

Nature-Inspired Distributed Computing:
Using Synapse Oriented Networks
for Anonymous Communication

by

Daniel Castro, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2008

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, and that unless otherwise stated, is my own work.

Daniel Castro

September 10, 2008

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Daniel Castro

September 10, 2008

Acknowledgments

Special thanks to Michelle Fung for all her help in the Neuroscience field and for giving me the inspiration and guts to base this dissertation on synapses and Schwann cell functionality. To Katherine Southall, I am eternally grateful for her untiring support, understanding and assurance this dissertation was written in prose of proper English. To my family, without their support and orientation this M.Sc. would not have been possible. Thanks to my supervisor, Dr. Mads Haahr, for his support and orientation, and for believing in the results of this dissertation. Finally, to God, on whose hands I put my life and which I believe is the reason of the good outcome of all things.

DANIEL CASTRO

University of Dublin, Trinity College

September 2008

Nature-Inspired Distributed Computing:

Using Synapse Oriented Networks

for Anonymous Communication

Daniel Castro, M.Sc.

University of Dublin, Trinity College, 2008

Supervisor: Mads Haahr

Nature has millions of years of successful evolution, therefore, the creation of new distributed systems techniques inspired by nature is very promising. This project has identified how neurons regenerate and build connections with help of other supportive cells and it applies those behaviours into peer-to-peer systems providing a way for peers to send anonymous messages between them. Within the privacy area, anonymity is very controversial and it has, so far, only been addressed for specific purposes such as Freenet and Anonymous Remailers do. A prototype has been built that works as a general purpose middleware for peer-to-peer applications that require a high degree of anonymity between peers. The results, in terms of anonymity, were successful but scalability challenges arose and were left for future work. The mapping of two very different domains (Neuroscience and Distributed Systems) was achieved. The amount of work was substantial with very interesting results that could inspire many other applications in the future.

Contents

Acknowledgments	iv
Abstract	v
List of Tables	x
List of Figures	xi
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 This Dissertation	2
1.3 Road-map	2
Chapter 2 Neuroscience Overview	4
2.1 The Scope within Neuroscience	4
2.2 Basic Terms and Concepts	6
2.3 Neuroplasticity	12
2.4 Central Nerve Regeneration	14
2.5 Peripheral Nerve Regeneration	15
2.5.1 Neuronal Survival and Reaction	15
2.5.2 Injury Types	16
2.5.3 Injury Classification	16
2.5.4 Regeneration	17

Chapter 3	Peer-to-peer Applications Overview	19
3.1	Characteristics of Peer-to-Peer	20
3.2	Components of Peer-to-Peer Systems	24
3.3	Classification of Peer-to-Peer Systems	25
3.4	Overlay Network Structure	27
3.4.1	Unstructured Architectures	27
3.4.2	Structured Architectures	30
3.4.3	Loosely Structured Architectures	33
3.5	Basic Algorithms	33
3.6	FreeNet	34
3.7	Remailers	35
3.7.1	Anonymous Remailers	36
3.7.2	Pseudonymous remailers	40
3.8	Peer-to-Peer Summary	41
Chapter 4	Design	42
4.1	Translation	43
4.1.1	Neuron Peer	43
4.1.2	Schwann Peer	44
4.1.3	Synapse Connection	45
4.1.4	NGF to build routes	45
4.1.5	General Mapping	46
4.2	Architecture of Synapse Oriented Networks	47
4.2.1	Directory Server	48
4.2.2	Peer Interaction	49
4.3	Expected Benefits	56
Chapter 5	Implementation	58
5.1	Prototype	58

5.1.1	Directory Server	59
5.1.2	Node	62
5.1.3	Protocols	64
5.2	API	65
5.2.1	Packages	65
5.2.2	Classes	65
5.2.3	Methods	65
5.2.4	Class Diagram	72
Chapter 6 Evaluation		73
6.1	Domain Mapping Challenges	73
6.2	Anonymity	74
6.3	Scalability and Fault Tolerance	75
6.3.1	Distributed Solution to the Directory Server	76
6.4	Possible Attacks	77
6.4.1	Eavesdropping	77
6.4.2	Data Modification	78
6.4.3	Identity Spoofing	78
6.4.4	Man-in-the-middle	79
6.4.5	Replay	79
6.5	Evaluation of the Implementation	79
6.5.1	Simplicity	79
6.5.2	Portability and Re-usability	80
6.5.3	Real IP Address	80
6.5.4	References to Parent Class	81
Chapter 7 Conclusions		82
7.1	Future Work	83

Appendix A	Abbreviations	84
Appendix B	SON Implementation Code	85
	Bibliography	86

List of Tables

4.1	A general mapping of Neuroscience concepts applied to P2P Systems. . . .	46
5.1	Table of the package distribution for the SON implementation.	65
5.2	Table of all the classes in each package for the SON implementation. . . .	66
5.3	The class only contains one static method used to determine the real IP address of the host under any platform.	67
5.4	The methods contained in the Main class.	67
5.5	The methods contained in the Schwann class.	68
5.6	The methods contained in the AnonymousForwarder class.	68
5.7	The methods contained in the Node class.	69
5.8	The methods contained in the Directory class.	70
5.9	The methods contained in the NodeHandler class.	71

List of Figures

2.1	Basic diagram of a human central nervous system (2) [37], the spinal cord (3) and the brain (1).	6
2.2	Diagram of a human neuron with some of its most relevant parts such as (a) dendrite, (b) Schwann cells, (c) axon terminal, (d) axon, (e) nucleus and (f) soma.	7
2.3	Diagram of a presynaptic and postsynaptic neuron joined by a synapse which shows what happens at the axon terminal and dentritic spine level. .	8
2.4	Wallerian degeneration. (A) Normal connections before an axon is severed. (B) Degeneration following severance of an axon. Degeneration following axonal injury involves several changes: the axon terminal degenerates, myelin breaks down and forms debris, and the cell body undergoes metabolic changes. Subsequently, presynaptic terminals retract from the dying cell body, and postsynaptic cells degenerate. Based on figure 4.2 in Neuroscience: Fundamentals for Rehabilitation [1].	9
2.5	Axonal Sprouting. The new growth of axons following injury involves two types of sprouting: A., collateral sprouting, in which a denervated neuron attracts side sprouts from nerby undamaged axons; and B., regenerative sprouting, in which the injured axon issues side sprouts to form new synapses with undamaged neurons. Based on figure 4.3 from [1].	12
3.1	(1) An example of a centralized topology and (2) a decentralized topology.	21

3.2	File Sharing Overlay Network. A peer-to-peer network showing several computers connected in a random order with a floppy diskette representing data storage. In the example, each peer is sharing CPU, bandwidth and storage resources.	26
3.3	An unstructured hybrid network. It represents the combination of centralized and pure P2P topologies.	29
3.4	An unstructured pure P2P network. It has no dependencies on a central server.	29
3.5	An unstructured centralized P2P network. Search queries are completed relative to a central server, yet connections are created directly between peers.	30
3.6	An example of a CHORD “ring like” topology in which the circles are peers and the rectangles are the resources shared, both identified by a key. The similarity between the peer keys and the resource keys can be appreciated. A peer is responsible for the resources within his scope.	32
3.7	An example of a CAN topology in which the circles are peers and the rectangles are the resources shared. A peer is responsible for the resources within its space in the Cartesian table.	32
3.8	A typical request sequence. Taken from Figure 1 in [25].	35
4.1	Diagram of two Neuron Peers (N1 and N2) involved in a Synapse Connection formed by several Schwann Peers (N3 to N8).	44
4.2	A representation of the interaction of the peer and the Directory Server at peer startup.	49
4.3	The message format used on the Peer startup messages as a request message to the Directory Service.	50
4.4	The message format used by the Directory Server as a response message to assign the peer its unique identifier.	50

4.5	A representation of the interaction between a peer and the Directory Server and the same peer and another peer when sending or forwarding a message. When sending an anonymous message this will happen on every hop the message goes through.	51
4.6	The table shows the routing of an anonymous message that originated at peer 3, the final destination is peer 6. Read left to right, each cell contain NGF values and highlighted is the peer at which the message is at each hop until it reaches its destination. Notice how after a message has been at a peer the NGF value decreases. In figure 4.7 a graphical representation can be seen.	52
4.7	A diagram representing the route of the message from figure 4.6.	53
4.8	The protocol used by the peers to send anonymous messages.	53
4.9	An interaction between a peer and the Directory Server when the peer updates its NGF value.	54
4.10	The protocol used by a peer when notifying the server of a NGF value update.	54
4.11	A representation of the interaction between a peer and the Directory Server when a peer requests a list of nodes.	55
4.12	The protocol used by a peer when requesting an updated list of peers to the server.	56
4.13	The message format used by the server when sending an updated list of peers.	56
5.1	A screenshot of the Main Menu.	59
5.2	A screenshot of the menu for the Directory Server.	61
5.3	A screenshot of the list of Nodes in the Directory Server.	61
5.4	A screenshot of the Node's menu.	62
5.5	A screenshot of the options needed at a Node to send an anonymous message.	63
5.6	A UML class diagram of SON2008.	72

Chapter 1

Introduction

1.1 Motivation

The possibility of borrowing ideas from nature and applying them to computer systems is something that few researchers have ventured to investigate. Some cases gained successful results, such as the *Abstract Schelling Algorithm* written by Sing *et al* [11]. This was based on a model from Sociology provided by Thomas Schelling in which he explains the existence of segregated models in neighborhoods in the United States of America. In a similar way, ideas from Neuroscience have been applied, such as the Blue Brain Project [26] in which a comprehensive attempt to reverse-engineer the mammalian brain is undertaken in order to understand brain function and dysfunction through detailed simulations. Such successful investigations motivate us to look into new ideas for Computer Science borrowed from other fields.

Since the emergence of Freenet [25] in the year 2000, peer-to-peer (P2P) applications have become popular and related research has increased in both academic and industrial fields. The fact that resources can be shared among many peers to create powerful systems with a specific purpose often gives P2P applications a wide range of new requirements that have to be met, and identified issues that need to be addressed. Generally, P2P systems are distributed, adaptive to different environments and they provide a very good

fault tolerance. These are very desirable characteristics of distributed systems. They are also self-organizing systems which involve difficult challenges like scalability. Such complex characteristics have intrinsic issues for which constant improvement is needed. It is therefore appropriate to look at biology, specifically in the Neuroscience field where investigations will be made into how neurons recover from injury and how connections are built with the help of supportive cells.

1.2 This Dissertation

The purpose of this research is to design a new nature-inspired distributed computing technique. Specific aspects of Biology will be analysed and applied to Computer Science. More specifically, this project will be looking into Neuroscience, at how neurons recover from injury in the peripheral nervous system and the supportive cells that work as helpers for neurons to build connections and find paths. Application of those behaviours and techniques into a P2P distributed computer network with the specific purpose of providing a way to ensure anonymity between peers will be attempted.

Another purpose of this document is to satisfy the dissertation requirements of a Master's in Science in Computer Science (Networking and Distributed Systems) degree, for the Computer Science Department in Trinity College Dublin, for the academic year of 2007/2008. It is also intended that the topics presented and how they are merged to different scientific fields will be of interest to all lecturers, academics and students who have any kind of knowledge of distributed systems.

1.3 Road-map

Firstly it is important to provide the necessary theoretical background to understand both Neuroscience and P2P applications aspects. In chapter 2, a Neuroscience Overview is given, provided for people who have never studied such topics before, to give the reader

a clear and simple understanding. Then, in chapter 3, a P2P Networks overview is given. In this case it is expected that the reader is more familiar with the topics therefore more specific details which are relevant to the research are provided.

In chapter 4 the merging of Neuroscience behaviours into Distributed Systems is undertaken and the design of what is called Synapse Oriented Networks is presented. A basic prototype is built which is then explained in chapter 5. Finally, an evaluation of the results are given in chapter 6.

Chapter 2

Neuroscience Overview

This chapter introduces Neuroscience with a view to borrowing concepts and ideas for use in self-organizing P2P networks. It can be thought of as “a computer scientist’s quick and simple guide to Neuroscience”. As it is not expected for the reader to have any background in Biology, a simple and understandable overview is provided, ultimately giving the reader the necessary tools to understand how neurons recover from injury in the Peripheral Nervous System. Once the reader has gained this insight, application of the concepts to P2P networks will seem very natural and straightforward.

2.1 The Scope within Neuroscience

Neuroscience, traditionally classified as a biological science, has recently been classified as several disciplines including cognitive and neuro-psychology, computer science, statistics, physics and medicine. Neuroscience is the study of the nervous system, covering aspects such as development, physiology, function, structure, evolutionary history, genetics, bio-chemisitry, informatics, pharmacology, computational neuroscience and pathology. Such a broad field has spawned many experimental and theoretical investigations of the central and peripheral nervous system. Neuroscience started with empirical methodologies that have evolved through the years and most recent theoretical advances have been aided

by the use of computational modeling. This helped the scientific studies of the nervous systems to increase significantly in the second half of the twentieth century, leading to a better understanding of the complex processes that occur inside and between neurons, and how they form networks. Many of the terms and concepts used in the following paragraphs are explained in detail in the next section of this chapter.

The nervous system, composed of a network of neurons and other supportive cells, form functional circuits responsible for specific tasks. It can be studied at many different levels, ranging from molecular to cellular, and from independent systems to the collective interactions between multiple systems. In *molecular neuroscience*, special interest is given to the mechanisms by which neurons express and respond to molecular signals and how axons form complex connectivity patterns in order to understand how neurons develop and die, and how genetic changes affect biological functions. Also the morphology, molecular identity and physiological characteristics of neurons are of considerable interest. *Cellular neuroscience* studies the mechanisms of how neurons process signals physiologically and electrochemically and how they are processed by the dendrites, somas and axons. It is also concerned with how neurotransmitters help those mechanisms. *Systems neuroscience* is concerned with physiological functions such as reflexes, sensory integration, motor coordination, emotional responses, learning, memory and others, and how they are produced by the circuits formed and used anatomically and physiologically. Finally, *cognitive neuroscience* studies how psychological and cognitive functions are produced by the neural circuitry. This has evolved with the emergence of new powerful measurement techniques that will map specific neural circuitries to specific human cognition and emotion.

As said at the beginning of this chapter, the main concern is to give the reader enough understanding about the Neuroscience concepts that are relevant for this research. Focus is given to in transmitting to the reader those concepts in a familiar language. Giving a computational science point of view should facilitate a contrasted comprehension against distributed systems.

2.2 Basic Terms and Concepts

As this document will be read by computer scientists mainly, it is essential to begin with an explanation of the basic terms and concepts from Neuroscience. This will give the reader the necessary background to understand how they will be applied and contrasted with distributed systems at a later stage. As explained before, the field of Neuroscience is wide, it is therefore impossible to cover everything. Instead, the most relevant concepts for this investigation have been identified.

Central Nervous System (CNS): this consists of the brain and spinal cord. The brain is protected by a hard outer-casing called the skull. The spinal cord is protected by a similar protective structure called vertebrae. As shown in figure 2.1 the brain (1) and the spinal cord (3) together form the CNS (2).

Peripheral Nervous System (PNS): this is the name given to all nerves and neurons that lie outside of the brain and spinal cord (or central nervous system). It is not protected by bone which makes it more vulnerable to injury.

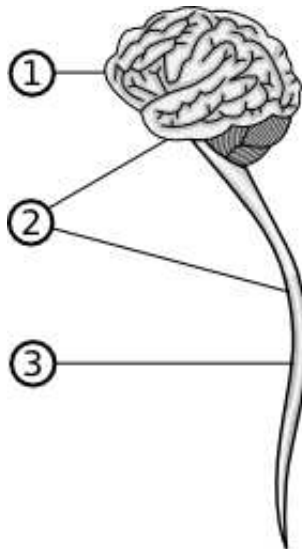


Figure 2.1: Basic diagram of a human central nervous system (2) [37], the spinal cord (3) and the brain (1).

Neurons: are cells in the nervous system that are electrically excitable and conform the basic computational unit of the brain. They process and transmit information. Many of these cells together form the core components of the brain and spinal cord in vertebrates, ventral nerve cord in invertebrates, and peripheral nerves on both. In figure 2.2, a basic representation of a neuron can be seen.

Presynaptic neuron: the transmitting neuron. Its synaptic terminals extend into synapses.

Postsynaptic neuron: the receiving neuron in a synapse; formed by a neuron's dendrite.

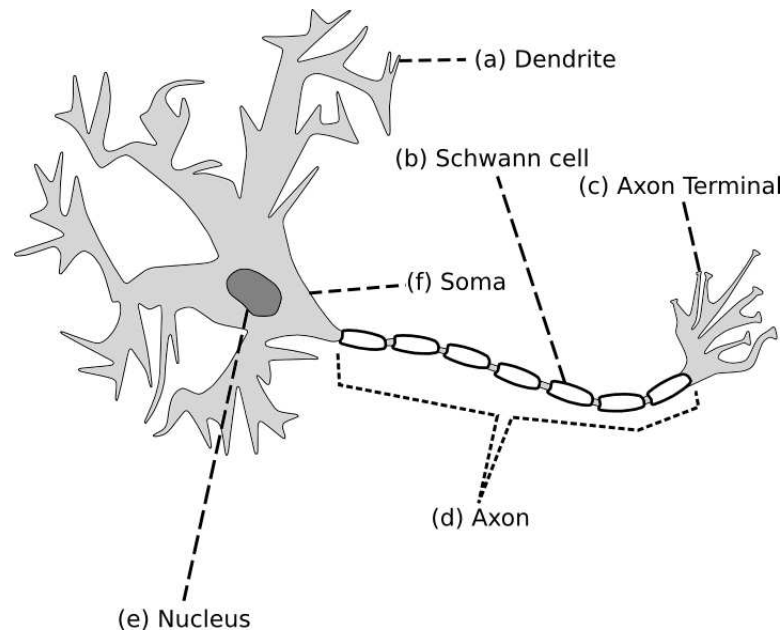


Figure 2.2: Diagram of a human neuron with some of its most relevant parts such as (a) dendrite, (b) Schwann cells, (c) axon terminal, (d) axon, (e) nucleus and (f) soma.

