Title : Replication and availability in decentralised online social networks

Name : Adil Hassan

REPLICATION AND AVAILABILITY IN DECENTRALISED ONLINE SOCIAL NETWORKS

ADIL HASSAN

MPHIL

2017

UNIVERSITY OF BEDFORDSHIRE

REPLICATION AND AVAILABILITY IN DECENTRALISED ONLINE SOCIAL NETWORKS

by

ADIL HASSAN

A thesis submitted to the University of Bedfordshire in partial fulfilment of the requirements for the degree of Master of Philosophy

REPLICATION AND AVAILABILITY IN DECENTRALISED ONLINE SOCIAL NETWORKS

ADIL HASSAN

ABSTRACT

During the last few years' online social networks (OSNs) have become increasingly popular among all age groups and professions but this has raised a number of issues around users' privacy and security. To address these issues a number of attempts have been made in the literature to create the next generation of OSNs built on decentralised architectures.

Maintaining high data availability in decentralised OSNs is a challenging task as users themselves are responsible for keeping their profiles available either by staying online for longer periods of time or by choosing trusted peers that can keep their data available on their behalf.

The major findings of this research include algorithmically determining the users' availability and the minimum number of replicas required to achieve the same availability as all mirror nodes combined. The thesis also investigates how the users' availability, replication degree and the update propagation delay changes as we alter the number of mirror nodes their online patterns, number of sessions and session duration. We found as we increase the number of mirror nodes the availability increases and becomes stable after a certain point which may vary from node to node as it directly depends on the node's number of mirror nodes and their online patterns. Moreover, we also found the minimum number of replicas required to achieve the same availability as all mirror nodes combined and update propagation delay directly depends on mirror nodes' number of sessions and session duration. Furthermore, we also found as we increase the number of sessions with reduced session lengths the update propagation delay between the mirror nodes starts to decrease. Thus resulting in spreading the updates faster as compared to mirror nodes with fewer sessions but of longer durations.

LIST OF CONTENTS

Abstract

LIST OF TABLES V

LIST OF FIGURES VI

Dedication VIII

Acknowledgements IX

CHAPTER 1: INTRODUCTION 1
1.1 Background and History of Online Social Networks 2
1.2 The Problem 3
1.3 Aim 5
1.4 Research Objectives 5
1.5 Hypothesis 5
1.6 Research Questions 5
1.7 Structure of the Thesis 6

CHAPTER 2: LITERATURE REVIEW 7
2.1 Centralised Online Social Networks 8
2.2 Decentralised Online Social Networks 9
2.3 Challenges and Opportunities 10
2.3.1 Storage and Data Availability 10
2.3.2 Overhead 11
2.3.3 Leverage Social Relationships 12
2.4 Related Work 12
2.4.1 PeerSon 12
2.4.2 Safebook 13
2.4.3 SuperNova 13
2.4.4 DECENT 13
2.4.5 Cachet 13
2.4.6 Vis-à-Vis 14
2.4.7 FOAF 14
2.4.8 Diaspora 14
2.5 Research Gap Analysis 16
2.6 Requirements 19
2.6.1 Functional Requirements 19
2.6.2 Non-Functional Requirements 19

CHAPTER 3: RESEARCH METHODOLOGY AND SYSTEM DESIGN 21
3.1 Research 22
3.2 Research Methodologies 22
3.2.1 Quantitative Research Methodology 22
3.2.1.1 Descriptive 23
i) Surveys 23
ii) Longitudinal 23
iii) Cross-Sectional 24
3.2.1.2 Correlational design 24
3.2.1.3 Group Comparison 24
i) Ex Post Facto Design 24
ii) True-Experimental Design 25
3.2.2 Qualitative Research Methodology 25
3.2.2.1 Action Research 26
3.2.2.2 Case Study 26
3.2.2.3 Grounded Theory 26

3.2.3 Design Science Research Methodology 27
3.2.3.1 Problem Awareness 27
3.2.3.2 Objective Setting 27
3.2.3.3 Design and Development 27
3.2.3.4 Demonstration 28
3.2.3.5 Evaluation 28
3.2.3.6 Communication 28
3.2.4 Conclusion 28
3.3 System Design 30
3.3.1 Availability 30
3.3.2 Update Propagation Delay 33
3.3.3 Replication Degree 35
3.3.4 Availability on Demand 36
3.4 Conclusion 38
CHAPTER 4: RESULTS AND DISCUSSION 38
4.1 The Context 40
4.2 Model (Objective 2) 41
4.3 Simulation 43
4.4 Availability (Objective 3) 44
4.5 Update Propagation Delay (Objective 4) 48
4.6 Replication Degree (Objective 5 and 6) 50
4.7 Availability on Demand (Objective 7) 53
4.8 Conclusion 56
CHAPTER 5: CONCLUSION AND FUTURE WORK 57
5.1 Conclusion 57
5.2 Limitations and Future Work 62
5.2.1 Enhancement of Data 62
5.2.2 Enhancement of Algorithms 62
5.2.3 Enhancement of Research Scope 63
GLOSSARY OF TERMS 64
PUBLICATION 65
REFERENCES 65
Appendix 1 – Availability Algorithm 71
Appendix 2 – Update Propagation Delay Algorithm 72
Appendix 3 – Replication Degree Algorithm 74
Appendix 4 – Availability on Demand Algorithm 76
Appendix 5 – Code: Availability, Update Propagation Delay, Replication Degree 77
Appendix 6 – Code: Availability on Demand 94

LIST OF TABLES

Table 1: Availability in decentralised online social networks _____ 15
Table 2: Research methodologies _____ 29
Table 3: Research objectives, assessment method and outcome _____ 41
Table 4: Simulation results of availability and replication degree algorithms _____ 46
Table 5: Simulation results of update propagation delay algorithm _____ 49

LIST OF FIGURES

Figure 1: Quantitative research methods _____ 23
Figure 2: Availability algorithm _____ 32
Figure 3: Update propagation delay _____ 33
Figure 4: Update propagation delay algorithm _____ 34
Figure 5: Replication degree algorithm _____ 36
Figure 6: Availability on demand algorithm _____ 37
Figure 7: Model _____ 42
Figure 8: Model - graph _____ 43
Figure 9: Availability - graph _____ 43
Figure 10: Core node's diurnal availability (Mirror Nodes: 25) _____ 47
Figure 11: Update propagation delay (Mirror Nodes: 25) _____ 47
Figure 12: Core node's diurnal availability by hour (Mirror Nodes: 25) _____ 48
Figure 13: Update propagation delay – 100 Mirror Nodes _____ 48
Figure 14: Update propagation delay - 500 Mirror Nodes _____ 49
Figure 15: Replication degree - Model A _____ 51
Figure 16: Replication degree - Model A - Result _____ 52
Figure 17: Replication degree - Model B _____ 52
Figure 18: Replication degree - Model B Result _____ 53
Figure 19: Model - Availability on demand _____ 54
Figure 20: Availability on demand - 3 days _____ 55
Figure 21: Availability on demand - 30 days _____ 55
Figure 22: Availability on demand - 60 days _____ 56

DECLARATION

I declare that this thesis is my own unaided work. It is being submitted for the degree of MPhil at the University of Bedfordshire.

It has not been submitted before for any degree or examination in any other University.

Name of candidate: Adil Hassan Signature: Adil Hassan

Date: 17th April 2017

Dedication

I dedicate this project to my parents Mr. Manzoor Hussain and Mrs. Jamshaid Akhtar and my lovely sister Miss. Farzana Yasmeen.

Acknowledgements

First, I would like to thank my ALLAH (S.W.T) for all HIS (S.W.T) blessings upon me and giving me the ability to write this Thesis. I would also like to thank my last Prophet Hazrat Muhammad (S.A.W) and HIS (S.A.W) Noble Family of Ahlul-Bait (A.I/S.A) for all their blessings upon me and my family.

I would like to thank and express my deepest gratitude to my supervisor Dr. Marc Conrad for his excellent guidance, motivation, patience and constant support throughout my research. During my studies I have been through some serious ups and downs of life but Dr. Marc Conrad was always there to support and encourage me to get through those difficult times. I truly and whole-heartedly believe without Dr. Marc Conrad's constant support this thesis would not have been possible.

I would also like to thank my parents, my sister and my elder brothers for supporting and encouraging me with their best wishes.

CHAPTER 1: INTRODUCTION

In general, Online Social Networks (OSNs) are digital representations of the individuals representing their interests, hobbies, and relationships to the outer world. The motivation for an individual to join a social network is to create a profile and share information with other members of the network or with the selected network of friends. Over the past few years, the popularity of OSNs has grown tremendously generating huge amount of users' sensitive data online (Falahi, Atif & Elnaffar, 2010).

The centralised architecture of OSNs and commercial nature of service providers raises a number of issues around users' privacy and security. Therefore, more effective and flexible security measures are required for the protection of users' privacy and for the continued growth of OSNs. For that reason, we envision a new paradigm shift towards the creation of next generation of OSNs that may address the drawbacks of the traditional OSNs and offer more secure social networking platform to its users.

1.1 Background and History of Online Social Networks

The history of OSNs goes beyond the birth of the Internet. The project, called *"Community Memory"* was the first public computerised bulletin board system established in 1973 in Berkeley, California. This allowed the users to enter and retrieve messages between different computer terminals and share information with other members of the community (Doub, 2016). Late in 1980's, with the passage of time and advancement in technology the world has seen a revolution. This revolution began with the advent of the Internet that has made the world a global village. With the growth of the Internet usage, people started to create web applications. This gave birth to a new era of OSNs. In 1995, when Classmates made its presence it attempted to reconnect people who had attended the same school/college and allowed them to stay in touch. At the time of its release, Classmates did not allow its users to create profiles or list their friends; these features were added in the later years (Boyd & Ellison, 2007). In 1997, SixDegrees was launched and was the first social network of its kind that supported features like creating profiles, listing friends and browse friends' friend list. It managed to attract millions of users, but shortly after 3 years of its launch, the service was closed in 2000. The next major social network, Friendster appeared in 2002 and was designed to compete with Match.com, a profitable online dating website. While most of the dating websites focused on introducing users to strangers sharing common interests, However, Friendster took a different approach and helped its users to find a better match through 'friend of a friend' relationship. It became successful in attracting millions of users, but with time Friendster lost its popularity for a number reasons including social collisions, technical issues and a rupture of trust between the users and the organisation (Boyd & Ellison, 2007). In 2003, MySpace was launched to compete with Friendster and few others. MySpace wanted to target the estranged Friendster users and with the support of indie-rock bands (who were expelled from Friendster) MySpace gained a rapid popularity in a short time. To attract more users MySpace allowed local promoters to advertise VIP passes for popular clubs. This also gave the opportunity to fans to connect with their favourite bands and vice versa (Boyd & Ellison, 2007) (Sherchan, Nepal & Paris, 2014). In 2004, a new era of social networking started with the birth of Facebook that initially had a user database of Harvard students/graduates only. Unlike other social networks, Facebook was designed to support distinct college networks and to join Facebook one must have to have a .edu account. The popularity of Facebook started to spread from Harvard to other universities, high schools and school-going students. In 2006, Facebook was open to public to create profiles, build relationships and find friends. After Facebook, we have seen a different kind of social network called Twitter. Twitter was categorised as a micro blogging service that uses a completely different relationship model i.e. follower and followee model to connect to people. In 2011, Google+ and Pinterest were released. Google+ extended the traditional relationship model of fiends to family, acquaintances, and following. It also allowed its users to create their own social circles and give them appropriate names. It also gave its users the ability to hangout (online video chat) with other users in the network. Pinterest, however, falls into a completely different category of OSNs. It is a pin board-style photo sharing website that allows its users to

create theme-based image collections. Users can browse through pins, comment and re-pin the images they like to their own pin boards. Among all OSNs Facebook is the most dominant and shares the most number of monthly active users i.e.1.65 billion users as of March 31, 2016 (Facebook, 2016). The popularity of OSNs is growing every day by leaps and bounds and where they have brought advantages to the people and communities they have also put the users' privacy and security at risk as well (Tatjana et al. 2010).

1.2 The Problem

The success of OSNs has changed the way people interact and communicate with each other today. The enormous reach of OSNs combined with the speed at which the information is disseminated around the globe is immense.

The traditional online social networking service providers are centralised in nature and their commercial nature has raised a number of issues around users' privacy and security among its users and in the research community. The market leaders like Facebook are striving to obtain more users' data by acquiring other social networking applications like Instagram (photo sharing application) and WhatsApp i.e. instant messaging application to target users who were either not on Facebook or to obtain more information about its existing users. This means the users' of Facebook, Instagram and WhatsApp are administered by a single organisation, typically a commercial provider who has complete control over users' data that they can use for various purposes (Beye et al. 2012). The service providers may exploit users' data in various different ways including selling users' data to third parties for data-mining and targeted advertisements (Shahriar et al, 2013).

The Facebook program Beacon is one of many examples that exploited users' privacy. Beacon was a part of Facebook's advertisement system that posts updates on users' profiles when they interact with its partner websites like Amazon. Just after two years of its launch, the program went offline due to privacy issues (Zamzami et al. 2010)(Boyd & Ellison, 2010)(Krishnamurthy & Wills, 2009). Moreover, traditional web-server based architectures of OSNs are viewed as information silos lacking interoperability across other OSNs (Yeung et al. 2009). Furthermore, users are often at the mercy of service providers' service terms and conditions that often compromises on users' privacy and property rights (Bielenberg et al. 2012)**.**

All the aforementioned issues are well known to the service providers and its users. Users either don't understand the risks of using these services or they don't want to leave the network because of their family and friends who they can connect with easily via OSNs. On the other hand, service providers can permanently fix or improve the protection of users' privacy by encrypting users' data and allowing them to decide with whom they want to share their data but service providers not implementing this is understandable. As doing that would deprive service providers to mine, analyze and sell users' data to third parties that is their major source of revenue (Shahriar et al. 2013).

Over the past few years, to overcome the drawbacks of the existing OSNs many academic researchers and practitioners around the world have been working on creating decentralized OSNs that may offer better data privacy and security to its users. Recent research conducted in this area has produced some interesting applications that differ greatly in their design and approach but all aim to solve the same problem i.e. preserving users' privacy while allowing the users to participate in OSNs. Few of the applications utilized permanently available resources, while others embraced mutual cooperation among the users to share resources, bandwidth, and storage and some adopted a Hybrid approach (Liu et al. 2011) (Sharma and Datta, 2012). Where each of these approaches tried to overcome the drawbacks of the existing systems, introduced some other shortcomings e.g. limited data availability, discrimination of users with few social connections, low adaptivity to user churn rates, and technical and economic feasibility to deploy these applications on large scale (Beye, 2012). In the next few chapters, we investigate how these approaches suffer from the multitude of shortcomings, which prevented them from being successful as the next social network.

In essence, before the paradigm of decentralized OSNs become a serious alternative to centralized approaches we must experimentally study and address some of the key challenges i.e. how to achieve high data availability with the minimum number of replicas possible, and how the users' availability, replication degree, and update propagation delay changes by altering the number of mirror nodes and their online patterns. We assume a certain threshold value to denote 'high' data availability. The results can then be applied (as shown in Section 4.4) for specific values of availability such as 90%, 99%, 99.9% and so on. The term "minimum number of replicas" denotes the smallest number of mirror nodes that achieve the same availability as all mirror nodes combined. This can be explained as if 300 mirror nodes achieve 23 hours of data availability, then we may find we can achieve the same availability with just 20, 30 or any number of nodes '*n'* where *'n'* is less than 300. In this case, we can then say the minimum number of replicas required to achieve the same availability as all mirror nodes combined i.e. 300 is 20, 30 or any number of nodes '*n'*.

Achieving high data availability is one of many challenges that we face in building decentralized OSNs (Shahriar et al. 2013), but it is also important to note that achieving high data availability itself depends on many other factors including number of replicas/mirror nodes, number of sessions and session duration. For the rest of the thesis, we interchangeably use replicas and mirror nodes, where appropriate, that refers to node's replica hosting locations unless otherwise specified. In the following sections we summarize the research aim and objectives, hypothesis and associated research questions.

1.3 Aim

The overall aim of this research is to develop a model and investigate into how the node's availability, replication degree, and update propagation delay (dependent variables) changes

on altering its number of mirror nodes, their online patterns, number of sessions and session duration (independent variables) by studying the effects of changing each of the independent variables on each of the dependent variables.

1.4 Research Objectives

1. To investigate into how the existing decentralised OSNs have addressed availability issues in their design.
2. To identify the relationship between the node's availability and the number of mirror nodes, number of sessions and session duration.
3. To identify the relationship between the node's update propagation delay and its number of mirror nodes, number of sessions and session duration.
4. To identify what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined from given number of mirror nodes and their online patterns.
5. To introduce the concept of availability on demand and help new joining nodes to find good mirrors.

1.5 Hypothesis

For given number of mirror nodes and their online patterns it is algorithmically possible to determine the minimum number of replicas required to keep the node's profile highly available, where highly availability may mean 90%, 99%, 99.9% etc. diurnal availability.

1.6 Research Questions

Based on the above hypothesis the research questions that we aim to answer are:

1. What are the challenges in existing decentralized OSNs in achieving high data availability?
2. How the node's availability changes as we alter its number of mirror nodes, their online patterns, number of sessions and session duration?
3. How the node's update propagation delay changes as we alter its number of mirror nodes, number of sessions and session duration?
4. For given number of mirror nodes and their online patterns what is minimum number of replicas required to achieve the same availability as all mirror nodes combined?
5. How the new joining nodes can find good mirrors and achieve desired availability targets, despite of having no or few social connections in the network?

1.7 Structure of the Thesis

The remainder of the Thesis is organised as follows: Chapter 2 discusses the literature review, outlines the research gap analysis and concludes with the functional and non-functional requirements of the system. Research Methodology and System Design is discussed in Chapter 3. In Chapter 4, we conduct extensive evaluation of our model and the algorithms via

simulations and present the results. Finally, in Chapter 5 we conclude and outline the directions for future work.

CHAPTER 2: LITERATURE REVIEW

Boyd and Ellison (2007) define OSNs as:

"*web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system"*.

OSNs allow users to create public or semi-public profiles and encourage the users to add personal information about themselves e.g. date of birth, education, workplace, telephone number, home address etc. Leakage of such personally identifiable information sometimes leads to undesirable consequences (Falahi et al. 2010). Traditional OSNs are built on centralised architectures where service providers have unprecedented privileges of access to users' private data. This has made privacy advocates and the users of OSNs worrisome alike.

2.1 Centralised Online Social Networks

The traditional centralised - client/server architectures have become a standard model for developing network applications and all the major OSNs like Facebook, Google+ and Twitter are built on centralised architectures. In centralised OSNs, users must trust service providers to enforce access control policies, not to leak or misuse users' data and to take appropriate

measures to protect the users' data from external attacks. The massive information aggregation of users' sensitive personal information on these central service providers is an inherent threat to users' privacy. In the past, leakage of users' personal data from the aforementioned social networks happened regularly both intentionally, in the form of selling users data to third parties and unintentionally, via outside attacks on service providers (Koll, Li & Fu, 2014).

Lam et al. (2008) believe that disclosing personal information in OSNs is like a double-edged sword. To fully benefit from the services of OSNs sometimes it's important for the users to provide personal information that is often misused by service providers. Moreover, the plethora of information related to users' personal lives may also invite external attacks as well including stalking, reputation slander, and phishing attacks (Lam, Chen and Chen, 2008). Greschbach et al. (2012) highlight the risks of massive central data aggregation of users' personal information in conjunction with an advertisement based business model of major social networking service providers, where the users are not customers, but primarily products. As service providers use its users' data for data mining and targeted advertisement, which in turn generates revenue for them that is their major source of income. Moreover, users of OSNs are often at the mercy of service providers with their constantly changing service terms and conditions, which often compromise on users' privacy and property rights (Koll, Li & Fu, 2014).

Moreover, Facebook and Google also own other social networking applications e.g. Facebook owns Instagram - a photo sharing application and WhatsApp – an instant messenger and Google acquired YouTube in 2006. This way service providers can obtain deep insights into users' personal and private information and sell it to third parties that in return generate revenues for them (Dwyer, 2011). Facebook annual report 2012 states that 85% of its annual income is generated from personalised advertisements (Olteanu & Pierre, 2012).

