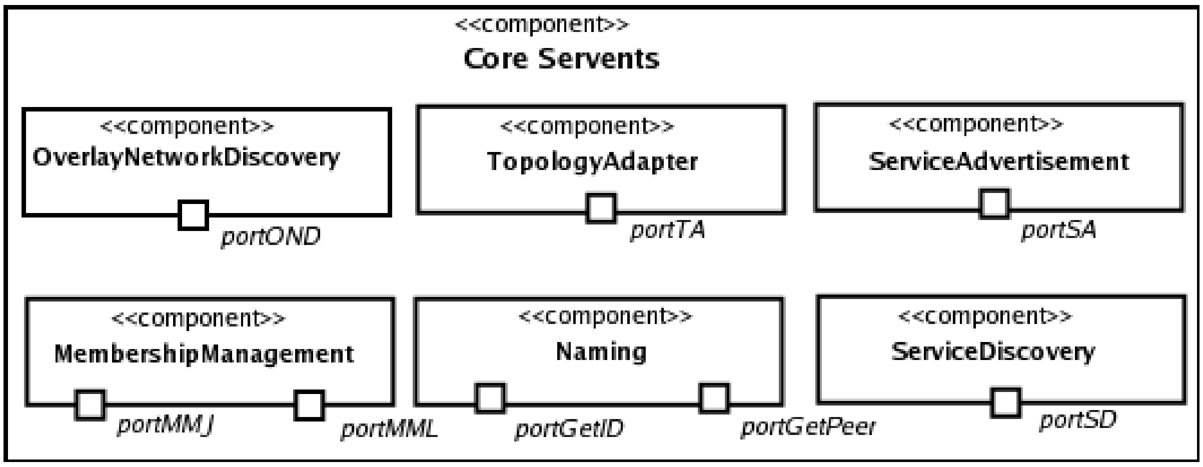


Fig. 6.2: A UML diagram depicting the conceptual view of the core servents of the reference architecture. The core servents handle most of the concerns which need to be addressed by a P2P application.

handle. The routing logic may vary for different services and is handled by the servent for which a request arrives. In PASTA an application consists of application servents (e.g., for file sharing or instant messaging) that are responsible for the application logic. PASTA supports multiple co-existing application servents. This means that an application developed using PASTA can be used to share multiple services. An instance of PASTA can have multiple instances of the core servents, security component and runtime. These instances can be used by an application to connect to different overlay networks.

6.4.1 Core Servents

There are six core servents (see table 6.3) which handle all the concerns except routing and access. Figure 6.2 shows a conceptual view of the core servents. The core servents along with the concerns they handle are shown in table 6.3. A servent can handle incoming messages from both remote and local servents and applications. The protocol

Core Servent	**Concern**
OverlayNetworkDiscovery (OND)	Search
MembershipManagement (MM)	Join (JN), Leave (LV)
ServiceAdvertisement (SA)	Advertise
ServiceDiscovery (SD)	Discovery
Naming (N)	Identification (ID), Resolution (RES)
TopologyAdapter (TA)	Topology

Table 6.3: The core servents of the reference architecture and the concerns they handle. The access concern is handled by the client and server components of the common runtime. The routing concern is handled by the individual servents.

used by the core servent ports is shown in table 6.4 using the notation suggested in [55]. The table does not show implementation-specific details. The protocols define the structure of the messages exchanged to invoke these servents.

Overlay Management

In this section we discuss the core servents that address the concerns related to the overlay management group. The OverlayNetworkDiscovery (OND) servent is used to obtain already connected peers on the overlay network to which a new peer can connect. The port portOND of the OND servent expects a single incoming message *getPeers* with the new peer's details and responds with information about a list of peers to which the new peer can connect. The MembershipManagement (MM) servent is used to handle requests to join and leave the overlay network. The port portMMJ of the MM servent expects an incoming message *joinOverlayNetwork* with the new peer's details. Upon receiving the *joinOverlayNetwork* message, the servent can reject, accept or redirect the new peer's request. The port portMML of the MM servent is used to leave the overlay network and expects a single incoming message called *leaveOverlayNetwork*. The TopologyAdapter core servent used to handle the topology concern is discussed in detail in section 6.4.3.

portMMJProtocol incoming joinOverlayNetwork(newPeerInformation) outgoing redirect(PeerInformation[]) reject accept(PeerInformation[])	**portONDProtocol** incoming getPeers(newPeerInformation) outgoing PeerInformation[]	**portSDProtocol** incoming findService(name, criteria) outgoing ServiceDetail[]
portSAProtocol incoming advertise(ServiceDetail) outgoing success fail	**portGetIdProtocol** incoming getId(ServiceOrResource Description) outgoing id	**portGetPeerProtocol** incoming getPeer(id) outgoing PeerInformation
portMMLProtocol incoming leaveOverlayNetwork outgoing	**portTAProtocol** incoming registerServentCache(serventCacheName, topologyAdaptationPolicy) outgoing accept reject	

Table 6.4: The protocol obeyed by the ports of the core-servents in the reference architecture.

The topology adaptation algorithm can be applied at the OND and MM core servents, so that a new peer joining the network establishes connections to peers on the overlay network in such a fashion that the desired topology is maintained. For a new peer, bootstrapping to the overlay network is a challenging task because of lack of information about all the peers currently connected to the network. The view taken in PASTA is that it is not appropriate to perform topology adaptation at this stage since it will make it even more difficult for a new peer to connect to the overlay network.

Service Management

This section describes the core servents that address the core concerns belonging to the service management group. The access concern is handled by the common runtime component discussed in section 6.4.2. A peer wishing to advertise its services sends an *advertise* message to the port portSA of its ServiceAdvertisement (SA) servent. The SA servent may advertise the service detail to the SA servent of the other peers on the overlay network and responds back to its peer with a *success* or *fail* message. The ServiceDiscovery (SD) servent is used to discover the details of services on the overlay network that match a search criterion. The portSD port of the SD servent expects a single incoming message *findService* with the search criteria and responds with the details of a list of services matching the search criteria. The service details could include information such as the network address and geographical location of the peer. The SD servent provides the search functionality specified in table 3.1 that can be used to search for peers with a given property on the overlay network. As mentioned earlier the routing is handled by the servent for which a request arrives.

Naming

Core servents that address the concerns belonging to the naming group are discussed in this section. The Naming (N) servent is used to assign a unique identifier to entities

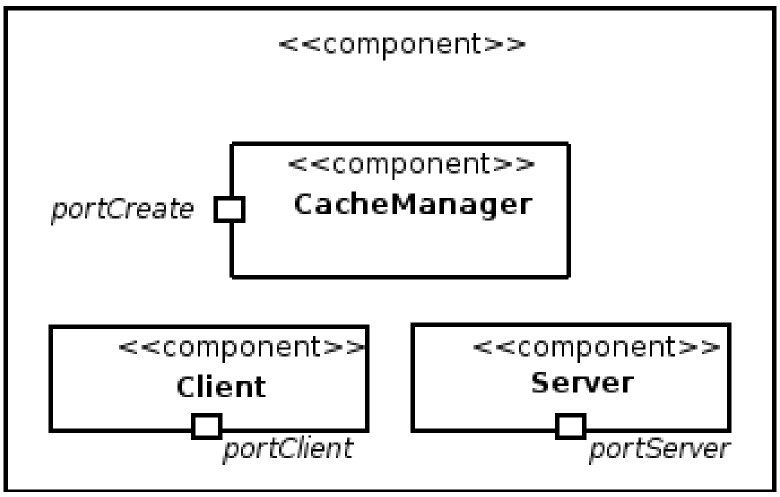


Fig. 6.3: A UML diagram presenting a conceptual view of the runtime component.

in a P2P system (using portGetID) and to resolve an entity's id to a network address (using portGetPeer).