Synapses: there are more than 100 billion neurons in a human brain, each of which has the ability to communicate with many other cells. To make this possible, highly efficient and sophisticated mechanisms are needed. This kind of communication is made possible by synapses, the functional contacts between neurons. Synapse refers to very specialized points of connection between two neurons. Through these junc-

tions, neurons signal to each other and also to other non-neuronal cells like muscles and glands, which means that they allow the nervous system to communicate with all other systems of the body. This can be seen as a chain of cells. Two different types of synapses are identified, electrical and chemical synapses. As explained by Purves *et al* [2] in electrical synapses, current flows through gap junctions, which are specialized membrane channels. In contrast, chemical synapses enable cell-to-cell communication via the secretion of neurotransmitters; these chemical agents released by the presynaptic neurons produce secondary current flow in postsynaptic neurons by activation of specific receptor molecules. As seen in figure 2.3 two neurons are connected by a synapse. At the far end of the axon terminal (a) there are neurotransmitters (b) enclosed within synaptic vesicles (g) which are released into the extracellular space known as the synaptic cleft (c) between the pre-synaptic neuron and the receptive post-synaptic neuron.

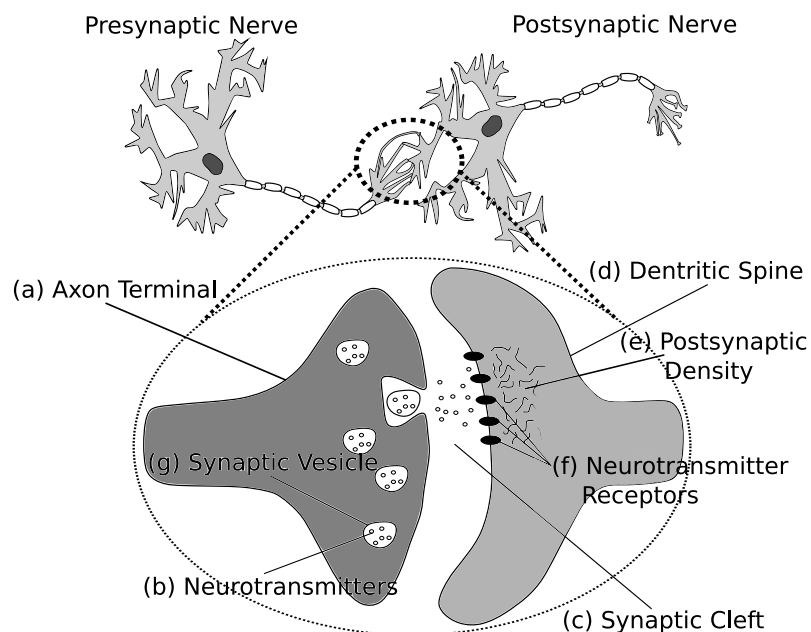


Figure 2.3: Diagram of a presynaptic and postsynaptic neuron joined by a synapse which shows what happens at the axon terminal and dendritic spine level.

Reinervate: this term is used to explain when a neuron, which has lost its connection, receives a new connection. This new connection could come from the same neuron

or another neuron, and it compensates for the lost connection.

Axon: is an extension from the nerve cell body that sends information away from it to its axon terminals. A neuronal process that carries the action potential from the nerve cell body to a target. In other words it is the medium in which messages are sent. In figure 2.2 it can be seen that the axon (d) extends away from the nerve. It is also observed that it goes inside a sheath formed by Schwann cells (b) which will provide support in the event of an injury.

Axotomy: is the process of cutting or severing an axon. In figure 2.4 B it can be seen that the nerve has suffered an injury and the part of the axon separated from the nerve degenerates. The Schwann cells remain in about the same position so they will act as a guide for a new axon to grow and innervate the effector.

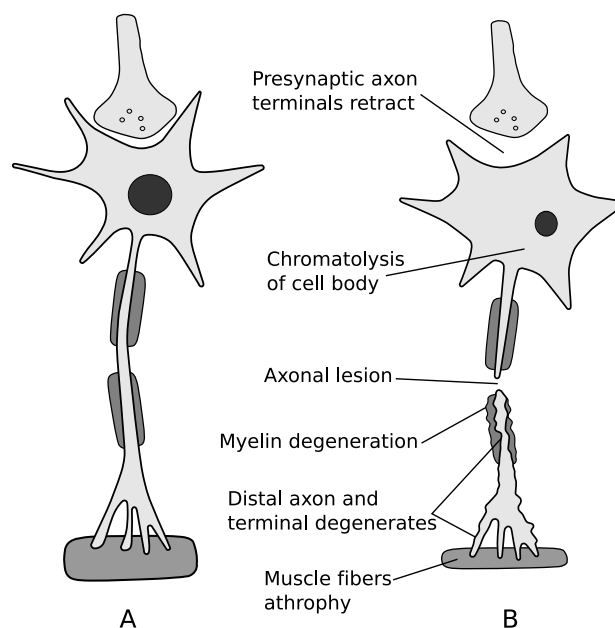


Figure 2.4: Wallerian degeneration. (A) Normal connections before an axon is severed. (B) Degeneration following severance of an axon. Degeneration following axonal injury involves several changes: the axon terminal degenerates, myelin breaks down and forms debris, and the cell body undergoes metabolic changes. Subsequently, presynaptic terminals retract from the dying cell body, and postsynaptic cells degenerate. Based on figure 4.2 in Neuroscience: Fundamentals for Rehabilitation [1].

Cell death: when a neuron is completely deprived of its source of growth factors and

exposed to high levels of toxic molecules of inflammatory attack, it can undergo cell death [5]. An example of this can be seen in figure 2.4 B.

Phenotype: the visible (or otherwise discernible) characteristics of an animal that arise during development [2].

Soma (Cell Body): the metabolic centre of any cell that contains the nucleus and the apparatus that produces and stores energy. The cell body of neurons also includes mechanisms to synthesize neurotransmitters [1]. An example of a cell body or soma (f) can be seen in figure 2.2.

Dendrites: derived from the Greek word *dendron* meaning “tree” it is a neuronal process arising from the cell body that receives synaptic input, forming branches that sprout out. The way the dendrites arrange is not very well known and defines how neurons will interact with other neurons in its environment in terms of communication. An example of dendrites (a) can be seen in figure 2.2.

Myelin: is a multilaminated wrapping that is formed around the axon of a neuron. This wrapping is formed by a special type of cells. In the PNS these cells are Schwann cells (explained below). Basically, this wrapping works as an insulation that protects the axon and the messages it transports from the environment and any potential disruptions it may cause. The simplest way to understand this is to think of a copper cable, in which the myelin would be the rubber or plastic that covers the conductor to protect it and the current it carries. In figure 2.2 myelin is found in bead-like bodies around the axon in the form of Schwann cells (b).

Demyelination: as the word implies, it is a loss of myelin. A demyelinated axon may maintain both its afferent and efferent connections but, due to loss of myelination, poor or failed conduction results [5]. In the PNS, destruction of Schwann cells impedes conduction of electrical signals along sensory and motor pathways [1].

Glial Cells: they provide support and protection to neurons in a very helpful way by giving them nutrients and oxygen, insulating them from each other, and removing pathogens. They basically provide a good environment for the neurons to survive, a key concept for nerve regeneration to be successful. Glial cells of the central nervous system include oligodendrocytes, astrocytes and microglia. Schwann cells are the support cells of the peripheral nervous system.

Pathogen: refers to all foreign things in the body to which, in most cases, the immune system will react. In other words, it refers to all strange things that could harm the host.

Schwann Cells: neuroglial cells (a type of glial cell) in the peripheral nervous system that elaborate myelin. As explained before, myelin is formed around axons as a sheath, in particular the Schwann cells will help a denervated neuron to find an axon (path) to reinervate another neuron after injury. In figure 2.2 it can be seen how the Schwann cells (b) surround the axon (d).

Axoplasm: the gelatinous fluid that fills the axon, it can be generally seen as the medium or content.

Neurotrophic factor: they are a family of proteins that are often associated with growth, cell differentiation and proliferation, essential for neuronal development. It is a large family that includes neurotrophins such as: nerve growth factor (NGF), brain-derived nerve growth factor (BDNF), neurotrophin-3 (NT-3), neurotrophin-4 (NT-4) [3].

Sprouting: axonal sprouting can happen on surviving neurons that suffered some level of injury. It refers to the process that will occur while the neuron regenerates. It is the actual process an axon follows when regrowing and forming a new connection. The process will typically fail when it encounters an inhibitory matrix or scar, loss of neurotrophin support, or presence of continuing inflammation or toxicity. In other

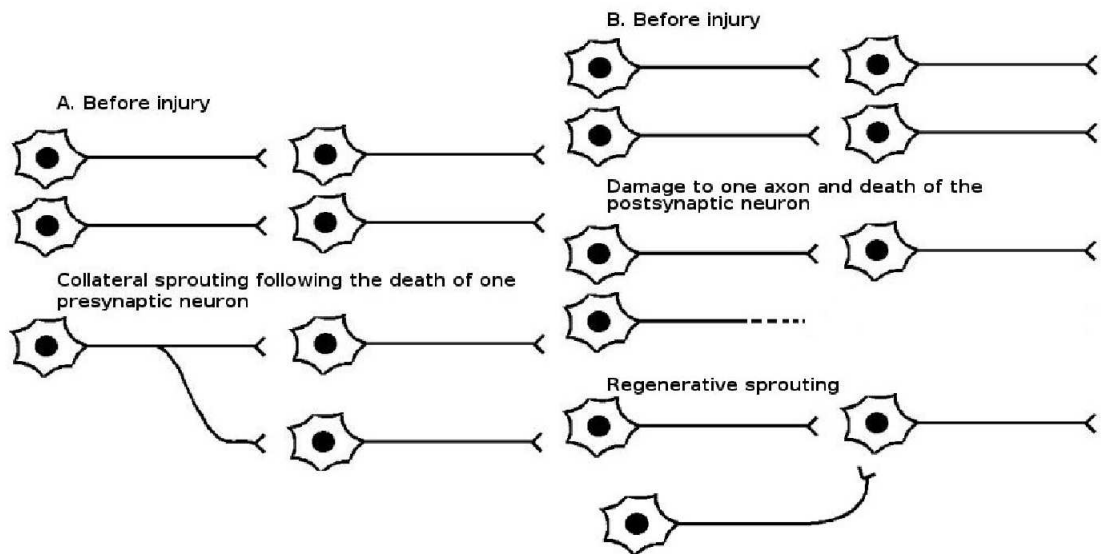


Figure 2.5: Axonal Sprouting. The new growth of axons following injury involves two types of sprouting: A., collateral sprouting, in which a denervated neuron attracts side sprouts from nearby undamaged axons; and B., regenerative sprouting, in which the injured axon issues side sprouts to form new synapses with undamaged neurons. Based on figure 4.3 from [1].

words, if the environment is not good, sprouting will not occur. Also, aberrant sprouting can occur when an axon reconnects to an inappropriate target. If this occurs, the synaptic conduction is restored but this pathway does not result in functional restoration. An example of nerve sprouting can be seen in figure 2.5. The diagram shows the two types of sprouting that can occur, collateral sprouting (a) and regenerative sprouting (b). More details of this are given in section 2.3 below.

2.3 Neuroplasticity

According to Purves *et al* [2] neuroplasticity refers to the capacity of the nervous system to change, and it shows during the development of neural circuits. However, the adult brain must also possess substantial plasticity in order to learn new skills, establish new memories, and respond to injury throughout life.

As explained by Lundy-Ekman [1] the term neuroplasticity refers to the way the brain adapts, caused by experience or changes in the environment. “By definition, neuroplasticity is any change in the nervous system that is periodic and has a duration of more than a few seconds.” [1] (p.58) There are three types of neuroplasticity that can be observed:

Habituation is the simplest form of plasticity. Basically the short-term changes of certain stimuli is repeated so that eventually it ceases to be received. An example of this is reflex behaviours in which one responds to a certain stimuli or situation always in the same way because the brain is trained to react in that certain way. Another example would be to cause a mild pain stimuli in a hand continuously; eventually the brain gets “used” to it and stops processing the pain feeling.

Learning and memory are the long-term changes that happen when certain stimuli are repeated. It triggers new proteins to grow and new synaptic connections to form, which results in a maintained response and memory. For example, in the initial phases of motor learning, large regions of the brain show synaptic activity. Those changes are then persistent and only minor further changes are observed. A perfect example of this are all the synaptic connections formed during the learning process of walking in an infant, under normal circumstances those connections will remain throughout their life with minor changes.

Recovery from injury which is possible when damage to the axons occur but may not result in cell death. It allows the neurons to reorganize its synapses creating new connections through axons. This was explained previously as sprouting.

As seen in figure 2.5, the regrowth of damaged axons is called sprouting which is divided into two types. *Collateral sprouting* (A) occurs when a denervated neuron attracts nearby undamaged axons, in other words, it will be reinervated by branches of intact axons. *Regenerative sprouting* (B) occurs when both axon and target have been damaged, in which the injured axon sends side sprouts that will then form new synapses with neighbouring undamaged neurons. Functional regeneration of axons occurs most frequently in

the PNS, partially because the production of nerve growth factor by the Schwann cells contributes to the recovery of peripheral axons. Sprouting by peripheral axons can cause problems when the wrong targets are innervated. For example, in motor connections this means a muscle can end up receiving wrong instructions of movements, which then could be fixed by a relearning process in the brain.

It is important to mention the difference between stimuli in the CNS and the PNS. In the CNS it is more related to learning and experience processes in which the brain changes and creates many new synaptic connections. In the PNS the stimuli can be related to more physical events like exercise or injury.

2.4 Central Nerve Regeneration

For many years it was accepted as a principle that injury to the adult central nervous system was devastating because of the inability of central neurons to regenerate correct axonal and dendritic connections. The consequences of injury are not just a break in communication between healthy neurons, but a cascade of events that can lead to neuronal degeneration and cell death. This principle has been reconsidered after recent discoveries, which indicate that neurons could regrow if they are provided access to the permissive environment of a conditioned sciatic nerve. It was also revealed that the failure of CNS neurons to regenerate is not an intrinsic deficit of the neuron, but rather a characteristic feature of the damaged environment that either did not support, or prevented, the regeneration process [5].

Regeneration in the adult CNS requires a multistep process. In general terms, the first step is to be for the injured neuron to survive. Then the damaged axon must extend its cut processes to its original neuronal targets and, once contact is made, the axon needs to be remylinated and functional synapses need to form on the surface of the targeted neurons. This regeneration process is only possible in the presence of a proper environment that will facilitate regrowth.

2.5 Peripheral Nerve Regeneration

In the PNS, after a nerve is injured, a complex and finely regulated sequence of events will start, it will remove the damaged tissue and then a reparative process will begin. There are several factors that come into play, and a remarkable cooperation between destructive and restorative forces occurs for proper nerve regeneration, as presented by Burnett *et al* [10].

Injury in the PNS is characterised by ruptures of the axonal connection which makes the axons distal to the lesion site disconnect from the neuronal body and degenerate. These injuries can include rupture of peripheral nerve fibres, axons and myelin sheaths far from the lesion site, which is known as a process called Wallerian degeneration, explained in detail in a subsequent section. The functional significance of regeneration is to replace the distal nerve segment lost during degeneration, allowing reinnervation of target organs and restitution of their corresponding functions as explained by Navarro *et al* [4]. Therefore, functional deficits caused by nerve injuries can be compensated by three neural mechanisms: the reinnervation of denervated targets by regeneration of injured axons, the reinnervation by collateral branching of undamaged axons in the vicinity, and the remodeling of nervous system circuitry related to the lost functions [4]. However, clinical and experimental evidences usually show that these mechanisms by themselves do not allow for a satisfactory functional recovery, especially after severe injuries. It is generally considered that in humans, for nerve gaps of less than 2 cm neurological recovery is moderate, but for gaps longer than 4 cm, recovery is close to non-existent [4].

2.5.1 Neuronal Survival and Reaction

After an axon has been cut or severed (postaxotomy) [4] neuronal death depends on several factors such as age, severity of injury and proximity of the injury to the cell body. Neurons in an adult are less susceptible to die than immature neurons, and lesions close to the cell body are much more likely to cause death than distal lesions. All these factors

determine the intensity and the time course of the neuronal response.

After a lesion has occurred *axonal injury signalling* takes place, in which signals responsible for the initiation and maintenance of the regenerative neuronal response include a variety of mechanisms acting at sequential time phases. Before regeneration can take place a series of degenerative processes must occur [10]. Firstly, seconds after a nerve lesion, the axoplasm of lesioned axons is in continuity with the extracellular medium before the plasmatic membrane is sealed. Then, a second set of signals are conveyed by retrograde axonal transport. These include the early deprivation of target-derived trophic factors and the arrival of activating signals from the own injured axons and non-neuronal cells [4].

The axonal signals induced in response to nerve injury activates signalling pathway genes in neuronal cell bodies that may lead to two opposing consequences: cell death or regenerative response [4]. After a severe nerve injury cell survival is not assured [10].

2.5.2 Injury Types

For this research focus will be placed on laceration injuries such as those created by a knife blade. These can be complete transections but more often some nerve element of continuity remains.

There are other kinds of injury where the nerve's capacity to stretch is exceeded. Also compression-related injuries, when radial nerve compression is applied but does not involve severance or tearing of the neuronal elements [10]

2.5.3 Injury Classification

In the previous section types of injury were divided according to what caused them. Now, laceration injuries will be divided according to the severity of the injury. The timing and success of the peripheral nerve repair depends on the extent of the injury [10].

Nerve injuries are divided by severity into neurapraxia, axonotmesis and neurotmesis.

This research will specifically focus on axonotmesis which occurs when there is complete interruption of the nerve axon. Then, axon and myelin degeneration will occur distal to the point of injury leading to complete denervation. In such injuries the chances of recovery are very good because of the remaining uninjured Schwann cells.