Another important perspective to this is that users' data is not only at the mercy of service providers but also the data at a single entity increases the risks of external attacks as well. If service provider's security measures are compromised by the external attacks then the attackers would be able to gain access to all the users' data as well. This kind of attack was seen 2012 when passwords of around 8 million users of LinkedIn were leaked (Koll, Li & Fu, 2014). Moreover, in the past numbers of cases have been reported when service providers were caught of selling users' data illegally to third parties breaching users' privacy (Nilizadeh et al. 2012).

The privacy and security risks associated with the centralised architectures of OSNs are often misunderstood, underestimated or completely ignored. Krishnamurthy and Wills (2009) believe that popularity of OSNs have accelerated the appearance of vast amount of users' personal information online. Their research has shown that it is possible for third parties to link personally identifiable information leaked via OSNs with information present on other non social networking websites. They also found that despite of privacy controls to limit access 55% to 90% of the users retain default privacy settings that is sometimes open to public access and to

even non-users of social network as well. Moreover, Malin (2005) suggests that it is possible to infer correct relation of seemingly anonymous data to explicitly identifying information. His results show that 87% of Americans can be uniquely identified from their date of birth, zip code and gender.

In the past, OSNs have also become victims of phishing attacks and a distribution channel for spreading malware. Through OSNs attackers can easily spread malware to millions of users in few seconds who have a certain level of trust for each other, and gathers users' personal information (Tim & Perez, 2010). Despite of the number of privacy and security issues in traditional OSNs, nothing or very little has been offered from the service providers to improve the situation. Implementing encryption and allowing users' to define customized access control policies would solve most of the problems but service providers not implementing that is however understandable. As, doing that would prevent them from mining, analyzing and selling users' data to third parties that is their main source of income (Koll, Li & Fu, 2014).

In summary, there exists an obvious need for an increased privacy in OSNs. Therefore, to address the privacy and security issues of the traditional OSNs number of attempts have been made in the literature to create the next generation of OSNs built on decentralised architectures where users can have complete control over their data (Koll, Li & Fu, 2014).

2.2 Decentralised Online Social Networks

As mentioned earlier, most of the Internet based applications are built on client/server architectures because of its number of advantages. The primary advantage of client/server approach over decentralised model is that the entire network is managed by dedicated servers, offering reliability, high data availability, accessibility, and ability to perform complex search queries but on the downside, service providers have full control over users' data that raises natural questions of trust and users' privacy.

In contrast, decentralised OSNs do not have any centralised controlling entity instead it is maintained by the participating individuals in various different ways depending on the system design that can vary from one to another. A generic system design for any application build on decentralised architecture is explained below.

In decentralised architectures data is stored on/by the participating individuals removing the dependency of any centralised entity or database. In decentralised OSNs users not only consume system resources but also contribute towards the storage and communication requirements of the system as well. To offer fine-grained access control policies the architecture offers sophisticated encryption algorithms to store encrypted data on participants' machines so that it is accessible only to/by the eligible users. This approach, however, solves the privacy issues of centralised OSNs in a quite forthright way but introduces some other

challenges e.g. how to achieve high data availability, what is the minimum number of replicas one must have to achieve the desired availability targets etc.

Looking at the bigger picture of the problem i.e. “building a social network”, the following arguments are typically brought forward to favour decentralisation over tradition centralised OSNs. Firstly, the presence of service providers is not necessary as the content generated and consumed in any OSN is by the users and not for the service providers – this eliminates the need of service providers. Moreover, data in decentralised OSNs is encrypted and can only be decrypted by the recipients with whom the author has intended to share. Whereas, in centralised OSNs service providers are not only trusted to protect users’ data but also to enforce access rights that the author of the data has defined (Buchegger & Datta, 2009). Secondly, not having a service provider would eliminate the risk of single point of failure and large-scale privacy breaches as well. Thirdly, it would prevent information silos that is a problem with the existing social networking applications where users have to create many profiles on different OSNs i.e. Facebook, Google+, Twitter and many more (Yeung et al. 2009).

2.3 Challenges and Opportunities

There are a number of challenges involved in decentralising the existing architectures while providing the same or even more sophisticated functionalities to the users of OSNs (Datta et al. 2010). It requires finding ways of distributing and storing data in the network (Hales, 2004), achieving high data availability, minimizing update propagation delay, implementing search (ability to find other peers in the network), robustness against churn, and mechanisms ensuring users’ privacy, security, confidentiality and data integrity is not compromised (Buchegger & Datta, 2009). In the next section, we outline some of the key requirements that must be addressed to build decentralised OSNs.

2.3.1 Storage and Data Availability

There are number of questions associated with the storage of users’ generated/consumed data in the network that must be addressed to successfully achieve high levels of data availability that is close to traditional OSNs. To achieve that there are few different options that can be considered as alternatives for the missing infrastructure of storage and communication requirements.

To achieve high data availability decentralised architectures can rely on permanently available resources like Cloud storage provided by Amazon and other service providers but this would result in a dependency to third parties for application to perform its operations (Baden et al. 2009). Moreover, it would also incur costs to users that might not be attractive to use paid service when the existing OSNs are free to use but cost users’ privacy. Furthermore, it would also violate the true essence of decentralised OSNs as well (Olteanu & Pierre, 2012). Additionally, in the recent years data privacy and unauthorised access to users’ data in the

cloud has become a major concern among the researchers and industry practitioners (Jansen, 2011)(Zhang, Yang & Zhang, 2012)(Wang, 2011). Using distributed hash table, trusted friends as proxies, super peers and user administered permanently available resources have also been proposed in the literature to serve as storage of users' data as well. However, in completely decentralised architectures, selecting trusted friends as proxies appears promising but is also challenging as well as users are dependent on their social connections to keep their profiles highly available when they themselves are not available. Moreover, the system must not discriminate between well established and new joining nodes and must offer equal opportunities to every node to make their profiles highly available. Another important question that arises of the aforementioned discussion about storage is where the data should be stored? Should it be stored on users' trusted nodes i.e. friends? Or should it be stored on random nodes to achieve high availability targets. Moreover, it is also important to determine what is the minimum number of replicas required to keep the users' profile highly available.

2.3.2 Overhead

Whilst designing the system, one may achieve high data availability by spreading the data on as many nodes as possible but might underestimate the overhead caused by it. If users' data is replicated across many different storage locations then it might achieve the first objective but would also consume network resources in maintaining those replicas as every time a user updates or uploads a new content it must be distributed to all the storage locations to keep the data synchronised across all mirror nodes. Therefore, it is important to recognize the trade-off between achieving high data availability and overhead caused by it (Stoica et al. 2001). The system must be intelligently designed such that it keeps the storage locations to minimum while achieving high data availability targets.

2.3.3 Leverage Social Relationships

Research shows applications that leverage social relations in their design have improved their performance (Viswanath et al. 2010)(Hui, Crowcroft & Yoneki, 2011). Therefore, to design a better-decentralised OSN the system must exploit social relationships between nodes to facilitate storage and mirror selection requirements. Moreover, the mutual co-operation between nodes would help every node in the network to distribute their data efficiently. One must consider that this must not discriminate users with few or weak social connections and should give equal opportunities to make their data highly available.

During the last two decades, traditional OSNs have proved themselves extremely successful offering services to its users, but lacks in providing data privacy and security. On the other hand, decentralised OSNs offer promising alternative to its users by offering data privacy and security but have number of other challenges to overcome for example offering high data availability is the major one.

In the next section, we present a comprehensive overview of the different systems that differ greatly in their design but all aim to solve the same problems i.e. how to preserve users' privacy while offering full set of services to the users that they experience in traditional OSNs. In completely decentralised architectures, users of OSN form a peer-to-peer network where everyone contributes towards the storage and communication requirements of the system. Whereas, partially decentralised architectures depend on distributed servers thus keeping the client-server paradigm, but giving freedom to the users to host their profiles on the servers they trust or to administer and host their profiles on their own on permanently available resources.

2.4 Related Work

2.4.1 PeerSon

To address the privacy issues in traditional OSNs Buchegger and Datta (2009) highlighted some of the key challenges and opportunities for peer-to-peer networks in the area of social networks and proposed PeerSon a system built on peer-to-peer architecture. The main building blocks of PeerSon are encryption and decentralisation. Encryption provides privacy and data integrity to users' data while decentralisation gives freedom to its users from the service terms and conditions and allows them to define fine grained access control polices. PeerSon was built on two-tier architecture, one tier serves as a look-up service and the second tier consists of peers and users' data. To facilitate asynchronous communication PeerSon uses OpenDHT that is a centrally managed deployment of the BambooDHT on PlanetLab. OpenDHT was used as a look up service that provides a mechanism to keep nodes connected as necessary and to store the messages of up to 800 characters for a maximum of 7 days. The authors of PeerSon recognized the issue of high data availability in their design but didn't provide any mechanism to address the issue.

2.4.2 Safebook

Cutillo et al. (2009) proposed another social network called Safebook built on peer-to-peer architecture to provide a decentralised general-purpose social networking experience assuring users' privacy, security, data integrity and availability. Safebook mainly focused on preserving users' privacy while offering full set of services that users can experience in traditional OSNs. It consists of three main components i.e. Trusted Identity Service (TIS), matryoshkas and P2P location substrate. TIS was used for authentication purposes. A set of concentric ring structures called matryoshkas, serves to store data of the inner most node to nodes in its outer shells. Data in Safebook was stored on users' trusted nodes that also served as proxies to core node in its absence. Any requests to/from the core node were routed through matryoshaksts to achieve communication anonymization and to obfuscate information flow (Cutillo, Molva & Onen, 2011). Moreover, to gain access to core node's data all nodes on the same path towards to inner-most shell need to be online simultaneously which is very unlikely as user sessions in OSNs are often short and volatile (Benevenuto et al. 2009). As Safebook's approach in achieving high data availability directly depends on one's number of friends which means it

would be difficult for users to achieve desired availability targets who maintain only few social connections.

2.4.3 SuperNova

Sharma and Datta (2012) suggest that primary motivation in creating decentralised OSNs is to achieve privacy and autonomy from big brotherly service providers. They also recognise that one of biggest problems in building decentralised OSNs is to achieve high data availability when the data owner is not available. They proposed SuperNova – a super-peer based decentralised OSN where Super-peers help bootstrap new joining nodes and serve as central directory to facilitate search. In SuperNova when a node joins a network, it relies on its super-peers to find good mirrors and increase its data availability. Later a node may find other nodes (friends or strangers) in the network who act as mirrors/storekeepers.

2.4.4 DECENT

Jahid et al. (2012) proposed DECENT – a fully decentralised peer-to-peer OSN with a special focus on privacy and security. It utilises distributed hash table to store data and implements sophisticated cryptographic techniques to ensure confidentiality and data integrity. To ensure availability users' data was replicated on multiple locations across the network. The authors of DECENT recognise that to address availability, number of replicas required in the network needs to be fine-tuned based on churn patterns of the network that was left to do in their future work.

2.4.5 Cachet

Nilizadeh et al. (2012) proposed another social network called Cachet built on peer-to-peer architecture. Like DECENT, replication in Cachet was system driven where users' data was replicated on random nodes to ensure high data availability. For efficient data retrieval and dissemination Cachet made use of social connections between the users which served as caches to store recent updates in the network. The idea behind social caching was that users who satisfy attribute based encryption and decryption policies were leveraged to provide and retrieve cached decrypted objects of other users. This helped the system to reduce cryptographic and communication overhead in the network.

2.4.6 Vis-à-Vis

Shakimov et al. (2011) proposed Vis-à-Vis, a decentralised framework for OSNs running on Amazon EC2 computing utility. It mainly focuses on preserving privacy of users' location information. In contrast to other approaches, to address availability, Vis-à-Vis took a philosophical departure from replication by trusting cloud service providers i.e. Amazon EC2 with access to unencrypted version of users' data that we believe is slightly ambitious. As in the past, a number of cases have been reported when cloud service providers were accused of privacy breaches (Jansen, 2011) (Zhang, Yang, Zhang. 2012) (Wang, 2011).

2.4.7 FOAF

Yeung et al. (2009) proposed FOAF (Friend Of A Friend) application built on dedicated trusted servers. It allowed its users to publish their profiles on the servers they trust or to administer and host their profiles on their own servers. This gave the users complete control and ownership of their data but had a number of other issues e.g. it did not provide any mechanism to verify the reverse links. Because of its distributed nature, it became very easy for anyone to claim that a user is a friend of another user by just specifying his or her WebID. The main weakness of the proposed model was that it failed to address how users will be able to communicate or access each other profiles when the users are offline or servers are down, as FOAF didn't offer any replication mechanisms.

2.4.8 Diaspora

Diaspora was built on a semi-decentralised architecture consisting of network of independent, federated Diaspora servers administered by individual users (Bielenberg et al. 2012). The architecture of Diaspora gave freedom to its users either to host their profiles on the Diaspora servers or on their own server(s) to keep complete control of their data (Narendul, Papaioannou and Aberer, 2012). One of the problems identified in the architecture of Diaspora was that when a user communicates with another user who hosts its profile on the Diaspora servers then the user's communication was stored on the Diaspora server(s) in an unencrypted form thus leaving the privacy in the hands of server administrators. Another problem identified by Bielenberg et al. (2012) was that for the successful transaction of communication between the users both the users have to be online at the same time or otherwise messages get lost in the network and the sender does not receive any acknowledgement either.

In Table 1 we present a summary of above systems, indicating how they have addressed availability in their design.

System Design	System	Availability		Replica Selection	Replica Placement
		Approach	Issues		
Peer-to-Peer	Safebook	Replication	i) Discriminating nodes with few social connections. ii) Simultaneous availability requirements of all mirror nodes along the same path for communication.	User Driven	Friends
	DECENT	Replication	Nodes selected randomly for replication	System Driven	Random
	Cachet	Replication & Caching	Nodes selected randomly for replication	System Driven	Random

Hybrid	SuperNova	Replication	Dependency on Super-Peers.	System Driven	Super-Peers Friends, Strangers
	PeerSon	-	Availability not addressed	-	-
Permanently Available Resources	FOAF	User Administered Servers	Dependency on permanently available resources. No replication mechanisms in place.	-	-
	Diaspora	User Administered Servers – Diaspora Administered Servers	Unencrypted data stored on untrusted servers	-	-
Other	Vis-à-Vis	Cloud	Dependency on third parties and privacy issues associated with it.	-	-

Table 1: Availability in decentralised online social networks

From the Table 1, we can see where the different decentralised OSNs tried to overcome the privacy and security issues of the existing OSNs have introduced some other challenges. i.e. i) limited success in achieving high data availability ii) discriminating nodes with few social connections iii) dependency on powerful nodes for different operations e.g. data storage and search iv) lack of encryption v) dependency on permanently available resources. Most of them suffer from the multitude of shortcomings and any one of the shortcomings can prevent the success of decentralised OSNs that can compete with the traditional OSNs. If a new social network built on decentralised architecture does not offer high data availability then it is not very likely that it will attract masses. Moreover, if a system offers high data availability but with a usage fee then it might not attract users either to use a paid service when the existing OSNs built on advertisement based business models are free to use costing nothing but users' privacy.

One can argue by exploiting the characteristics of Peer-to-Peer networks and file sharing applications, that have already proved themselves scalable and extremely successful, we can build a social network built on decentralised architecture. We must recognize that there are few major differences between the two. First, the online patterns of users sharing/seeding files over the network and users of OSNs differ greatly. In file sharing applications, users are online for longer periods spanning from hours to days whereas users of OSNs often have short and abrupt online patterns (Benevenuto et al. 2009). Moreover, in traditional Peer-to-Peer applications high bandwidth offering users would be able to download the files faster than the users offering lower bandwidth. In contrast, the same principle in context of OSNs is very different. Users in OSNs can achieve high data availability even if they are not online for most of the time given they have chosen good mirrors that can keep their data highly available

(Hales, 2004). Among all the shortcomings, one critical drawback of decentralised OSNs is low data availability that must be addressed when designing a new decentralised OSN that is competitive in nature with the traditional OSNs.

2.5 Research Gap Analysis

In centralised OSNs, service providers ensure high system availability by providing dedicated resources that can keep users' profiles highly available. Whereas in decentralised OSNs it is the responsibility of the users to keep their profiles highly available either by storing data locally on the devices, setting up their own servers, relying on third party cloud storage providers, or replicating data on their friends' machines who can act as a proxies when the users themselves are not available. Therefore, achieving high data availability in decentralised OSNs is a challenging issue and must be addressed. From the literature, we realised the researchers and practitioners have adopted the following two approaches to address the issue.

1. **Replication:** Users replicate their profiles on a set of other users who act as proxies when the users themselves are not available or are offline.
2. **Permanently available resources:** In this case users choose to host their profiles on their own either by setting up their own servers or by relying on cloud storage providers. This approach however offers high data availability with low overhead cost but requires all users to be technically capable and able to configure and setup their own servers that we believe is slightly ambitious and impractical. Moreover, choosing cloud storage has its own privacy and security issues associated with them.

Systems that adopted replication as their preferred choice failed to address the issues around availability (e.g. Safebook), update propagation delay (e.g. Diaspora), replication degree, and system overhead (e.g. DECENT), and how the new joining nodes can achieve high data availability when they don't have enough friends or social connections in the network to choose as mirrors (e.g. Safebook).

From the different systems, that we have discussed in Section 2.4 PeerSon acknowledges the issue of high content availability in decentralised OSNs but doesn't provide any mechanisms to address the issue. Safebook adopted replication to address high data availability and created replication groups in the form of matryoshaks concentric ring structures (i.e. shells) around the core node based on their friendship relationships. It encourages users to cooperate and create a temporarily available storage space for each other. This saves them from being dependent on any permanently available resources and their drawbacks.

The major challenge Safebook had to overcome was to offer reliably high data availability to its users, which it failed to address for two major reasons. i.e. i) To gain access to core node's data all nodes on the same path towards to inner-most shell need to be online simultaneously which is very unlikely as user sessions in OSNs are often short and volatile (Benevenuto et al.

2009) ii) Safebook's approach in achieving high data availability directly depends on one's number of friends which means it would be difficult for users to achieve desired availability targets who maintain only few social connections.

Diaspora adopted the second approach of setting up permanently available resources to achieve high data availability that didn't require any sophisticated data synchronisation and replication mechanisms but instead gave freedom to its users to either host profiles on their own or select one of the diaspora federated servers to host their profiles. It allowed its users to spread their profiles on multiple different administrative domains but eventually left the privacy and security of their data in the hands of server administrators.

FOAF followed the same model as Diaspora and allowed its users to publish their profiles on servers they trust or host their profiles on their own.

Vis-à-Vis addressed availability in its design by running virtual individual servers in a paid cloud computing utility Amazon EC2. This however offers high data availability but leaves privacy in the hand of service providers. In Vis-à-Vis users' data was stored in an unencrypted form and service provider were trusted not to misuse or share users' personal information with third parties that we believe was slightly ambitious leaving privacy in the hands of service providers. In the past, a number of cases have been reported when cloud service providers were accused of privacy breaches (Jansen, 2011) (Zhang, Yang, Zhang. 2012) (Wang, 2011).

Moreover, availability in DECENT and Cachet was addressed via replication that was mainly system driven, which means the system was responsible to select appropriate replicas that may offer high availability. Replica selection in their design was completely random. The authors of DECENT however believed number of replicas in their design must be fine-tuned and was left as future work to do.

To address the availability issues in decentralised OSNs Shahriar et al. (2013) proposed a different approach, in which data was stored and replicated with in small user groups, which ensures that for any given time at least Beta members of the replication group are online. Their results show 2 availability grouping policy delivers high data availability. One of the problems with their simulation setup was that they performed their experiments on data set obtained from the users of file sharing applications that doesn't truly depict the online patterns of users of OSNs.

Another research conducted by Fu and his colleagues (2014) in finding relationship between data availability and storage capacity of the devices. They found maintaining high data availability in decentralised OSNs is one of the biggest challenges and it is often assumed that friends of a user can always contribute towards the storage requirements to store their friends' replicas. But this is not always true because most users often use smart phones to access online social networks and their storage capacity may jeopardize the data availability.

In 2014, Fu et al. (2014) proposed Cadros – a cloud assisted data replication technique for decentralized OSNs. Fu and his colleague believe that full replication can improve data availability but pure decentralized OSNs may not be able to deliver sustainable data availability targets. Therefore, they envision a cloud assisted data replication scheme to improve availability in decentralized OSNs. Li and Dabek (2006), Shakimov et al. (2009) Gracia-Tinedo et al. (2012) and Sun et al. (2009) also proposed hybrid replication schemes to achieve high data availability targets.