6.4.2 Common Runtime

The common runtime can be further decomposed into three components (see figure 6.3) which are: Client, Server and CacheManager. The server component receives messages from the network and dispatches them to the appropriate servent. The client component helps servents to invoke services provided by servents on other peers. Together the Client and the Server components handle the access concern. Table 6.5 shows the protocols used by the ports of the common runtime components. The portClient port of the Client component is used to send a given Message to a destination address. The Message format is determined by the protocols used by the port to which the message is destined. The portServer of the Server component is used to receive messages. The server directs an incoming messages to an instance of the servent type (specified in destination Address) for which the message is intended.

portCreateProtocol	**portClientProtocol**	**portServerProtocol**
incoming	incoming	incoming
createServentCache(serventCacheName, setOrKeyValuePair, topologyAdaptation-Policy)	sendMessage(dstnAddress, Message)	receiveMessage(Message)
outgoing	outgoing	outgoing
ServentCache	success	
	fail	

Table 6.5: The protocol obeyed by the ports of the runtime components.

CacheManager and Servent Cache

The CacheManager component shown in figure 6.3, is responsible for maintaining a cache called the servent cache. As defined in section 1.4.3, the topology of an overlay network is the graph in which the peers are the vertices and the connections between them are the edges. The servent cache of a peer P constitutes its local view of the overlay network's topology and provides information (such as the network address of the peer) about all the peers to which P is connected. The aim of the topology adaptation algorithms is to change these connections so that a desired global topology can be created. The servent cache may also contain information about other peers on the network, and this information may be used to find peers with a desired property with which P might like to establish a connection.

The servent cache can also be used to process both incoming and outgoing requests for a service. For processing requests the servent cache may be implemented as a table of (key, value) pairs. The key is an application specific entity (e.g., file id for a file-sharing application) that is required to process the request. The value contains information such as the details of a peer that may process the request and cached response to the request for a given key.

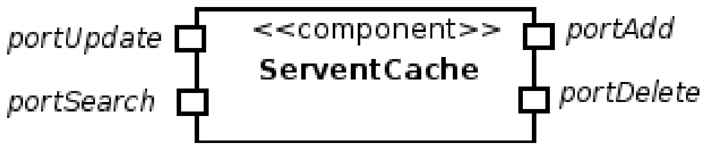


Fig. 6.4: A UML diagram presenting a conceptual view of the ServentCache component.

portAddProtocol	**portSearchProtocol**
incoming	incoming
addEntry(	search(
serventCacheEntry)	criteria)
outgoing	outgoing
success	serventCacheEntry[]
fail	
portUpdateProtocol	**portDeleteProtocol**
incoming	incoming
updateEntry(	deleteEntry(
serventCacheEntry)	serventCacheEntry)
outgoing	outgoing
success	success
fail	fail

Table 6.6: The protocol obeyed by the ports of the ServentCache component.

A servent instance may ask the CacheManager to create a servent cache of a desired type (set or table). The data stored in a servent cache will vary with the service. For example, for a file sharing service the servent cache can be a set of (file id, host) pairs. One servent cache can be shared across multiple servents. Table 6.5 presents the protocols for the CacheManager component. The port portCreate of the CacheManager component is used to create a servent cache. In the incoming message, the port expects a name for the ServentCache, the topology adaptation step to be used on the servent cache and the description of the servent cache (set or table of (key, value) pairs). The port creates a new servent cache if one with the given name and properties does not exist and then returns a reference to it.

Figure 6.4 presents the component diagram of the ServentCache component and table 6.6 presents the protocols. The portAdd, portUpdate, portDelete, portSearch ports of the CacheManager are used to add, update, remove and search entries from a servent cache.

6.4.3 Topology Adaptation

The ServentCache contains information about the peers on the overlay network to which the local peer is connected. A request for a service would be redirected to one or more of these connected peers if the local servent handling the request can not handle it. The TopologyAdapter servent works on the servent cache. Its role is to ensure that a peer is connected to other peers which will help in an optimal processing of the service.

PESTO presented in section 3.3 is a possible approach to implement this component. The concrete realizations of PESTO described in the previous two chapters can be used to manage the ServentCache so that a desired global topology is created. PASTA provides support for the operations described in table 3.1 to implement these algorithms. The concrete instantiations can use the search facility offered by the SD servent and the servent cache to find peers with a desired property. Neighbors can be added and removed by using the portAdd and portUpdate of the ServentCache. The portSearch of the ServentCache can be used to find connected peers with a desired property.

As discussed in sections 2.1 and 2.2, the optimal topology depends on the application. PASTA allows multiple application servents with different topology requirements to co-exist in an application. In PASTA, multiple topologies are allowed in the application through the topology subsystem, which incorporates the topology adapter core servent, the CacheManager and the ServentCache components. An application can have multiple instances of the topology adapter and ServentCache servents.

6.4.4 Generic Servent

Figure 6.5 shows the conceptual view of a generic P2P servent. During initialization, a servent may do either of the following:

- advertise its services using the port portSA of the ServiceAdvertisement servent,
- use the port portGetId of the naming servent to get an id for the servent,
- use the port portAdd of the CacheManager to obtain a ServentCache.

The Implementation component provides the actual service implementation. The Implementation component uses a ServentCache (if used by the servent) to find an existing response for an incoming request. If a response is available from the ServentCache, then it is returned. However if a cached response is not available, then the Implementation component tries to process the request locally, and send a response. If it can not process the request locally then it invokes the RemoteAccess component that uses other peers on the overlay network to process the request and generate a response. The Implementation component directs a request to the Router component for processing if the request's destination is not the servent's peer. Before returning the response, the Implementation component updates the ServentCache (if required) by using its portUpdate.