Injury in the nervous system is mainly known as axonal injury which happens when an axon is severed, the part connected to the cell body is referred to as proximal segment, and the part isolated from the cell body is called the distal segment, as explained by Lundy-Ekman [1]. Immediately after the lesion, axoplasm leaks from the cut ends and makes the segments retract away from each other. The isolated part will go through a process called wallerian degeneration.

The **Wallerian degeneration** process, as seen in figure 2.4, describes how the distal segment of the axon undergoes deterioration, the myelin sheath pulls away from the segment and how the axon swells and breaks into short segments. This is all followed by death of the entire distal segment.

In the Wallerian degeneration process, explained in the last paragraph, the Schwann cells play a key role. At the later stages of the process, the Schwann cells will form into stacks or cell columns known as the **bands of Büngner** and they become important guides for sprouting axons during the expected reinervation.

2.5.4 Regeneration

After the degeneration processes, explained previously, and in an ideal case where regeneration is to happen, the environment will be ready to support regeneration. The relevant factors that will lead to regeneration are mostly caused by the Schwann cells. The tip of each axon sprout will contain filopodia which act by adhering to the Schwann cells and use them as a guide [10]. It has also been proved [10] that Schwann cells will produce nerve growth factors at the site of injury. The growth factors along with the bands of Büngner act as the main source of regeneration and they make it possible for axons to

reinervate old targets. The NGF uptaken by the axon is then transferred to the cell body and provides a constant stimuli for growth as well as the before mentioned guide for the advancing axon. This means that the amount of NGF at the Schwann cells will decrease each time an axon uses them to grow.

In more simple terms, the regeneration process is possible because the Schwann cells will guide the injured neuron's axon to find its path through its previous receptive neuron. After injury has occurred the Schwann cells will remain at about the same position they were before injury. For that reason, in most cases, the injured neuron is able to find its old receptive neuron.

Chapter 3

Peer-to-peer Applications Overview

An excellent paper which conveys the concepts of peer-to-peer computing is written by Milojevic *et al* [14] in which they say that the term “peer-to-peer” refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. P2P has received lots of attention in research, product development, and investment circles. Some benefits of P2P are that they improve scalability by avoiding dependency on centralized points, they reduce costly infrastructure by enabling direct communication among clients and they enable resource aggregation.

A simplistic way to identify when a P2P system is formed is when there are several computers that share resources and services by direct exchange. Also, Shirky gives his definition saying that “P2P is a class of applications that takes advantage of resources (storage, cycles, content, human presence) available at the edges of the Internet” [38]. If “peer” is defined as “like each other”, a P2P system, then, is one in which autonomous peers depend on other autonomous peers [14]. This can be taken further by saying that this dependency between peers often turns into a collaboration between them.

A P2P topology starts as soon as there is communication between peers. From that communication many applications are derived, such as telephony users who talk to each other directly. Also, in computer networks where computers communicate, or in games where players interact directly. However, a comparison between client-server and P2P

computing is significantly more complex and intertwined along many dimensions [14].

In this chapter the common characteristics of P2P applications will first be analysed, followed by a description of the components that form such systems. A classification dividing P2P networks into structured and unstructured components will be presented in order to set a common base of understanding for this research. Finally, applications that have some relation with anonymity are presented, along with some Neuroscience concepts that inspired this dissertation.

3.1 Characteristics of Peer-to-Peer

There are many tasks that can be performed by P2P systems, which will always depend on the user's needs. This makes these types of applications task specific, and it will define many of the most common characteristics of such systems, as explained below.

Decentralization: when data is stored and processed only on centralised servers and the contents accessed via request-response protocols, decentralisation is impossible. Even when it is ideal for some applications, it is not the case for many P2P applications. P2P systems provide a solution to this with an increased autonomy by eliminating the need of central processes making decisions. This means that no peer is responsible for a specific task, other peers could take over and provide the same services.

This is shown in figure 3.1 where node A acts as a centralised connection between the rest of the nodes. If node A fails, the whole topology breaks and all the nodes will lose communication. Contrary to this, topology (2) is an example of a decentralized topology. If node A fails there are still other connections among the rest of the nodes that could still be used to perform the needed tasks which makes it able to cope with failure without disrupting the topology.

Scalability: is the ability of a system to grow while maintaining its characteristics.

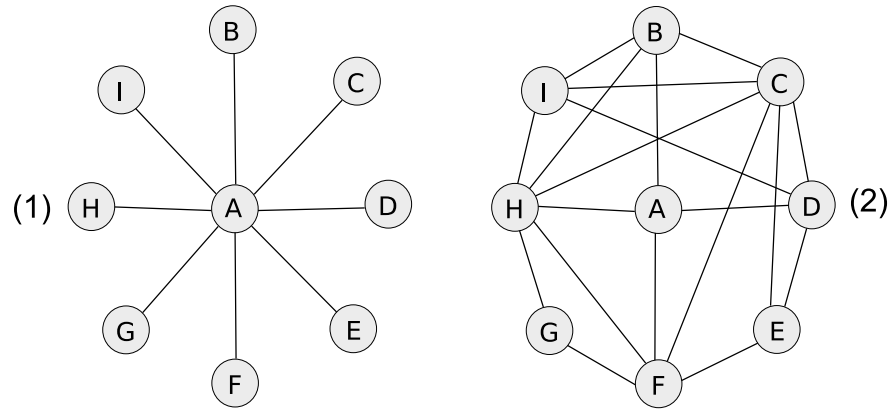


Figure 3.1: (1) An example of a centralized topology and (2) a decentralized topology.

This is usually limited by factors such as the amount of centralised operations that need to be performed, the amount of state that needs to be maintained, the degree of parallelism an application exhibits, and the programming model that is used to represent the computation. Scalability improves by ensuring that the system can grow by simply adding more computers and, ideally, making it more reliable in case of failures.

As explained before, centralised operations are all those which need to be performed by a specific resource, usually a server. This limits scalability because functionality depends on one or just a few systems.

The amount of state refers to the amount of storage space required to maintain information of the current state of the system. This will vary and the information stored will depend on the needs. In most cases, it will include data like routing tables or sequence of processes. This limits scalability because the availability of this information will determine how peers behave after failure, reconnection or when joining for the first time.

The parallelism of a P2P system refers to the amount of tasks that will be performed at the same time, usually in a coordinated way. This basically involves ensuring that the processes performed by the peers are coordinated somehow.

Anonymity: this refers to protecting information such as author, publisher, reader, server, documents, queries and any other sensitive information that could in any way expose the user's identity. The purpose of anonymity is to hide the final details of each peer. Most of the time users do not like other peers to know sensitive information about them or about what they are doing on the network.

Self-Organization: happens when there is a system with processes where the organization spontaneously increases or decreases. This happens without control from the system but from the environment or an encompassing or otherwise external system. In P2P systems, self-organisation is needed in order to provide scalability, fault resilience, allow intermittent connection of resources, and reduce the cost of ownership by distributing it among peers. They can scale in terms of the number of systems, number of users, and load. This allows dynamism by allowing change if the needs require. Usually the communication protocols change little, but there can be constant changes at the application layer.

Self-organization also allows for resource aggregation and interoperability, which makes it possible to utilize several different resources that are distributed in an unknown way. It puts them together which eventually creates a self-organizing distributed system.

Cost of Ownership: reduces the cost of owning the systems and the content, and the cost of maintaining it. It shares the costs and reduces it by distributing the tasks to be performed among many users and computers. The tasks could be data storage, processing power or network bandwidth.

Ad-Hoc Connectivity: collaborative users are increasingly expected to use mobile devices, making them more connected to the Internet and available for collaboration. The term ad-hoc refers to providing a solution for a specific problem or task. Under that line, creating ad-hoc connections happens when making connections for specific purposes. It is important to remember that this is not talking about physical

connections in which there would be a direct connection between nodes, rather, this is referring to virtual connections in which the path and what is between two nodes is unknown.

Performance: it aims to improve performance by aggregating distributed storage capacity and computing cycles of devices spread across a network. In many systems a lot of attention is given to the bandwidth each peer will have. Performance can be approached in several ways but probably the most challenging is from the application layer which will always be in constant change.

Security: relates to common concerns of distributed systems like trust between peers and shared objects, session key exchange schemes, encryption, digital digests, and signatures. Also relates to some new concerns as multi-key encryption, digital rights management, reputation and accountability.

Transparency and Usability: this was traditionally associated with the ability to transparently connect distributed systems into a seamlessly local system. The primary form of transparency was location transparency, but other forms included transparency of access, concurrency, replication, failure, mobility, scaling and others. A good example of transparency would be a BitTorrent application in which all a user needs is the torrent file. The application will transparently connect to other peers and start downloading and sharing data. For the BitTorrent application, who the user is or where the user is located, are not important.

Fault Resilience: one of the main priorities is to avoid a central point of failure. Most P2P systems already do this, but disconnections, unreachability, partitions, and node failures are still experienced by connected hosts. The ability of a system to recover from failures will define the level of resilience it has.

Interoperability: the idea behind this is to allow different types of P2P systems to exchange resources and cooperate. Since all are new implementations not many

of these techniques have been covered yet [14]. A good example would be current BitTorrent applications in which you can have different applications created by different developers running on different platforms yet all operating together and seamlessly sharing data.

3.2 Components of Peer-to-Peer Systems

There are a few components that, when put together, form P2P systems. Milojevic *et al* [14] talks about these components as follows.

The **Communication** component. There are many ways to understand this, the most common would be desktop machines connected via stable, high-speed links over the Internet at the end points of the P2P systems. Another communication component could be considered with small wireless devices such as PDAs or even sensor-based devices connected in an ad-hoc manner via a wireless medium. No matter what the medium is, there will always be some sort of communication between nodes.

The **Group Management** component which includes discovery of other peers in the community, location and routing between those peers. This can be resolved by each peer or by several peers in collaboration.

The **Robustness** component which will include things like security, resource aggregation, and reliability. Reliability in P2P systems is a difficult problem to solve. The inherently distributed nature of peer networks makes it difficult to guarantee a reliable behaviour. The most common solution to reliability across P2P systems is to take advantage of redundancy.

3.3 Classification of Peer-to-Peer Systems

Applications such as historical systems, distributed computing, file sharing, collaborative systems and P2P platforms are presented below. All of these are presented by Milojevic *et al* [14]. The following are the most common applications but other types of P2P systems have appeared and will continue to appear.

Historical Systems: years ago early distributed applications were already P2P, most users were from a technical or academic background, and were using either time-sharing systems or engineering workstations. P2P applications seemed the most natural approach because resources on machines were very limited, forcing users to share them. An example of two historical systems is the Simple Mail Transfer Protocol and Usenet News [14].

Public Resource Computing: the use of spare computing resources to achieve a common goal has been addressed for some time by traditional distributed computing systems, such as the Beowulf project from NASA [33] in which high performance was achieved by using a number of normal machines. In general terms the way this works is by splitting the computational problem to be solved into small independent parts. Then, each of the parts are processed in individual hosts and the results are collected on a central server. This central server distributes job items to PCs on the Internet. All hosts that are part of this have a client software which takes advantage of inactivity periods and performs some computation requested by the server. Once completed, the results are sent back to the server, and new jobs are allocated to the client.

File Sharing: this is one of the areas where P2P technology has been most successful. Multimedia content, for instance, inherently requires large files. In these cases P2P applications have solved bandwidth limitations that make large file transfers unacceptable. Some other features P2P applications provide through file sharing are

in file exchange areas, highly available safe storage, anonymity and manageability. There are also some technical issues in file sharing systems, most of them are network bandwidth consumption, security, and search capabilities. In figure 3.2 a simple file storage and exchange representation can be seen.

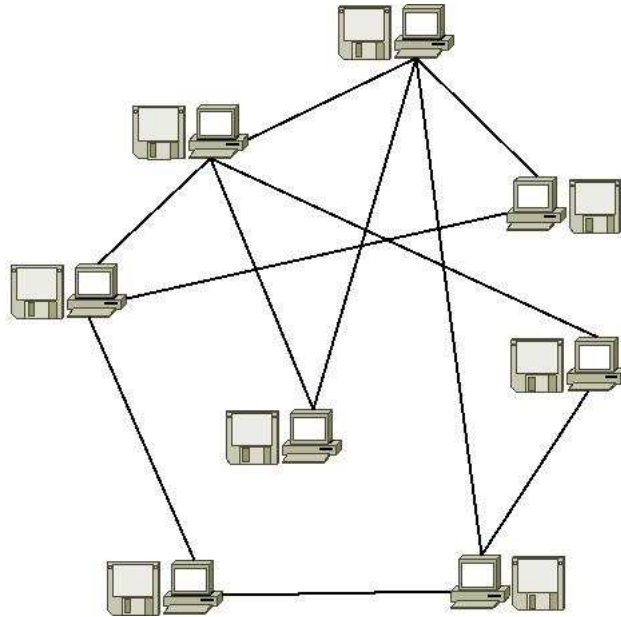


Figure 3.2: File Sharing Overlay Network. A peer-to-peer network showing several computers connected in a random order with a floppy diskette representing data storage. In the example, each peer is sharing CPU, bandwidth and storage resources.

Collaboration: these types of applications aim to allow application-level collaboration between users. The inherently ad-hoc nature of distributed systems gives a good fit for user-level collaborative applications. These applications are usually event-based, where peers forming a group begin a given task. The group may only be two peers collaborating directly or it could be a larger group. When a change happens in one peer, an event is generated and sent to the rest of the group notifying the changes. Some of the technical challenges to be considered are fault tolerance and real-time constraints.

Platforms: operating systems are less important when it comes to environments for applications to run. Middleware solutions, such as Java Virtual Machines, or Web

browsers and servers, are the dominant environment that is of interest to users as well as to developers of applications. This means that it is likely that future systems will increasingly depend on some other sort of platform that will be a common denominator for users and services.

3.4 Overlay Network Structure

A network of computers is a complex system formed by several interconnected computers. Between these computers many different network devices can be found such as routers, switches and hubs. Communication is possible through a physical medium such as copper cable, optical fibre or wireless connections. On top of the physical network an *overlay network* can be built. The main difference is that computers are connected through virtual or logical links and not through physical connections. There is always logic behind an overlay network which defines the configuration it has.

Overlay networks are classified by their structure. They can be created in a non-deterministic way (ad-hoc) as nodes and content are added, or their creation can be based on specific rules, as explained by Androutsellis-Theotokis *et al* [24] in their paper “A Survey of Peer-to-Peer Content Distribution Technologies”. The most relevant things in a overlay network are nodes and contents and the way they are arranged within the overlay network determines its structure. Under this scheme, structured, unstructured and loosely structured networks are derived as explained below.

3.4.1 Unstructured Architectures

In an unstructured network the way the content is placed is completely irrelevant to the way the overlay network is set [24]. This means that the connections between peers are built in a somehow arbitrary way. This raises a big challenge because locating the contents is difficult. Different searching mechanisms need to be implemented, ranging from brute force methods such as those inspired in flooding algorithms to more sophisticated

content hashed efficient methods with the purpose of preserving resources by doing random walks and using routing indexes. Important aspects such as availability, scalability and persistence are directly affected by the searching mechanisms that are put in place.

In the following paragraphs the common types of unstructured architectures will be discussed, they are classified according to the degree of their structure.

In *hybrid decentralized* overlay networks each client stores contents which they share with the rest of the network. Then each client connects to a central directory server which knows about all connected users and all contents that each user has. When a new peer joins it will register its contents with the server. Client computers make search queries to the server, and then reply back with information on how to find those contents. The client then opens a direct connection with the peer who has the desired content. These systems are considered decentralized because there are several peers acting as servers, but from an edge peer point of view there is still dependency on one server. These systems have the advantage of being easy to implement and files are located quickly and efficiently [24]. However, they do have disadvantages such as not being scalable systems and being vulnerable to censorship, legal action, surveillance, malicious attack, and technical failure. An example of a hybrid decentralized network can be seen in figure 3.3 where there are several servers represented as bigger nodes.

In *purely decentralized* overlay networks there is no central coordination of the activities in the network and users connect to each other directly through a software application that functions both as a client and a server. For this reason users are referred to as *servents*, as explained by Androutsellis-Theotokis *et al* [24]. When a new peer joins the network, it goes through a bootstrap process which involves acquiring a known list of connected users and from there it is able to announce itself and make queries. If a node fails or leaves the network, the system is able to remain in a stable state. An example of a purely decentralized network can be seen in figure 3.4.

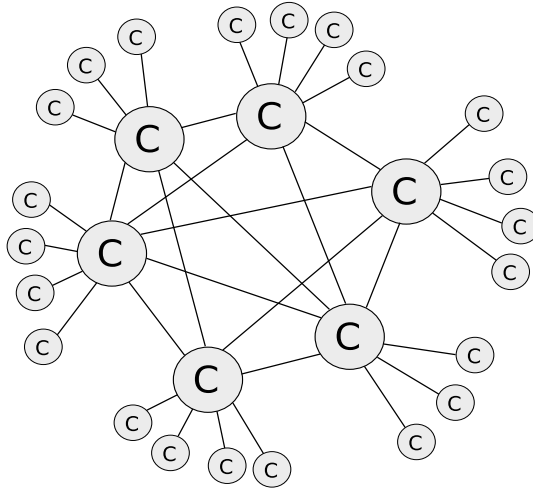


Figure 3.3: An unstructured hybrid network. It represents the combination of centralized and pure P2P topologies.

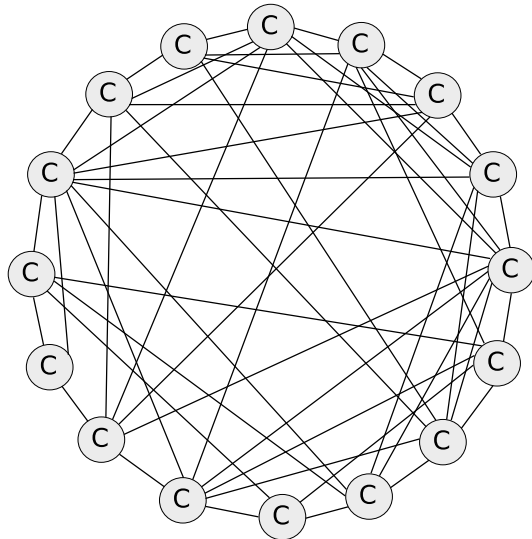


Figure 3.4: An unstructured pure P2P network. It has no dependencies on a central server.