Koll et al. (2014) suggests exchanging the recommendations between friends about their mirror nodes would greatly help in recognizing good and bad mirrors. Olteanu and Pierre (2012) suggests replica placement should be user driven instead of system driven and preference should be given to users' trusted friends to host the replicas and when all friends are offline data then must be stored on nodes, which are not in users' friend circle.

Tegeler et al. (2011) proposed a similar approach called Gemstone. Gemstone stores users' data in the so-called data holding agents (DHAs). If a DHA itself is offline then the data must be passed on to its DHAs and so on.

Over the past two decades an extensive amount of research has gone into this domain. Achieving high data availability in decentralized OSNs is a complicated task for a number reasons. Firstly, user sessions in OSNs are often short and volatile. Secondly, if only friends are chosen as possible replica hosting locations then users with few social connections or friends may suffer greatly as they may not have enough replicas in the network to keep their profiles highly available. Existing work in improving data availability in decentralized OSNs only focuses on how to store users' replicas across the network. But none of the approaches attempt to answer what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined, how the node's availability, replication degree, and update propagation delay changes by changing its number of mirror nodes and their online patterns. Therefore, in order to address the aforementioned questions the following section outlines the functional and non-functional requirements of the system.

2.6 Requirements

2.6.1 Functional Requirements

According to Sommerville (2015) functional requirements are:

"Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do."

Following the guidelines of RFC-2119 (Bradner, 1997) the functional requirements of our system as follows:

1. The system must output how the node's availability changes as we alter its number of mirror nodes, their online patterns, number of sessions and session duration. (Discussed in detail in Section 3.3.1 and 4.4).
2. The system must output how the node's update propagation delay changes as we alter its number of mirror nodes, their online patterns, number of sessions and session duration. (Discussed in detail in Section 3.3.2 and 4.5)
3. The system must output what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined and the ids' of chosen mirror nodes in order. (Discussed in detail in Section 3.3.3 and 4.6)
4. The system must be able to analyse mirror nodes' past online patterns and find their average daily and hourly availability and the hour during which they are most available. (Discussed in detail in Section 3.3.4 and 4.7)

2.6.2 Non-Functional Requirements

According to Sommerville (2015) non-functional requirements are:

"Constraints on the services or the functions offered by the system. They include timing constraints, constraints on the development process and standards. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual features or services."

Following the guidelines of RFC-2119 (Bradner, 1997) the non-functional requirements of our system as follows:

1. The system must be driven by a given model, which is used to simulate the online patterns of mirror nodes as in traditional OSNs.
2. The system must output node's diurnal (24 hour) availability in minutes from a given model.
3. The system must output node's replication degree in numbers from a given model.
4. The system must output the results in form of JavaScript arrays format that are then used to visualize data on the web.
5. The user must be able to visualize the results in the form of charts and graphs on the web.

There are some interdependencies between the functional and non-functional requirements of the system stated above. For example, the model, which is a by-product of this research, is a part of the non-functional requirements but is a vital component of the whole system and plays a very important role in realizing the functional requirements of the system e.g. calculating node's availability, replication degree, update propagation delay, and availability on demand.

We have also created a website to accompany with this Thesis mainly because the number of simulations performed and the results obtained from them couldn't be contained and presented in the thesis alone. Therefore, we may refer to links in the website for more detailed simulation results. However, the results presented in this thesis are self-sufficient and enough to draw conclusions.

Website Link: www.adilhassan.com/mphil

CHAPTER 3: RESEARCH METHODOLOGY AND SYSTEM DESIGN

In this chapter, we present an overview of different research methodologies and investigate if there exist set of methodologies that are widely accepted in research community for research projects embarking on creating new artefacts, models and algorithms. Moreover, we then present a case why our chosen set of methodologies suit best for our needs and how it helps in conducting, validating and evaluating our research. Furthermore, we then present the system design driven from the research objectives stated in Chapter 1.

3.1 Research

Gratton and Jones (2003) define research as:

"A systematic process of discovery and advancement of human knowledge."

In just a single statement, Gratton and Jones have identified several key descriptors of research. First, it is a systematic process, which means it involves several steps that must be executed in specific order to obtain reliable and reproducible results. Second, research involves discovery, which implies that it is a process in which answering one question inevitably reveals new questions and one's quest of answering these questions creates new knowledge. Lastly, research involves advancement of human knowledge, which suggests research by its very nature is an iterative and a cyclic process that contributes to the body of knowledge both by expanding the knowledge in a given domain and other related disciplines of the research under study.

3.2 Research Methodologies

The research project under study aims to understand the relationship between availability, replication degree, and update propagation delay with one's number of mirror nodes, their online patterns, number of sessions and session duration via simulations. This suggests that our research broadly falls into Information Systems research category which involves creating artifacts, models, algorithms, and understanding relationships between different variables via simulations and experiments. In the next section, we review different types of research methodologies and identify a set of methodologies that fit best for our research.

3.2.1 Quantitative Research Methodology

Creswell (2013) defines quantitative research as:

"A means of testing objective theories by examining the relationship among variables. These variables in turn, can be measured, typically on instruments, so that numbered data can be analysed using statistical procedures."

Quantitative approaches mainly focus on analysing data and finding relationships between variables. There are numerous research designs that fall under quantitative research and the ones' that we will be discussing in this section are as shown in Figure 1.

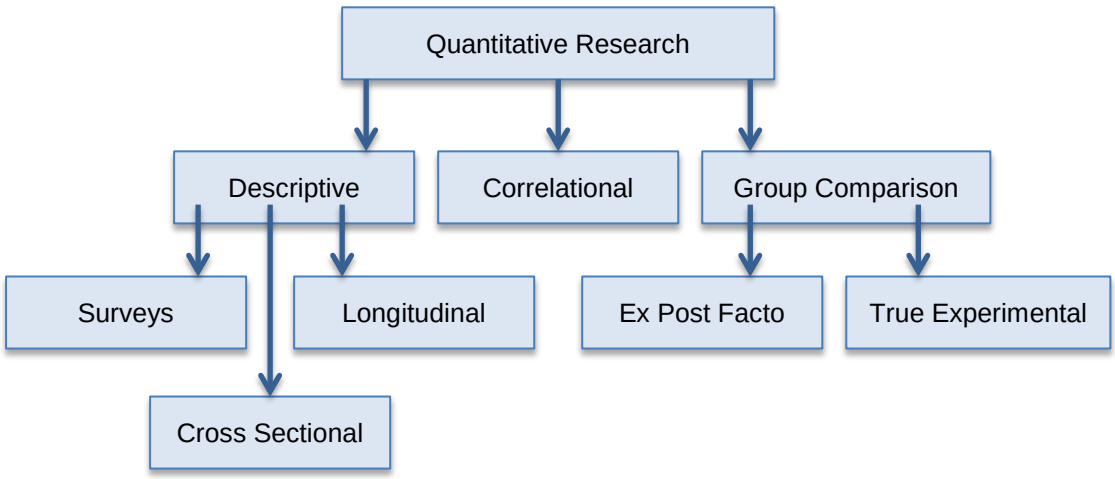


Figure 1: Quantitative research methods

3.2.1.1 Descriptive

Descriptive designs are set out to establish associations between variables and describe and interpret "*what is"* in a specific situation (Cohen et al. 2000) (Hopkins, 2000). They are often used to identify problems for further and more sophisticated research. According to Best (1970) descriptive designs are concerned with:

"*Conditions or relationships that exist; practices that prevail; beliefs, points of views, or attitudes that are held; processes that are going on; effects that are being felt; or trends that are developing. At times, descriptive research is concerned with how, what is, or what exists is related to some preceding event that has influenced or affected a present condition or event."*

In descriptive studies, researchers don't attempt to manipulate or exert control over events under study but instead they only observe and measure events as they occur. Moreover, descriptive studies neither examine causal relationships nor they have any dependent or independent variables to manipulate. Descriptive designs are mainly divided into three main categories i.e. Surveys, Longitudinal and Cross-Sectional studies (Cohen et al. 2000).

i) Surveys

Surveys are typically used to scan a wide field of issues, populations and programs to measure or describe any generalised features. Morrison (1994) suggests that surveys represent wide target population, generate numerical data, provide descriptive and explanatory information, and make generalizations to support or refute hypothesis.

ii) Longitudinal

Longitudinal research involves variety of studies conducted over a period of time. In longitudinal studies, researchers gather data over an extended period of time that can span from several weeks or months to even years. During the research, data gathering is done from the same respondents and successive measures are taken at different points in time (Cohen et al. 2000) (Gall et al. 2006).

iii) Cross-Sectional

Unlike longitudinal studies that span over extended period of time, cross-sectional studies produce snapshot of population at a particular point in time. Moreover, cross-sectional studies are usually less effective in identifying and establishing causal relationship between different variables (Cohen et al. 2000).

3.2.1.2 Correlational design

A correlational design is a statistical procedure used to determine if there exists a relationship between two or more variables or if one variable can predict the behaviour of another variable in study. Moreover, it also examines to what extent the variables are related to each other as well. Furthermore, correlation studies also tend to answer what's the magnitude and the direction of relationship between the variables under study. It is often chosen to conduct exploratory or beginning research to determine if more rigorous research is needed. Therefore, if correlational design is chosen as a preferred choice then it is very important to ensure that

the significance and role of each variable under study is fully understood and presented (Cohen et al. 2000)(Morrison, 1994).

3.2.1.3 Group Comparison

i) Ex Post Facto Design

Ex Post Facto design aims to determine possible cause and effect relationship between two or more variables. In ex post facto studies, researchers retrospectively examine the relationship between independent and dependent variables and subsequently establish causal link between them (Cohen et al. 2000). If there exists a strong relationship between the two then the researchers can make the following three interpretations for dependent and independent variables (Kerlinger, 1970) (Borkowsky, 1970) i.e.

- Variable x has caused y
- Variable y has caused x
- Unidentified variable z has caused x and y

One of the problems with Ex Post Facto Designs is that researchers are unable to manipulate independent variables and therefore are unable to establish which one of the above three interpretations are correct. Moreover, researchers are not always confident whether causation factor was included or even identified in the study (Kerlinger, 1970) (Spector, 1993)). Despite of aforementioned limitations and weaknesses, Ex Post Facto Designs proved to be valuable exploratory tools for researcher to identify *'what goes with what and under what studies'.* Moreover, they are very useful when possible cause and effect relationships are being explored and true experimental approaches are not possible to conduct the study.

ii) True-Experimental Design

True Experimental Design is the most rigorous design to examine cause and effect relationship between dependent and independent variables. True Experimental Designs are mainly distinguished from other research designs by following three factors (Cohen, 2000) (Campbell & Stanley, 1996).

a) Manipulation: Researchers can manipulate independent variables and study its effects on dependent variables.

b) Control: Researchers have high degree of control on the conditions and variables under study. They also have the ability to give special treatments to one or more variables whilst keeping other variables constant and study its effects on the dependent variable.

c) Randomization: With the high degree of control researcher can randomly assign one or more experimental groups to one or more treatment conditions and compare their results with groups that didn't receive any special treatment. Because of the high degree of control and

random assignment to conditions True Experiments provide unambiguous and reproducible results regarding the effects of independent variables on dependent variables that strengthens the internal validity.

3.2.2 Qualitative Research Methodology

Maanen (1979) defines qualitative research as:

"An umbrella term covering an array of interpretive techniques which seek to describe, decode, translate and otherwise come to terms with the meaning, not the frequency, of certain more or less naturally occurring phenomena in the social world."

According to Punch (1998) qualitative research is a research paradigm that focuses on collecting subjective data and the data is mostly in the form of written and spoken words and are especially useful for exploring the full nature of little understood phenomenon. In addition, in qualitative research *words* are used as data and are analysed in all sorts of ways whereas in quantitative research *numbers* are used as data and are analysed using statistical techniques (Braun & Clarke, 2013). There are number of techniques to carry out qualitative research and many of them hold a number of characteristics in common but there are some major differences as well. The three major research methods that we will be discussing in this section are:

1. Action Research
2. Case Study
3. Grounded Theory

3.2.2.1 Action Research

Blum (1955) explains action research as two stage process i.e. Diagnostic stage and Therapeutic stage. In diagnostic stage hypotheses are formulated and in therapeutic stage changes are introduced and their effects are studied. In order to achieve scientific rigor in action research Susman and Evered (1978) proposes five phase iterative/cyclic process starting with i) diagnoses, ii) action planning, iii) action taking, iv) evaluating and v) specifying learning outcomes. Diagnoses involve identification of problems followed by rigorous action planning and implementation. In phase four and five researchers retrospectively evaluate the outcomes and learn from their experiences and the cycle continues. Following Action Research principles researchers typically unfold the study through spiral cycle of planning, acting, observing, and reflecting (Kuhne & Quigley, 1997). Action research in essence attempts to link theory and practice achieving both practical and research objectives of the study.

3.2.2.2 Case Study

Yin (2014) defines case study as:

"A case study is an empirical inquiry that investigates a contemporary phenomenon (the 'case') with in its real-life context, especially when the boundaries between phenomenon and context may not be clearly evident."

What distinguishes case study research from other qualitative research methods is that it delimits the object of study i.e. the case. As Stake (2005) suggests *"a case study is less of a methodological choice than a choice of what to be studied"* and according to Smith (1978) *"what"* in case study approach is a bounded system, a single entity surrounded by a fence or a boundary. The *"case"* however can be a single person, an institution, organization or a program that is studied within a confined boundary and anything outside of the boundary is out of interest. In essence, case study research only focuses on studying single unit of analysis within a bounded system (Merriam, 2009). Therefore, to conduct an effective case study research one must understand how it differs in its design from other qualitative research methods.

3.2.2.3 Grounded Theory

Strauss and Corbin (2008) defines grounded theory as:

"A Grounded Theory is one that is inductively derived from the study of the phenomena it represents"

Grounded Theory in essence focuses on generating theoretical ideas or hypotheses from the data that is simultaneously collected and analysed during the different phases of the project. Early data analyses helps to develop theories and explanations of the phenomenon under investigation and indicates what data to collect next (Glaser & Strauss, 1967). Strauss and Corbin (2008) explain three stages of grounded theory in the form of open coding, axial coding and selective coding.

1. Open coding: involves identifying different categories from the data.
2. Axial coding: involves exploring relationships between different categories and making connections between them.
3. Selective coding: involves identifying core categories as central phenomenon and producing discursive set of theoretical propositions from them.

Once researchers go through the aforementioned series of stages they then look for the causal conditions i.e. what factors influences the central phenomenon and develop purposeful and goal oriented strategies to address the phenomenon.

3.2.3 Design Science Research Methodology

Design Science Research (DSR) involves creation of new knowledge through the design of novel or innovative artefacts and these artefacts include (but not limited to) algorithms, models, and HCI (Human Computer Interfaces). To better understand design science Owen (1997) explains a general model of DSR as *"learning through building"* in which knowledge is used in building artefacts and these artefacts are then retrospectively evaluated that contributes to the existing knowledge base and the cycle continues. March and Smith (1995) suggest DSR products attempt to create things that serve human purposes and are evaluated by their value or utility. In essence, DSR mainly consists of two main activities i.e. building and evaluating. During the last few decades number of design science research process models have been proposed. Among all most of them share a number of phases in common but differ considerably

in activities carried within these phases. The model that we will be discussing in this section is adopted from Peffers et al. (2008) that begins with Problem Awareness followed by Setting Objectives, Design and Development, Demonstration, Evaluation, and Communication.

3.2.3.1 Problem Awareness

Awareness of a problem may come from multiple sources including literature review, new developments in industry and advancements in reference discipline. The output of this phase is normally in the form of a research proposal that outlines the research problem, captures its complexity and atomizes it for better understanding of the problem.

3.2.3.2 Objective Setting

Once the research problem is fully understood, objectives are then set. The set objectives can be either qualitative or quantitative in nature but it is important to remember the set objectives must be inferred directly and rationally from the problem definition stated in phase 1 of the DSR model.

3.2.3.3 Design and Development

During design and development, researchers propose a Tentative Design that determines the artefacts desired functionality. The tentative design after going through several iterations of improvement becomes ready to go into the next phase i.e. Development. During development the proposed design is then implemented, the implementation details however may vary depending on artefact to be created. After the artefact is fully developed it enters into the next phase i.e. Demonstration.

3.2.3.4 Demonstration

During demonstration, the efficiency of the artefact is determined to solve the problem. This can be done via experimentation, simulation, case study, or by other appropriate activity.

3.2.3.5 Evaluation

During evaluation, researchers observe and measure how well the artefact performs to the problem and whether it meets the set objectives laid in phase 2 of DSR model. After the artefact is thoroughly evaluated researchers can either decide to go back to phase 3 i.e. Design and Development and make improvements to the artefact or to continue to the next phase i.e. Communication and leave further improvements for future work.

3.2.3.6 Communication

During communication, the nature of the problem, its importance, proposed design, the artefact, its utility and novelty is shared and communicated with other researchers and practicing professionals in the field via scholarly research publications and other appropriate means.

3.2.4 Conclusion

From the aforementioned set of research methodologies, the Design Science Research Methodology is best fit for our research study for the following reasons. It is widely accepted among Information System (IS) researchers and provides a strong conceptual process and mental model for the production and presentation of IS research. Moreover, DSR offers a paradigm for conducting applicable yet rigorous research. As Denzin (1978) suggests "no single method ever adequately solves the problem of rival causal factors. Because each method reveals different aspects of empirical reality, multiple methods of observations must be employed". Therefore, we believe triangulation of research methods (i.e. DSR along with Qualitative and Quantitative Research Methods) would help in conducting and checking the validity of our research from multiple perspectives. In addition, methodological triangulation may also reveal unique findings that might not be evident by just using a single research method. Therefore, borrowing principles of Correlational and True Experiments research designs from Quantitative methods helps in determining causal relationship between two or more variables and if one variable can predict the behaviour of another variable respectively. Moreover, Case Study Designs from Qualitative methods helps in studying different components/units of our research within confined boundaries. Table 2 outlines how the chosen set of research methodologies were used in this thesis.

Research Methodology	How it was used?	In Thesis
Design Science Research Methodology	The Design Science Research Methodology was used throughout the research project i.e. from awareness of the problem to communication and conclusion.	
	Problem Awareness	Section: 1.2 and 2.5
	Objective Setting	Section 1.4 and 2.6
	Design and Development	Section 3.3, 4.2, 4.4, 4.5, 4.6 and 4.7
	Demonstration and Evaluation	Chapter 4
	Communication	Production of Thesis
	Conclusion	Chapter 5
Correlational (Quantitative Research)	The Correlational research method was predominantly used to determine if there exists a relationship between different variables i.e. number of mirror nodes, number of sessions, and session duration and the system properties i.e. Availability, Update Propagation Delay and Replication Degree.	Section 4.4, 4.5 and 4.6
True Experiment (Quantitative Research)	The True Experiment research method was used to find the relationship between dependent and independent variables of our system i.e. number of mirror nodes, number of sessions, and session duration. It also enabled us to control different variables in our experiments and study their effects on system properties e.g. controlling the diurnal availability of 40 minutes for Models A, B and C and studying its effects on update propagation delay and minimum number of replicas required.	Section 4.4, 4.5 and 4.6
Case Study (Qualitative Research)	Following the principles of case study research methods enabled us to test different components/units of our system in isolation and within a confined boundary. It helped us in	Section 4.4, 4.5 and 4.6

	identifying and modelling different cases as well e.g. Models A, B and C.	

Table 2: Research methodologies

The following section discusses the iteration roadmap of Design Science Research Methodology in the context of this project.

The algorithms and the simulation tool presented in this thesis went through several cycles of iteration during different phases of this project including Design and Development, Demonstration and Evaluation.

During the design phase, the construction of different algorithms and simulation tool was mainly conceptual and involved discovery through multiple thought and paper trails of their details. Following the design, the artefact then entered into the next phase i.e. development where the artefact was developed and implemented in a programming language of choice i.e. Java. The initial deliverable of this phase was a working prototype of the artefact, which was retrospectively evaluated and refined through several cycles of iteration. Following the design and development, the artefact then entered into the next phase i.e. demonstration where the efficiency of the artefact was determined via simulations and experiments. Following demonstration, the artefact then entered into the next phase i.e. evaluation. During evaluation, minor redesigns of the artefact occurred on several occasions. By the end of this phase, the developed artefact was rigorously tested and evaluated against the set objectives and system requirements. One of the biggest advantages of following design science research methodology was that any of its phases could be spontaneously revisited from any of the other phases, which greatly helped in making continuous improvements to the algorithms and the system design.