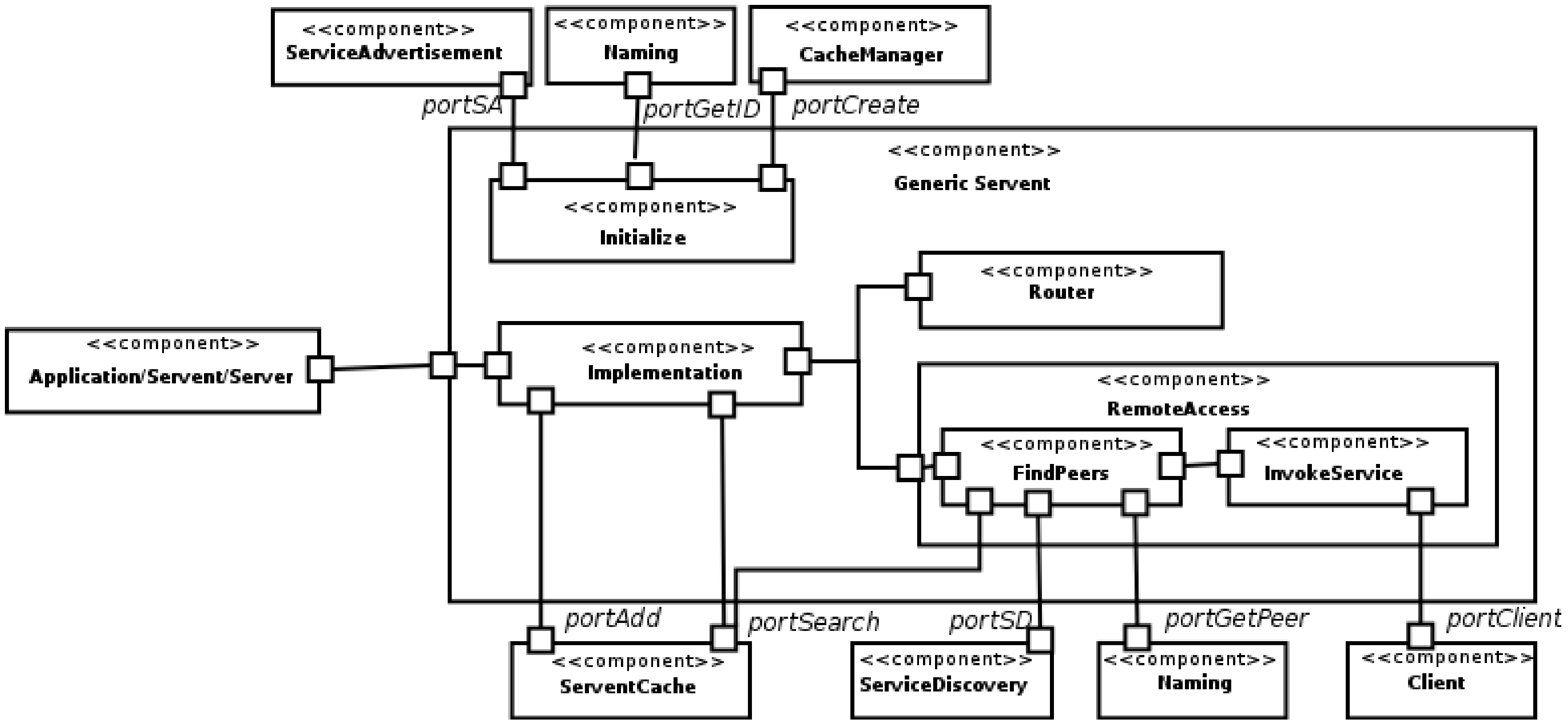


Fig. 6.5: A UML diagram of the conceptual view of a generic servent. All the connectors depict local method invocation. The Routing component may use the Client component.

The RemoteAccess component of a servent is used to process a service request by using other peers on the overlay network if it can not be processed locally by the servent. The RemoteAccess component uses its FindPeers component to find details of peers hosting similar servents that can be used to process the request. The FindPeers component uses the ServentCache and/or the ServiceDiscovery servent to find these peers. The FindPeers component uses the port portGetPeer of the Naming servent to resolve a peer id to its IP address. The InvokeService component then processes the incoming request using the servents on the peers supplied by the FindPeer component.

6.4.5 Interoperability

Interoperability is the problem of ensuring that the several P2P applications can interoperate with each other. As discussed in section 1.2, interoperability is a forthcoming challenge for the developers of P2P applications. Existing P2P systems such as JXTA [67] try to solve the interoperability problem by defining standards (format and on-wire form of the message) for the messages exchanged between peers. However messaging standards alone do not ensure interoperability. For example, a peer (P) might not implement the SA servent and instead use a multicast on the overlay network policy for the SD servent to discover services on the overlay network. An overlay network (ON) with a small number of peers might expect its peers to use a policy of multicast to all known peers for the SA servent and to use the information provided by the SA servent to perform service discovery (SD). When P joins ON, messaging standards, for example those set by JXTA, ensure that peers on ON can understand the SD messages sent by P. However, the peers on ON will not know how to process the messages sent by peer P.

PASTA complements the notion of using messaging standards to ensure interoperability by allowing an overlay network to be defined in terms of the policies it uses to implement the different components of the reference architecture. Applications based on PASTA can be designed so that they allow a new peer to join an overlay network only if it has the set of instances of the reference architecture elements that implement the policies that the overlay network desires. This will ensure that only the peers that can interoperate with other peers join an overlay network. Further if the applications uses a plugin-based approach for implementing the different components then the new peer may obtain the plugins that provide implementation for components with the desired policies from a central repository or a search on the existing overlay networks to which it is connected, so that it may join the overlay network.

6.5 Validation of PASTA

As discussed in section 1.2.2 a reference architecture can be used to design new applications and it provides a vocabulary that can be used to describe and compare the structure of existing applications. All the components of PASTA are abstractions that can have multiple instances implemented using different policies. The existing applications can be described using PASTA by identifying the application servents and the policies that will be used to implement the different components of PASTA. This section describes a range of existing P2P applications and middlewares using PASTA.

Based on the application domain, the existing P2P applications can be divided into three major categories: parallel computing, content management and collaborative [73]. Parallelizable P2P applications split a large computation-intensive task into smaller sub-pieces that can execute in parallel over a number of independent peer nodes. Content Management applications focus on storing and retrieving information from various peers in the network. Collaborative applications allow users to collaborate in real time, without relying on a central server to collect and relay information.

In this section we first describe Gnutella which is an influential content management system. This is followed by Freenet which is one of the earliest content management system that provides censorship ability. We then describe Jabber an application for collaboration that can also be used as a middleware for developing collaborative applications. Finally we describe JXTA which is an influential multi-purpose P2P middleware. JXTA has been used to develop JNGI [7] a parallel computing application.

For each P2P system the routing algorithm and the ServentCache structure is described. The routing policy and the ServentCache used by all the servents in the P2P system is documented using tables.

6.5.1 Gnutella

Gnutella [4, 92] is a decentralized P2P file-sharing network. Gnutella creates an unstructured, self-organizing overlay network of peers. There is no constraint on the position of the files. A peer uses a breadth-first search on the overlay network to locate a file. Once the peer containing the file is located, a direct connection is used to transfer the file.

All the peers in Gnutella implement two application servents which are: *File getFile(String fileName)* which is used by other peers to download files and *PeerInformation getPeers()* which is a network service used by a connected peer to discover other peers on the overlay network. Table 6.7 shows the policies used by Gnutella to implement the different core servents. The table also describes the routing policy and the ServentCache used by the different servents. New peers join the Gnutella network using an available peer from a list of well-known peers which are connected to the Gnutella overlay network. When the new peer joins the network, the peer accepting the connection provides the new peer with information about other peers on the network. Naming service is provided by the Gnutella application. The application generates a unique 16-byte identifier for each new message.