In *partially centralized* overlay networks the concept of nodes that are dynamically assigned the task of servicing a subpart of the peer network is introduced. These nodes will index and cache files contained therein [24]. Such nodes are known as *supernodes*. Usually, characteristics such as sufficient bandwidth and processing power will determine if a node is elected as a supernode. These supernodes will index the contents shared in the segment of the network they are responsible for, and proxy search requests on behalf of the peers it looks after. This means that all queries are therefore initially directed to supernodes. An example of a partially centralized network can be seen in figure 3.5.

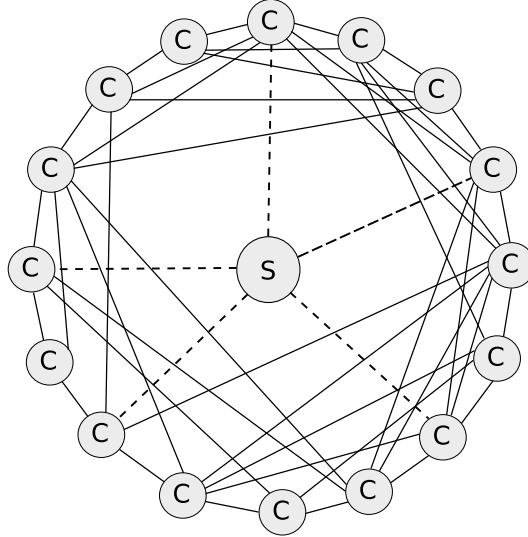


Figure 3.5: An unstructured centralized P2P network. Search queries are completed relative to a central server, yet connections are created directly between peers.

3.4.2 Structured Architectures

In unstructured networks, scalability issues are usually faced. As a result, structured networks emerged in an attempt to address those issues. Such topologies are inherently purely decentralized. There is a rigid control over the overlay topology, the contents (or pointers to them) are placed at precise and controlled locations. In principle, the way these systems work is by providing a mapping between content and location in some sort

of routing tables that will lead searches, yielding the correct results in an efficient way. A disadvantage of this is that it is hard to maintain the structure required for efficiently routing messages when dealing with very transient peers joining and leaving at a high rate [24].

Some of the most interesting and representative mechanisms for routing messages and locating data are discussed in the next paragraphs.

A *loosely structured* system is formed by peers with the ability to produce and locate content using node identifier similarity [24]. When undertaking a search, it will yield with close precision (not exact) which node is most likely responsible for certain content. This avoids broadcasting queries to unnecessary peers. Instead, search messages are forwarded through the peers, based on decisions made by each peer, and this forms a chain or sequence of hops. Similarity in the identifiers is achieved by using some sort of hashing algorithms that will be common to all users. The input and output of these algorithms will vary, as well as the logic within the algorithm.

A *finger table* system, exemplified by CHORD [34], performs a mapping of content identifiers with node identifiers. Both nodes and contents are assigned a key which identifies them. This key is produced using a deterministic function. The contents will be stored at the nodes which closely match its identifier. The nodes are arranged in a “ring like” way. An example of a CHORD topology can be seen in figure 3.6.

A *n-dimensional Cartesian coordinate* system also known as a Content Addressable Network (CAN), is a distributed hash table that maps content name with its location on the network [24]. It supports the insertion, lookup and deletion of “pairs” formed of a key and a value in the table. The network is divided into adjacent zones. Each node is assigned responsibility for its correspondent zone. A new node that joins the CAN system is allocated its own portion of the coordinate space by splitting the allocated zone of an existing node in half. An example of a CAN topology can be seen in figure 3.7.

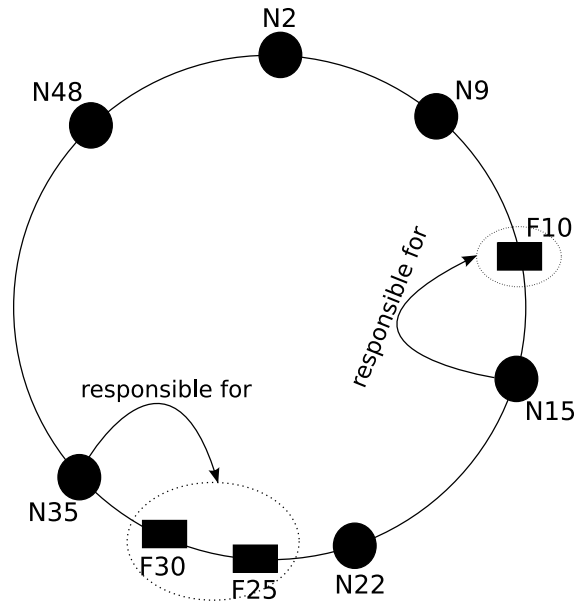


Figure 3.6: An example of a CHORD “ring like” topology in which the circles are peers and the rectangles are the resources shared, both identified by a key. The similarity between the peer keys and the resource keys can be appreciated. A peer is responsible for the resources within his scope.

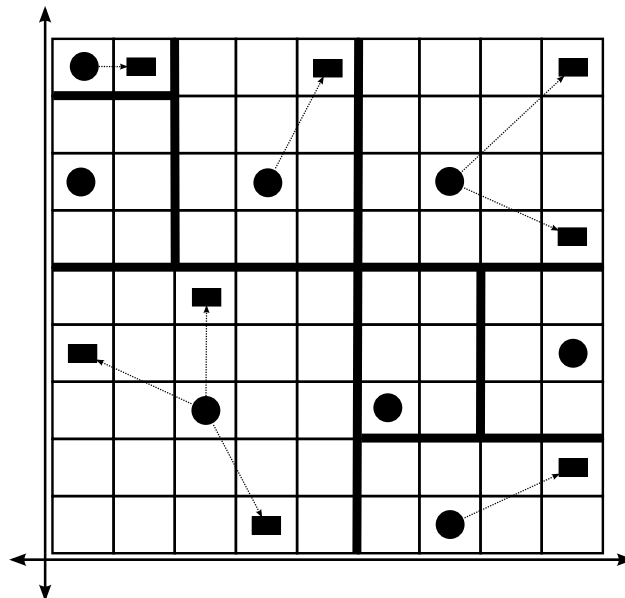


Figure 3.7: An example of a CAN topology in which the circles are peers and the rectangles are the resources shared. A peer is responsible for the resources within its space in the Cartesian table.

3.4.3 Loosely Structured Architectures

When the network topology falls between structured and unstructured they are referred to as loosely structured networks. The location of contents is not specified completely, but it is somehow affected by some routing entries. As in the structured architectures, these loosely structured topologies are inherently purely decentralized.

3.5 Basic Algorithms

The algorithms used to implement P2P systems define the type of topology. The common ones are detailed below.

Centralised directory model: upon request from a peer, a central index will match the request with the best peer in its directory. The term “best” in this case will be defined by the kind of resources shared in the system.

Flooded requests model: each request from a peer is flooded (broadcast) to directly connected peers, which themselves flood their peers, until the request is answered or a maximum number of flooding steps occur.

Document routing model: each peer from the network is assigned a random ID and each peer also knows a given number of peers. When a document is published (shared) on such a system, an ID is assigned to the document based on a hash of the documents contents and its name. Each peer will then route the document towards the peer with the ID that is most similar to the document ID. This process is repeated until the nearest peer ID is the current peers ID. Each routing operation also ensures that a local copy of the document is kept. When a peer requests the document from the P2P system, the request will go to the peer with the ID most similar to the document ID. This process is repeated until a copy of the document is found. Then, the document is transferred back to the request originator, while each peer participating the routing will keep a local copy [14].

3.6 FreeNet

Freenet is a distributed data store system with strong anonymity which benefits directly freedom of speech. It “permits the publication, replication, and retrieval of data while protecting the anonymity of both authors and readers” [25]. It is a totally distributed system, no broadcast search or centralized location index is used. It is not possible to find who originated or who is the destination of a file that is in the network. Even a node operator will find it difficult to know what contents are in his data space.

The architecture of the system is based on nodes that query one another to store and retrieve data files which are named by location independent keys. There are three types of file keys: Keyword-Signed Key (KSK), Signed-Subspace Key (SSK) and Content-Hash Key (CHK). Each file will also have a descriptive text associated which is also used to generate the KSK and SSK keys. For the CHK keys the whole file is hashed. Each node should have its own data store which is made available to the network for reading and writing. Each node will also hold a dynamic routing table containing addresses of other nodes and the keys of the files they store. When a node wants to perform a search the query will be passed along from node to node through a chain of requests in which each node makes a local decision about where to send the request next.

Once a file is located the reply will return following the same route it did when completing the search and it will cache the file at each node which gives the system high redundancy. In figure 3.8 a typical request sequence can be seen. This means the files will eventually be stored at more than one location even if the originator of the file shuts down. To deal with the problem of finite storage capacity, each node handles its data by least recently used. This means that data items that have had no recent request would be deleted if more space is needed. This gives the advantage of allowing outdated documents to be removed naturally, and documents that are still in constant use to remain in the network.

To insert a file into the network, a node will first calculate the key for it and check

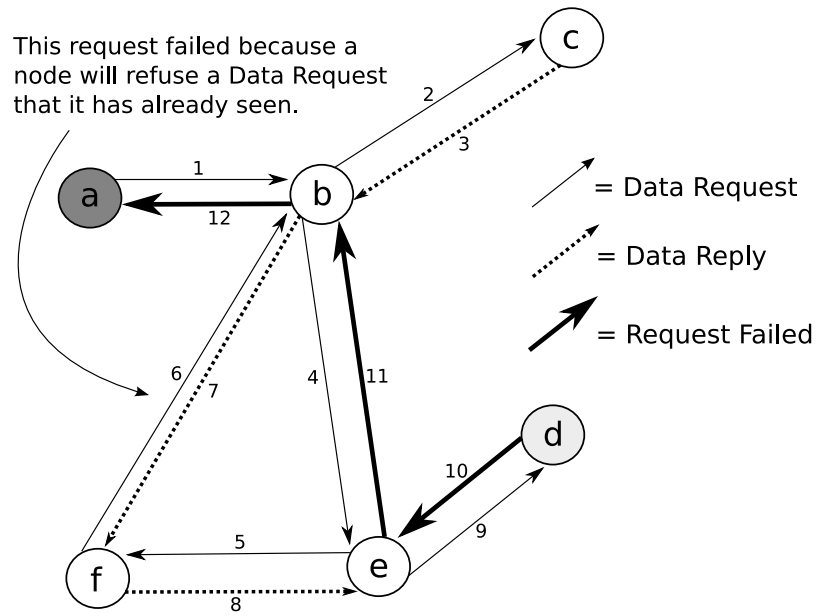


Figure 3.8: A typical request sequence. Taken from Figure 1 in [25].

that there are no collisions with keys it already has. Then it will determine a hops-to-live value which will represent the number of nodes the file will spread out to and then it sends the key insertion out. If the hops-to-live is reached without any key collisions then the file gets uploaded and replicated along the same search path.

Opposed to the early Freenet implementations called Opennet, the latest versions allow users to connect directly only to users they know and trust. This is called the Darknet and it is based on small-world networks, in which human relationships tend to follow a small path.

3.7 Remailers

The first concept of Anonymous Remailers was introduced by Chaum [30] about 27 years ago. His proposal focused mainly on encrypting messages to hide who a sender communicates with as well as the contents of the messages sent. The main goal was not to provide anonymous communication in terms of the receiver not knowing who sent the messages, but he mentions anonymity to be one of the side effects. As presented by du Pont [28]

anonymity can be seen as both a good and a bad thing for society.

In simple words a remailer is a computer located in the Internet, designed to strip out the sender's identification marks from all email messages it receives, replace them with new identification marks that denote the remailer as the message originator and forwards, or "re-mails" them to the intended recipient, as explained by Mostyn [27].

Remailers are divided into anonymous and pseudonymous and each one will be explained in detail in the following sections.

3.7.1 Anonymous Remailers

As it was mentioned before, a remailer is a computer that delivers messages without disclosing information of who sent the message. The message itself will contain information of who the intended receiver is in a specific format, therefore, they apply to messages intended for particular recipients and not for general communications. Anonymous remailers in particular will only provide one way communication, which ensures that the sender remains anonymous at all times. The latest implementations try to address this by allowing two-way communication, but the server still cannot differentiate between sent messages or replies.

There have been a few implementations of anonymous remailers through the years with differences that depend on choices made by their designers and some specific needs at the moment they were designed. In general terms, they have been classified into three types which will be explained below.

Type I: Cypherpunk

The word "cypherpunk" is listed in the Oxford English Dictionary [39] as "a person who uses encryption when sending emails in order to ensure privacy, especially from government authorities". In the context of anonymous remailers it refers to an informal group of people with the purpose to achieve privacy and security through cryptography

[43].

A type I remailer sends a message to a recipient after stripping out the sender's identity. The recipient can not answer the message received [43]. A sender will usually encrypt the message sent, then the remailer will decrypt it and send it, although this is not a requirement. It is also possible to chain more than one remailer. One of the main characteristics is that the sender does not need any special email software installed on the computer, which makes it accessible to more people.

Such remailers will take messages encrypted with PGP, GPG or simply in plain text. In general terms the sender will obtain the remailer's public key and then write the message with a specific format and fields, then it can be encrypted using the mentioned key. If the sender were to use several remailers he would need to repeat those steps encrypting the message that was already encrypted but with another remailer's key. Cypherpunk remailers do not keep logs of transactions [43].

After looking at a few different remailer websites [40, 41, 42] the general steps that a sender must follow to send an encrypted message through a cypherpunk remailer have been extracted. They are summarised below.

1. Retrieve the remailer's public key, generally by sending an email to the remailer with "remailer-key" in the subject field. An email that contains the public key will be sent back.
2. Compose and save the message on any text editor using the following template:

```
::  
Anon-To: <Recipient Email Address>  
  
##  
Subject: <Subject>  
<Message Text>
```

3. Use PGP or GPG to encrypt the message. To do this a valid PGP or GPG program must be installed, then it will run on the file that contains the message using the

key obtained from the remailer.

4. Send the encrypted message to the remailer, placing the remailer's address in the "To:" field, any subject and the body using the following template:

```
::
Encrypted: PGP

- - - - - BEGIN PGP MESSAGE - - - - -
<place encrypted output here>
- - - - - END PGP MESSAGE - - - - -
```

As mentioned before, the sender can use several remailers which means that the steps need to be repeated for each remailer, encrypting messages that have already been encrypted with other remailer's keys. Alternatively, a sender can use a simpler method without encryption which consists of the following:

1. Create a new message in email software or using any available webmail clients (Gmail, Hotmail, etc.).
2. Insert the address of the remailer being used as the recipient in the "To:" field.
3. Give the message a subject.
4. On the first line of the body of the email, type "::" (two colons).
5. On the line immediately following, type "Anon-To: ", followed by the address of the final recipient of the message.
6. Leave one blank line and then type the message.
7. Send the message.

Type II: Mixmaster

It is suggested that the term "Mixmaster" is probably derived from the word "mix" which was first used by David Chaum in his paper about "Untraceable Electronic Mail, Return

addresses, and Digital Pseudonyms” [30] in which, as commented in section 3.7, the first concepts of remailers were conceived. There, Chaum says that a sender knows not only about his intended recipient but about “a computer called a “mix” that will process each item of mail before it is delivered”. Clearly he was referring to the email server, which later was to be identified as an anonymous remailer.

Its main characteristic is that a message will be split into fixed-size packets and re-ordered [44] in an unknown way before sending them. This will make it harder for eavesdroppers to monitor communications, increasing security for the sender. This means that there is a need for specific software installed on the sender’s computer, which limits the amount of people able to understand and use it. Also, the sender has more control on how the remailer(s) will handle the message. An example would be a value that delays when the message should be sent to the intended recipient which can be specified by the sender. A receiver is still not able to reply to a message, therefore two-way communication is not yet possible.

Most of the current Mixmaster implementations can be adjusted to support type I Cypherpunk’s format without the need of special software, giving it backwards compatibility. The steps required to do this were previously explained. As handling messages by Mixmaster servers is more complex, there is a necessity to install and use dedicated software applications for sending messages [46]. This basically consists of installing the Mixmaster packages on the preferred Linux distribution [45], then retrieve the necessary public keys for encryption. Non-frequent users should do this before every use, for frequent users there are applications called pingers that automate a regular retrieval and update of the keys. Finally the client application can be run and will provide a comprehensive interface.

Type III: Mixminion

The implementation of Mixminion made it possible to send and receive anonymous email without affecting security. Anonymity is possible by using a “mix” network architecture

that prevents attackers from linking senders with receivers. People interested in anonymity and freedom of speech will volunteer all over the world, they will run servers called mixes. All together for the Mixminion network that receives the messages, decrypt, reorder and retransmit them passing by several remailers.

This implementation is an enhancement to mixmaster that addresses issues like replies, forward anonymity, replay prevention and key rotation, exit policies, integrated directory servers and dummy traffic [47].

The goals of Mixminion were to provide a system simple to deploy and simple for clients, as this had been one of the main barriers in previous implementations (type I and II). Support for type II protocols was also important in order to keep some backward compatibility. Currently there is a working software designed to run on Windows, but plans to make it compatible for Windows and Linux are in place. The design goals were not to provide a low latency system.

3.7.2 Pseudonymous remailers

Also known as “nym servers” they allow people to send pseudonymous messages to Usenet groups and via email. This characteristic is in direct contrast with any of the forms of anonymous remailers presented before. It will take away the email address of the sender, give a pseudonym to the sender and send the message to the intended recipient who can use this pseudonym to answer those messages. From all the different remailer implementations shown before, this is perhaps the most similar to Chaum’s [30] proposal and its first product.