The algorithms presented in this thesis went through several iterations of improvements. Their details can found in section 3.3.1, 3.3.2, 3.3.3, and 3.3.4.

3.3 System Design

In this section we present the core components of the system i.e. availability, update propagation delay, replication degree and availability on demand that are derived from the requirements set in Section 2.6.

For decentralized OSNs to become a viable option there are a number of feasibility checks that must be performed. In this section, we present a comprehensive overview of various system properties including availability, replication degree, update propagation delay and availability on demand along with different parameters that influence them i.e. number of mirror nodes, their online patterns, number of sessions and session duration.

Since the sole force behind the creation of decentralized OSNs is the privacy issues in traditional OSNs. Therefore, in our work we explore the case where user replicas are placed only on trusted nodes (friends) unless a user is in bootstrapping mode i.e. a user that has recently joined a network and is looking for suitable replicas. In bootstrapping mode, a node receives availability statistics from other nodes about their mirror nodes in the network. Based on those statistics the new joining nodes can then choose suitable replica(s) that meets its availability requirements; this is discussed in detail in Section 3.3.4 and 4.7.

The four core components of the system that we will be discussing in this section are availability, replication degree, update propagation delay, and availability on demand in conjunction with the core node's number of mirror nodes, their online patterns, number of sessions and session duration. The term 'core node' only refers to the 'node of interest'. The 'core node' is no different from any other node in the network. It only refers to a node whose availability, replication degree or update propagation delay we want to find.

3.3.1 Availability

Availability of a user in decentralized OSNs is defined as the fraction of a time in a day during which the user's profile is available, either directly from a user or via its mirror nodes. In decentralized OSNs availability is twofold i.e. i) availability for friends ii) 24/7 data availability. The difference between the two is, in the second approach user's profile is available for everyone to access anytime during a day. Whereas in the first approach user's profile is available to its friends either directly from a user or via its mirror nodes but this doesn't guarantee 24/7 data availability.

In this study, we assume all friends of a user are trusted and can host user replica. We assume user u*'s* replicas/mirror nodes appear online several times during a day and every time they appear online they spend some time t (in minutes). Then, the maximum availability a user u can achieve (i.e. T_u) is the union of online times of all its mirror nodes minus the overlapping times between them.

In this case, profile of a user u is accessible to another user w if there exists an overlapping time between them such that $T_w \cap T_u \neq \emptyset$. The algorithm shown in Figure 2 just takes the following four parameters i.e. number of replicas to simulate, each replica's number of sessions, session duration and the timings during which they are more likely to appear online that together form the core node's availability for any given time. For more detailed version of the algorithm see Appendix 1.

Input : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ], sessionsLowerBound [ ][ ], sessionsUpperBound [ ][ ])
Output: 1) Core node's diurnal profile availability in minutes via its mirror nodes
2) Core node's profile availability during each hour of the day

```
1  begin
2      Create a list of mirror nodes of size numberOfMirrorNodes
3      Set the number of sessions for each of the mirror node in numberOfSessions property from
         sessionsDistribution array
4      Set the session duration for each of the mirror node's sessions in sessionDuration property from
         sessionsDuration array
5      Set the lower and upper bounds for each of the mirror node's sessions in sessionLowerBound and
         sessionUpperBound properties from sessionsLowerBound and sessionsUpperBound arrays
6      Set the online patterns to each of the mirror node based on the above parameters
7      Find the mirror nodes in descending order of their availability and
8      Assign it to variable mostAvailableMirrorNodeList
9      Create a variable onlineTimeList
10     Compare the mostAvailableMirrorNode's online patterns with other mirror nodes in
         mostAvailableMirrorNodeList
11     Remove the overlapping times from the mirror nodes that overlap with the mostAvailableMirrorNode
12     Find the mirror nodes in descending order of their availability i.e. from highest to lowest available nodes
         and
13     Assign it to variable mostAvailableMirrorNodeList
14     Add the mostAvailableMirrorNode's total online time (in minutes) to onlineTimeList
15     if secondMostAvailableMirrorNode ≠ mostAvailableMirrorNode then
16         Add the online times of secondMostAvailableMirrorNode to the previously selected
             mostAvailableMirrorNode's online time list
17     Repeat steps 10 to 16 till the mostAvailableMirrorNode and the secondMostAvailableMirrorNode are
         the same
18     Create variables onlineTimeInMinutes and offlineTimeInMinutes
19     Assign the last entry in onlineTimeList to onlineTimeInMinutes
20     Subtract onlineTimeInMinutes from numberOfMinutesInADay and
21     Assign it variable offlineTimeInMinutes
22     Create an array replicaAvailabilityByHour of size numberOfHoursInADay
23     Loop through mostAvailableMirrorNode's online time list
24     Find the hour and duration of each instance from mostAvailableMirrorNode's online time list
25     Increment the hour index of replicaAvailabilityByHour by the number of minutes in the duration
26     End Loop
27     Loop through the entries in replicaAvailabilityByHour
28     Print the value of each entry in replicaAvailabilityByHour that indicates the core node's profile availability
         during each hour of the day
29     End Loop
30     Print the value of onlineTimeInMinutes and offlineTimeInMinutes that indicates the core node's diurnal
         profile availability via its mirror nodes
```

Figure 2: Availability algorithm

There are different approaches to calculate availability i.e.

1. Users make use of all of its social connections.
2. Users make use of only selected number of social connections as mirrors.

Comparing the above two approaches, we found the second approach however achieves the same availability as first but with fewer replicas and also incurs less overhead cost too as there will be fewer replicas to synchronize whenever an update occurs on core node's replica. This is explained in detail in Sections 3.3.3 and 4.6.

Availability algorithm went through few iterations of improvements, which are summarized as below.

During iteration 1, the algorithm had few issues i.e. as per the algorithm instructions the maximum availability a node could achieve was 1440 minutes (24 hours) but during the implementation we found sometimes a node could achieve more than 24 hours of availability during a day. This is because while calculating the node's total overall availability the algorithm was also taking that into account the overlapping time between the mirror nodes as well. The issue was fixed in 2nd iteration of improvements. Further testing the algorithm we found few issues with randomness of mirror nodes appearing online during different times of the day. As, sometimes mirror nodes were appearing online twice at the same time of the day and that was not correct. The issue was fixed in 3rd iteration along with few other minor improvements in the algorithm design. In the 4th and final iteration, all the hard coded values from the algorithm were removed and replaced with variables.

3.3.2 Update Propagation Delay

Update propagation delay, in the context of decentralized OSNs, is the delay between the occurrence of an update on one node and its arrival on another. It directly depends on the amount of overlapping times between the mirror nodes. High overlapping between the mirror nodes would spread the updates faster and vice versa. Figure 3 shows a simple example of how an update propagates through the mirror nodes.

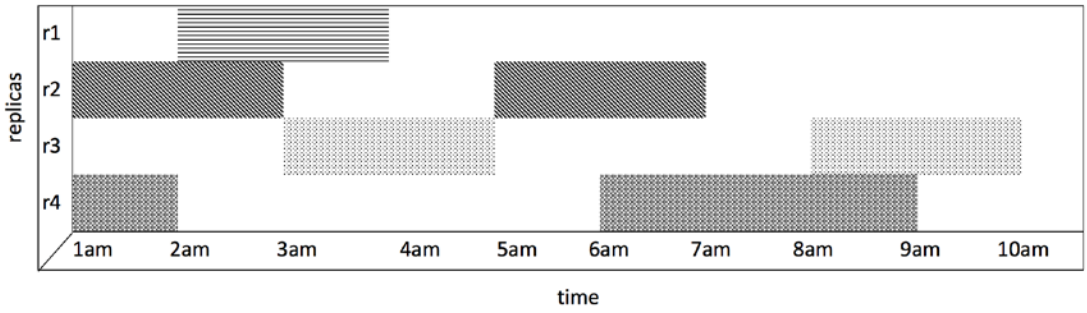


Figure 3: Update propagation delay

In Figure 3, we have indicated four different replicas i.e. $r_1, r_2, r_3,$ and r_4. Assume an update arrives at replica r_1 at 2am then, it will be immediately propagated to r_2 because of overlapping time between them i.e. 2am to 3am. Update on r_3 arrives at 3am from r_1 and not from r_2 because when $r_3$becomes online r_2 goes offline. Similarly, replica r_4 receives an update from replica r_2 at 6am. In r_1 absence r_2 and r_3 are responsible to respond to any incoming requests for core node's updated replica. When r_4 appears online at 6am and finds r_2 is online and has the latest copy of core node's profile. It requests r_2 for the core nodes' updated profile and updates its local copy.

In Section 4.5, we analyse how changing the number of mirror nodes, number of sessions and session duration affects the update propagation delay while keeping the total online time (for

each of the mirror node) constant. The algorithm to calculate the update propagation delay is shown Figure 4. For more detailed version of the algorithm see Appendix 2.

```
Input  : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ],
         sessionsLowerBound [ ][ ], sessionsUpperBound [ ][ ])
Output: 1) Number of mirror nodes that received the core node's updated profile/replica
2) Number of mirror nodes that received the core node's updated profile/replica by hour
3) Time stamps during which the mirror nodes received core node's update profile/replica

1  begin
2      Create a list of mirror nodes of size numberOfMirrorNodes
3      Set the number of sessions for each of the mirror node in numberOfSessions property from
         sessionsDistribution array
4      Set the session duration for each of the mirror node's sessions in sessionDuration property from
         sessionsDuration array
5      Set the lower and upper bounds for each of the mirror node's sessions in sessionLowerBound and
         sessionUpperBound properties from sessionsLowerBound and sessionsUpperBound arrays
6      Set the online patterns to each of the mirror node based on the above parameters
7      Find the most available mirror node from list of mirror nodes and
8      Assign it to variable mostAvailableMirrorNode
9      Set the mostAvailableMirrorNode's received core node's updated profile/replica flag to true. The time
         when the mostAvailableMirrorNode appears online for the first time indicates it has the core node's most
         updated profile replica and the node is responsible for spreading it to other mirror nodes
10     Create a variable numberOfMirrorNodesReceivedUpdatedReplica
11     Increment the numberOfMirrorNodesReceivedUpdatedReplica variable to +1
12     Create an array numberOfMirrorNodesReceivedUpdatedReplicaByHour
13     Create a list variable coreNodeAvailability
14     Assign the online appearances of mostAvailableMirrorNode to coreNodeAvailability
15     Compare the online appearances of coreNodeAvailability with other mirror nodes
16     If they overlap
17     Increment the numberOfMirrorNodesReceivedUpdatedReplica variable to +1
18     Set the overlapping time of the mirror node to its 'core node's updated replica received time stamp'
         property
19     Remove the overlapping time from the mirror node
20     Add the remaining non-overlapping time from the mirror node to coreNodeAvailability
21     Add the other online appearances of the mirror node to coreNodeAvailability from overlapping instance
         onwards
22     Repeat steps 15 to 21 till all online appearances in coreNodeAvailability are compared against all mirror
         nodes' online appearances
23     Loop through each of the mirror nodes 'received core node updated replica time stamp' property
24     If it is not null
25     Print the mirror nodes' ids and the time stamp when they received core node's updated replica
26     Find the hour in the time stamp
27     Increment the hour index of numberOfMirrorNodesReceivedUpdatedReplicaByHour array by +1
28     EndLoop
29     Loop through the numberOfMirrorNodesReceivedUpdatedReplicaByHour array
30     Print the value in the hour index of numberOfMirrorNodesReceivedUpdatedReplicaByHour that indicates
         the number of mirror nodes that received core node's updated replica during each hour of the day
31     EndLoop
32     Print the number of mirror nodes that received the core node's updated replica i.e.
         numberOfMirrorNodesReceivedUpdatedReplica
33     Print the number of mirror nodes that were unable to receive core node's updated replica i.e.
         numberOfMirrorNodes − numberOfMirrorNodesReceivedUpdatedReplica
```

Figure 4: Update propagation delay algorithm

The algorithm shown in Figure 4 went through few iteration of improvement during different phases of the project. The major problem in the algorithm was identified during the implementation phase, when nodes were receiving updates from other mirror nodes at times that were in the past. For example as shown in Figure 3, replica (r4) receives the latest copy of the node at 6am from replica (r2) and that's correct but the problem was initially it was showing as replica (r4) had received the updated copy of the node from replica (r2) at 1am because it was recording the timestamp when the nodes first overlapped in time despite of the fact that

replica (r2) didn't even had the updated copy of the node itself at 1am. The issue was fixed in the 2nd iteration of improvements. Further testing the algorithm and its implementation, we could see a pattern in nodes receiving updates during early hours of the update. This led us in introducing a new metric i.e. capturing the number of nodes receiving updates during differrnt hours of the day. Following the principles of design science research, which encourages researchers to revist and make continuous improvements to different phases of the project, assisted us in going back to the design phase and add more metrics in our algorithm. This helped us in drawing conclusions and making relationships between different variables under study. These metrics were added during the 3rd iteration of improvements. In the 4th and final iteration, all the hard coded values from the algorithm were removed and replaced with variables.

3.3.3 Replication Degree

It is the number mirror nodes users choose to host their data, which act as proxies in their absence. This keeps the users profiles available when the users themselves are offline. The availability algorithm presented in Section 3.3.1 takes the following four parameters i.e. number of mirror nodes, number of sessions, session duration and hours during which mirror nodes are more likely to appear online as input and output the maximum achievable availability. In this section, we discuss what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined. With this approach, users don't necessarily have to spread their data on all their social connections but to only selected ones' that achieves the same availability as all mirror nodes combined. Moreover, this approach also optimizes the overhead cost that comes with replicating users' data to multiple locations to increase availability. Furthermore, the algorithm also shows which nodes to choose and in which order to increase the availability. Moreover, it also shows how the core node's availability increases as the algorithm chooses more and more replicas and become stable after a certain point. The algorithm is shown in Figure 5 and simulation results are in Section 4.6 and Table 4. For more detailed version of the algorithm see Appendix 3.

Input : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ], sessionsLowerBound [ ][ ], sessionsUpperBound [ ][ ])
Output: 1) IDs' of mirror nodes (in order) that can achieve the same availability as all mirror nodes combined
2) Minimum number of replicas required to achieve the same availability as all mirror nodes combined

```
1  begin
2      Create a list of mirror nodes of size numberOfMirrorNodes
3      Set the number of sessions for each of the mirror node in numberOfSessions property from
         sessionsDistribution array
4      Set the session duration for each of the mirror node's sessions in sessionDuration property from
         sessionsDuration array
5      Set the lower and upper bounds for each of the mirror node's sessions in sessionLowerBound and
         sessionUpperBound properties from sessionsLowerBound and sessionsUpperBound arrays
6      Set the online patterns to each of the mirror node based on the above parameters
7      Find the mirror nodes in descending order of their availability and
8      Assign it to variable mostAvailableMirrorNodeList
9      Create a variable minimumNumberOfReplicasRequired
10     Create a variable chosenReplicaList
11     Create a variable onlineTimeList
12     Add the mostAvailableMirrorNode from mostAvailableMirrorNodeList to chosenReplicaList
13     Increment the variable minimumNumberOfReplicasRequired to +1
14     Compare the mostAvailableMirrorNode's online patterns with other mirror nodes in
         mostAvailableMirrorNodeList
15     Remove the overlapping times from the mirror nodes that overlap with the mostAvailableMirrorNode
16     Find the mirror nodes in descending order of their availability i.e. from highest to lowest available nodes
         and
17     Assign it to variable mostAvailableMirrorNodeList
18     Add the mostAvailableMirrorNode's total online time (in minutes) to onlineTimeList
19     if secondMostAvailableMirrorNode ≠ mostAvailableMirrorNode then
20         Add the secondMostAvailableMirrorNode to chosenReplicaList
21         Increment the minimumNumberOfReplicasRequired to +1
22         Add the online times of secondMostAvailableMirrorNode to the previously selected
             mostAvailableMirrorNode's online time list
23     Repeat steps 14 to 22 till the mostAvailableMirrorNode and the secondMostAvailableMirrorNode are
         the same
24     Print the ids' of mirror nodes from chosenReplicaList i.e. these are the ids' of mirror nodes (in order)
       that can achieve the same availability as all mirror nodes combined.
25     Print the entries in onlineTimeList that indicates the number of unique minutes contributed by each of
       the nodes in chosenReplicaList in increasing order; Print the value of
       minimumNumberOfReplicasRequired that can achieve the same availability as all mirror nodes combined
```

Figure 5: Replication degree algorithm

Replication degree algorithm also went through few iterations of improvements as well. In the 1st iteration, the algorithm had everything in place to find the number of mirror nodes/replicas required to achieve the same availability as all mirror nodes combined but had few missing metrics i.e. identifying chosen nodes by their ids and the number of minutes each contributed to nodes total overall availability. These metrics were added during the 2nd iteration of improvements. During the 3rd iteration, minor improvements were made in how we output the data such that it's easy to interpret and visualize on the web. In the 4th and final iteration, all the hard coded values from the algorithm were removed and replaced with variables.

3.3.4 Availability on Demand

Availability on demand is to help new joining nodes to achieve the same availability as other well established nodes in the network despite of having no or few social connections. This is to provide equal opportunities to every node to keep their profiles highly available. This also ensures nodes with few social connections are not discriminated as seen in Safebook where nodes with few social connections were unable to achieve desired availability targets. Every

node maintains a history of online patterns of its mirror nodes for any given number of days. The online patterns of mirror nodes are then analysed and presented to new joining nodes in the form of their average daily and hourly availability and the hour during which their availability is maximum. This is to help new joining nodes to find mirror nodes that are highly available during certain hours of a day. For example, if a new joining node only wants to keep its profile available during 3pm to 6pm then the node may find 3 replicas are enough for its requirements. The availability on demand algorithm is shown in Figure 6 and the simulation results are in Section 4.7. For more detailed version of the algorithm see Appendix 4.

Input : model (numberOfDays, numberOfMirrorNodes, nodes [ ], numberOfSessions [ ][ ], sessionDuration [ ][ ], lowerBound [ ][ ], upperBound [ ][ ])
Output: 1)Average daily availability of each of the mirror node
2) Average hourly availability of each of the mirror node and the hour during which mirror node's availability is maximum

1 **begin**
2 **Create** *a list of mirror nodes of size* numberOfMirrorNodes
3 **Set** *the number of sessions for each of the mirror node in* numberOfSessions *property*
4 **Set** *the session duration for each of the mirror node's sessions in* sessionDuration *property*
5 **Set** *the lower and upper bounds for each of the mirror node's sessions in* sessionsLowerBound and sessionsUpperBound *properties*
6 **Set** *the online patterns to each of the mirror node based on the above parameters for given number of days*
7 **Create** *a two dimensional array i.e.* availability[numberOfHoursInADay][numberOfDays]. *This is used to store mirror node's availability during each hour for given number of days*
8 **Loop** *through each of the mirror node's online times and*
9 **If** *the session spans for hours*
10 **Then** *cut the session in multiple sessions such that each of the session ends and starts with a new hour*
11 **Loop** *through each of those sessions and*
12 **Add** *the number of minutes a mirror node is online during that hour and day in* availability$[hour][day]$ *array*
13 **If** *the session lasts only for few minutes*
14 **Then** *Add the number of minutes a mirror node is online during that hour and day in* availability$[hour][day]$ *array*
15 **Loop** *through* availability$[hour][day]$ *array and*
16 **Find** *the mirror node's average daily and hourly availability and store them in* averageDailyAvailability *and* averageHourlyAvailability *respectively and*
17 **Find** *the hour during which mirror node's availability is maximum and store it in* maximumAvailabilityHour *and*
18 **Find** *the maximum availability for that hour and store it in* MaximumAvailability
19 **Print** *the mirror node's* averageDailyAvailability
20 **Print** *the mirror node's* averageHourlyAvailability
21 **Print** *the hour during which mirror node's availability is maximum i.e.* maximumAvailabilityHour
22 **Print** *the maximum average availability for that hour i.e.* MaximumAvailability
23 **Repeat** *steps* **8 to 22** *for all mirror nodes*

Figure 6: Availability on demand algorithm

Availability on demand algorithm also went through few iterations of improvements as well. In the 1st iteration, we simulated the developed algorithm against the session durations that only lasted for couple of minutes and everything worked fine until we started performing simulations for session durations that spanned over hours. During the 2nd iteration, we improved the algorithm by splitting the session duration (spanning over hours) into smaller sessions such that we could capture the time spread across different hours correctly and use it appropriately in the algorithm to perfrom different calculations. In the 3rd and final iteration, all the hard coded values from the algorithm were removed and replaced with variables.