ServiceDiscovery is implemented as a network service. The ServiceDiscovery, OverlayNetworkDiscovery and $getPeers()$ servents share a single ServentCache which is a

ServentCache	
Name	*Description*
PeerCache (GPC)	Set of Peer Information

Routing Policy

Name: Multicast (GMC)

Algorithm:

```
if (isRequest(message)) {
  if (!hasSeen(message.id)) {
    message.TTL (Time to live) = message.TTL - 1
    message.TTL ! = 0 ? forwardMessage(GPC - message.peer) :
    sendMessageResponse(message.peer)
    cache(message.id and message.peer mapping)
  }
} else {
  if (hasSeen(message.id)) {
    forwardMessage(findPeerWhichSendRequest(message.id))
  }
}
```

Application Servents		
Name	*Routing Policy*	*Servent Cache*
PeerInformation getPeers()	GMC	GPC
File getFile(String fileName)	none	none

Core Servents			
Name	*Policy*	*Routing Policy*	*Servent Cache*
OND	Well-known peers	none	GPC
MM	Peers on the overlay network	none	GPC
SA	none	none	none
SD	Multicast to peers in PeerCache	GMC	GPC
N(ID)	none	none	none
N(RES)	none	none	none
TA	none	none	none

Table 6.7: Describing Gnutella using PASTA.

set of addresses of the peers on the overlay network. A peer searches for a desired file it wants by using the ServiceDiscovery servent. The ServiceDiscovery servent sends out a multicast message containing a search request for a getFile service, which can provide a file matching the peer's requirements to all the peers in the ServentCache. A peer receiving the request responds with its address and a list of files matching the search query. If the time to live for the request has not elapsed, then the peer forwards the request to all the nodes in its ServentCache. The $getPeer()$ network service is accessed by sending a multicast to all the peers in the ServentCache. The peer receiving the request responds with information about itself. If the request has not reached its time to live then the peer propagates the request to the peers in its ServentCache.

6.5.2 Freenet

Freenet [36, 2] is a censorship-proof P2P content storage and retrieval system. It provides complete anonymity to the publishers and users of content. Freenet is designed to reduce the *slashdot effect*[1], by caching files on multiple nodes closer (in terms of node hops) to the content's users. All data stored on Freenet is associated with a key.

Table 6.8 presents the ServentCache and the routing policy used in Freenet. All the peers in Freenet implement two application servents which provide network services. They are $DatagetData(Keyk)$ which returns data for a given key, and $voidinsertData(key, data)$ which is used to publish data associated with a key. Table 6.9 shows the policies used by Freenet to implement the core servents of PASTA. The table also describes the routing policy and the ServentCache used by the different core servents. New peers join the Freenet network using an available peer from a list of well-known peers which are connected to the Freenet overlay network. When a new peer joins, the existing peers work together to assign the new node a portion of the key space to manage. The

[1] *Slashdot effect* refers to the slowing down or temporarily closing of a web-site because of a heavy influx of web-traffic caused by a mention of the web-site on Slashdot, a popular technology news and information site.

ServentCache	
Name	*Description*
DataCache (FDC)	table of (dataKey, (hostAddress, data)) tuples

Routing Policy

Name: Freenet Routing Algorithm (FRA)

Algorithm:

```
if (isRequest(message)) {
  if (FDC.hasEntryForKey(message.dataKey)) {
    if(FDC.hasDataForKey(message.dataKey)) {
      sendMessageResponse(message.peer)
    } else {
      forwardMessage(FDC.getHost(message.dataKey))
      cache(message.id and message.peer mapping)
    }
  } else {
    if (FDC.closestKey(message.dataKey) == thisPeer.id) {
      sendMessageResponse(message.peer)
    } else {
      cache(message.id and message.peer mapping)
      forwardMessage(FDC.getHost(FDC.closestKey(message.dataKey)))
    }
  }
} else {
   if (hasSeen(message.id)) {
      forwardMessage(findPeerWhichSendRequest(message.id))
   }
}
```

Table 6.8: The ServentCache and the Routing Policy used in Freenet.

Application Servents		
Name	*Routing Policy*	*Servent Cache*
Data getData(Key k)	FRA	FDC
void insertData(key k, data d)	FRA	FDC

Core Servents			
Name	*Policy*	*Routing Policy*	*Servent Cache*
OND	Well-known peers	none	none
MM	Key space to manage	FRA	FDC
SA	none	none	none
SD	none	none	none
N(ID)	unique id generated by consulting existing peers on the network	FRA	FDC
N(RES)	none	none	none
TA	Prune ServentCache	none	none

Table 6.9: Describing Freenet Using PASTA.

Naming servent generates a unique key for files to be stored on Freenet using SHA-1 hash function. The SHA-1 algorithm ensures that same key is generated for identical files. Freenet does not allow the resolution of the identifier to a network address to ensure anonymity of the peer storing the file.

The *getData*(...) and *insertData*(...) servents share a single ServentCache which is a table of ($dataKey$, ($hostAddress$, $data$)) tuples. The ServentCache is pruned by the TopologyAdapter to ensure that its size does not exceed a specified limit. A least recently used algorithm is used to remove the data for a key. The host address is still cached so that the data can be retrieved at a later time if required.

The *getData*(...) and *insertData*(...) servent implementations propagate a request to the host (in the ServentCache) whose key is closest to the key in the request. The *getData*(...) request is not propagated further, and a response is sent back if the key in the request is already present in the ServentCache along with the data, otherwise the request is forwarded to the host in the ServentCache whose key is closest to the key in the request. The *getData*(...) response is sent back to the peer which forwarded the request which in turn sends the response back to the peer from which it received the request. The response is not directly sent back to the peer from which the request originates in order to preserve anonymity of the peer interested in the resource. The *insertData*(...) request is not propagated further and data is added to the ServentCache if the id of the peer is closest to the key in the request than any other key already present in the ServentCache, otherwise the request is forwarded to the host in the ServentCache whose key is closest to the key in the request.

6.5.3 JXTA

Described in section 2.3.2, JXTA [8, 80] is a specification for developing P2P applications. The specifications can be implemented as a framework which can be used for P2P application development. The Project JXTA reference implementation [121] is

JXTA Protocol	**PASTA Elements**
PDP	ServiceDiscovery, ServiceAdvertisement
PRP	Client, Server
RVP	Application Servents
ERP	Application Servents
PIP	Application Servents

Table 6.10: A mapping between the JXTA protocols and PASTA.

one such framework. As discussed in section 2.3.2, JXTA can not be classified as a reference architecture because it does not describes and defines the components of an application. It concentrates on providing a messaging standard for P2P applications so that they can interoperate.

JXTA specifications define a set of six protocols for P2P applications. JXTA protocols work together to perform services required by a peer. The protocols use XML schemas to describe the format of the messages exchanged between peers to offer a service. JXTA is policy agnostic and does not specify how the service provided by a protocol will be implemented. Table 6.10 shows a mapping between the JXTA protocols and PASTA. The Peer Resolver Protocol (PRP) defines the structure of XML request and response messages which are exchanged between peers. The rest of the protocol messages are embedded within a PRP message. The client and server component in the common runtime can use this as the format of the messages exchanged between peers. A peer can advertise and discover resources and services on the overlay network using Peer Discovery Protocol (PDP). The ServiceDiscovery and ServiceAdvertisement elements can exchange messages using the format specified in the PDP protocol.

There is no direct mapping possible between the next three protocols and PASTA elements. An application that desires the services supported by these protocols can implement the desired services as application servents. These JXTA protocols can be used by the application servents. The rest of this paragraph describes the three protocols that do not have a direct mapping with PASTA. The Peer Information Protocol (PIP) in JXTA, that can be used to find the details about a peer. In JXTA the client may use Rendezvous Protocol (RVP) to find peers if the message has to be propagated over the overlay network. PASTA does not provides a dedicated servent to find peers that can propagate a message on the overlay network. This has been done to keep the list of core servents minimal. In PASTA a servent can use the ServentCache or the ServiceDiscovery servent to find peers which can propagate a message over the overlay network. If JXTA cannot find a direct connection to a destination peer then the Endpoint Routing Protocol (ERP) is used to find intermediate hosts which can route the information to the destination peer. In PASTA routing is the responsibility of the servent handling the message. PASTA does not stipulate a specific service that is responsible for finding peers that can be used to route messages and suggests using the ServiceDiscovery component to find peers that can route the message.