This type of remailer is less secure in terms of anonymity because the server needs to map senders with pseudonyms in order to handle the reply messages. If the server is compromised in any way, it means that third parties are able to eventually find out who is sending the messages, as happened with the anon.penet.fi remailer’s controversial case [28], in which a Scientology minister posted some confidential information on Usenet

which could do potential harm to the Church of Scientology. A complaint was made to Interpol and the Finish police and after a lot of bad press, a warrant was obtained to confiscate the computer. The remailer administrator chose to give them just the identity of who posted the message and decided to discard the rest of the information and identities ultimately shutting down the server.

3.8 Peer-to-Peer Summary

According to an analysis made by Karagiannis *et al* [18] around the year 2004, popular media sources reported a sharp decline in P2P traffic during that year, stating that user population dropped by a half. This, of course, was in direct contrast with the increase of P2P activity during previous years. This was attributed to legal issues and the overwhelming threats of copyright lawsuits and fines.

The real scenario is that current file-sharing networks provide users with a variety of options. Furthermore, an increasing number of P2P networks intentionally camouflage their traffic. In general, it is observed that P2P activity did not diminish. On the contrary, P2P traffic still represents a significant amount of Internet traffic and is likely to continue to grow in the future.

It is fair enough to say that P2P is here to stay. Link level measurements performed [18] show that P2P traffic was at least comparable to the previous year's levels, or maybe even increased. The problem is that measuring P2P traffic has become problematic, with results that underestimate the reality of things. The use of non-standard or arbitrary ports is the main reason why this happens but also includes packet encryption which makes payload heuristics inapplicable.

On the last few sections of this chapter Freenet and Remailers were looked which are P2P application with strong anonymity. It is of great importance to understand those applications for this research.

Chapter 4

Design

SON is presented here as the design of a general purpose middleware for P2P applications that require a high degree of anonymity between peers. This has been developed by translating ideas and behaviours from Neuroscience to Distributed Systems. After looking at the common P2P applications in chapter 3 and at Anonymous Remailers and Freenet special interest is paid to those applications that deal with privacy and anonymity.

Characteristics such as anonymity and scalability make applications suitable for implementation using SON. The main characteristic, anonymity, will have other side effects such as improving privacy and enabling freedom of speech.

Each peer can be considered to have a set of information and functionality that makes them useful in the network. Focusing on the information, challenges of **managing** the information and **using** the information were encountered. Managing the information has to do with how peers join the network, the way they communicate to and how they keep their information and other peer's information updated. How this is handled in this design is described in section 4.2.1. Using the information has to do with how each peer interacts in the network based on other peer's properties. This will provide the main focus and details are given in section 4.2.2.

In this chapter common concepts between Neuroscience and Distributed Systems will be presented giving a basic explanation of how they fit into the SON design. Then,

the architecture of the design will be presented giving details of the establishment of connections, node interaction and overall protocol descriptions.

4.1 Translation

In this section a mapping between two domains is presented, Neuroscience and Distributed Systems, important to understand the design. As the overall purpose of this dissertation is to explore the potential of a new nature-inspired distributed computing technique based on Neuroscience, it is necessary to set all common concepts in the way they should be interpreted. Interesting new terms such as *Neuron Peers*, *Schwann Peers* and *Synapse Oriented Networks* (SON) have been created which will be explained below.

4.1.1 Neuron Peer

In section 2.2 it was explained what a neuron is by saying that “they conform the basic computational unit of the brain” and that “they process and transmit information”. If this is viewed from an informatics standpoint, there is a lot of similarity with a single computer in a distributed system. In fact, the brain can be considered a distributed system in its own right. Regardless of the computer type available, they all process and transmit information. The information can be transmitted internally to a hard drive for example, or it can also be transmitted externally to the network and other computer systems. In chapter 3 it was said that a peer was defined as “like each other” which fits very well with saying that many neurons together form the core components of the brain and spinal cord. In the same way many peers form a P2P network.

The term *Neuron Peer* is defined as all those peer computers in a P2P network which have a sender-receiver relationship. Depending on the peer’s function as a message originator or a message receiver, the following terms are defined respectively:

- **Presynaptic Neuron Peer** is the name given to the peer which acts as the message originator.

- **Postsynaptic Neuron Peer** is the name given to the peer which acts as the message receiver or final destination.

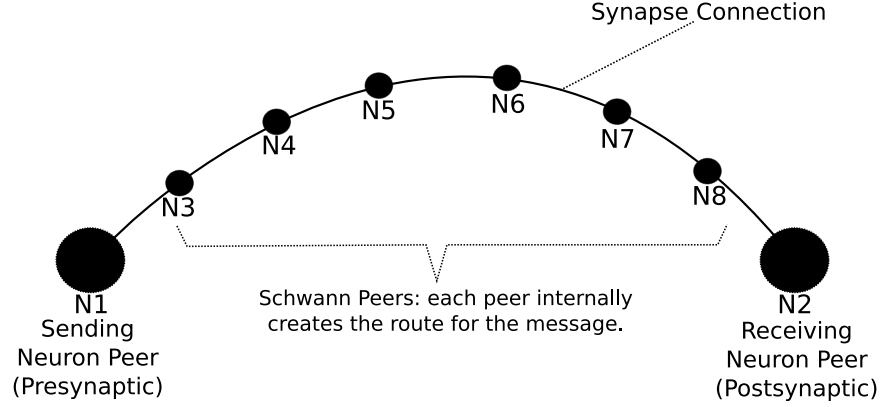


Figure 4.1: Diagram of two Neuron Peers (N1 and N2) involved in a Synapse Connection formed by several Schwann Peers (N3 to N8).

A more detailed explanation of how these Neuron Peers work within this dissertation will be given in section 4.2. In figure 4.1 a simple example of a Presynaptic and a Postsynaptic Neuron Peer can be seen.

4.1.2 Schwann Peer

When it comes to nerve regeneration in the PNS after injury, Schwann cells are essential. As explained in section 2.2, they “will help a denervated neuron to find an axon (path) to reinervate another neuron after injury”. The word “reinervate” here refers to building a connection between two neurons.

Schwann Peers are defined as all those peers in a P2P network that do not depend on each other and do not have any specific relationship between them but rather work as a pathway between two Neuron Peers. These peers will act as packet forwarders within the network.

A more detailed explanation of how these Schwann Peers work to help forward packets will be given in section 4.2. In figure 4.1, a simple example of the Schwann cells in a Neuron Peer relationship.

4.1.3 Synapse Connection

When a relationship between two Neuron Peers is formed, a Synapse Connection is constructed. As was explained in section 4.1.1, there can be two types of Neuron Peers according to their sending or receiving functions. On a given synapse connection there will only be one presynaptic and one postsynaptic peer. Figure 4.1 displays a Synapse Connection formed by two Neuron Peers and several Schwann Peers.

It is important to note that in this implementation, there is no direct connection between a Presynaptic and a Postsynaptic Neuron Peer. Otherwise it would be impossible to assure anonymity for the sending peer, as the IP datagrams will contain identifiable information. A *Synapse Connection* is defined as a packet sent by a peer and then received at its destination, taking into account that there will be many Schwann Peers in between. Figure 4.1 shows a clear example of a Synapse Connection. There is no way to keep a trace of the route the packet follows and neither the sender, the intermediate Schwann Peers or the receiver will know of peers other than their immediate ones. This indirection is the method used to implement anonymity.

4.1.4 NGF to build routes

In section 2.2 it was explained that Nerve Growth Factors “are a family of proteins that are often associated with growth, cell differentiation and proliferation, essential for neuronal development”. Specifically in nerve regeneration, NGF is contained within the Schwann cells. When such cells have high amounts of these proteins, they attract an axon and guide it towards a final destination. For this dissertation NGF is considered to be an important characteristic every node has, and it will define how suitable a node is at a specific moment in time to carry packets through the network.

In nature, proteins in cells are consumed. In the same way, NGF in peers will be consumed each time the peer acts as a packet forwarder. When a Schwann Peer receives a packet and it identifies that the packet still has more hops to complete before it reaches

the final destination, the Schwann Peer will forward the packet to the next peer with highest NGF and it will decrease its own NGF value. This gives the network very high churn and assures that new, different routes are always built.

4.1.5 General Mapping

The following table 4.1, presents a general mapping which can be used as a reference for future work or just to understand and clearly acknowledge that the two domains can be easily related.

Neuroscience Concept	Applied to Distributed Systems
Neuron	Peer, node, host.
Central Nervous System	A computer and all its internal components.
Peripheral Nervous System	Multiple computer systems interconnected in a network.
Presynaptic Nerve	A peer who is the originator of something.
Postsynaptic Nerve	A peer who is the receiver or destination of something.
Synapse	A connection between two peers.
Reinervate	To reconnect or rebuild a connection between two peers.
Axon	A communication channel.
Axotomy	The failure or break of a communication channel.
Cell Death	Failure of a peer.
Phenotype	A property of a peer.
Myelin	Packaging or encoding information to protect it from the exterior.
Demyelination	Extracting (decoding) information.
Glial Cells	Helper peers.
Neurotrophic Factor	Helper peers with specific characteristics and for specific needs.
Schwann Cells	Helper peers, a type of neurotrophic factor.
Pathogen	Noise in a network.
Axoplasm	The data that is being transported in the network.
Sprouting	Building routes between peers.
Regeneration	Recovery from failure.
Injury	Network or peer failure.

Table 4.1: A general mapping of Neuroscience concepts applied to P2P Systems.

4.2 Architecture of Synapse Oriented Networks

In chapter 2 it was identified how neurons build connections following a path of proteins contained in the Schwann cells. Basically these proteins form NGF which will attract growing axons and will guide them to reach another neuron. This characteristic will be focused upon and applied into the SON networks.

From the wide variety of P2P network implementations that exist, there is special interest in those with strong anonymity and security concerns, such as Freenet and Anonymous Remailers. Using the above mentioned characteristics of NGF a way to route packets in a system of peer computers in such way that peer anonymity can be assured is proposed.

It is not explained in detail the resources that are shared between the peers as it is of no importance for this project and, in the end, will depend on the application's requirements. Messages or packages could be encrypted and split in many ways allowing the network to share a wide variety of information. This architecture has simple message routing with simple message formats and it is explained in detail below.

Every peer on the network will act as a Schwann Peer, and each will have certain amount of NGF. At the same time, peers can act as Neuron Peers in a specific case in which one peer wants to send an anonymous packet to another. Under that scenario, the sender acting as Presynaptic Neuron Peer, will indicate which is the intended recipient of the packet and the number of hops the packet should perform before it is delivered. This is the key part of the system. Each time a packet is forwarded, by a Schwann Peer, the peer will not keep any traceable information that could relate the packet with previous peers, and the decision of which will be the next peer is made based on the amount of NGF other peers have at that moment in time. The peer with highest NGF in the network is chosen as the next forwarding peer. When a peer acts as a forwarder, it will reduce its NGF value, representing that it has consumed NGF to help an axon grow, or a connection happen. This way the network is given very high churn when routing the packages as the values of the NGF are constantly changing. Finally, when a message has gone through

all necessary hops it will be delivered to the intended destination which will be acting as the Postsynaptic Neuron Peer completing the Synapse Connection.

4.2.1 Directory Server

There is a need to maintain an updated view of the peers in the network and their NGF values. Because of time constraints, the first implementation of the SON networks contains a Directory Server that will keep track of all peers that join the network. The Directory Server maintains a global view of the network. In other words, the management of information is, for now, implemented with this simple and fast solution. This is just a temporary solution as it provides very poor scalability and it is vulnerable to attacks. When a peer starts it will contact this “known” server, the server will assign a unique identifier to the peer and it will add it to its list of peers. The peer will tell the server the NGF value it has and the port on which the peer will be listening to other peers. This means that the information the Directory Server keeps of every peer is: unique identifier, IP address, port and NGF.

The information the server keeps from each peer is kept as simple with the view to remove this temporary solution and replace it with a robust and scalable distributed solution. The Directory Server maintains information but exercises no control over the network, and the way the network behaves is determined entirely by the peers. The server’s only functions are to keep the updated list of registered peers and send that list to peers upon request. A set of protocols are used for communication depending on the needs. These protocols are explained in detail later.

The connection between peers and the Directory Server is attained using a TCP connection on port 7777. For reasons of simplicity the connection between the peers and the Directory Server is to be maintained for as long as the peer is participating in the network. Otherwise there will be a need for an algorithm on the server to constantly check if the peers are still alive. This will only increase the centralised dependency making it harder

to merge to a distributed solution in the future. With this simple implementation, if a connection is lost, it represents that the peer has left the network. In a later decentralised solution this will have to be handled by the peers.

4.2.2 Peer Interaction

A set of interactions can happen at the peer level which will be explained next. Although simple, all interactions together are what creates SON and determines the way it behaves. An analysis of the network cost will be given for each peer interaction in which *network cost* is defined as data sent by the participants in a SON.

Peer Startup

When a peer starts, its first task is to join the network and register itself in order for other peers to learn about its existence. In this case, this is achieved through the Directory Server. At startup, a peer will initialize its own NGF value with a random number between zero and one, and it will start a listening service for other peers to connect. This listening service has the ability to choose between a list of possible ports, which means that several peers can run on the same host. This service will receive connections and data from other peers and will process it depending on the contents of that data.

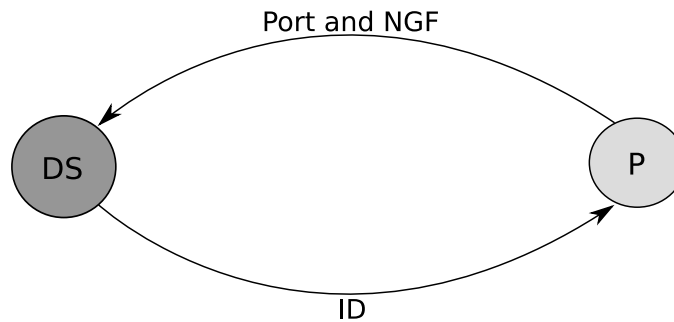


Figure 4.2: A representation of the interaction of the peer and the Directory Server at peer startup.

As mentioned in section 4.2.1, a TCP connection is used between the peer and the Directory Server. The listening service that runs on each peer also uses TCP connections

for other peers to connect and send messages. These connections can be opened on any port that ranges from 7770 to 7776 which means that on a single host seven instances of a peer can be run at the same time. There is no specific reason for this decision, it can be changed to meet specific needs. The way a peer handles this is by trying first the highest port (7776). If that port is available the peer will use it, otherwise it will try with the next lower one (7775) until it finds an available port. If no available ports are found, the peer can not run and execution is aborted.

In the network there will be N Nodes which can send M messages. In this case M refers to single messages between any two peers or between peer and server. The network cost of a peer startup is represented by:

$$StartupCost \in O(2MN)$$

As shown in figure 4.2, for each peer there are request/response messages. One message is sent to the Directory Server that contains the port other peers should use to contact it and the peer's own NGF value. The second message is sent from the server to the peer and contains the unique identifier for this peer. The protocol used for the first message can be seen in figure 4.3. The first tag <START> tells the server this is a startup message sent by the peer. The protocol used by the server to assign the peer an identifier can be seen in figure 4.4. The first tag <ID> tells the peer that the message contains the ID by which it will be identified on the network.

<START>port>ngf<END>

Figure 4.3: The message format used on the Peer startup messages as a request message to the Directory Service.

<ID>id<END>

Figure 4.4: The message format used by the Directory Server as a response message to assign the peer its unique identifier.

Peer Sending/Forwarding Messages

After a peer has initialised, it is ready to send or forward any number of anonymous messages. If the peer is to start an anonymous message, it has to provide basic information that consists of the unique identifier of the destination peer, number of hops it wants the packet to do before it reaches the final destination (represented by H) and the message or packet itself.

It is important to stress the fact that when peers send messages it can be when they start a new message as mentioned before, but in most cases it will be because they are forwarding a message that was previously initiated by another peer. In either case the protocol used is the same.

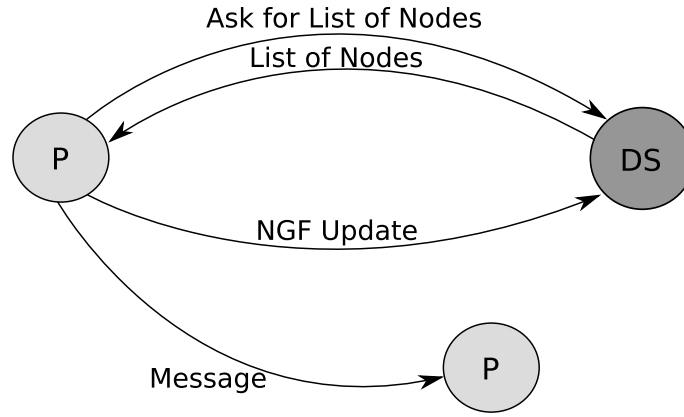



Figure 4.5: A representation of the interaction between a peer and the Directory Server and the same peer and another peer when sending or forwarding a message. When sending an anonymous message this will happen on every hop the message goes through.

The connection established between peers is a TCP connection and, as it was explained in section 4.2.2, it can use any port that ranges from 7770 to 7776. In these case the connections are temporary, they last just while the message is being sent. As soon as the entire message is transferred from one peer to another, the connection is closed.

The network cost of peers sending anonymous messages is represented by:

$$SendMessageCost \in (4NMH)$$

As shown in figure 4.5, for each peer that sends an anonymous message it needs to get an updated list of registered peers from the Directory Server which consists of two messages, one sent from the peer to the server requesting the list and another one from the server to the peer containing the list.