3.4 Conclusion

In this chapter, we reviewed different research methodologies including Quantitative, Qualitative and Design Science Research Methodology. To benefit from all the aforementioned research methodologies we decided to take a triangulation approach that checks the consistency of our findings from multiple views. We then presented our system design and the algorithms derived from the research objectives stated in Chapter 1 and requirements in Chapter 2. In the next chapter, we evaluate and check the validity of our algorithms, build relationships between different system parameters and draw conclusions from the findings.

CHAPTER 4: RESULTS AND DISCUSSION

In this chapter, we experimentally study the relationship between availability, update propagation delay, and replication degree with core node's number of mirror nodes, their online patterns, number of sessions, and session duration. Availability describes for how long the core node's profile is available for others to access via its mirror nodes. Update propagation delay captures the time it takes for an update to spread to core node's mirror nodes and Replication degree finds the minimum number of replicas required to achieve the same availability as all mirror nodes combined. In this chapter, we also discuss the concept of availability on demand that allows new joining nodes to find good mirrors and achieve desired availability targets despite of having no or few social connections.

4.1 The Context

The table below shows the research objectives set in Chapter 1 along with the assessment method and their outcomes.

No	Research Objectives	Assessment Method	Outcome
1	To investigate into how the existing decentralised OSNs have addressed availability issues in their design	During literature review investigate into how the different decentralised OSNs have addressed or overcome the availability issues in their design.	From the literature review conducted in chapter 2 and research gap analysis presented in section 2.5 we found most of the decentralised OSNs discussed in section 2.4 suffer from the availability issues. Existing work in improving data availability in decentralized OSNs only focuses on how to store users' data across the network. But none of the approaches attempt to answer what is the minimum number of replicas required to achieve high data availability targets, how the node's availability, replication degree, and update propagation delay changes as we alter its number of mirror nodes, their online patterns, number of sessions and session duration.
2	To identify the relationship between the node's availability and the number of mirror nodes, number of sessions and session duration	During simulations change the core node's number of mirror nodes, their number of sessions and session duration while keeping the total online time constant.	From the simulation results we found as we increase the number of mirror nodes the availability increases and becomes stable after it reaches 100% i.e. 24 hours (1440 minutes). We also found as we move from Model A to C (discussed in detail in section 4.4) the probability of overlapping time between the mirror nodes starts to increase. Thus, resulting in individual mirror nodes contributing fewer unique minutes to core node's total overall availability and therefore requiring more mirror nodes for Model C to achieve 100% availability than Model A or B. The same applies for Model B when compared with Model A.
3	To identify the relationship between the node's update propagation delay and the number of mirror nodes, number of sessions and session duration.	During simulations change the core node's number of mirror nodes, their number of sessions and session duration while keeping the total online time constant.	From the simulation results we found as we increase the number of sessions with decreasing session lengths, such that the total availability remains constant, the update propagation delay starts to decrease. With Models A, B and C, discussed in detail Section 4.4, we found for any given number of mirror nodes the update propagates faster with Model C than Model A or B. The same applies for Model B when compared with Model A. This is because as we move from Model A to C the probability of overlapping times between the mirror nodes increases. Thus resulting in spreading the updates faster.
4	To identify what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined from given number of mirror nodes and their online patterns.	During simulations change the core node's number of mirror nodes, their number of sessions and session duration (while keeping the total availability constant) and observe how minimum number of replicas required (MNoRR) value changes.	From the simulations results we found once the availability reaches 100% i.e. 1440 minutes and we increase the total number of mirror nodes to simulate the minimum number of replicas required (MNoRR) starts to decrease for each of the Models A, B, and C and becomes stable at 1100 and 1200 nodes. Moreover, we also found the MNoRR value for Model A is always less than Model B and C. The same applies for Model B when compared with Model C (This is true when MNoRR in not 100%).

5	To introduce the concept of availability on demand and help new joining nodes to find good mirrors.	The model must be flexible, configurable and easy to use. The model must also support analysing nodes' historical online patterns for last 30 days.	It helps new joining nodes to find good mirrors despite of having no or few social connection in the network. It also helps new joining nodes to decide which nodes to choose as mirrors, for how long and during what time.

Table 3: Research objectives, assessment method and outcome

We have structured the rest of the chapter based on the research objectives stated in Table 3.

4.2 Model

To understand how the core node's availability and content dissemination changes by changing the number of mirror nodes and their online patterns, we needed a model that is flexible, configurable and easy to use. Before we get into the details of our model we must cognize the online patterns of users in traditional OSNs that would give deep insights into users' behaviour in existing OSNs and set the requirements for our model. Bachrach et al. (2012) study in examining the correlations between users' personality and the properties of their Facebook profiles revealed Facebook users on average spend 40 minutes online daily. Similar study conducted by Muangngeon and Erjongmanee (2015) in 'analysing usage of Facebook activities through network and human perspectives' revealed an interesting perspective of users' online patterns. In their study, they examined the distribution of user sessions and session duration(s) at different times of a day. They found majority of the users i.e. 28.44% of their sample size (38,355 users) mostly logged in between 09:00 – 11:59 hours and 24.93% of users logged in for 0.5 to 1 hour. Benevenuto et al. (2009) study on the analysis of user workloads in OSNs based on detailed clickstream data collected over a 12-day period from 37,024 users revealed deep insights of users' behaviour in existing OSNs. Their analysis found 63% of the users only logged in once over a 12-day period and most frequently signing users accessed the site 4.1 times a day (on average). Moreover, they also found 51% of the users just spent 10 minutes online over a 12-day period, 14% of the users spent just over an hour and 2% of the users spent more than 12 hours making an average of an hour a day. Furthermore, they also found the amount of time a user spends online is not strongly tied to the number of times they appear online or login to a social network. Junco's (2011) study in finding relationship between multiple indices of Facebook usage and academic performance of students, found, in their sample of 1778 students, most users spent a substantial amount of time on Facebook with a mean of 79 and 106 minutes during two days in 5 and 6 sessions with each session lasting for an average of 22 minutes. Gyarmati and Trinh (2010) conducted a similar study on 80,000 users over six weeks period and found mean daily usage on 5 different social networking sites i.e. Bebo (30.48 mins), MySpace (26.37 mins), MySpace Friends (53.13 mins), Netlog (68.28 mins) and tagged (77.31 mins). Therefore, to construct the online patterns of users that are close to traditional OSNs our model must be configurable with the following parameters:

1. Number of mirror nodes to simulate.

2. How the mirror nodes are grouped into smaller groups depending on their online patterns?
3. Time intervals during which the grouped mirrors nodes are more likely to appear online.
4. Number of sessions for each of the grouped mirror nodes.
5. How sessions are distributed during a day?
6. Session duration for each of the grouped mirror nodes sessions

Ability to configure the above parameters enabled us to construct online patterns of users as in traditional OSNs. With the code shown in Figure 7 we can simulate users' availability as in traditional OSNs.

```
1.  private static int numberOfNodes = 500;
2.  private static int[] nodes = {320, 80, 50, 30, 15, 5};
3.  private static int[][]    sessionsDistribution = {{10, 5, 5}, {1, 2, 2, 10}, {10,
    10, 5}, {20, 20, 5}, {25},{10, 10, 10, 10}};
4.  private static int[][]    sessionsDuration = {{2, 3, 5}, {5, 5, 5, 5}, {5, 5, 5},
    {5, 5, 10}, {20}, {30, 30, 15, 15}};
5.  private static int[][]    sessionsLowerBound = {{0, 6, 18},{13, 15, 16, 17},{0, 0,
    0},{0, 0, 0},{0},{0, 0, 0, 0}};
6.  private static int[][]    sessionsUpperBound = {{3, 2, 2},{1, 1, 1, 2},{24, 24,
    24},{24, 24, 24},{24},{24, 24, 24, 24}};
```

Figure 7: Model

In the Figure 7, at line 1 we set number of mirror nodes to simulate i.e. 500. At line 2, we break down the total number of mirror nodes to simulate in groups of 320, 80, 50, 30, 15, and 5 depending on their sessions distribution, session duration and online patters that we set at lines 3, 4 and 5-6 respectively. Code in Figure 7 is interpreted as:

"*Out of 500 mirror nodes there are 320 nodes that on average spend 60 minutes online (each), during a day, in 20 different sessions. During 00:00 to 03:00 users stay online for 20 minutes in 10 different sessions with each session lasting for 2 minutes only. During 06:00 to 08:00 users stay online for 15 minutes in 5 different sessions with each session lasting for 3 minutes only. During 18:00 to 20:00 users stay online for 25 minutes in 5 different sessions with each session lasting for 5 minutes only.*"

In the above example, timings, hours and duration during which users appear online is calculated as follows:

In the *nodes* array 320 is present at index 0 and represent number of mirror nodes in group 1. If look at the index 0 of *sessionsDistribution* array we will find {10, 5, 5} that represent mirror nodes' total number of sessions during a day i.e. 10 + 5 + 5 = 20. Next, if we look at index 0 of *sessionsDuration* array we will find {2, 3, 5} that represent for how long mirror nodes stay online during each of their sessions i.e. for 10 sessions they stay online for 2 minutes each, for 5 sessions they stay online for 3 minutes each and for next 5 sessions they stay online for 5 minutes each that together add up to 60 minutes i.e. (10 * 2) + (5 * 3) + (5 * 5) = 60. At lines 5 and 6, *sessionsLowerBound* and *sessionsUpperBound* arrays represent hours during which mirror nodes appear online during a day. The three arrays i.e. *sessionsLowerBound:* {0, 6, 18} and *sessionsUpperBound:* {3, 2, 2} and *sessionsDistribution:* {10, 5, 5} together can be interpreted as 320 mirror nodes appear online 10 times during 00:00 to 03:00 hours, 5 times during 06:00 to 08:00 hours and 5 times during 18:00 to 20:00 hours respectively.

Some of the outputs of the model can be seen in Figures in 8 and 9.

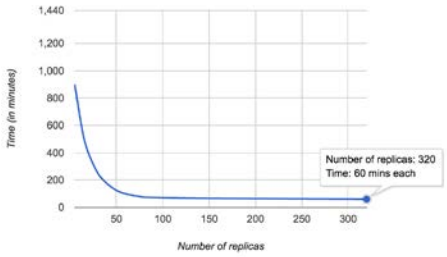


Figure 8: Model – graph

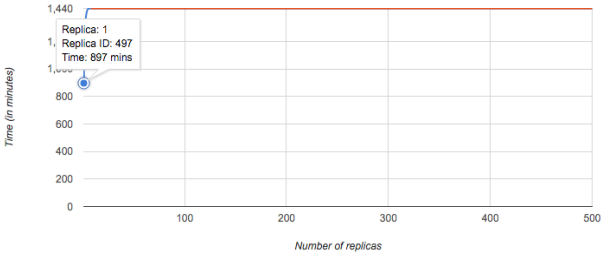


Figure 9: Availability - graph

More detailed simulation results of the Model shown in Figure 7 can be found at www.adilhassan.com/mphil/model.html that also shows core node's update propagation delay in the form of scatter and pie charts and timeline for each of the mirror node's online appearance.

4.3 Simulation

There are a number of simulation tools available to simulate and model communication networks, multiprocessors and other distributed or parallel systems. The most prominent among all are OMNET++[1] (Objective Modular Network Testbed), NS3[2] (Network Simulator), OPNET[3] (Optimum Network Performance), PeerSim[4], and NetSim[5]. They are mainly used to design, simulate, verify, and analyze the performance of different networking protocols (Varga & Hornig, 2008). The aforementioned simulation tools also come with some pre-loaded models and algorithms as well or one can find the models separately online that can be imported and tweaked to user needs to run the simulations. In our case, we could not find any readymade models, algorithms or even the data to perform our simulations. Therefore, we created our own model, with the parameters that we were interested in to generate the data that resembles closely to the online patterns of users of the existing online social networks. This led us to write our own algorithms to analyze the data and draw the conclusions from our findings.

Therefore, we built our own Java based simulation tool that processes the model inputs and generates data. The data generated from the model is then used in different algorithms (discussed in chapter 3) to compute core node's diurnal availability, replication degree, and update propagation delay. The simulator outputs the results in JavaScript arrays that are then used to visualise the data in the form of graphs and charts on the web. The simulator not only outputs core node's diurnal availability but also hourly availability as well (in minutes). The simulator also outputs what is the minimum number of replicas required that can achieve the

1 https://omnetpp.org/

2 https://www.nsnam.org/

3 https://www.riverbed.com/gb/products/steelcentral/opnet.html

4 http://peersim.sourceforge.net/

5 http://www.boson.com/netsim-cisco-network-simulator

same availability as all mirror nodes combined and the list of best nodes to choose as replicas in order. Furthermore, the simulator also outputs the core node's update propagation delay through and via mirror nodes in the form of scatter and pie charts. The scatter and pie charts indicate how the update propagates through the mirror nodes as the time passes and percentage or number of mirror nodes that received the update during any hour of the day respectively. Moreover, the simulation tool also generates and process the online patterns of mirror nodes for up to 30 days and outputs each of the mirror node's average daily and hourly availability and the hour during which the mirror nodes are most available to help newly joining nodes to find good mirrors when they don't have enough friends in the network.

4.4 Availability (Objective 2)

To understand the relationship between the core node's availability and the number of mirror nodes, number of sessions and session duration we modelled the online patterns of mirror nodes by manipulating two different variables i.e. number of sessions and session duration while keeping the total online time constant i.e. 40 mins. We chose 40 minutes of online time as a conservative approach from our findings in section 4.2. We also assume users are geographically distributed and can appear online anytime during a day.

We ran our simulations against three different models/cases, indicated below, with variable number of mirror nodes ranging from 25 to 1200. We stopped our simulations at 1200 nodes because we couldn't see any further improvements to core nodes' total overall availability and the minimum number of replicas required (MNoRR) to achieve the same availability as all mirror nodes combined. Table 4 shows the simulation setup and the results obtained from them.

- **Model A:** Mirror nodes appear online 5 times a day and every time they appear online spend 8 minutes online.
- **Model B:** Mirror nodes appear online 10 times a day and every time they appear online spend 4 minutes online.
- **Model C:** Mirror nodes appear online 20 times a day and every time they appear online spend 2 minutes online.

No.	Mirror Nodes	Model	Diurnal Availability (mins)	MNoRR
1	25	A	717 49.7%	25 100%
		B	735 51.0%	25 100%
		C	748 51.9%	25 100%
2	50	A	1064 73.8%	50 100%
		B	1083 75.2%	50 100%
		C	1087 75.4%	50 100%
3	100	A	1340 93.0%	83 83.0%

		B	1346 93.4%	91 91.0%
		C	1355 94.0%	96 96.0%
4	150	A	1417 98.4%	85 56.6%
		B	1416 98.3%	100 66.6%
		C	1417 98.4%	109 72.6%
5	200	A	1434 99.5%	89 44.5%
		B	1435 99.6%	99 49.5%
		C	1436 99.7%	111 55.5%
6	250	A	1437 99.7%	79 31.6%
		B	1439 99.9%	91 36.4%
		C	1438 99.8%	106 42.4%
7	300	A	1439 99.9%	75 25.0%
		B	1439 99.9%	88 29.3%
		C	1440 100%	100 33.3%
8	350	A	1439 99.9%	74 21.1%
		B	1440 100%	88 25.1%
		C	1440 100%	95 27.1%
9	400	A	1440 100%	75 18.7%
		B	1440 100%	83 20.7%
		C	1440 100%	91 22.7%
10	450	A	1440 100%	73 16.2%
		B	1440 100%	80 17.7%
		C	1440 100%	89 19.7%
11	500	A	1440 100%	70 14.0%
		B	1440 100%	80 16.0%
		C	1440 100%	90 18.0%
12	600	A	1440 100%	68 11.3%
		B	1440 100%	78 13.0%
		C	1440 100%	86 14.3%
13	700	A	1440 100%	67 9.5%
		B	1440 100%	76 10.8%
		C	1440 100%	83 11.8%
14	800	A	1440 100%	65 8.1%
		B	1440 100%	75 9.3%
		C	1440 100%	82 10.2%
15	900	A	1440 100%	65 7.2%

		B	1440 100%	74 8.2%
		C	1440 100%	82 9.1%
16	1000	A	1440 100%	65 6.5%
		B	1440 100%	74 7.4%
		C	1440 100%	80 8.0%
17	1100	A	1440 100%	66 6.0%
		B	1440 100%	73 6.6%
		C	1440 100%	78 7.0%
18	1200	A	1440 100%	66 6.0%
		B	1440 100%	73 6.6%
		C	1440 100%	78 7.0%

Table 4: Simulation results of availability and replication degree algorithms

From the simulation results in Table 4, we can infer, as we increase the number of mirror nodes the availability increases and becomes stable after it reaches 100% i.e. 1440 minutes. The availability and minimum number of replicas required (MNoRR) was calculated by running the simulations against each of the aforementioned models 5 times and then taking the average. The relationship between availability, number of mirror nodes, number of sessions and session duration can be explained by analysing the data presented in Table 4 as below.

Case 1

For the first two data sets i.e. 25 and 50 mirror nodes, with MNoRR values 100%, we can see the availability achieved with Model A is slightly less than the availability achieved with Model B and C. The same applies for Model B when compared with Model C. This is because as we move from Model A to C the probability of overlapping (time) between the mirror nodes increases. And if the sessions overlap then we only lose fraction of a time because of the reduced session lengths as in Model C. This implies the probability of overlapping in Model A is however low but if the nodes overlap then we lose relatively more time in overlapping as compared to Model B and C. The same applies to Model B when compared with Model C. This explains why the availability achieved with Model A is relatively less than the availability achieved with Models B and C.

Case 2

When the availability reaches 100% for each of the Models A, B and C; and we increase the number of mirror nodes the MNoRR value starts to decrease and becomes stable after a certain point i.e. 1100 and 1200 nodes. From the results shown in Table 4, we find MNoRR value for Model C is always greater than Model B and MNoRR value for Model B is always greater then Model A. This is because (as said before) moving from Model A to C the probability of overlapping (time) between the mirror nodes increases. Thus, resulting in individual mirror

nodes contributing fewer unique minutes to core node's total overall availability. Therefore, we see more number of mirror nodes required for Model C to achieve the same availability as Model A and B i.e. 1440 minutes. This explains why MNoRR value for Model C is always greater than MNoRR values of Model A and B. The same applies for Model B when compared against Model A. This also supports our argument presented in case 1.

Please note the difference between case 1 and case 2 is in case 1 overlapping cannot be avoided but in case 2 the algorithm avoids overlapping by selecting mirror nodes with least or no overlapping at all with the ones that are already chosen.

Some of the outputs of simulation results can found in Figures 10, 11, and 12.