6.5.4 Jabber

Jabber [16, 21, 39] provides specifications for developing instant messaging (IM) applications. Jabber implementations provide a realization of these specifications. Jabber is intended for IM applications, but the infrastructure provided by it can be used for developing other types of P2P applications.

Table 6.11 shows the application servents that Jabber provides and describes how Jabber implements the core servents. Jabber uses a hybrid P2P architecture. The Jabber server is used for authenticating peers and resolving identifiers to network addresses. However, peers can exchange data directly by establishing connections which are brokered using Jabber servers. A new peer connects to the central server to join the overlay network. The central server provides the connecting peer with information (e.g., availability of peer and peer address) about other peers in which the connecting peer is interested. The Jabber server provides three presence re-

Application Servents
void receiveMessage(String mesg)
boolean subscribeToPresenceInformation(String peerId)
String[] getSubscribedPeers()
boolean getPresenceInformation(String peerId)

Core Servents	
Name	Policy
OND	central server
MM	online peers
SA	none
SD	none
N(ID)	central server
N(ID)	central server
TA	none

Table 6.11: The application servents provided by Jabber and the policies used by Jabber to implement the different core servents. The servents in Jabber do not use a ServentCache. The servent implementations in Jabber do not have routing capability.

lated services: $subscribeToPresenceInformation(...)$ which can be used to tell the Jabber server about interest in a peer's (with id peerId) availability information, $getSubscribedPeers()$ gives a list of all the peers in which the peer invoking this service is interested and $getPresenceInformation(...)$ can be used to receive presence information about a peer (with id peerId) on the network. All the peers implement a service called $receiveMessage(...)$ which can be used to send a chat message to them.

6.5.5 Conclusion

This section has successfully described a range of existing applications and middlewares using PASTA. The applications can be compared using the basis of the policies that they use to implement the different PASTA components. Currently we think that the core servents in PASTA provide the common services required by a range of applications. However, if required later the core servents may be extended to include new servents that provide services such as a peer information service specified by PIP in JXTA. We have not used PASTA to design an application, but we feel that it should be easy as it can be used to describe the design of a variety of existing applications and middlewares.

6.6 Comparison with Essence of P2P

Presented in section 2.3.1 "The Essence of P2P" [19] is an existing reference architecture for the P2P domain. The concept of reference architecture has been defined and its advantages has been discussed in section 1.4.2. Both PASTA and "The Essence of P2P" can be used to describe and compare the structure of an existing application and also used as a starting point for developing the software architecture of a new application. A key difference between "The Essence of P2P" and PASTA is that PASTA has been validated (see section 6.5) by using it to describe the structure of existing P2P applications. PASTA also complements “The Essence of P2P” by providing support for topology adaptation and by taking a service-centric view.

As discussed in sections 1.2 and 2.1, the topology of an application affects its performance. As defined in section 1.4 topology adaptation involves creating and maintaining a desired topology and is a major challenge for P2P application developers. “The Essence of P2P” does not specify how topology adaptation can be performed in a P2P application. The primary contribution of this thesis is PESTO an abstract algorithm for designing topology adaptation algorithms. PASTA builds on PESTO by providing a template for developing applications that support topology adaptation.

A service-centric view has been taken to simplify PASTA because the notion of services lends itself well to modelling P2P applications and middlewares. A P2P overlay network is used to share both resources and services among peers on the overlay network. However resources can be accessed through services and so PASTA takes a service-centric view. In PASTA peers on the overlay network work together to use services offered by each other. The shared services can be used to access resources shared on the overlay network. The service-centric view is another key difference between PASTA and “The Essence of P2P”.

6.7 Summary

This chapter identified the core concerns that a P2P application needs to address. The chapter presented PASTA, a reference architecture for the P2P domain. PASTA uses a service-oriented approach that facilitates the development of a software implementation (e.g., a P2P file-sharing application) because the software implementation can use the support for service-oriented architecture provided by the existing middlewares. The chapter validated PASTA by using it to describe four prominent P2P applications and middlewares.

While PASTA is a general reference architecture it can also be used as a template for developing applications that support topology adaptation using PESTO. Chapters 4 and 5 presented three concrete realizations of PESTO that can be used for topology adaptation. These three algorithms are possible solutions that can be used to implement the TopologyAdapter core servent in PASTA, to design applications that require a hub or a clustered topology.

Chapter 7

Conclusion

This chapter presents the conclusion of the work presented in this thesis. We first list the achievements of the thesis and then describe future work. Finally we present the closing remarks.

7.1 Achievements

The overlay network's topology is important in P2P applications because it affects the performance of tasks such as routing of messages and search on the overlay network. Topology adaptation algorithms can be used to create and maintain a desired topology. As discussed in section 1.2, application developers have until now developed topology adaptation algorithms from scratch. The lack of a central peer, makes it challenging to design algorithms that can create a required topology. The flux of peers on an overlay network makes it difficult to maintain the topology as required by an application. This thesis presented an abstract algorithm PESTO that can be used as a template for designing self-organizing topology adaptation algorithms for decentralized unstructured P2P overlay networks. PESTO was compared with the only other approach from the literature that can be used to create a topology adaptation algorithm and was found to be more flexible.

This thesis has shown how PESTO can be used to create a family of topology adaptation algorithms. Concrete realizations of PESTO were presented to validate its utility as a template for designing topology adaptation algorithms. Two algorithms called the SelflessClustering and the SelfishClustering based on PESTO were presented in chapter 4 that can be used to create a clustered topology. An algorithm called the HubAlgorithm based on PESTO was presented in chapter 5 that can be used to create a hub topology. A possible approach to constructing a concrete realization of PESTO to create a power-law topology was presented in section 3.3.

The SelflessClustering, the SelfishClustering and the HubAlgorithm are self-organizing in nature. Algorithms were validated using simulations, which showed that the algorithms can create the desired topologies and maintain them under a continuous arrival of peers joining the network. The simulation results show that the clustering algorithms and the HubAlgorithm converge within 10 simulator iterations, which is a good result.

As discussed in section 1.2.2, the P2P domain has reached a level of maturity that a reference architecture should be proposed for it. PESTO is presented in the context of a reference architecture PASTA that can be used by application designers as a starting point for developing the software architecture of P2P applications that supports topology adaptation. In PASTA, the topology adaptation core servent is responsible for topology adaptation, and PESTO can be used to implement this core servent. PESTO with PASTA provides useful abstractions for designing applications that support topology adaptation.

While PASTA allows for topology adaptation, it is a generic reference architecture that can be used to describe and compare the structure of existing applications. PASTA has been validated by describing existing applications and middlewares. PASTA was compared with another contemporary P2P reference architecture and was found to be an improvement over the existing reference architecture by providing support for topology adaptation and by taking a service-centric view. The latter enhancement facilitates the implementation of an application designed using PASTA because an existing middleware that provides support for Service-oriented architecture could be used to implement the application.