	Hops to Destination 					
	5	4	3	2	1	0
Peers	1	0.91234017	0.91234017	0.729872136	0.729872136	0.583897709
	2	0.84432016	0.84432016	0.675456128	0.67545613	0.67545613
	3	0.43788122	0.350304976	0.350304976	0.350304976	0.350304976
	4	0.22113341	0.22113341	0.22113341	0.22113341	0.22113341
	5	0.63112344	0.63112344	0.63112344	0.63112344	0.63112344
	6	0.27776611	0.27776611	0.27776611	0.27776611	0.27776611
	7	0.77654132	0.77654132	0.77654132	0.621233056	0.621233056

Peer that originates the message: 3
Destination Peer: 6
Number of hops: 5

The route of the message based on NGF: 3 - 1 - 2 - 7 - 1 - 6

Figure 4.6: The table shows the routing of an anonymous message that originated at peer 3, the final destination is peer 6. Read left to right, each cell contain NGF values and highlighted is the peer at which the message is at each hop until it reaches its destination. Notice how after a message has been at a peer the NGF value decreases. In figure 4.7 a graphical representation can be seen.

Every time a peer sends or forwards a message, its NGF gets consumed, meaning that the value will be decreased. On the prototype implementation a third message being an update of the new NGF value is sent to the Directory Server.

After the above tasks are completed the actual message or packet is delivered, which counts also as a message. All of these are hopped the amount of times specified by the sending peer. The protocol used for anonymous messages can be seen on figure 4.8. The first tag <MSG> tells the peer an anonymous message has been received and it will act according to the hops value.

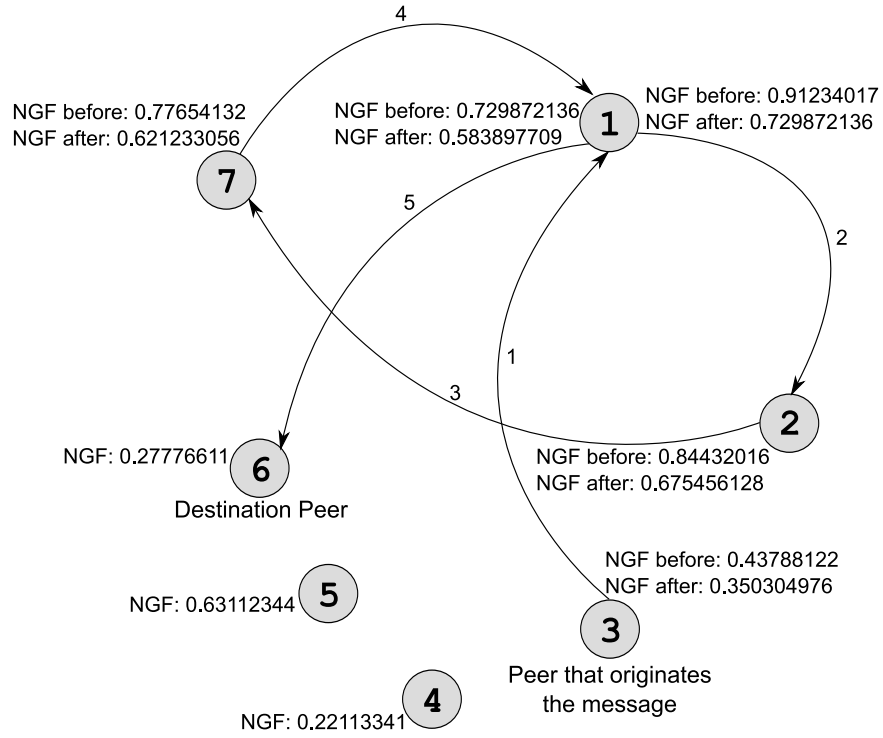


Figure 4.7: A diagram representing the route of the message from figure 4.6.

```
<MSG>id_destination>hops_left>message<END>
```

Figure 4.8: The protocol used by the peers to send anonymous messages.

Peer Updates

As it was explained before, each time a peer processes a message, either if it is generating a new one or forwarding one, it will consume some of its own NGF. In the prototype implementation this needs to be updated at the Directory Server because all peers rely on this value to decide the route packets to follow. This update consists of a single message sent from the peer to the server.

The network cost of a peer updating its NGF value with the server is represented by:

$$UpdateCost \in O(NM)$$

As shown in figure 4.9, a single message from the peer to the server is all it takes. Other peers receive this updated information when they request an updated list of peers as seen in the next section. The protocol used for the update messages can be seen in figure 4.10. The first tag `<NGF>` tells the server that this peer is updating its NGF value.

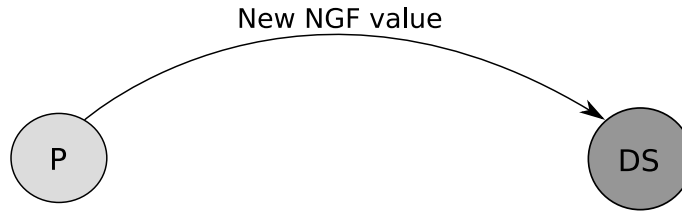


Figure 4.9: An interaction between a peer and the Directory Server when the peer updates its NGF value.

`<NGF>ngf<END>`

Figure 4.10: The protocol used by a peer when notifying the server of a NGF value update.

Peer Get List of Nodes

Before making the decision of the route a message should follow, each peer needs an updated list of the peers participating in the network. As was explained previously, this

list will contain each peer's NGF value. The size of this list will be $O(N)$. With this value, the peer is then able to determine the peer with highest NGF value and choose it as the next peer to deliver the message to.

The connection used for this is the same TCP connection established at startup using port 7777 which was explained in section 4.2.1.

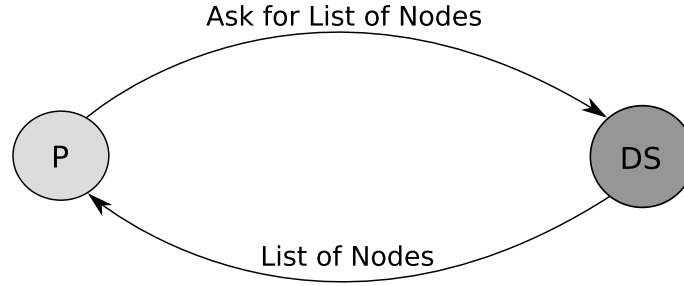


Figure 4.11: A representation of the interaction between a peer and the Directory Server when a peer requests a list of nodes.

The network cost of a peer requesting an updated list of peers to the Directory Server is represented by:

$$GetListCost \in O(2NM)$$

As shown in figure 4.11, for each peer a message is sent to the server requesting the list. The protocol used for this message can be seen in figure 4.12. In response, a message is received back from the server with the updated list. The updated list will contain different NGF values or will reflect the absence of peers that have left the network. An example of the protocol received from the server that contains the list of peers can be seen in figure 4.13. The first tag <LIST> differentiates this message from others. If what comes after is GET then it is the message sent to the server asking for the list of peers. Otherwise it will contain the list of peers.

The sign “ > ” is used to separate each value. The order and amount of each value is known and must be respected. In this way processing it with a string tokenizer is an easy task.

<LIST>GET<END>

Figure 4.12: The protocol used by a peer when requesting an updated list of peers to the server.

<LIST>id>ipaddress>port>ngf>...<END>

Figure 4.13: The message format used by the server when sending an updated list of peers.

Peer Shutdown

To reduce the amount of messages it was defined that if the connection of a peer is lost it means that the peer has left the network. It does not matter for the network if the peer left by its own will or because there was an error. This would be the case either from the server or from other peers perspectives.

There is no cost involved in the peer's shutdown, which is represented by:

$$ShutdownCost \in O(0NM)$$

4.3 Expected Benefits

The main goal, anonymity, is obtained in a very satisfactory way. The way the routing of messages is completed assures anonymity by eliminating any traceable information of previous peers. Of course, on top of that, there are encryption concerns of the messages which are not addressed here as the amount of work needed to implement this exceeds the scope of this project. Those aspects are left for future work.

The NGF value being the decisive factor when creating routes, and the fact that each time a message is processed by a peer this NGF value changes, gives SON high churn when building routes which is a very desirable aspect when trying to hide where the messages are coming from.

This implementation is simple but it gives a good base for achieving the main goal for this project and it can be adapted for specific needs. From these simple anonymous messages, many more complex systems can be built on top, depending on the specific application needs.

In terms of efficiency the system is not expected to be fast, but it should be scalable. At the moment, the Directory Server is a constraint to this, but should be addressed with future work that will replace it with a distributed solution. In chapter 6 possible attacks SON can suffer will be examined.

Chapter 5

Implementation

In this chapter the prototype implementation that gives proof of concept to SON is presented. An API with all classes and methods is given followed by a class diagram that shows the structure of the software.

5.1 Prototype

The first prototype of this design has been written in Java. It has been designed with full modularity and portability in mind. The idea is to provide a basic API other developers can use if they have a need for sending anonymous messages in a P2P way over the network for whatever reason. The API's scope is limited to the basic algorithms that provide the anonymity, as it is assumed that it will be adjusted later to each specific need.

Some basic guidelines will be followed:

- Each node will act as a Schwann Peer always.
- A node originating an anonymous message will act as a Presynaptic Neuron Peer and a node for which an anonymous message is intended will act as a Postsynaptic Neuron Peer only for the transaction of that message.
- A Directory Server will keep a record of all nodes that join and leave the network

and their corresponding properties.

- Protocols for all data that will be passed are defined according to the needs.

At startup, a Main Menu is presented which provides the user with the option of running as a Directory Server or as a Node, as can be seen in figure 5.1. This was done to give full portability and to keep everything in a single jar package. It is important to note that the Directory Server should be run at the host which address is hard coded into the implementation. It is done this way because nodes need to know about the “known” address of the Directory Server.



```
File Edit View Terminal Tabs Help
dancasmo@BlackBeauty:~$ java -jar "/home/dancasmo/Desktop/MSCNDS 2007-2008/Disse
rtation Project/SON_Implementation/SON2008/dist/SON2008.jar"
-----
SON --> Synapse Oriented Networks
A middleware API for Anonymous Communications
-----

+-----+
+- MAIN MENU -+
+-----+

0. Terminate
1. Run as Directory Server
2. Run as a Node

Option: █
```

Figure 5.1: A screenshot of the Main Menu.

5.1.1 Directory Server

Once an instance of a Directory Server is started it will:

1. Opens a socket on port 7777 and keeps listening for new connections on that port (son2008.server.Directory.listenSocket()).

2. When a new connection is received it will create an instance to handle this connection and it will be started as a separate thread so that the server can keep listening for more connections. This new thread is an instance of `NodeHandler`.
3. The new threaded instance (`son2008.server.NodeHandler`) of each new connected node will:
 - (a) Send the node a unique identifier which is a consecutive counter kept by the server.
 - (b) Keep listening for any data coming from the node.
 - (c) When data is received from the node it is processed according to its contents (`son2008.server.NodeHandler.receivedFromNode()`):
 - Tell the server the port it is listening on for other nodes to use it as a forwarder node.
 - Request a list of all the nodes in the network and their properties. There is also an option to print this information in the Directory Server as seen in figure 5.2.
 - Tell the server that the node has updated its NGF value.
4. When a connection between a node and the server is lost the node is removed from the list of nodes (`son2008.server.Directory.removeNode(int id)`).
5. Relevant output is constantly printed in the console which displays what is going on in the network as a basic monitoring tool. Of course in a production environment this will be disabled for security reasons and no output or logging will be kept. At any time the server will accept an input in the console for which “0” terminates the program and “1” prints a list of the current nodes as is shown in figure 5.3. If the program is terminated, all nodes terminate their program too.

```
File Edit View Terminal Tabs Help
+-----+
+-- MAIN MENU --+
+-----+

0. Terminate
1. Run as Directory Server
2. Run as a Node

Option: 1
Real IP address: 10.5.60.241
Note: If you are getting more than one IP address you might be behind a VPN.
Starting SON Directory Server...
Port 7777 opened on 10.5.60.241
----+----

+-----+
+-- DIRECTORY SERVER MENU --+
+-----+

0. Terminate
1. Show registered Nodes.

Option: █
```

Figure 5.2: A screenshot of the menu for the Directory Server.

```
File Edit View Terminal Tabs Help
+-----+

0. Terminate
1. Show registered Nodes.

Option: 1
----+----

Nodes registered with the Directory Server
-----
Node: 1 @ 10.5.60.241:7776 NGF: 0.1744484
Node: 2 @ 10.5.60.241:7775 NGF: 0.94250596
Node: 3 @ 10.5.60.241:7774 NGF: 0.79368067
Node: 4 @ 10.5.60.241:7773 NGF: 0.33028173

+-----+
+-- DIRECTORY SERVER MENU --+
+-----+

0. Terminate
1. Show registered Nodes.

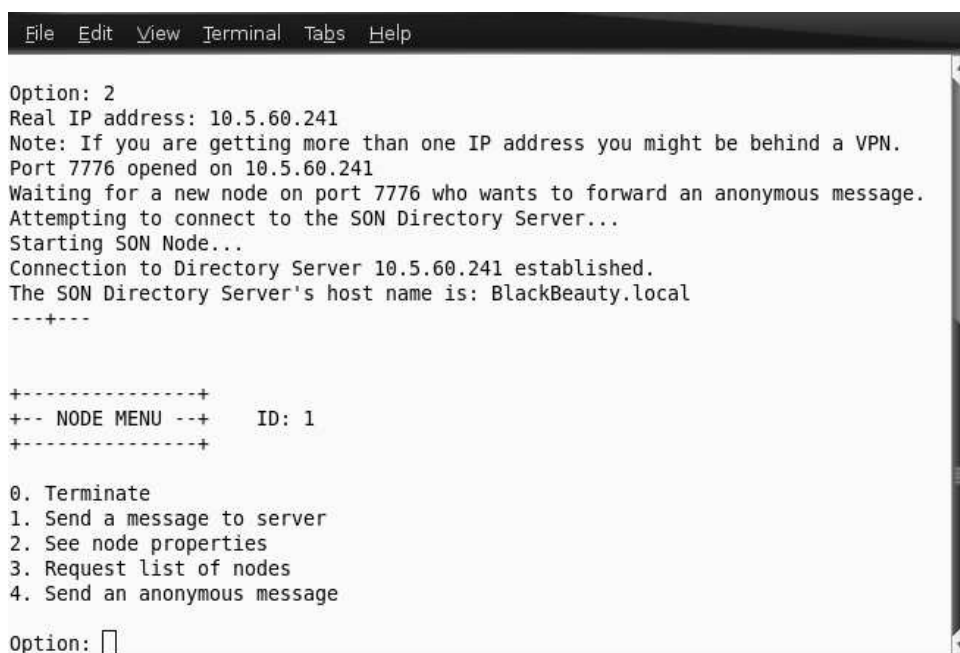
Option: █
```

Figure 5.3: A screenshot of the list of Nodes in the Directory Server.

5.1.2 Node

Once an instance of a Node is started it will:

1. It will attempt to connect to the Directory Server using it's "known" IP address and port 7777.
2. After a connection to the Directory Server is successful it will keep waiting for data sent from the server.



```
File Edit View Terminal Tabs Help

Option: 2
Real IP address: 10.5.60.241
Note: If you are getting more than one IP address you might be behind a VPN.
Port 7776 opened on 10.5.60.241
Waiting for a new node on port 7776 who wants to forward an anonymous message.
Attempting to connect to the SON Directory Server...
Starting SON Node...
Connection to Directory Server 10.5.60.241 established.
The SON Directory Server's host name is: BlackBeauty.local
---+---

+-----+
+-- NODE MENU --+   ID: 1
+-----+

0. Terminate
1. Send a message to server
2. See node properties
3. Request list of nodes
4. Send an anonymous message

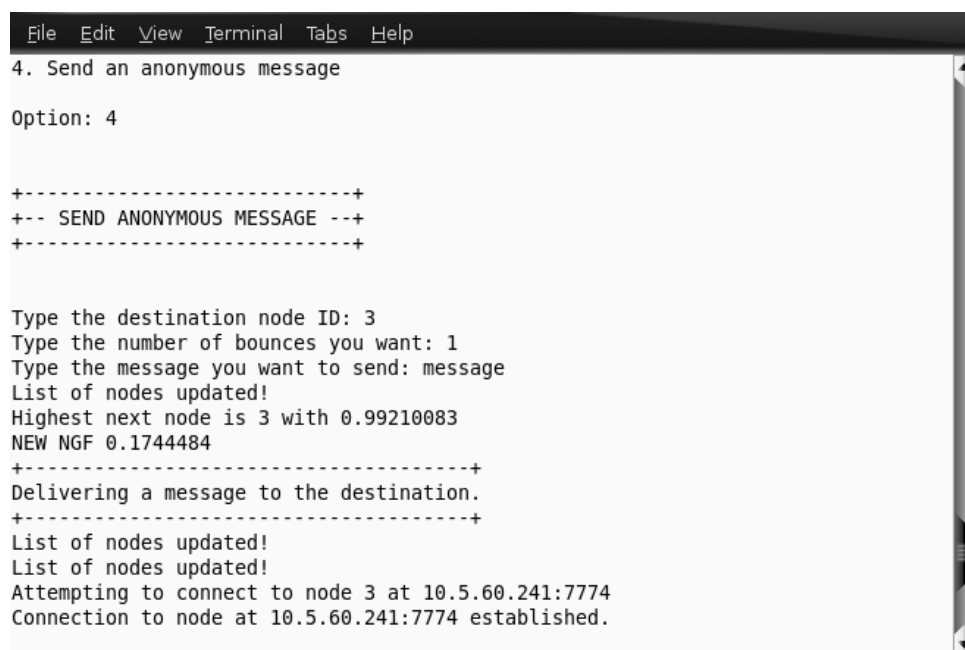
Option: 
```

Figure 5.4: A screenshot of the Node's menu.

3. It will start a forwarder service. It opens a socket on ports 7770-7776 and keeps listening for new connections from other nodes.
4. At any time, the node will accept an input in the console on its menu as seen in figure 5.4 for which "0" terminates the program, "1" allows a message to be sent to the server, "2" prints the node's properties, "3" requests a list of nodes and their properties to the server and "4" allows an anonymous message to be sent to another node specifying the destination node's IP, the number of bounces before the message is delivered and the message itself which can be seen in figure 5.5.