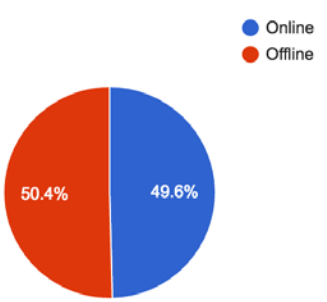


Figure 10: Core node's diurnal availability (Mirror Nodes: 25)

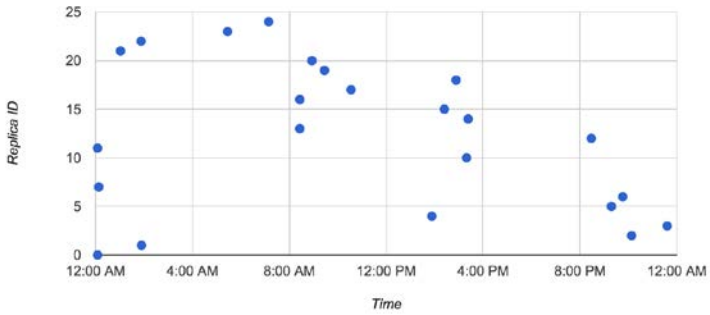


Figure 11: Update propagation delay (Mirror Nodes: 25)

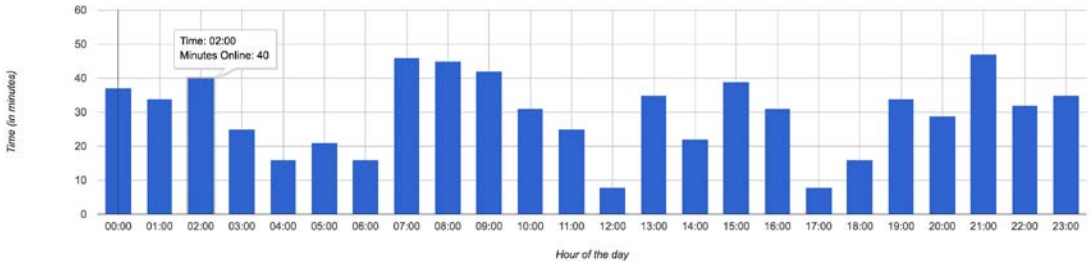


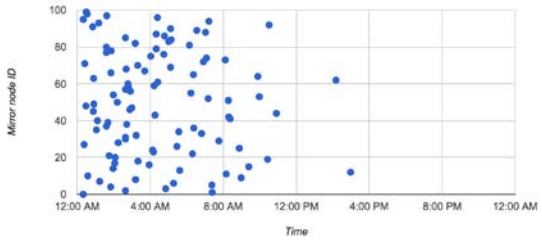
Figure 12: Core node's diurnal availability by hour (Mirror Nodes: 25)

More detailed simulation results can be found at www.adilhassan.com/mphil that contains output of all our simulations ranging from 25 to 1200 nodes.

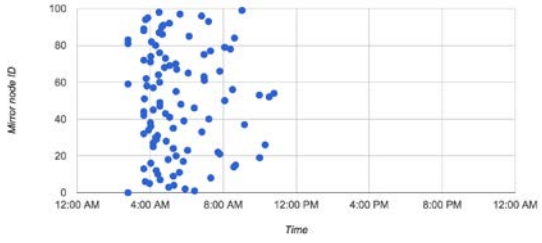
4.5 Update Propagation Delay (Objective 3)

To identify the relationship between the core node's update propagation delay and number of mirror nodes, number of sessions and session duration we needed to model the availability of mirror nodes such that they are online for the same amount of time but differ in number of sessions and session duration. Therefore, we reuse the data generated in section 4.4. From Figures 13 and 14, we can infer as we increase number of sessions with decreasing session lengths, the spread between the nodes receiving updates or in other words the update propagation delay between the mirror nodes decreases. This is because as we move from

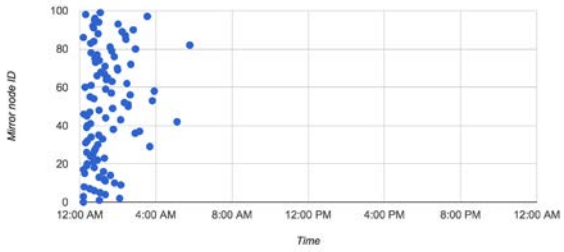
Model A to C the probability of overlapping times with other mirror nodes starts to increase. Thus resulting in spreading the updates faster to other mirror nodes. This can be seen in figures 13 and 14 as we move from Model A to B and C we can see a significant increase in the number of nodes receiving updates during the first few hours of the update.



100 Mirror Nodes – Model A – Data set 1

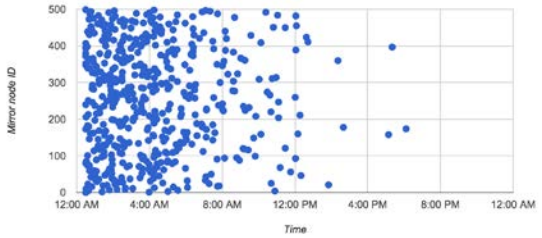


100 Mirror Nodes – Model B – Data set 1

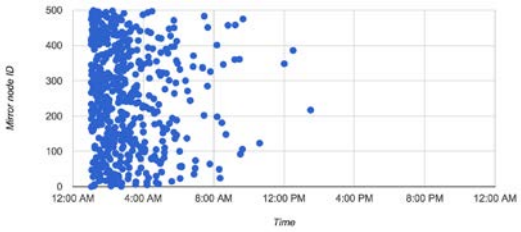


100 Mirror Nodes– Model C – Data set 1

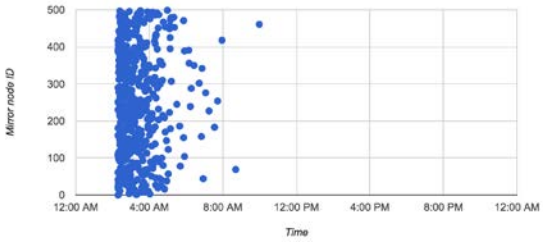
Figure 13: Update propagation delay – 100 Mirror Nodes



500 Mirror Nodes – Model A – Data set 1



500 Mirror Nodes – Model B – Data set 1



500 Mirror Nodes – Model C – Data set 1

Figure 14: Update propagation delay - 500 Mirror Nodes

We were also interested in finding the percentage of nodes receiving updates during the first 3 hours of the update and found some interesting results shown in Table 5.

Number of Mirror Nodes	Model		
	A	B	C

	Percentage of nodes receiving update during first 3 hours		
25	32%	29.6%	51.2%
50	30.4%	39.6%	78%
100	38.8%	58.6%	88%
150	39.46%	64.82%	89.06%
200	44.8%	64%	91.1%
250	44.92%	69.52%	90.24%
300	42.4%	71.14%	90.26%
350	45.08%	68.94%	90.54%
400	45.76%	67.08%	90.02%
450	40.5%	70.66%	89.68%
500	42.8%	72.44%	91.16%
600	47.3%	65.88%	89.08%
700	41.82%	65.06%	88.78%
800	44.42%	65.16%	86.36%
900	44.5%	70.74%	91.08%
1000	42.88%	67.88%	87.04%
1100	45.72%	68.48%	90.38%
1200	48.76%	70.14%	89.38%

Table 5: Simulation results of update propagation delay algorithm

From Table 5, we can see as we move from Model A to C the percentage of nodes receiving update during the first 3 hours of the update increases with an exception '25' mirror nodes that does not follow a pattern as in seen in rest of the table. We highlighted some of the cells in table in red to indicate the outliers in our data, as they don't closely match in numbers when compared with other cells of their respective models. If we look at the percentage of nodes receiving update during the first 3 hours for 200 nodes in model A (i.e. 44.8%) and compare it with 250, 350, 400, 800, 900, and 1100 nodes we find the percentage of nodes receiving update is almost the same with ±1% difference. We also see a similar pattern in Model B and Model C as well. Moreover, if we look at the percentage of nodes receiving update during first three hours for 150 nodes in Model C (i.e. 89.06%) and compare it with 1200 nodes (89.38%) we find the difference is just 0.32% but if we compare the number of nodes receiving updates for each of the two datasets we find a significant difference between the two i.e. 133 of 150 and 1072 of 1200 mirror nodes receiving updates. This indicates the percentage of nodes receiving updates, during the first 3 hours of the update, for different number of mirror nodes is almost the same with some degree of difference. The difference however decreases as we move from Model A to C.

More detailed simulation results can be found at www.adilhassan.com/mphil that contains the output of all our simulations ranging from 25 to 1200 mirror nodes.

4.6 Replication Degree (Objective 4)

To identify the best and minimum number of replicas required to keep the core node's profile highly available we performed our simulations on two different data sets. The first, whilst keeping the total availability constant we altered the total number of mirror nodes, number of sessions and session duration till we reached a point where we couldn't see any further improvements in MNoRR (minimum number of replicas required) value and total availability achieved. This gave us interesting statistics on how MNoRR value changes on changing the number mirror nodes, number of sessions and session duration. Second, to test the algorithm and see how the availability changes as we alter the online patterns of the mirror nodes such that very few of them are available for most of time and most of the mirror nodes are available for just under few minutes. We expected to see the algorithm to choose the most available mirror node first followed by the second most available mirror node and so on. For the first set of simulations, we reuse the data from Section 4.4. From the Table 4, we can see as we increase the number of mirror nodes the MNoRR value (percentage) starts to decrease for each of the Models A, B, and C. The reason for that is as we increase the total number of mirror nodes the algorithm finds more and more nodes with least overlapping times. Thus, resulting in contributing more unique minutes to core node's total overall availability with relatively less nodes. From the data set in Table 4, we also see once the availability reaches 100% the MNoRR value for each of the Model A, B and C starts to decrease and becomes stable after a certain point i.e. 1100 and 1200 nodes. Moreover, we can also see the MNoRR value for Model A is always less than the MNoRR value for Models B and C. The same applies for Model B when compared with Model C. The reason for that is as we increase the number of sessions from Model A to C the probability of overlapping time between the mirror nodes starts to increase. Thus, resulting in individual mirror nodes contributing fewer unique minutes to core node's total overall availability. Therefore, we see more number of mirror nodes required for Model C to achieve the same availability as Model A and B i.e. 1440 minutes. In table 4, the MNoRR value indicates the minimum number of replicas required to keep the core node's profile highly available where 'high availability' may mean 90%, 99%, 99.9% etc. diurnal availability. From the simulation results, we also see a strong correlation between the MNoRR value and the update propagation delay (when availability reaches 100%) i.e. low MNoRR value suggests high spread and high MNoRR value suggests low spread in update propagation delay. To see how the algorithm selects the best nodes or the most available nodes first from the list mirror nodes we performed simulations on two different models i.e.

1. All mirror nodes are available for the same number of minutes in a day i.e. 40 minutes.

Model

```
1.  private static int numberOfNodes = 500;
2.  private static int[] nodes = {500};
3.  private static int[][] sessionsDistribution = {{20}};
4.  private static int[][] sessionsDuration = {{2}};
5.  private static int[][] sessionsLowerBound = {{0}};
6.  private static int[][] sessionsUpperBound = {{24}};
```

Figure 15: Replication degree - Model A

In the model shown in Figure 15, we simulate 500 mirror nodes where every node is online for 40 minutes during a day in 20 different sessions with each session lasting for 2 minutes only. Figure 16 shows the results of our algorithm choosing different replicas. The algorithm chose Replica ID 0 as the first most available mirror node contributing 40 minutes to core node's total overall availability. It then chooses Replica ID 3 again contributing 40 minutes to core node's overall availability. The reason our algorithm chose Replica ID 3 as a second best choice instead of Replica ID 1 and Replica ID 2 is because of their overlapping times with Replica ID 0. Replica ID 1 overlaps with Replica ID 0 at 12:17pm for 1 minute and Replica ID 2 overlaps with Replica ID 0 at 11:58am for 2 minutes. If our algorithm would have chosen Replica ID 1 and Replica ID 2 as second best choice then they could only contribute 39 and 38 unique minutes respectively to core node's total overall availability instead of 40minutes as we have seen in choosing Replica ID 3 as the second best choice. The algorithm repeatedly finds and chooses most available mirror nodes until it reaches maximum achievable availability.

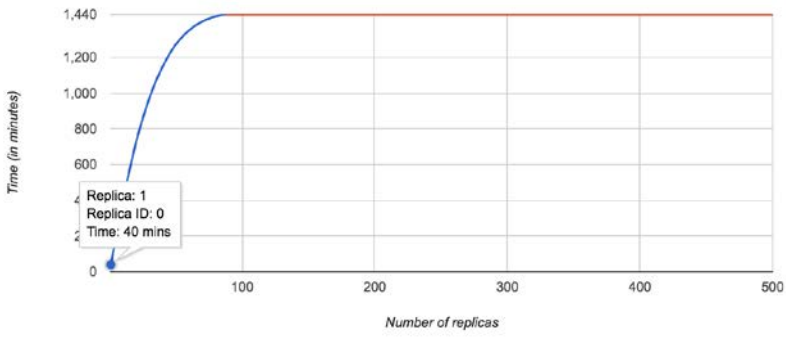


Figure 16: Replication degree - Model A - Result

More detailed simulation results of the model shown in Figure 15 can be found online at www.adilhassan.com/mphil/MNoRR1.html that also contains the output of core node's update propagation delay in the scatter and pie charts and the timeline of mirror nodes' individual online appearances during a day.

2. Some of the mirror nodes are available for slightly longer than the others.

Model

```
1.  private static int numberOfNodes = 500;
2.  private static int[] nodes = {320, 80, 50, 30, 15, 5};
3.  private static int[][] sessionsDistribution = {{10, 5, 5}, {1, 2, 2, 10}, {10, 10,
    5}, {20, 20, 5}, {25}, {10, 10, 10, 10}};
4.  private static int[][] sessionsDuration = {{2, 3, 5}, {5, 5, 5, 5}, {5, 5, 5}, {5,
    5, 10}, {20}, {30, 30, 15, 15}};
5.  private static int[][] sessionsLowerBound = {{0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0},
    {0, 0, 0}, {0}, {0, 0, 0, 0}};
6.  private static int[][] sessionsUpperBound = {{24, 24, 24}, {24, 24, 24, 24}, {24,
    24, 24}, {24, 24, 24}, {24}, {24, 24, 24, 24}};
```

Figure 17: Replication degree - Model B

With the model shown in Figure 17, we simulate 500 mirror nodes in groups of 320, 80, 50, 30, 15, and 5 depending on their number of sessions, session duration and timings during which

they are most available. From the model, we can see 320 of 500 nodes are available for just an hour in a day (each) and 5 of 500 nodes are available for 900 minutes each. Therefore, we expect our algorithm to choose the nodes that are most available first. Looking at the results shown in Figure 18 we find the algorithm chooses Replica ID 496 as the first mirror contributing 900 minutes to core node's total overall availability. The second mirror node it chooses is Replica ID 498 contributing 348 unique minutes to core node's overall availability and so on until reaches 100% availability i.e. 1440 minutes.

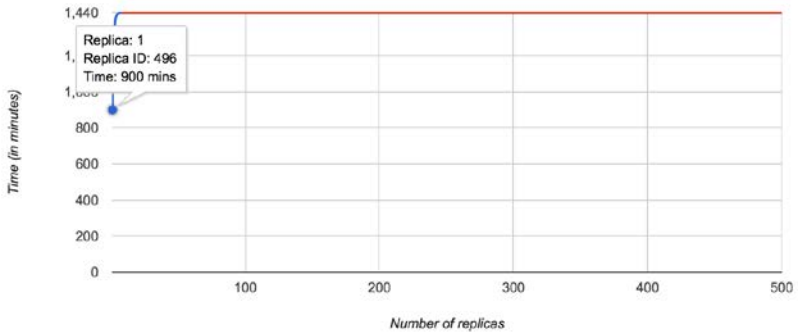


Figure 18: Replication degree - Model B Result

More detailed simulation results of the model shown in Figure 17 can be found online at www.adilhassan.com/mphil/MNoRR2.html that also contains the output of core node's update propagation delay in the scatter and pie charts and the timeline of mirror nodes' individual online appearances during a day.

4.7 Availability on Demand (Objective 5)

To help new joining nodes to find good mirrors, when they don't have enough social connections in the network to choose as replicas, we introduce the concept of availability on demand. It helps new joining nodes to find good mirrors by receiving statistics from other nodes about their mirror nodes in the network. The mechanisms of statistics collection, collation and verification are out of scope of this thesis. We assume nodes are well behaving, trustworthy, cooperative and maintains history of online patterns of its mirror nodes for last 30 days. When requested our algorithm converts the historical online patterns of mirror nodes into useful information and present it to requesting nodes in the form of average daily and hourly availability and the hour during which the mirror nodes are most available. The new joining nodes can then choose that node as a possible replica-hosting location either for number of hours or for just an hour during which it is most available depending on the node's availability requirements.

To test the algorithm we developed a model that is flexible, configurable and easy to use. We can customize our model with the following parameters:

1. Total number of mirror nodes to simulate.
2. How the mirror nodes are grouped into smaller groups depending on their online patterns.
3. For how many days do we want to perform our simulation?
4. Number of sessions for each of the grouped mirror nodes per day.

5. Session duration for each of the grouped mirror nodes per day.
6. Lower and upper bounds of time during which grouped mirror nodes are more likely to appear online per day.

The figure below shows a model that we will be using to test our algorithm.

```
1.  private static int numberOfDays = 3;
2.  private static int numberOfNodes = 6;
3.  private static int[] nodes = {3, 2, 1};
4.  private static int[][] numberOfSessions = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
5.  private static int[][] sessionDuration = {{2, 4, 6}, {8, 10, 12}, {14, 16, 18}};
6.  private static int[][] lowerBound = {{0, 0, 0}, {4, 5, 6}, {7, 10, 13}};
7.  private static int[][] upperBound = {{24, 24, 24}, {4, 4, 4}, {5, 5, 5}};
```

Figure 19: Model - Availability on demand

In Figure 19, at line 1, we set the number of days we want to run our simulation for i.e. `3`. At line 2, we set the number of mirror nodes to simulate i.e. `6`. At line 3, we break down the mirrors nodes into groups of `{3, 2, 1}` depending on their online patterns defined at lines 4 to 7. At lines 4 and 5, we set number of sessions and session duration for each of the grouped mirror nodes per day respectively. At lines 6 and 7, we set the lower and upper bounds of time during which the grouped mirror nodes are more likely to appear online.
The model in Figure 19 is interpreted as:

We want to simulate 6 nodes for 3 days. From total of 6 nodes, 3 nodes appear online anytime during a day. On the first day, they appear online once and spend 2 minutes online. On the second day, they appear online 2 times and spend 4 minutes online during each instance. On the third day, they appear online 3 times and spend 6 minutes online during each instance.

From the total of 6 nodes, 2 nodes appear online at various times during a day i.e. between 4am and 10am. On the first day, they appear online 4 times between 4am and 8am and spend 8 minutes online during each instance. On the second day, they appear online 5 times between 5am and 9am and spend 10 minutes online during each instance. On the third day, they appear online 6 times between 6am and 10am and spend 12 minutes during each instance.

From the total of 6 nodes, 1 node appears online at various times during a day i.e. between 7am and 6pm. On the first day, the node appears online 7 times between 7am and 12noon and spends 14 minutes online during each instance. On the second day, the node appears online 8 times between 10am and 3pm and spends16 minutes online during each instance. On the third day, the node appears online 9 times between 1pm and 6pm and spends 18 minutes online during each instance.

In the above interpretations number of sessions, session duration, days and online timings is calculated as follows:

At line 1, we set the number of days we want to run our simulation for i.e. `3`. At line 2, we set the number of mirror nodes to simulate i.e. `6`. At line 3, we break down the nodes in group of `3`, `2`, and `1` node(s) depending on their online patterns. At line 4, we set number of sessions for each of the grouped mirror nodes per day with a two-dimensional array i.e. `{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}`. Each row in the *`numberOfSessions`* array is linked to individual elements in the *`nodes`* array such that `{1, 2, 3}` in *`numberOfSessions`* array belongs to 3 nodes in *`nodes`* array at index 0, Similarly `{4, 5, 6}` in *`numberOfSessions`* array belongs to 2 nodes and `{7, 8, 9}` belongs to 1 node. Furthermore, each column in each of these rows represent a new day. The *`sessionDuration`* array can be interpreted following the same rules as *`numberOfSessions`* array. The hours in lower and upper bound arrays are calculated as follows: If we look at the index 2 of *`lowerBound`* and *`upperBound`* arrays we find `{7, 10, 13}` and `{5, 5, 5}` respectively. These indicate the hours during which the node(s) appears online during a day. The *`lowerBound`* and *`upperBound`* arrays when seen together with *`numberOfSessions`* and *`sessionDuration`* can be interpreted as follows:

1. A node appears online 7 times during a day between 7am and 12noon (7 + 5 = 12) and spends 14 minutes online during each appearance.
2. A node appears online 8 times during a day between 10am and 3pm (10 + 5 = 15) and spends 16 minutes online during each appearance.
3. A node appears online 9 times during a day between 1pm and 6pm (13 + 5 = 18) and spends 18 minutes online during each appearance.

Figures 20, 21, and 22 show the output of model for a node that appears online between 7am and 6pm i.e. index 2 of nodes array. From the figures, we can see how the node's daily and average hourly availability changes and the hour during which it is most available.