7.2 Future Work

A limitation of the simulations is that we have not evaluated the scenario when peers leave the overlay network. In the PESTO-based topology adaptation algorithms presented in this thesis, a peer periodically checks its neighbors to decide if it is satisfied with them. A peer can execute topology adaptation steps if it is dissatisfied because of its neighbors leaving the overlay network. We expect that the topology adaptation steps would maintain the desired topology in this case also. However simulations are required to estimate the rate of departure of peers under which the algorithms can perform effectively.

The topology adaptation algorithms presented in this thesis depend on the choice of parameters such as $PNSP_{desired}$ for the clustering algorithms and H_{max} for the HubAlgorithm. Currently, simulations are the only way to decide the best value for these parameters and this is a major disadvantage of the topology adaptation algorithms in this thesis. Further research is required to design TAS and SC that do not depend on parameters that require simulations to decide their best value.

It is difficult to mathematically analyze decentralized topology adaptation algorithms because of the large number of variables. Section 4.1 presented an analysis of a specific case (when $PNSP_{desired}$ is 100%) of the SelflessClustering and SelfishClustering algorithms and showed that it is difficult to analyze the other cases (i.e., when $PNSP_{desired}$ is less than 100%). A future work based on this thesis is to present mathematical analysis that explains the results from the simulations.

We have not developed an application using PASTA, however while validating the architecture we found that PASTA is generally applicable because it can be easily used to express applications from different application domains. We expect that PASTA would provide a good starting point for developing an application.

As discussed in section 1.2, a reference architecture provides the basis that can be used to solve the interoperability problem, so that peers on different overlay networks using different protocols and policies could interoperate with each other. The interoperability aspect of PASTA has not been evaluated in this thesis and remains a future work. However, we feel that the modular design of PASTA, using components that can be implemented using different policies facilitates interoperability. An application based on PASTA could be implemented using a plugin-based approach so that by changing the plugins for the different components it could be made to easily interoperate with other applications.

7.3 Closing Remarks

The October 2006 issue of the reputed magazine *Communications of the ACM* (CACM) recognizes nature inspired computing as an emerging approach for solving computing problems [68]. As discussed in section 1.4.6 the nature-inspired solutions are interesting because typically they are self-organizing. This thesis has proposed a solution to designing topology adaptation algorithms using the emerging approach of nature-inspired computing. The solution is based on an agent-based model proposed by Thomas Schelling and is one of several agent-based models in literature. We hope that this thesis motivates exploration of other agent-based models for solving P2P and distributed system problems.

Bibliography

[1] *The aspectj programming guide*, http://www.eclipse.org/aspectj/doc/released/progguide/index.html, last accessed: June 2007.

[2] *Freenet project*, http://freenetproject.org/.

[3] *Gnuplot*, http://www.gnuplot.info/.

[4] *Gnutella*, http://www.gnutella.com/.

[5] *imdb*, http://www.imdb.com.

[6] *Java agent development framework (jade)*, http://jade.tilab.com/.

[7] *Jngi p2p distributed computing framework*, http://jngi.jxta.org/, last accessed: June 2007.

[8] *Jxta v2.0 protocols specification*, http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html, last accessed: June 2007.

[9] *Kazaa*, http://www.kazaa.com.

[10] *Merriam-webster online dictionary*, http://www.m-w.com.

[11] *Napster*, http://www.napster.com, 1999 version, 1999.

[12] *Planetlab*, http://www.planet-lab.org/.

[13] *Skype*, `http://www.skype.com`.

[14] *Soap specification*, `http://www.w3.org/TR/soap/`, last accessed: June 2007.

[15] *Starlogo*, `http://education.mit.edu/starlogo/`.

[16] *What is jabber?*, `http://www.jabber.org/about/overview.shtml`, last accessed: June 2007.

[17] *Wikipedia peer-to-peer*, `http://en.wikipedia.org/wiki/Peer-to-peer`, last accessed: June 2007.

[18] *Isc internet domain survey*, `http://www.isc.org/index.pl`, 2007, last accessed: June 2007.

[19] Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Seif Haridi, and Manfred Hauswirth, *The essence of p2p: A reference architecture for overlay networks.*, Peer-to-Peer Computing, 2005, pp. 11–20.

[20] Lada A. Adamic, Bernardo A. Huberman, Rajan M. Lukose, and Amit R. Puniyani, *Search in power law networks*, Physical Review E **64** (2001), 46135–46143.

[21] DJ Adams, *Programming jabber*, O'Reilly and Associates, 2002.

[22] Abhishek Agrawal and Henri Casanova, *Clustering hosts in p2p and global computing platforms*, Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), 2003.

[23] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi, *Error and attack tolerance of complex networks*, Nature **406** (2000), 378, last accessed: June 2007.

[24] A. L. Barabási, R. Albert, and H. Jeong, *Mean-field theory for scale-free random networks*, Physica A **272** (1999), 173–187.

[25] David Barkai, *Peer-to-peer computing*, Intel Press, 2001.

[26] Salman A. Baset and Henning Schulzrinne, *An analysis of the skype peer-to-peer internel telephony protocol*, `http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0412017`, 2004, last accessed: June 2007.

[27] M. Bawa, G. Manku, and P. Raghavan, *Sets: Search enhanced by topic segmentation*, 26th Annual International ACM SIGIR Conference, 2003.

[28] Sean Blanchfield, *Lattice: An anonymous, scalable peer-to-peer system*, citeseer.ist.psu.edu/blanchfield01lattice.html, last accessed: June 2007.

[29] Bela Bollobas and Oliver Riordan, *The diameter of a scale-free random graph*, Combinatorica **24** (2004), 5–34.

[30] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The unified modeling language user guide*, Addison-Wesley, 1998.

[31] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner, *Experiments on graph clustering algorithmś*, Lecture Notes In Computer Science (2003), 568–579.

[32] Schmidt Douglas C., Rohnert Hans, Stal Michael, and Schultz Dieter, *Pattern-oriented software architecture: Patterns for concurrent and networked objects*, John Wiley & Sons, Ltd, 2000.

[33] Scott Camazine, Nigel R. Franks, James Sneyd, Eric Bonabeau, Jean-Louis Deneubourg, and Guy Theraula, *Self-organization in biological systems*, Princeton University Press, Princeton, NJ, USA, 2001.

[34] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker, *Making gnutella-like p2p systems scalable*, SIGCOMM '03: Proceedings

of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (New York, NY, USA), ACM Press, 2003, pp. 407–418.

[35] Ronald W. Clark, *Einstein: The life and times*, Avon Books, 1994.

[36] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley, *Protecting free expression online with freenet*, IEEE Internet Computing **6** (2002), no. 1, 40–49.

[37] Bram Cohen, *Incentives build robustness in bittorrent*, Proceedings of the Workshop on Economics of Peer-to-Peer Systems, 2003.

[38] Dollimore J. Couloris G. and Kindberg T., *Distributed systems, concepts and design*, Addison Wesley, 2001.

[39] Cathreine Dodson, *Jabber technical white paper*, `http://xml.coverpages.org/`, Aug 2000, last accessed: June 2007.

[40] John Donne, *The works of John Donne*, Parker, 1839.

[41] Gamma E., Helm R. Johnson, and R. Vlissides J., *Design patterns: Elements of reusable object-oriented software*, Addison-Wesley, 1995.