5. If data is received from the server it will be processed according to its contents:

- ID given. Immediately after a connection is established, the server sends the node its correspondent ID which the node updates in its properties.
- List of nodes received. This happens as a result of this node asking the server for the list so the node is expecting it. If the node was not expecting this no action is taken.
- A message, which could be meant for this node if the node were the final destination, or it could be meant for any other node in which case this node forwards it.

A screenshot of a terminal window with a dark title bar containing menu items: File, Edit, View, Terminal, Tabs, Help. The terminal text shows a sequence of commands and outputs for sending an anonymous message. It starts with '4. Send an anonymous message', followed by 'Option: 4'. A dashed border separates a section of text: 'Type the destination node ID: 3', 'Type the number of bounces you want: 1', 'Type the message you want to send: message', 'List of nodes updated!', 'Highest next node is 3 with 0.99210083', and 'NEW NGF 0.1744484'. Another dashed border follows, then 'Delivering a message to the destination.', another dashed border, and finally 'List of nodes updated!', 'List of nodes updated!', 'Attempting to connect to node 3 at 10.5.60.241:7774', and 'Connection to node at 10.5.60.241:7774 established.'

```
File Edit View Terminal Tabs Help
4. Send an anonymous message

Option: 4

+-----+
+-- SEND ANONYMOUS MESSAGE --+
+-----+

Type the destination node ID: 3
Type the number of bounces you want: 1
Type the message you want to send: message
List of nodes updated!
Highest next node is 3 with 0.99210083
NEW NGF 0.1744484
+-----+
Delivering a message to the destination.
+-----+
List of nodes updated!
List of nodes updated!
Attempting to connect to node 3 at 10.5.60.241:7774
Connection to node at 10.5.60.241:7774 established.
```

Figure 5.5: A screenshot of the options needed at a Node to send an anonymous message.

6. If a node receives a new connection on port 7770-7776, it will inspect the message and:

- (a) Determine the final destination and how many hops it has left.

- (b) If this hop is determined to be the last one, then the message is delivered to the final destination.
- (c) If the message still needs to be forwarded, it will decrease the hop value (H) and his own NGF value.

5.1.3 Protocols

For all communications between peers, protocols are defined according to the needs. These protocols are to be followed in order for the system to function.

- **Node to server**, startup message: `<START>port>ngf<END>`
- **Server to node**, list of nodes: `<LIST>id>ipaddress>port>ngf>...<END>`
- **Server to node**, assign ID: `<ID>id<END>`
- **Node to server**, ask for the list of nodes: `<LIST>GET<END>`
- **Node to node**, anonymous message: `<MSG>iddestination>bouncesleft>message<END>`
- **Node to server**, update of the NGF value: `<NGF>newngf<END>`

5.2 API

A language-dependant Application Programming Interface (API) that gives a set of classes and procedures that support the SON implementation will be explained next.

5.2.1 Packages

The software is divided into three packages and each package contains the necessary classes and methods as can be seen in table 5.1.

Packages	Description
son2008	Provides all the shared classes for Directory Server and Node.
son2008.node	Provides all the classes for a Node.
son2008.server	Provides all the classes for Directory Server.

Table 5.1: Table of the package distribution for the SON implementation.

5.2.2 Classes

As was shown in table 5.1, the implementation is divided into a package for Directory Server and another one for Nodes. There is also a third package which contains classes that are shared among both. A detail of the classes can be seen in table 5.2.

5.2.3 Methods

Each class presented in table 5.2 has methods which all together give functionality to SON. More details of the methods for each class can be seen in tables 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9.

Class	Description
son2008	
IPAddress	This class is created to address a limitation of Java. The method <code>InetAddress.getLocalHost().getHostAddress()</code> will not work properly on Unix or Linux environments. It will return the localhost addresss (127.0.0.1) and not the network address assigned to the computer. To make this implementation fully portable the method <code>getRealNetworkIP</code> is given.
Main	The main class that brings a command line menu UI. The user can choose to run as a Directory Server or as a Node. Depending on it running as a Directory Server or as a Node another menu is given for each.
Schwann	This class contains the node's properties. It also provides ways to get and update the values.
son2008.node	
AnonymousForwarder	Each Node will run one instance of this class. It will start a socket to listen for connections from other nodes at the provided ports. It provides methods to send and forward anonymous messages.
Node	This is the main class for a Node. It will start by connecting to the Directory Server, which for this version of the implementation, is a single known host and the port the Directory Server is listening to is the default 7777.
son2008.server	
Directory	This is the main class for the Directory Server. A node will connect to it and will receive a unique identifier. The Directory Server will keep a record of all connected nodes and the address to get to them. If a node wants to send an anonymous message, this class will send the node the list of all available nodes in the network.
NodeHandler	One instance of this class is created by the Directory Server for each client that connects to it. This class will always be threaded and it will keep listening on the socket connection waiting for new data from the node. If new data is received, it process it according to the content.

Table 5.2: Table of all the classes in each package for the SON implementation.

Class: IPAddress

Method	Description
String getRealNetworkIP()	This method will return the real network IP. It looks at all network interfaces on the computer and returns the first IP address it finds that does not start with 127 and does not contain ":" which means it is a IPv6 addresss.

Table 5.3: The class only contains one static method used to determine the real IP address of the host under any platform.

Class: Main

Method	Description
main (java.lang.String[] args) showMainMenu()	This is the main block of code. It runs the main menu and from there it loops into other menus depending on it being a Directory Server or a Node. This method displays the Main Menu where the user can choose between starting as a Directory Server or as a Node.
showDirectoryMenu()	If the option "Directory Server" was chosen from the main menu then this menu will be displayed which gives the relevant options for a Directory Server.
showNodeMenu()	If the option "Node" was chosen from the main menu then this menu will be displayed which gives the relevant options for a Node. These options are: 0. Terminate 1. Send a message to server 2. See node properties 3. Request list of nodes 4. Send an anonymous message
getAnonymousMsgOptions()	This will be called from the node menu when a node wants to send an anonymous message. It will ask for the relevant information which is: destination node id, number of hops the message should complete before it is delivered and the message to be sent.

Table 5.4: The methods contained in the Main class.

Class: Schwann

Method	Description
Schwann (int cell-Body, float NGF, String address) int getCellBody() setCellBody(int id) float getNGF() setNGF (float ngf) String getAddress() useNGF() int getPort() setPort (int p)	Constructor of this class. Receives almost all the properties as parameters. The port parameter is passed in later as in the nodes it will be chosen at run time. This method will return the value of the unique identifier of this node. This method will assign a value to the unique identifier of this node. This method will return the NGF of this node. This method will assign a new NGF value for this node. This method returns the address of this node. This method reduce the amount of NGF this node has each time the method is called. This method returns the port the node is listening on. This method will assign a new port number to this node.

Table 5.5: The methods contained in the Schwann class.

Class: AnonymousForwarder

Method	Description
AnonymousForwarder (Node p) listenSocket() forward(String msg) sendAnonymous-Message (int dest, int bounce, String msg, int next) int determineNext-Node()	This is the constructor of this class. It makes reference to the parent instance which is sent as a parameter. This method initialises the socket and loops waiting for new Nodes to connect. It will use any available port from the vector of ports provided. This is done so that several nodes can be run on the same host. This message handles all the logic on a node when a message is received. It will determine if it has completed the amount of hops determined by the original sender. If so then it will deliver the message to the final destination, if not, it will forward it to the node with the highest NGF. On each forward it will decrease the hop value for that message and it will reduce the NGF value on this node. This method forwards a message to the next node. It receives the values that form the actual message (dest, hops, msg) and it forms the message in the form of <MSG>dest>bounce>msg<END>. Once the message is formed it connects to the node which ID corresponds to next and delivers the message. This method will determine the node with the highest NGF value in the network. It will return the node's ID.

Table 5.6: The methods contained in the AnonymousForwarder class.

Class: Node

Method	Description
Node()	Constructor of this class. It will determine the node's real IP address. It starts with a random NGF value.
listenSocket()	This method establishes the connection with the Directory Server and then it loops waiting for data sent by it. When data is received it will call the method receivedFromDirectory().
receivedFromDirectory (String data)	This method will take the data received from the Directory Server and will process it according to its contents. The protocol of the messages are: For a message from the Directory Server with the id assigned to this node: <ID>id<END> For a message from the Directory Server that contains an updated list of nodes: <LIST>id>ipaddress>port>ngf>id>ipaddress>ngf>...<END> For a message from another node with an anonymous message: <MSG>iddestination>bouncesleft>message<END>
printNodes()	This method will print an updated list of nodes.
useNGF()	This method will decrease the NGF value by 20%.
updateNGF()	This method sends a message to the Directory Server with a new value for its own NGF. The protocol is: <NGF>ngfvalue<END>
convert2List()	This method converts the list of nodes received to an ArrayList of Schwann objects which makes it easier for further manipulation. The protocol to follow is: <LIST>id>ipaddress>ngf>id>ipaddress>ngf>...<END>
sendMsg2Server (String packet)	This method will send a message to the Directory Server with the value passed as a parameter.
askListOfNodes()	This method will send a message to the server asking for a list of nodes.
printProperties()	This method prints the node's own properties.
startAnonymous- Forwarder()	This method will start an instance of AnonymousForwarder. It will open a socket connection in the chosen port and will keep listening for nodes that will connect and deliver anonymous messages.
sendMessage (int to, int hops, String msg)	This method will start an anonymous message using the values received as parameters.
String getNodeIP (int id)	This method will return the IP address of a node given the ID as a parameter. It will return null if the node is not found.
int getNodePort (int id)	This method will return the port of a node given the ID as a parameter. It will return 0 if the node is not found.

Table 5.7: The methods contained in the Node class.

Class: Directory

Method	Description
Directory()	The constructor of this class. It will start the list of all nodes and it will determine its real IP using the IPAd-dress class.
listenSocket()	This method initialises the socket and loops waiting for new Nodes to connect to the Directory Server. For each new Node a new instance of NodeHandler is created, and a new identifier is assigned to it. Then, it is added to the list of Nodes and spawned as an individual thread.
removeNode (int id)	This method removes a node from the registered node's list. The id of the node to be removed is received as a parameter.
showNodes()	This method prints the list of Nodes that the Directory Server has registered.

Table 5.8: The methods contained in the Directory class.

Class: NodeHandler

Method	Description
NodeHandler (Directory p, Socket client, int id) run()	The constructor of this class. Receives client which contains the socket connection and id which is the unique identifier. This is the method that runs when the thread is started. It will send the node its unique identifier and it will keep listening on the socket for new data sent from the node. If the socket is closed and/or the connection is lost, it will take it as if the node has left the network.
receivedFromNode (String data)	This method will be called when data is received from a Node. It will receive the data as a parameter and it will decide what to do depending on the content received. The protocol of the messages are: For a message coming from the node with an update of the ngf value: <NGF>new_ngf_value<END> For a message coming from the node at the node's startup: <START>port>ngf<END> For a message coming from the node asking for an updated list of nodes: <LIST>GET<END>
sendList2Node()	This method will build a list of nodes and send it to the node. After receiving a message from the node asking for the list this method is called. The protocol of this message is: For the message sent to the node that contains the list: <LIST>id>ipaddress>port>ngf>id>ipaddress>ngf>...<END>

Table 5.9: The methods contained in the NodeHandler class.

5.2.4 Class Diagram

A UML diagram of all the classes, the methods and attributes can be seen on figure 5.6.

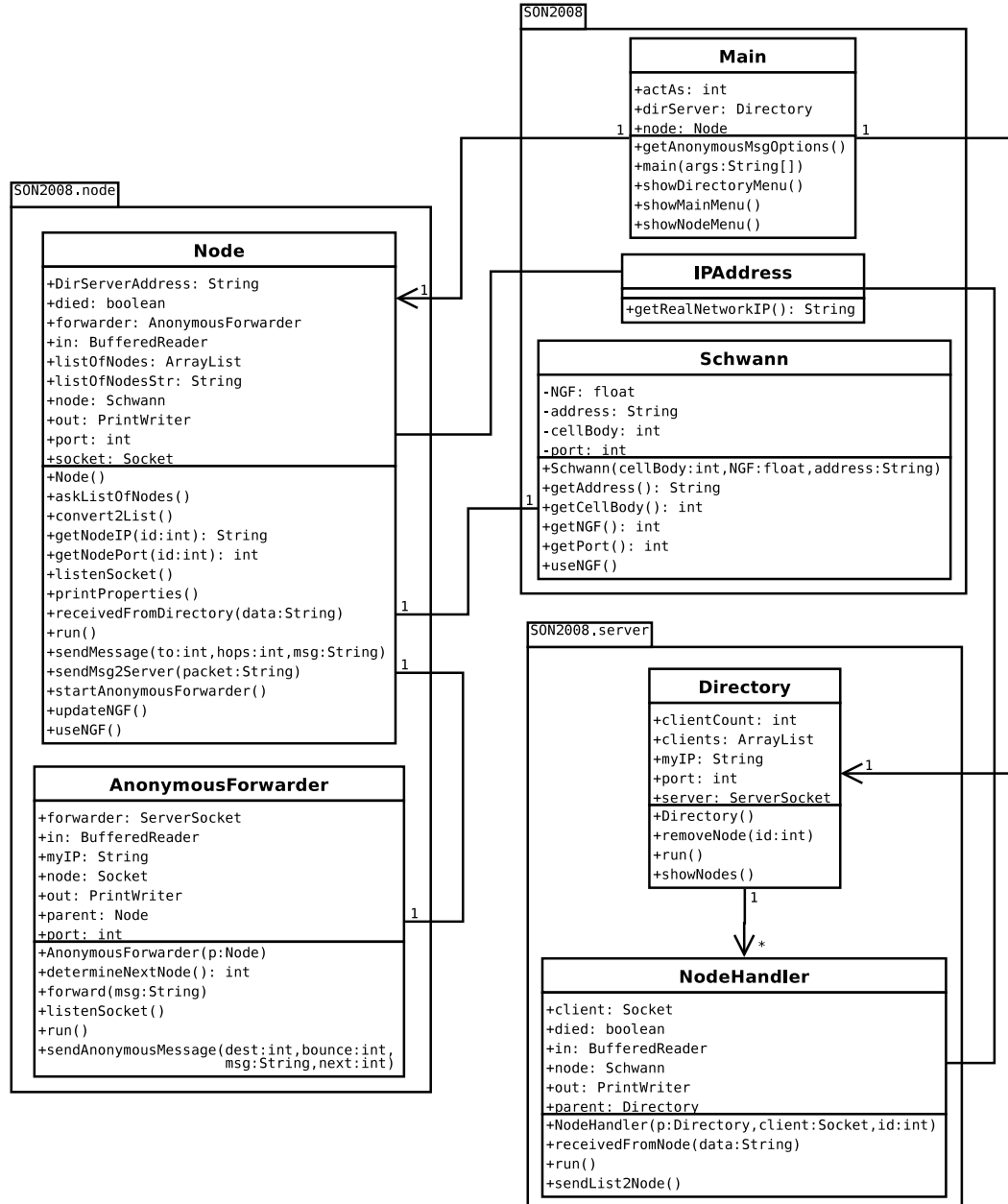


Figure 5.6: A UML class diagram of SON2008.

Chapter 6

Evaluation

Following the design presented in chapter 4, a general purpose middleware for P2P applications that required a high degree of anonymity between peers was implemented. This can be seen in chapter 5. The prototype produced successful results over anonymity which was the main goal. The prototype also produced some other drawbacks which will be evaluated in this chapter.

With the prototype built, a proof of concept was achieved which provided material for an analytical evaluation. Looking at the goals presented in chapter 1, it is important to evaluate how nature inspired the design of SON. This chapter will start by looking at the mapping done between Neuroscience and P2P systems, then it goes over anonymity. An evaluation of scalability and fault tolerance is given along with a possible alternative to minimize this. Finally an evaluation of the typical security attacks.

6.1 Domain Mapping Challenges

At the beginning of this dissertation, chapters 2 and 3 provided a in-depth introduction to Neuroscience and P2P Systems respectively. This was done to provide the reader with a solid background to understand the aspects that provided the inspiration to build SON. Even though it was challenging, the dissertation was very successful in terms of translating

Neuroscience to Distributed Systems.

In chapter 1 it was mentioned that the project was going to look at how neurons recover from injury in the peripheral nervous system and the supportive cells that work as helpers for neurons to build connections and find paths. The final outcome was that this behaviour was partially mapped into the design of SON. A table with all the mappings relevant to this project can be seen in section 4.1.5. The concept of “recovery from injury” gives an idea of re-building lost connections, which was irrelevant for the purposes of sending anonymous messages, therefore it was not taken into consideration. However, the creation of lost connections is a very appealing characteristic that could be applied when there is need to preserve paths and connections even after failure. The second concept of “the supportive cells that work as helpers for neurons to build connections” is very well applied. In fact, it constitutes the base for the SON model. The Schwann Peers are message forwarders (helpers) and they contain NGF which is the decisive factor to create routes.

6.2 Anonymity

As was specified in chapter 4, the main goal of the design was to provide a way to send anonymous messages in a P2P overlay network through indirection. By using basic algorithms and building routes based on NGF values, anonymity of peers was assured with high confidence.

No peer keeps a record of previous peers which assures anonymity at an application layer. At the IP datagram layer, anonymity is assured in terms of the sender (Presynaptic Neuron Peer) and receiver (Postsynaptic Neuron Peer) having no knowledge of each other. The IP datagrams only have knowledge of their immediate sender and receiver which, in this case, happens at each hop. The recipient can only know about the peer who delivered the message, which is highly unlikely to be the one who originated it, because the sender is able (and required) to specify the amount of hops it wants the message to complete

before it reaches the final destination. By varying the number of hops for each message (e.g., randomly) the sender ensures that intermediate Schwann Peers cannot deduce the source. In theory, anything above two hops is enough to ensure that the final destination does not know who sent the message. After a message completes the second hop it has successfully been inserted into the network anonymously. Any further hops from there on will just create network activity and make a difficult environment for an attacker.