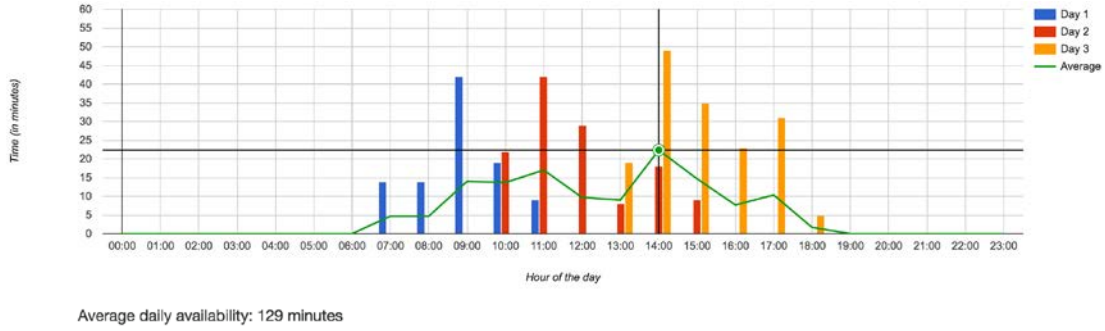


Figure 20: Availability on demand - 3 days

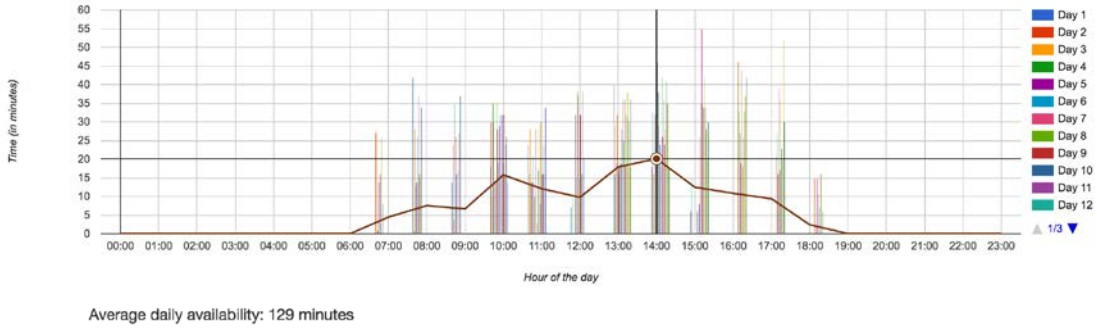


Figure 21: Availability on demand - 30 days

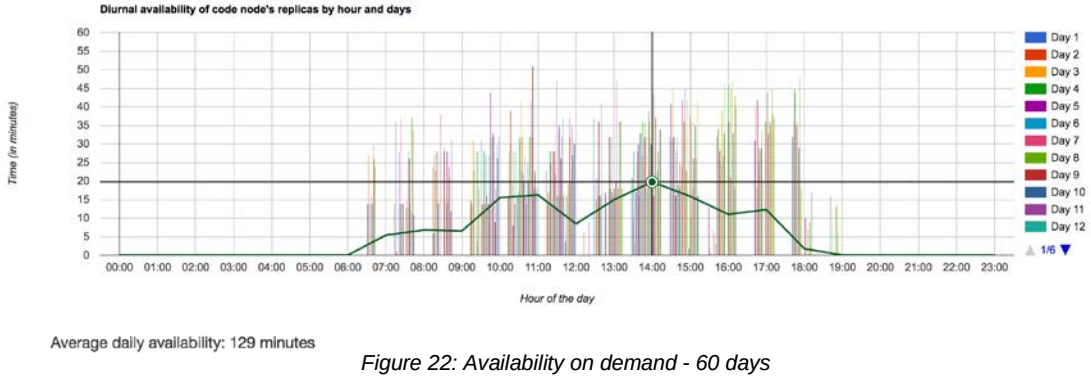


Figure 22: Availability on demand - 60 days

From the information shown in Figures 20, 21, and 22 a new joining node can either choose that node as a mirror for hours during which it is most available or find a better replica-hosting node depending on its availability requirements.

The graphs shown in Figures in 20, 21, and 22 can also be found online at www.adilhassan.com/mphil/AoD1.html, .../AoD2.html and .../AoD3.html that allow users to scale and zoom in to show more detailed information.

4.8 Conclusion

In this chapter, we experimentally studied different system properties including availability, availability on demand, replication degree, and update propagation delay in conjunction with other parameters i.e. total number of mirror nodes, number of sessions and session duration. We have also performed extensive evaluation of our algorithms via simulations and analysed the results obtained from them. We conclude that the implementation of decentralised OSNs is feasible but under certain requirements of user online times and the number of mirror nodes that determines the maximum achievable availability, replication degree and update propagation delay.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

The success of online social networks has changed the way people interact and communicate with each other today. Where their success has brought advantages to the people and communities it has also put the users' privacy and security at risk as well. This thesis contributes in performing feasibility checks of creating next generation of online social networks built on decentralised architectures. The overall aim of the research was:

"To develop a model and investigate into how the node's availability, replication degree and update propagation delay (dependent variables) changes on altering its number of mirror nodes, their online patterns, number of sessions and session duration (independent variables) by studing the effects of changing independent variables on each of the dependent variables"

To achieve this aim a Model (Section 4.2) was developed that helped to construct and simulate the online patterns of users as in traditional online social networks. The data generated from the model was then used as input for different algorithms presented in this thesis including availability, replication degree, update propagation delay and availability on demand. Chapter 3 (System Design – Section 3.3) discusses the aforementioned algorithms in detail. The algorithms presented in Chapter 3 are derived from the research objectives set in Chapter 1 and requirements in Chapter 2. In Chapter 4 (Results and Discussion), we performed extensive evaluation of our algorithms and model via simulations, investigated into how the node's availability, replication degree and update propagation delay changes on altering its number of mirror nodes, their online patterns, number of sessions and session duration and drawn conclusions from our findings.

This thesis addresses the following research objectives:

Research Objective 1: *To investigate into how the existing decentralised OSNs have addressed availability issues in their design.*

In Chapter 2, we reviewed a number of different decentralised OSNs that differ greatly in their design but all aim to solve the same problem i.e. how to preserve users' privacy while offering full set of services to the users of OSNs. We found researchers typically used replication as their primary choice in achieving high data availability but we have also seen dependency on third parties in the form cloud storage providers e.g. Amazon EC2 and permanently available resources as another means of achieving high data availability in decentralised OSNs as well. The latter two, however, do not completely justify the true essence of decentralised OSNs as they may leave the privacy of the users in the hands of the service providers (e.g. Vis-à-Vis and Diaspora). This is discussed in detail in Section 2.4 and 2.5.

Research Objective 2: *To identify the relationship between the node's availability and the number of mirror nodes, number of sessions and session duration.*

This research objective is addressed in Chapter 4 – Section 4.4. To identify the relationship between the node's availability and its number of mirror nodes, number of sessions and sessions we modelled the online patterns of mirror nodes by manipulating two different variables i.e. number of sessions and session duration while keeping the total availability constant. We found as we increase the number of mirror nodes the availability increase and becomes stable after it reaches 100% i.e. 1440 minutes (24 hours). We also found as we increase the number of sessions with reduced sessions lengths such that the total availability remains constant the probability of overlapping times between the mirror nodes increases. Thus, resulting in individual mirror nodes contributing fewer unique minutes to node's total overall availability.

Research Objective 3: *To identify the relationship between the node's update propagation delay and its number of mirror nodes, number of sessions and session duration.*

This research objective is address in Chapter 4 – Section 4.5. We found as we increase the number of sessions with reduced session lengths such that the total availability remains constant the update propagation delay between the mirror nodes starts to decrease. This is because as we increase the number of sessions with reduced session lengths the probability of overlapping time between the mirror nodes starts to increases; thus, resulting in spreading the updates faster as compared to mirror nodes that appear online relatively fewer number of times but for longer durations.

Research Objective 4: *To identify what is the minimum number of replicas required, to achieve the same availability as all mirror nodes combined from given number of mirror nodes and their online patterns.*

This research objective is address in Chapter 4 – Section 4.6. We found once the availability reaches 100% and we increase the number of mirror nodes, the minimum number of replicas required (MNoRR) starts to decrease and becomes stable after a certain point. Moreover, we also found once the availability reaches 100% i.e. 1440 minutes (24 hours) the MNoRR value for mirror nodes that appear online fewer number of times but for longer durations is always

less than the mirror nodes that appear online relatively more often but for shorter durations. This is because as we increase the number of sessions with reduced session lengths the probability of overlapping times between the mirror nodes starts the increase; thus, resulting in individual mirror contributing fewer unique minutes to node's total overall availability and therefore requiring more number of mirror nodes to achieve the same availability (i.e. 1440 minutes or 24 hours) as mirror nodes that appear online fewer number of times but for longer durations.

Research Objective 5: *To introduce the concept of availability on demand and help new joining nodes to find good mirrors.*

This research objective is addressed in Chapter 4 – Section 4.7. We introduced the concept of availability on demand that helps new joining nodes to find good mirrors when they don't have enough friends/social connections in the network to choose as mirror nodes. From well-established nodes, the new joining nodes receive the following statistics i.e. average daily and hourly availability and the hour during which their mirror nodes are most available. The new joining nodes then decide which nodes to choose as possible mirror nodes/replica hosting locations and for how many hours or for just an hour during which they are most available depending on the new joining nodes' availability requirements.

The thesis also addresses the following research questions:

Research Question 1: *What are the challenges in existing decentralized OSNs in achieving high data availability?*

Over the past few years an extensive amount of research has gone into creating next generation of OSNs built on decentralized architectures. Where the different decentralised OSNs have tried to overcome the privacy and security issues of the existing OSNs have also introduced some other challenges as well e.g. how to achieve high data availability with minimum number of replicas possible, how to prevent the system discriminating nodes with few social connections and offer equal opportunities to everyone to keep their profiles highly available, and how to achieve desired availability targets without depending on third parties (e.g. cloud storage providers) or permanently available resources. To overcome these issues Shahriar and his colleagues (2013) proposed a different technique called 'grouping policy' where they found 2 availability grouping policy for replication delivers high data availability but one of the problems with their simulation setup was that they used the data set (online patterns) of users of file sharing applications which does not truly depict the online patterns of users of OSNs. Similar studies were conducted by other researchers in determining relationship between users availability and storage capacity of their devices, using cloud assisted data replication techniques in improving data availability in decentralised OSNs, exchanging recommendations between friends in recognizing good and bad mirrors, and fostering user driven replication instead of random system driven replica placement to achieve high data availability. From the literature, we found existing working in improving data availability in

decentralised OSNs only focuses on how to store users' data across the network but none of the approaches attempt to answer for any given number of mirror nodes and their online patterns what is the minimum number of replicas required to achieve the same availability as all mirror nodes combined, how changing the node's number of mirror nodes and their online patterns affects the node's availability and update propagation delay and how the nodes can achieve desired availability targets despite of having no or few social connections in the network.

Research Question 2: *How the node's availability changes as we alter its number of mirror nodes, their online patterns, number of sessions and session duration?*

As we increase the number of mirror nodes the availability increases and becomes stable after it reaches 100% i.e. 1440 minutes (24 hours). We also found when the MNoRR value is 100% then the availability achieved with mirror nodes that appear online fewer number of times but for longer durations is slightly less than than the availability achieved with mirror nodes that appear online several times during a day but only spend fraction of a time during each of their appearance. This is because as we increase the number of sessions with reduced session lengths the probability of overlapping between the mirror nodes starts to increase and if the nodes overlap then we only lose fraction of a time because of their reduced session lengths. The probability of overlapping for mirror nodes that appear online fewer number of times during a day is however low but if the nodes overlap then we lose a significant amount in overlapping which cannot be compensated as there are no other mirror nodes availabile to avoid that overalppping.

Research Question 3: *How the node's update propagation delay changes as we alter its number of mirror nodes, number of sessions and session duration?*

From the simulation results, we found as we increase the number of mirror nodes the update propagation delay between the mirror nodes remains almost the same in terms of the percentage of nodes receiving updates during first few hours of the update. Moreover, we also found as we increase the number of sessions with reduced session lengths whilst keeping the total availability constant, the update propagation delay between the mirror nodes starts to decrease. This is because as we increase the number of sessions with reduced session lengths the probability of overlapping time between the mirror nodes starts to increase; thus, resulting in spreading the updates faster.

Research Question 4: *For given number of mirror nodes and their online patterns what is minimum number of replicas required to achieve the same availability as all mirror nodes combined?*

The minimum number of replicas required to achieve the same availability as all mirror nodes combined depends on the following two parameters:

i) Total number of mirror nodes

ii) Mirror nodes' online patterns

If we have the number of mirror nodes and their online patterns as input we can then find the minimum number of replicas required to achieve the same availability as all mirror nodes combined. We found as we increase the number of mirror nodes the availability increases and becomes stable after a certain point but MNoRR value keeps dropping and becomes stable at a different point. This is because as we increase the total number of mirror nodes, we find more and more mirror nodes with least overlapping time between them; thus, resulting in each mirror node contributing more unique minutes to node's total overall availability. Moreover, comparing the MNoRR values for mirror nodes that appear online several times during a day and only spend fraction of a time during each of their appearance with mirror nodes that appear online only fewer number of times but spend a significant amount of time during each of their appearance we found the MNoRR values of the latter is always less than the other (given MNoRR is not 100%). This is because as we increase the number of sessions with reduced session lengths, whilst keeping the total availability constant, the probability of overlapping between the mirror nodes start to increase; thus, resulting in each mirror node contributing relatively fewer unique minutes to nodes total overall availability. And therefore, nodes that appear online more often but for shorter durations require more mirror nodes to achieve the same availability as mirror nodes that appear online less often but for longer durations.

Research Question 5: *How the new joining nodes can find good mirrors and achieve desired availability targets, despite of having no or few social connections in the network?*

To help new joining nodes to find good mirrors we introduced the concept of availability on demand. New joining nodes receives statistics from other well established nodes in the network about their mirror nodes in the form of their average daily and hourly availability and the hours during which they are most available. This helps new joining nodes to identify mirror nodes that meet their availability requirements.

Referring back to the hypothesis stated in Chapter 1 – Section 1.5:

For given number of mirror nodes and their online patterns it is algorithmically possible to determine the minimum number of replicas required to keep the nodes profile highly available, where highly availability may mean 90%, 99%, 99.9% etc. diurnal availability.

The work presented in this thesis investigates in checking the feasibility of creating next generation of online social networks built on decentralized architectures which involved creating artifact, developing algorithms, performing simulations and analysing results. We analysed the affects of changing independent variables including number of mirror nodes, their online patterns, number of sessions and session duration on the dependent variables i.e. node's availability, replication degree and update propagation delay. From the results presented in this thesis we found it is algorithmatically possible to determine the minimum number of replicas required to keep the node's profile highly available, where high availability may mean 90%, 99%, 99.9% diurnal availability.

We conclude the feasibility of such decentralized OSNs is possible but under certain requirements of node's number of mirror nodes and their online patterns that determines the node's maximum achievable availability, replication degree and update propagation delay.

5.2 Limitations and Future Work

The work presented in this thesis uncovers the relationship the node's between availability, replication degree and update propagation delay with the node's number of mirror nodes, their online patterns, number of sessions and session duration. The research directions arising for this work as are as follows:

5.2.1 Enhancement of Data

1) The results presented in this Thesis used the following three Models:
    a. Model A: Mirror nodes appear online 5 times a day and every time they appear online spend 8 minutes online.
    b. Model B: Mirror nodes appear online 10 times a day and every time they appear online spend 4 minutes online.
    c. Model C: Mirror nodes appear online 20 times a day and every time they appear online spend 2 minutes online.

In each of the models, the mirror nodes were available for a constant time i.e. 40 minutes but differ in their online patterns. With this setup, we found some interesting results that we have discussed in Chapter 4. Changing the constant availability of 40 minutes to 30 or 50 minutes and observing its effects on availability, replication degree and update propagation delay may reveal new findings. Furthermore, studying the effects of changing the online patterns of mirror nodes and their geographical distribution may also reveal new and interesting findings.

2) The model presented in this thesis has elements of randomness e.g. when the nodes should appear online during different times of the day given it has the upper and lower bounds of time for randomness. Future work will consider changing the existing model into a probabilistic model for all the differernt parameters we have in the existing model and observe its effects on node's availability, replication degree, update propagation delay, and availability on demand.

5.2.2 Enhancement of Algorithms

1) In the availability on demand algorithm, verification of the data received from the nodes about their mirror nodes in the network was left to do as a future work. Identifying nodes sending false/contradicting information about mirror nodes would help in identifying and filtering out malicious or unreliable nodes from the network. This can be done by checking if multiple nodes who share one or more mirror node(s) in common are sending contradicting information about them. Nodes sending false information can then be

identified and announced as malicious or unreliable nodes and prevented from sending any data to existing or new joining nodes in the network.

2) The algorithms presented in this thesis were evaluated in a simulation-based learning environment. As future work, it would be interesting to see how these algorithms perform and behave when integrated in real world applications.
3) The algorithms presented in this thesis only consider links/connections between nodes. Gilbert and Karahalios (2009) however, believe the existence just of a relation between nodes itself only contributes very little to its tie strength. Future work will take the strength into account as part of the simulation.
4) The algorirthms presented in this thesis assume every node participating in the network can accommodate the storage requirements of its mirror nodes. This however may not always be true because nodes have limited storage capacities that they can offer to host the replicas of their friends/social connections. Cogitating the available storage space and the strength of the relationship between nodes could be additional parameters to consider when choosing to host the replicas of other nodes in the network.

5.2.3 Enhancement of Research Scope

Future work will investigate into the privacy and security issues of the existing online social networks. The research objectives of this study could be to identify and characterize differernt types of attacks, the damages that they may cause to users' privacy and how they can be prevented.

GLOSSARY OF TERMS

DSR – Design Science Rearch

FOAF – Friend of a Friend

HCI – Human Compuer Interface

IS – Information Systems

MNoRR – Minimum Number of Replicas Required

NS3 – Network Simulator

OMNET++ – Object Modular Network Testbed

OPNET – Optimum Network Performance

OSN – Online Social Network

PUBLICATION

Conrad, M., **Hassan, A.,** Koshy, L., Kanamgotov, A., Christopoulos, A. (2017) 'Strategies and Challenges to Facilitate Situated Learning in Virtual Worlds Post-Second Life'*. Computers in Entertainment - ACM,* 15(1) doi: 10.1145/3010078

REFERENCES

1. Bachrach, Y., Kosinski, M., Graepel, T., Kohli, P. & Stillwell, D. (2012) "Personality and Patterns of Facebook Usage", *Proceedings of the 4th Annual ACM Web Science Conference,* Evanston, Illinois New York, NY, USA: ACM. p24-32

2. Baden, R., Bender, A., Spring, N., Bhattacharjee, B. & Starin, D. (2009) 'Persona: An Online Social Network with User-defined Privacy', *SIGCOMM Comput.Commun.Rev.,* 39 (4), pp.135-146.

3. Banville, C. and Landry, M. (1989) Can the field of MIS be disciplined? *Communications of the ACM*, 32, pp. 48-61

4. Benevenuto, F., Rodrigues, T., Cha, M. & Almeida, V. (2009) "Characterizing User Behavior in Online Social Networks", *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference,* Chicago, Illinois, USA New York, NY, USA: ACM. p49-62.

5. Best, J.W. (1970) Research in Education, 2nd Edition, Australia: Englewood Cliffs, N.J. : Prentice Hall

6. Beye M., Jeckmans, A.J.P., Zekeriya, E., Tang, Q., Lagendijk, R.L. & Tang, Q., (2012) 'Privacy in Online Social Networks', In A. Abraham (Ed), *Computational Social Networks: Security and Privacy,* pp. 87-113. London: Springer

7. Bielenberg, A., Helm, L., Gentilucci, A., Stefanescu, D. & Zhang, H. (2012) "The growth of Diaspora - A decentralized online social network in the wild", *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS),* March. p13-18.

8. Blum, F. (1955) Action Research – a scientific approach? *Philosophy of Science*, 22, pp. 1-7.

9. Borkowsky, F.T. (1970) The relationship of work quality in undergraduate music curricula to effectiveness in instrumental music teaching in the public schools*. Journal of Experimental Education*, 39, pp. 14–19.