[42] W Keith Edwards, Mark W. Newman, and Jana Z. Sedivy, *The case for recombinant computing*, Tech. report, Xerox Palo Alto Research Center, 2001.

[43] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, Trevor F. Smith, Dirk Balfanz, D. K. Smetters, H. Chi Wong, and Shahram Izadi, *Using speakeasy for ad hoc peer-to-peer collaboration*, CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work (New York, NY, USA), ACM Press, 2002, pp. 256–265.

[44] Joshua M. Epstein and Robert L. Axtell, *Growing artificial societies: Social science from the bottom up*, The MIT Press, 1996.

[45] P. Erdos and A. Rènyi, *On random graphs*, Publicationes Mathematicae (Debrecen) **6** (1959), 290–297.

[46] F. Ke Fessant, S. Handurukande, A. M. Kermarrec, and L. Massoulie, *Clustering in peer-to-peer file sharing workloads*, LNCS **3279** (2004), 217–226.

[47] Brian P. Gallagher, *Using the architecture tradeoff analysis method to evaluate a reference architecture: A case study*, Tech. report, CMU/SEI, 2000.

[48] Jesus Gomez-Gardenes and Yamir Moreno, *Local versus global knowledge in the barabasi-albert scale-free network model*, Physical Review E **69** (2004), 037103.

[49] R. Guimerà, A. Díaz-Guilera, F. Vega-Redondo, A. Cabrales, and A. Arenas, *Optimal network topologies for local search with congestion*, Phys. Rev. Lett. **89** (2002), no. 24, 248701–+.

[50] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., and Clifford, *Introduction to algorithms*, second ed., ch. 24.3, pp. 595–601, MIT Press and McGraw-Hill, 2001.

[51] Ahmed E. Hassan and Richard C. Holt, *A reference architecture for web servers*, WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00) (Washington, DC, USA), IEEE Computer Society, 2000, p. 150.

[52] Hao He, *What is service-oriented architecture?*, `http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html`, Sep. 2003, last accessed: June 2007.

[53] Rainer Hegselmann and Andreas Flache, *Understanding complex social dynamics: A plea for cellular automata based modelling*, Journal of Artificial Societies and Social Simulation **1** (1998).

[54] Michi Henning and Steve Vinoski, *Advanced corba programming with c++*, Addison Wesley, 1999.

[55] Christine Hofmeister, Robert Nord, and Dilip Soni, *Applied software architecture*, Addison-Wesley, 2000.

[56] Christine Hofmeister, Robert L. Nord, and Dilip Soni, *Describing software architecture with uml*, WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1) (Deventer, The Netherlands, The Netherlands), Kluwer, B.V., 1999, pp. 145–160.

[57] Intel, *Peer-to-peer computing in the enterprise*, Tech. report, Intel, 2002.

[58] James Ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl, and Jaime Rodrigo Oviedo Silva, *Documenting component and connector views with uml 2.0*, Tech. report, Carnegie Mellon Software Engineering Institute, 2004.

[59] Mark Jelasity and Ozlap Babaoglu, *T-man: Fast gossip-based construction of large-scale overlay topologies*, Tech. Report UBLCS-2004-7, Department of Computer Science, University of Bologna, 2004.

[60] ———, *T-man: Gossip-based overlay topology management*, Lecture Notes In Computer Science (2006), no. 3910, 1–15.

[61] Gian Paolo Jesi, *Peersim howto: Build a new protocol for the peersim 1.0 simulator*, `http://peersim.sourceforge.net/tutorial1/tutorial1.html`, last accessed: June 2007.

[62] Steven Johnson, *Emergence : The connected lives of ants, brains, cities and software*, Alien Lane, The Penguin Press, 2001.

[63] Rick Kazman, Leonard J. Bass, Mike Webb, and Gregory D. Abowd, *SAAM: A method for analyzing the properties of software architectures*, International Conference on Software Engineering, 1994.

[64] Ramnivas Laddad, *Aspectj in action*, Manning Publications Co., 2003.

[65] Paul Celements Len Bass and Rick Kazman, *Software architecture in practise*, Addison Wesley, 2000.

[66] Bo Leuf, *Peer to peer: Collaboration and sharing over the internet*, Addison Wesley, 2002.

[67] Sing Li, *Making p2p interoperable: The jxta story*, `http://www.ibm.com/developerworks/library/j-p2pint1.html`, Aug 2001, last accessed: June 2007.

[68] Jiming Liu and Kwok Ching Tsui, *Toward nature-inspired computing.*, Commun. ACM **49** (2006), no. 10, 59–64.

[69] Alexander Loser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden, *Semantic overlay clusters within super-peer networks*, Lecture Notes in Computer Science **2944** (2004), 33–47.

[70] Qin Lv, Sylvia Ratnasamy, and Scott Shenker, *Can heterogeneity make gnutella scalable?*, First International Workshop on Peer-to-Peer Systems(IPTPS), 2002.

[71] Jr. Melvin B. Greer, *The web services and service oriented architecture revolution: Using web services to deliver business value*, iUniverse, Inc., 2006.

[72] Microsoft, *Introduction to windows peer-to-peer networking*, Tech. report, Microsoft Corporation, 2003.

[73] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu, *Peer-to-peer computing*, Tech. Report HPL-2002-57R1, HP Labs, 2002.

[74] Remi Monasson, *Diffusion, localization and dispersion relations on small-world lattices*, European Physical Journal B **12** (1999), 555–568.

[75] Alberto Montresor, *A robust protocol for building superpeer overlay topologies*, Tech. Report UBLCS-2004-8, Department of Computer Science, University of Bologna, Italy, May 2004.

[76] Gordon E. Moore, *Cramming more components onto integrated circuits*, Electronics Magazine **38** (1965), no. 8.

[77] M. E. J. Newman, *The structure and function of complex networks*, SIAM Review **45** (2003), 167–256.

[78] M. E. J. Newman and D. J. Watts, *Renormalization group analysis of the small-world network model*, Physics Letters A **263** (1999), 341.

[79] Cheuk Hang Ng and Ka Cheung Si, *Peer clustering and firework query model*, The Eleventh International World Wide Web Conference, 2002.

[80] Scott Oaks, Bernard Traversat, and Li Gong, *Jxta in a nutshell*, O'Reilly and Associates Inc., 2002.

[81] Andy Oram (ed.), *Peer-to-peer: Harnessing the power of disruptive technologies*, O'Reilly, 2001.

[82] George Orwell, *Animal farm*, Nick Hern Books, 1945.

[83] N.H. Packard and S. Wolfram, *Two-dimensional cellular automata*, Journal of Statistical Physics **38** (1985), 901–946.

[84] Michale S. Pallos, *Service-oriented architecture: A primer*, Align Journal (2001).

[85] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal, *Building low-diameter p2p networks*, Annual Symposium On Foundations Of Computer Science **42** (2001), 492–499.

[86] M. P. Papazoglou and D. Georgakopoulos, *Introduction*, Communications of the ACM **46** (2003), no. 10, 24–28.

[87] Paul Valéry, *The collected works of paul valéry*, Princeton University Press, 1969.

[88] Christian Prehofer and Christian Bettstetter, *Self-organization in communication networks: principles and design paradigms*, Communications Magazine, IEEE **43** (2005), no. 7, 78–85.