Another aspect that help anonymity is routing. It is performed based on a parameter that constantly changes. This assures that the routes created are always different and unpredictable. As was explained in chapter 4, each peer has an NGF value which is the factor that determines the route to follow. This value is initiated randomly at each peer and it decreases by a percentage which makes the building of routes totally unpredictable.

6.3 Scalability and Fault Tolerance

The major drawback identified is scalability. In most modern distributed systems this is a required characteristic because systems need to be able to support growth without affecting functionality. There is a difference between managing and using the information. This prototype resolves the problem of using the information. As was clearly stated in section 4.2.1, for the managing of information a quick and simple directory server was implemented, which scales poorly.

Having this solution does not interfere with the proof of concept of the main goal which was to achieve anonymity.

The Directory Server is a centralized point that manages information about the participating peers which can be very easily attacked breaking the system's functionality. It was stated that this was only a temporary solution which needs to be changed in future work with a distributed solution.

6.3.1 Distributed Solution to the Directory Server

It is clear that the Directory Server provides very poor scalability and fault tolerance. Due to time constraints an implementation of a distributed solution was not possible. However, a proposal to address this is outlined below. At peer startup, a way to choose neighbours based on their NGF value is proposed. The NGF value is the decisive property to create routes when forwarding messages. Making use of this value to choose neighbours should provide very good results when trying to build a totally random and unpredictable layout. The algorithm to achieve this can be seen below.

1. All Schwann Peers will have a multicast listener which is initialized at startup.
2. When a new peer wants to join the network it will send out a message with its NGF value, IP address and port to a Internet multicast communication group. All other Schwann Peers that receive this message will compare the peer's NGF value to their own and if the value is within a configurable proximity then they will reply back with their properties through a direct TCP connection on the IP address and port provided.
3. The new peer will start receiving connections of peers with their properties and will build a list of them. This list will have a configurable size limit. For purposes of this example consider the limit to be one hundred.
4. From the initial list the peer will then create the actual list of neighbour peers. This list is also of configurable size. For purposes of this example consider that size to be ten.
5. The pattern proposed in section 6.4.2 to solve the NGF forgery problem will also be applied here. The new peer will instantiate a random value and choose the closest NGF value to that random value. Once found it will add that peer to the list of neighbours. It will do this until the list of neighbours is complete.

6. Once the list of neighbours is built the peer can start behaving as a Neuron or Schwann Peer.

6.4 Possible Attacks

Some other typical security attacks create threats that could affect the system's functionality or the data being handled. Fortunately the vulnerabilities can be identified and isolated, so that specific solutions can be implemented for each. Some of the possible attacks and recommendations on how to face them will be given below. They were compiled from a paper written by Foner [35].

6.4.1 Eavesdropping

Most network communications occur in an unsecured format, which makes it easy for an attacker who has gained access to the network to interpret the traffic. This technique is referred to as sniffing or eavesdropping. The ability of an eavesdropper to monitor the network is a big security problem that network administrators face. Without strong encryption services that are based on cryptography, data can be read by others as it traverses the network.

Intercepting a Presynaptic Neuron Peer

An eavesdropper will have special interest in intercepting the first message before it becomes anonymous. If the attacker succeeds in doing this, anonymity cannot be assured no matter how many hops the message completes.

The model addresses this issue through its message format. There is no difference made between new messages or forwarded messages because they all have the same format. The attacker is therefore unable to tell which are new messages. Though guessing is always an option for the attacker, it will provide no practical utility.

6.4.2 Data Modification

Once an attacker has accessed the data, it can be altered. Data modification can be done by the attacker without the knowledge of the sender or receiver. Even when confidentiality is not an issue, messages should not be modified during transit at any time because it can affect functionality, but mainly because the receiver will receive data which was not issued by the sender and which could be harmful.

NGF Value Forgery

A malicious peer will always have the option to fake its own NGF value with the highest value possible. The effects of such an action are that the attacker will constantly be chosen as the next hop, increasing the chances for the attacker to intercept new messages before they are introduced into the network anonymously.

The following solution to this attack is proposed: Instead of choosing the highest NGF value as the next hop, the forwarding peer will instantiate a random number and choose the closest NGF value to that random number as the next hop. With this solution faking the NGF value will have no useful results for the attacker.

6.4.3 Identity Spoofing

In most networks and operating systems the IP address is used to identify a valid host's entry. It is possible for an attacker to falsify an IP address creating identity spoofing. With that power the attacker might also construct IP packets that appear to originate from a valid address.

If access to the network is gained and a valid IP address is faked, subject to IP routing constraints, the attacker can modify, reroute, or delete data.

6.4.4 Man-in-the-middle

This happens when an attacker between two peers that are communicating is actively monitoring, capturing, and controlling the communication transparently. For example, the attacker can re-route a data exchange.

Man-in-the-middle attacks are like someone assuming a peer's identity in order to read the messages intended for that peer. The peer on the other end might believe it is the right peer it is communicating to, the attacker might be actively replying as it, to keep the exchange going and gain more information.

6.4.5 Replay

This happens when an attacker is able to grab packets off of the network. After packets are captured, the attacker can simply extract information from the packets and analyse it. Once the information is extracted, the captured data can be placed back on the network or replayed.

6.5 Evaluation of the Implementation

Implementation challenges specific to the programming language where faced and they are explained below.

6.5.1 Simplicity

As can be seen in chapter 5 the prototype implemented is very simple with just enough functionality to provide a general middleware that allows peers to send messages anonymously. The code is well commented and self-explanatory.

6.5.2 Portability and Re-usability

The implementation of SON was built not for a specific platform but thinking on full portability. That is the main reason why Java was the chosen programming language because any Java Virtual Machine (JVM) will run the code. An IP addressing cross-platform issue was encountered which is explained further in the following section. It was identified and fixed giving consistent results over different operating systems.

6.5.3 Real IP Address

Java provides a class `java.net.InetAddress.getHostAddress()` which is supposed to return the IP address of the host in textual presentation. Under Windows calling this method works well and it returns the correct network address of the host. Unfortunately under Unix/Linux operating systems calling this method will, in most cases, return the local host IP address which as a standard is 127.0.0.1.

Fortunately, the prototype was implemented on such operating systems so the bug was identified at an early stage. The method `IPAddress.getRealNetworkIP()` was implemented, which will loop on each network interface present on the system and look at the IP address for each interface and will discard all those that start with 127 (localhost), 169 (invalid IP address, probably no DHCP response) and the ones that contain a colon (":") which means they are IP version 6. Any address remaining will be the real network address. It could happen that a computer has more than one network address in which case it will return the first one found.

This class was then tested over several different operating systems giving positive results on all of them. Solving this issue was of great importance for the prototype because the IP address of each peer is necessary for the system to work.

6.5.4 References to Parent Class

Due to the modular nature of the code, child objects often need to access their parent's data. That data had to be updated and consistent for it to be usable. The way this was approached was by sending a reference of the parent class on the child's constructor when initialising the instance, and assigning it internally to an object of the same type that could be accessed at any moment.

Chapter 7

Conclusions

This project is based on how the Schwann cells act as helpers for neurons to build connections between them. This characteristic has been taken and applied to P2P systems to provide a general middleware to send anonymous messages. In cases where a peer wants to remain anonymous, this is very useful, especially when social concerns are involved. For example, someone who needs to send an anonymous message to report a problem that, if they do it publicly, the affected (reported) party can identify them, it might get them into trouble. Providing a basic middleware to build such anonymous applications will become popular to address this type of concerns.

With this prototype of SON, Peers use NGF values to decide how the routes are built in real time. The fact that the NGF value gets updated each time a message is forwarded gives high churn and unpredictable routing. This kind of unpredictable routing gives stronger anonymity because the route of a message is harder to trace. Messages are sent by a Peer and received at the intended destination giving real anonymity to the sender. With this prototype, it was possible to give a proof of concept to the algorithms implemented, and it also demonstrated that on top of a simple implementation more application-specific processes can be put in place.

For this dissertation a new field was investigated which had not been studied previously by the author. As was appreciated in chapter 2, Neuroscience is a very wide and complex

field. In fact, it is so extensive that just a few aspects relevant for this dissertation were identified and elaborated on. It is a fact that the way things work in nature can be compared to computer systems which is enough proof that nature concepts can be applied to distributed systems. The difficult part was finding a sufficiently interesting aspect that could be applied to something that has not been addressed yet or had been addressed but in a different way or not to its full extent.

The amount of research completed and results obtained are satisfactory. The implementation and evaluation parts of this dissertation are not as big and elaborated but the theoretical part behind is considerable and concise.

7.1 Future Work

A huge part of this dissertation was dedicated to provide a good understanding of the multiple domains covered: Neuroscience, P2P Systems and security. Therefore, it gives an elaborated theoretical background. Future work is expected on SONs to make it a robust anonymous platform. Future researchers are encouraged to look at what is considered to be the most relevant aspects to work on in the near future. These are to provide a distributed solution for managing the information and to provide encryption and secure channels to send information.

It is of interest that this dissertation will motivate people to look into Nature aspects and implement more Nature-Inspired Distributed Systems. Ideally, if someone takes this dissertation as a base for more research and elaborate more on the implementation part of it to build a more robust, secure and stable architecture of SON.

Appendix A

Abbreviations

Short Term	Expanded Term
P2P	Peer-to-Peer
API	Application Programming Interface
PC	Personal Computer
CPU	Central Processing Unit
PDA	Personal Device Assistant
CAN	Content Addressable Network
KSK	Keyword-Signed Key
SSK	Signed-Subspace Key
CHK	Content-Hash Key
PGP	Pretty Good Privacy
SON	Synapse Oriented Network
TCP	Transfer Control Protocol
JVM	Java Virtual Machine
CNS	Central Nervous System
PNS	Peripheral Nervous System
NGF	Nerve Growth Factor
BDNF	Brain-Derived Nerve Growth Factor
NT-3	Neurotrophin-3
NT-4	Neurotrophin-4

Appendix B

SON Implementation Code

Bibliography

- [1] Laurie, Lundy-Ekman. (1998) *Neuroscience: Fundamentals for Rehabilitation*. W.B. Saunders Company, USA.
- [2] Purves, D., Augustine, G., Fitzpatrick, D., Hall, W., LaMantia, A., McNamara, J.,Williamos, S. (2004) *Neuroscience:Third Edition* Sinauer Associates, Inc., USA.
- [3] Longstaff, A. (2005) *Neuroscience* Second Edition. Taylor and Francis Group, USA.
- [4] Navarro, X., Vivo, M., Valero-Cabre, A. (2007) *Neural plasticity after peripheral nerve injury and regeneration*. Progress in Neurobiology 82, 163201.
- [5] Horner, P., Gage, F. (2000) *Regenerating the damaged central nervous system*. NATURE, Vol. 407, 963-970.
- [6] Silver, J., Miller, J. (2004) *REGENERATION BEYOND THE GLIAL SCAR*. Nature Publishing Group, Vol. 5, 146-156.
- [7] Ramer, M., Priestley, J., McMahon, S. (2000) *Functional regeneration of sensory axons into the adult spinal cord*. NATURE, Vol. 403, 312-316.
- [8] Schwab, M. (2002) *Repairing the Injured Spinal Cord*. SCIENCE, Vol. 295, 1029-1031.
- [9] Pearse, D., Pereira, F., Marcillo, A., Bates, M., Berrocal, Y., Filbin, M., Bunge,

- Mary. (2004) *cAMP and Schwann cells promote axonal growth and functional recovery after spinal cord injury*. NATURE MEDICINE.
- [10] Burnett, M., Zager, E. (2004) *Pathophysiology of peripheral nerve injury: a brief review*. Neurosurg. Focus, Vol. 16.
- [11] Singh, A., Haahr, M. (2006) *CREATING AN ADAPTIVE NETWORK OF HUBS USING SCHELLING'S MODEL*. Communications of the ACM, Vol. 49, 69-73.
- [12] Chakravarti, A., Baumgartner, G., Lauria, M. (2005) *The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network*. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS PART A: SYSTEMS AND HUMANS, Vol. 35, 373-384.
- [13] Daly, E., Haahr, M. (2007) *Social Network Analysis for Routing in Disconnected Delay-Tolerant MANETs*. Distributed Systems Group, Trinity College Dublin.
- [14] Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z. (2003) *Peer-to-Peer Computing*. HP Laboratories Palo Alto.
- [15] Stutzbach, D., Rejaie, R. (2006) *Understanding Churn in Peer-toPeer Networks*. ACM, 189-201.
- [16] Lui, S., Kwok, S. (2001) *Interoperability of Peer-To-Peer File sharing Protocols*. ACM SIGecom Exchanges, Vol. 3, No. 3, 25-33.
- [17] Sit, E., Morris, R. (2002) *Security Considerations for Peer-toPeer Distributed Hash Tables*. Springer-Verlag Berlin Heidelberg: IPTPS, 261-269.
- [18] Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M. (2004) *Is P2P dying or just hiding?* Proceedings of the GLOBECOM 2004 Conference, IEEE Computer Society Press.

- [19] Daswani, N., Garcia-Molina, H., Yang, B. (2003) *Open Problems in Data-Sharing Peer-to-Peer Systems*. Springer-Verlag Berlin Heidelberg: ICDT, 1-15.
- [20] Castro, M., Druschel, P. Ganesh, A., Rowstron, A., Wallach, D. (2002) *Secure routing for structured peer-to-peer overlay networks*. OSDI '02.
- [21] Ramanathan, M., Kalogeraki, V., Pruyne, J. (2002) *Finding Good Peers in Peer-to-Peer Networks*. Proceedings of the International Parallel and Distributed Processing Symposium.
- [22] Liben-Nowell, D., Balakrishnan, H., Karger, D. (2002) *Analysis of the Evolution of Peer-to-Peer Systems*. ACM, 233-242.
- [23] Yang, B., Garcia-Molina, H. (2000) *Comparing Hybrid Peer-to-Peer Systems*. Technical report.
- [24] Androutsellis-Theotokis, S., Spinellis, D. (2004) *A Survey of Peer-to-Peer Content Distribution Technologies*. ACM Computing Surveys, Vol. 36, No. 4, pp. 335-371.
- [25] Clarke, I., Sandberg, O., Wiley, B., Hong, T. (2000) *Freenet: A Distributed Anonymous Information Storage and Retrieval System*. Lecture Notes in Computer Science, Vol. 2009 pg. 46.
- [26] Markram, H. (2006) *The Blue Brain Project*. Nature Reviews, Vol. 7 pp. 153-160.
- [27] Mostyn, M. (2000) *The Need for Regulating Anonymous Remailers*. International Review of Law, Computers & Technology, 14:1, 79-88.
- [28] du Pont, G. (2001) *THE TIME HAS COME FOR LIMITED LIABILITY FOR OPERATORS OF TRUE ANONYMITY REMAILERS IN CYBERSPACE: AN EXAMINATION OF THE POSSIBILITIES AND PERILS*. JOURNAL OF TECHNOLOGY, LAW & POLICY, Vol. 6 pg. 176-218.

- [29] Danezis, G., Dingledine, R., Mathewson, N. (2003) *Maxminion: Design of a Type III Anonymous Remailer Protocol*. IEEE Symposium on Security and Privacy.
- [30] Chaum, D. (1981) *Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms*. Communications of the ACM, Vol. 24, No. 2.
- [31] Reed, M., Syverson, P., Goldschlag, D. (1998) *Anonymous Connections and Onion Routing*. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, Vol. 16, No. 4.
- [32] Denning, D., Baugh, W. (1999) *HIDING CRIMES IN CYBERSPACE*. Information, Communication & Society, 2:3, 251-276.
- [33] Becker, D., Sterling, T., Dorband, J., Savarese, D., Ranawake, U., Packer, C. (1995) *BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION*. In Proceedings of the 24th International Conference on Parallel Processing.
- [34] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H. (2001) *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*. ACM, pg. 149-160.
- [35] Foner, L. (1996) *A Security Architecture for Multi-Agen Machmaking* The First International Conference on Multi-Agent Systems.
- [36] Fung, M. (2008) *The expression of neurotrophins in a longevity study for learning and memory retention by morris water maze paradigm and long term potentiation in aged rats*. Unpublished MSc.Neuroscience, Department of Physiology. Trinity College Dublin. Personal Communication.
- [37] User: Grm_wnr from Wikipedia. "Diagram of a human central nervous system." http://en.wikipedia.org/wiki/Image:Central_nervous_system.svg. Accessed on July 5th, 2008.

- [38] Shirky, C. (2000) *What It P2P ... And What Isn't?* O'Reilly Network Article, <http://www.openP2P.com/pub/a/472>. Accessed on May 26th, 2008.
- [39] Oxford English Dictionary. <http://dictionary.oed.com/> Accessed on July 31st, 2008.
- [40] Anonymous Email, Privacy and Security Information and Tutorials. <http://anonymous.to>, Accessed on August 1st, 2008.
- [41] Cypherpunk anonymous remailer.
http://en.wikipedia.org/wiki/Cypherpunk_anonymous_remailer, Accessed on August 1st, 2008.
- [42] How to Send an Anonymous Email Message.
<http://email.about.com/cs/anonemailtips/qt/et041304.htm>, Accessed on August 1st, 2008.
- [43] Anonymous remailer. http://en.wikipedia.org/wiki/Anonymous_remailer, Accessed on August 2nd, 2008.
- [44] Mixmaster anonymous remailer.
http://en.wikipedia.org/wiki/Mixmaster_anonymous_remailer, Accessed on August 2nd, 2008.
- [45] Howto use a Type II Anonymous Remailer (Mixmaster).
<http://feraga.com/node/74>, Accessed on August 2nd, 2008.
- [46] Mixmaster. <http://mixmaster.sourceforge.net/>, Accessed on August 2nd, 2008.
- [47] Mixminion. <http://en.wikipedia.org/wiki/Mixminion>, Accessed on August 2nd, 2008.
- [48] Mixminion: A Type III Anonymous Remailer. <http://mixminion.net/>, Accessed on August 2nd, 2008.