[89] Lakshmish Ramaswamy, Bugra Gedik, and Ling Liu, *Connectivity based node clustering in decentralized peer-to-peer networks*, Third International Conference on Peer-to-Peer Computing (P2P03), 2003.

[90] S. Redner, *Citation statistics from more than a century of physical review*, `http://www.citebase.org/abstract?id=oai:arXiv.org:physics/0407137`, 2004, last accessed: June 2007.

[91] Mitchel Resnick, *Turtles, termites, and traffic jams: Explorations in massively parallel microworlds (complex adaptive systems)*, The MIT Press, January 1997.

[92] Matei Ripeanu, *Peer-to-peer architecture case study: Gnutella network*, First International Conference on Peer-to-Peer Computing (P2P01), 2001.

[93] Luis Rodero, Antonio Fernndez, Luis Lopez, Jose Antonio Perez, Juan Francisco-Morcillo, and Vicent Cholvi, *Dante: A self-adaptable unstructured peer-to-peer network*, The 25th Conference on Computer Communications (IEEE INFOCOM), 2006.

[94] Winn L. Rosch, *Hardware bible*, Que Publishing, 2003.

[95] Antony I. T. Rowstron and Peter Druschel, *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*, Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (London, UK), Springer-Verlag, 2001, pp. 329–350.

[96] Jan Sacha, Jim Dowling, Raymond Cunningham, and René Meier, *Using aggregation for adaptive super-peer discovery on the gradient topology.*, SelfMan, 2006, pp. 73–86.

[97] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy, *An analysis of internet content delivery systems*, OSDI '02: Proceedings of the Fifth symposium on Operating systems design and implementation (New York, NY, USA), ACM Press, 2002, pp. 315–327.

[98] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble, *Measuring and analyzing the characteristics of napster and gnutella hosts*, Multimedia Systems Journal **9** (2003), no. 2, 170–184.

[99] Thomas C Schelling, *Models of segregation*, American Economic Review **59** (1969), no. 2, 488–93.

[100] ______, *Dynamic models of segregation*, Journal of Mathematical Sociology **1** (1971), no. 2, 143–186.

[101] Mario Schlosser and Sepandar Kamvar, *Simulating a file-sharing p2p network*, Proceedings of the First Workshop on Semantics in P2P and Grid Computing, 2003, `http://citeseer.ist.psu.edu/550289.html`.

[102] Detlef Schoder and Kai Fischbach, *Peer-to-peer prospects*, Commun. ACM **46** (2003), no. 2, 27–29.

[103] Rüdiger Schollmeier, *A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications.*, Peer-to-Peer Computing, 2001, pp. 101–102.

[104] Jeremy G. Siek, Arthur A. Lumsdaine, and Lie-Quan Lee, *The boost graph library: user guide and reference manual*, Addison-Wesley Longman Publishing Co., Inc., 2002.

[105] Paul Silvey and Laurie Hurwitz, *Adapting peer-to-peer topologies to improve system performance*, HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7 (Washington, DC, USA), IEEE Computer Society, January 2004, p. 70199.1.

[106] Atul Singh and Mads Haahr, *A survey of p2p middlewares*, `http://www.cs.tcd.ie/publications/tech-reports/reports.07/TCD-CS-2007-28.pdf`, last accessed: June 2007.

[107] ______, *Topology adaptation in p2p networks using schelling's model*, Proceedings of the First Workshop on Games and Emergent Behaviours in Distributed Computing Environments, colcated with PPSN-VIII, sep 2004.

[108] ______, *Creating an adaptive network of hubs using schelling's model*, Proceedings of the IFIP/IEEE International Workshop on Self-Managed Systems & Services (SelfMan2005), May 2005, last accessed: June 2007.

[109] ______, *Creating an adaptive network of hubs using schelling's model*, Communications of the ACM **49** (2006), no. 3, 69–73.

[110] ______, *A peer-to-peer reference architecture*, Proceedings of the First International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE), Jan 2006.

[111] ______, *Decentralized clustering in pure p2p overlay networks using schelling's model*, IEEE International Conference on Communications (ICC2007), jun 2007.

[112] Ray Solomonoff and Anatol Rapoport, *Connectivity of random nets*, Bulletin of Mathematical Biology **13** (1951), no. 2, 107–117.

[113] Dilip Soni, Robet L. Nord, and Christine Hofmeister, *Software architecture in industrial applications.*, Proceedings of the 17th International Conference on Software Engineering, 1995, pp. 196–207.

[114] Murray R. Spiegel and Larry J. Stephens, *Schaum's outlines of statistics, 3rd edition*, McGraw-Hill, 1998.

[115] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang, *Efficient content location using interest-based locality in peer-to-peer systems*, International Conference on Computer Communications (INFOCOM), vol. 3, 2003, pp. 2166–2176.

[116] Thad Starner, *Thick clients for personal wireless devices*, IEEE Computer **35** (2002), no. 1, 133–135.

[117] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Chord: a scalable peer-to-peer lookup protocol for internet applications*, IEEE/ACM Trans. Netw. **11** (2003), no. 1, 17–32.

[118] Ratnasamy Sylvia, Handley Mark, Karp Richard, and Shenker Scott, *Topologically-aware overlay construction and server selection*, Proceedings of IEEE INFOCOM 2002 (New York, NY), June 2002.

[119] Schelling C. T., *Micromotives and macrobehaviour*, Nortan and Company: W. W. Norton,, 1978.

[120] Thuan L. Thai, *Learning dcom*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999, Designed By-Nancy Priest.

[121] Bernard Traversat, Mohamed Abdelaziz, Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul, Bill Yeager, Ahkil Arora, and Carl Haywood, *Project jxta 20 superpeer virtual network*, `http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf`, last accessed: June 2007.

[122] S. Voulgaris, A. Kermarrec, L. Massoulié, and M. van Steen, *Exploring semantic proximity in peer-to-peer content searching*, Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS04), 2004.

[123] A. Wachowski and L. Wachowski, *The matrix*, 1999.

[124] Duncan J. Watts and Steven H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature **393** (1998), 440–442.

[125] Oscar Wilde, *The picture of dorian gray*, Penguin Books, 1891.

[126] Uri Wilensky, *Netlogo*, `http://ccl.northwestern.edu/netlogo/`, 1999, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[127] Beverly Yang and Hector Garcia-Molina, *Designing a super-peer network*, Proceedings Of the 19th International Conference On Data Engineering, 2003, pp. 49–62.

[128] Hezi Yizhaq, Boris A Portnov, and Ehud Meron, *A mathematical model of segregation patterns in residential neighbourhoods*, Environment and Planning **36** (2004), 149–172.

[129] H. Peyton Young, *Individual strategy and social structure*, Princeton University Press, 1998.

[130] Huia Zhang, Ashish Goel, and Ramesh Govindan, *Using the small-world model to improve freenet performance*, IEEE INFOCOM **3** (2002), 1228–1237.

[131] Junfu Zhang, *A dynamic model of residential segregation*, Journal of Mathematical Sociology **28** (2004), 147–170.

[132] ———, *Residential segregation in an all-integrationist world*, Journal of Economic Behaviour and Organization **54** (2004), 533–550.

[133] Wei Zhang, Shen Zhang, Yi Ouyang, Fillia Makedon, and James Ford, *Node clustering based on link delay in p2p networks*, ACM symposium on Applied Computing (SAC), 2005.