10. Boyd, D.M. & Ellison, N.B. (2007) 'Social Network Sites: Definition, History, and Scholarship', *Journal of Computer-Mediated Communication,* 13 (1), pp. 210-230.

11. Bradner, S*., Key words for use in RFCs to Indicate Requirement Levels,* BCP 14, RFC 2119, March 1997. Available at: http://www.ietf.org/rfc/rfc2119.txt (Accessed: 20 June 2016).

12. Braun, V., Clarke, V. (2013) *Successful Qualitative Research: A Practical Guide for Beginners*. London: Sage Publications

13. Buchegger, S. & Datta, A. (2009) "A Case for P2P Infrastructure for Social Networks - Opportunities & Challenges", *Proceedings of the Sixth International Conference on Wireless on-Demand Network Systems and Services,* Snowbird, Utah, USA Piscataway, NJ, USA: IEEE Press. p149-156.

14. Buchegger, S., Schioberg, D., Vu, L. & Datta, A. (2009) "PeerSoN: P2P Social Networking: Early Experiences and Insights", *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems,* Nuremberg, Germany New York, NY, USA: ACM. p46-52.

15. Campbell, D.T., Stanley, J.C. (1996) *Experimental and Quasi-experimental Designs for Research*. USA: Houghton Mifflin Company

16. Cohen, L., Manion, L., Morrison, K. (2000) Research Methods in Education, 5th Edition, London: RoutkedgeFalmer

17. Corbin, J.M., Strauss, A. (2008), *Basics of Qualitative Research Techniques and Procedures for Developing Grounded Theory.* 3rd Edition, California: Sage Publications

18. Creswell J.W. (2013) *Research Design: Qualitative, Quantitative and Mixed Methods Approaches,* 4th Edition, California: Sage Publications, Inc.

19. Cutillo, L.A., Molva, R. & Strufe, T. (2009) 'Safebook: A privacy-preserving online social network leveraging on real-life trust', *IEEE Communications Magazine,* 47 (12), pp.94-101.

20. Cutillo, L.A., Molva, R. & Önen, M. (2011) "Safebook: A distributed privacy preserving Online Social Network", *IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM),* June. p1-3.

21. Datta, A., Buchegger, S., Vu, L., Strufe, T., Rzadca, K. (2010). 'Decentralized Online Social Networks' in Furht, B. (ed) *Handbook of Social Network Technologies and Applications.* Springer: US. Pp.349-378.

22. Denzin, N. (1978), The Research Act: A Theoretical Introduction to Sociological Methods, New York: McGraw-Hill

23. Doub, B. (2016), *Community Memory: Precedents in Social Media and Movements.* Available at: http://goo.gl/vnKUop (Accessed: 5 April 2016).

24. Dwyer, C. (2011) 'Privacy in the Age of Google and Facebook', *IEEE Technology and Society Magazine,* 30 (3), pp.58-63.

25. Facebook. (2016), *Company Info.* Available at: http://newsroom.fb.com/company-info/ (Accessed: 10 April 2016).

26. Falahi, K.A., Atif, Y. & Elnaffar, S. (2010) "Social Networks: Challenges and New Opportunities", *Green Computing and Communications (GreenCom), 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing (CPSCom),* p804-808.

27. Fu, S., He, L., Liao, X., Huang, C., Li, K., Chang, C. & Gao, B. (2014) "Modelling and Predicting the Data Availability in Decentralized Online Social Networks", *Web Services (ICWS), 2014 IEEE International Conference on,* June. p161-168.

28. Fu, S., He, L., Liao, X., Huang, C., Li, K., Chang, C. & Gao, B. (2014) "Cadros: The Cloud-Assisted Data Replication in Decentralized Online Social Networks", *Services Computing (SCC), 2014 IEEE International Conference on,* June. p43-50.

29. Gall, M.D., Gall, J.P., Borg, W.R. (2006) *Educational Research: An Introduction,* 8th Edition, Pearson.

30. Gilbert, E., and Karahalios, K., (2009), 'Predicting tie strength with social media', *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2009,* ACM, Boston: MA (USA), p211-220

31. Glaser, B. G., and Strauss, A. (1967) *The discovery of Grounded Theory: Strategies for qualitative research*. Chicago: Aldine

32. Gratton, C., Jones, I. (2003) *Research Methods for Sports Studies*, London: Routledge

33. Greschbach, B., Kreitz, G. & Buchegger, S. (2012) "The devil is in the metadata - New privacy challenges in Decentralised Online Social Networks", *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),* March. p333-339.

34. Gracia-Tinedo, R., S'nchez-Artigas, M. & García-López, P. (2012) "F2Box: Cloudifying F2F Storage Systems with High Availability Correlation", *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on,* June. p123-130.

35. Gyarmati, L. & Trinh, T.A. (2010) "Measuring User Behavior in Online Social Networks", *Network Magazine of Global Internet working.,* 24 (5), pp.26-3

36. Hales, D. (2004) "From selfish nodes to cooperative networks - emergent link-based incentives in peer-to-peer networks", *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on,* Aug. p151-158.

37. Hopkins, H.W. (2000) Qualitative Research Design. *Sportsscience*, 4 (1) pp. 12-21.

38. Hui, P., Crowcroft, J. & Yoneki, E. (2011) 'BUBBLE Rap: Social-Based Forwarding in Delay-Tolerant Networks', *IEEE Transactions on Mobile Computing,* 10 (11), pp.1576-1589.

39. Jahid, S., Nilizadeh, S., Mittal, P., Borisov, N. & Kapadia, A. (2012) "DECENT: A decentralized architecture for enforcing privacy in online social networks", *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),* March. p326-332.

40. Jansen, W.A. (2011) "Cloud Hooks: Security and Privacy Issues in Cloud Computing", *44th Hawaii International Conference on System Sciences (HICSS),* Jan. p1-10.

41. Junco, R. (2012) "Too Much Face and Not Enough Books: The Relationship Between Multiple Indices of Facebook Use and Academic Performance", *Computer Human Behaviour.,* 28 (1), pp.187-198

42. Kerlinger, F.N. (1970) *Foundations of Behavioural Research*. New York: Holt, Rinehart & Winston

43. Koll, D., Li, J. & Fu, X. (2014) "SOUP: An Online Social Network by the People, for the People", *Proceedings of the 15th International Middleware Conference,* Bordeaux, France New York, NY, USA: ACM. p193-204.

44. Krishnamurthy, B. & Wills, C.E. (2009) "On the Leakage of Personally Identifiable Information via Online Social Networks", *Proceedings of the 2Nd ACM Workshop on Online Social Networks,* Barcelona, Spain. New York, NY, USA: ACM. p7-12.

45. Kuhne, G., And Quigley, B. A. (1997) Understanding and using action research in practice setting. *New Directions for Adult & Continuing Education*, 1997(73) pp. 23-40

46. Lam, I., Chen, K. & Chen, L. (2008) "Involuntary Information Leakage in Social Network Services", *Proceedings of the 3rd International Workshop on Security: Advances in Information and Computer Security,* Kagawa, Japan Berlin, Heidelberg: Springer-Verlag. p167-183.

47. Li, J. & Dabek, F. (2006) *'F2F: Reliable storage in open networks',* in *IPTPS Proceedings of the 5th international workshop on peer-to-peer systems*.

48. LinkedIn. 2012. *An Update on LinkedIn Member Passwords Compromised*. [Online] Available at: https://goo.gl/1hSaJN (Accessed: 30 May 2016).

49. Liu, D., Shakimov, A., C'aceres, R., Varshavsky, A. & Cox, L.P. (2011) "Confidant: Protecting OSN Data Without Locking It Up", *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware,* Lisbon, Portugal Berlin, Heidelberg: Springer-Verlag. p61-80.

50. Malin, B. (2005) 'Betrayed by my shadow: learning data identity via trail matching', *Journal of Privacy Technology,* 2005 pp.20050609001.

51. Manen, J. V. (1979) Reclaiming qualitative methods for organisational research: A preface *Adminstrative Science Quarterly* 24(4), 520-526

52. March, S., Smith, G. (1995). Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4), pp. 251-266

53. Merriam, B.S. (2009), *Qualitative research: A Guide to Design and Implementation*, California: Jossey-Bass.

54. Morrison, K. (1994) Planning and Accomplishing School-Centred Evaluation. *Bristish Journal of Educational Studies,* 42(4), pp. 417-419

55. Muangngeon, A. & Erjongmanee, S. (2015) "Analysis of facebook activity usage through network and human perspectives", *Knowledge and Smart Technology (KST), 2015 7th International Conference on,* Jan. p13-18.

56. Narendula, R., Papaioannou, T.G. & Aberer, K. (2012) "Towards the Realization of Decentralized Online Social Networks: An Empirical Study", *2012 32nd International Conference on Distributed Computing Systems Workshops,* June. p155-162.

57. Nilizadeh, S., Jahid, S., Mittal, P., Borisov, N. & Kapadia, A. (2012) "Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching", *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies,* Nice, France New York, NY, USA: ACM. p337-348.

58. Olteanu, A. & Pierre, G. (2012) "Towards Robust and Scalable Peer-to-peer Social Networks", *Proceedings of the Fifth Workshop on Social Network Systems,* Bern, Switzerland New York, NY, USA: ACM. p10:1-10:6.

59. Owen, C.L (1997) Understanding Design Research. Towards an Achievement of Balance, *Journal of the Japanese Society for the Science of Design*, 5(2), pp. 36-45

60. Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research, Journal of Management Information Systems, 24(3), pp. 45-77.

61. Punch, F. (1998). I*ntroduction to Social Research: Quantitative and Qualitative Approaches*. London: Sage Publications

62. Shahriar, N., Chowdhury, S.R., Sharmin, M., Ahmed, R., Boutaba, R. & Mathieu, B. (2013) "Ensuring Beta-Availability in P2P Social Networks", *IEEE 33rd International Conference on Distributed Computing Systems Workshops,* July. p150-155.

63. Sharma, R. & Datta, A. (2012) "SuperNova: Super-peers based architecture for decentralized online social networks", *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012),* Jan. p1-10.

64. Shakimov, A., Varshavsky, A., Cox, L.P. & C'aceres, R. (2009) "Privacy, Cost, and Availability Tradeoffs in Decentralized OSNs", *Proceedings of the 2Nd ACM Workshop on Online Social Networks,* Barcelona, Spain New York, NY, USA: ACM. p13-18.

65. Shakimov, A., Lim, H., Cáceres, R., Cox, L.P., Li, K., Liu, D. & Varshavsky, A. (2011) "Vis-a-Vis: Privacy-preserving online social networking via Virtual Individual Servers", *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011),* Jan. p1-10.

66. Sherchan, W., Nepal, S. & Paris, C. (2013) 'A Survey of Trust in Social Networks', *ACM Computer Survey,* 45 (4), pp.47:1-47:33.

67. Smith L. M. (1978). An evolving logic of participant observation, educational ethnography, and other case studies. *Review of research in education*, 6, pp. 316 -377.

68. Sommerville, I. (2015) *Software Engineering,* 10th ed. Edinburgh: Pearson

69. Spector, P.E. (1993) Research designs. In M.L. LewisBeck (ed.) Experimental Design and Methods. *International Handbook of Quantitative Applications in the Social Sciences*, 3, London: Sage Publications, pp. 1–74.

70. Stake, R. E. (2005) Qualitative case studies. *The Sage Handook of qualitative research*, 3rd Edition, pp. 443-466. Thousands Oaks, California: Sage

71. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. & Balakrishnan, H. (2001) "Chord: A scalable peer-to-peer lookup service for internet applications", *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,* San Diego, California, USA New York, NY, USA: ACM. p149-160.

72. Sun, Y., Liu, F., Li, B. & Zhang, X. (2009) "FS2You: Peer-Assisted Semi-Persistent Online Storage at a Large Scale", *IEEE INFOCOM 2009,* April. p873-881.

73. Susman, G., Evered, R (1978) An assessment of the scientific merits of action research, *Administrative Science Quarterly* 23, pp. 582-603

74. Tatjana, T., Elena, Aristodemou, E., Georgina, S., Yiannis, L. & Aysu, A. (2010) 'Disclosure of personal and contact information by young people in social networking sites: An analysis using Facebook profiles as an example', *International Journal of Media & Cultural Politics,* 6 (1), pp.81-101.

75. Tegeler, F., Koll, D. & Fu, X. (2011) "Gemstone: Empowering Decentralized Social Networking with High Data Availability", *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE,* Dec. p1-6.

76. Timm, C., Perez, R., (2010). *Seven Deadliest Social Network Attacks.* Syngress

77. Varga, A. & Hornig, Rudolf. 2008, 'An overview of the OMNET++ simulation environment' *Simutools'08 Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops,* ACM, Marseille, France

78. Vorakulpipat, C., Marks, A., Rezgui, Y. & Siwamogsatham, S. (2011) "Security and privacy issues in Social Networking sites from user's viewpoint", *2011 Proceedings of PICMET '11: Technology Management in the Energy Smart World (PICMET),* July. p1-4.

79. Viswanath, B., Post, A., Gummadi, K.P. & Mislove, A. (2010) "An Analysis of Social Network-based Sybil Defenses", *Proceedings of the ACM SIGCOMM 2010 Conference,* New Delhi, India New York, NY, USA: ACM. p363-374.

80. Wang, Z. (2011) "Security and Privacy Issues within the Cloud Computing", *International Conference on Computational and Information Sciences (ICCIS),* Oct. p175-178.

81. Yeung, C.A., Liccardi, I., Lu, K., Seneviratne, O. & Berners-lee, T. (2009) "Decentralization: The future of online social networking", *In W3C Workshop on the Future of Social Networking Position Papers*.

82. Yin, R. K. (2014) Case Study research: Design and methods, 5th Edition. Thousands Oaks, California: Sage Publications

83. Zamzami, I.F., Olowolayemo, A., Bakare, K.K. & Kindy, D.A. (2010) "Sensitivity to online privacy in social networking sites", *International Conference on Information and Communication Technology for the Muslim World (ICT4M),* pB-21-B-26.

84. Zhang, G., Yang, Y., Zhang, X., Liu, C. & Chen, J. (2012) "Key Research Issues for Privacy Protection and Preservation in Cloud Computing", *Second International Conference on Cloud and Green Computing (CGC),* Nov. p47-54.

Appendix 1 – Availability Algorithm

```
Input  : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ], sessionsLowerBound [ ][ ],
         sessionsUpperBound [ ][ ])
Output: Core node's diurnal profile availability in minutes via its mirror nodes
2) Core node's profile availability during each hour of the day

1  begin
2      node ← List of mirror nodes
3      mostAvailableMirrorNodeList ← List of highest to lowest available mirror nodes
4      replicaAvailabilityByHour ← Array of core node's availability in minutes during each hour of the day
5      onlineTimeList ← List of core node's availability in minutes in ascending order - This indicates how the core node's availability
         changes with chosen replicas
6      mostAvailableMirrorNode ← int value
7      duration ← new Duration(startTime, endTime)
8      n ← int value // this is a node counter and node ID
9      for i ← 0 to numberOfMirrorNodes do
10         Add new Node(n) object to node list
11         Increment n to +1
12     n ← 0
13     for x ← 0 to nodes.length do
14         for y ← 0 to nodes [x] do
15             for z ← 0 to sessionsDistribution[x].length do
16                 Set sessionsDuration [x][z] for node n
17                 Set sessionsLowerBound [x][z] for node n
18                 Set sessionsUpperBound [x][z] for node n
19                 for l ← 0 to sessionsDistribution[x][z] do
20                     Generate random start time t that is between node n's session lower and upper bounds
21                     Add start time t to node n's start time list
22                     Calculate the duration of node n's start time and end time and
23                     Assign it to a variable duration
24                     nodesAvailabilityInMinutes [n] ← nodesAvailabilityInMinutes [n] + duration in minutes
25             Sort node n's start time and end time list
26             Increment n to +1
27     Find the mirror nodes in descending order of their availability and
28     Assign it to variable mostAvailableMirrorNodeList
29     Compare the mostAvailableMirrorNode's online patterns with other mirror nodes in mostAvailableMirrorNodeList
30     Remove the overlapping times from the mirror nodes that overlap with the mostAvailableMirrorNode
31     Find the mirror nodes in descending order of their availability i.e. from highest to lowest available nodes and
32     Assign it to variable mostAvailableMirrorNodeList
33     Add the mostAvailableMirrorNode's total online time (in minutes) to onlineTimeList
34     if secondMostAvailableMirrorNode ≠ mostAvailableMirrorNode then
35         Add the online times of secondMostAvailableMirrorNode to the previously selected mostAvailableMirrorNode's online time list
36     Repeat steps 29 to 35 till the mostAvailableMirrorNode and the secondMostAvailableMirrorNode are the same
37     Create variables onlineTimeInMinutes and offlineTimeInMinutes
38     Assign the last entry in onlineTimeList to onlineTimeInMinutes
39     Subtract onlineTimeInMinutes from 1440 and
40     Assign it variable offlineTimeInMinutes
41     for i ← 0 to mostAvailableMirrorNode's online time list size do
42         hour ← node (mostAvailableMirrorNode).getStartTime().get(i).getHourOfDay()
43         duration ← duration of mostAvailableMirrorNode's online time instance during hour: hour
44         replicaAvailabilityByHour [hour ] ← nodesReceivedUpdatedReplicaByHour [hour ] + duration
45     for i ← 0 to number of entries in replicaAvailabilityByHour do
46         Print replicaAvailabilityByHour[i] that indicates core node's profile availability during each hour of the day
47     Print the value of onlineTimeInMinutes and offlineTimeInMinutes that indicates core node's diurnal profile availability via its
         mirror nodes
       // Steps 29 to 35 are described in more detail in ReplicationDegree Algorithm
```

Appendix 2 – Update Propagation Delay Algorithm

```
Input  : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ], sessionsLowerBound [ ][ ],
         sessionsUpperBound [ ][ ])
Output: 1) Number of mirror nodes that received the core node's updated profile/replica
2) Number of mirror nodes that received the core node's updated profile/replica by hour
3) Time stamps during which the mirror nodes received core node's update profile/replica

Part 1:
begin
    node ← List of mirror nodes
    nodesAvailabilityInMinutes ← Array of mirror node's availability in minutes
    highestAvailableNodesList ← List of highest to lowest available mirror nodes
    nodesReceivedUpdatedReplica ← List of mirror nodes that received the core node's updated replica
    replicaAvailabilityTimelineStartTime ← List of time stamps ('time from') when the core's updated replica is available
    replicaAvailabilityTimelineEndTime ← List of time stamps ('time to') when the core's updated replica is available
    nodesReceivedUpdatedReplicaByHour ← Array - number of mirror nodes receiving updated replica by hour
    numberOfMirrorNodesReceivedReplica ← int value
    highestAvailableNode ← int value
    duration ← new Duration(startTime, endTime)
    n ← int value // this is a node counter and node ID
    for i ← 0 to numberOfMirrorNodes do
        Add new Node(n) object to node list
        Increment n to +1
    n ← 0
    for x ← 0 to nodes.length do
        for y ← 0 to nodes [x] do
            for z ← 0 to sessionsDistribution[x].length do
                Set sessionsDuration [x][z] for node n
                Set sessionsLowerBound [x][z] for node n
                Set sessionsUpperBound [x][z] for node n
                for l ← 0 to sessionsDistribution[x][z] do
                    Generate random start time t that is between node n's session lower and upper bounds
                    Add start time t to node n's start time list
                    Calculate the duration between node n's start time and end time and
                    Assign it to a variable duration
                    nodesAvailabilityInMinutes [n] ← nodesAvailabilityInMinutes [n] + duration in minutes
            Sort node n's start time and end time list
            Increment n to +1
```

```
1  Part 2:
2  begin
3      Find the ids of highest to lowest available mirror nodes from nodesAvailabilityInMinutes array
4      Add the ids of highest to lowest available mirror nodes to highestAvailableNodesList
5      highestAvailableNode ← highestAvailableNodesList.get(0)
6      Add the highestAvailableNode's start times and end times list to replicaAvailabilityTimelineStartTime and
       replicaAvailabilityTimelineEndTime respectively
7      Set the highestAvailableNode's received core node's updated replica flag property to true
8      Set the highestAvailableNode's received core node's updated replica time stamp property to highestAvailableNodesList.get(0)'s
       first instance of start time i.e. highestAvailableNodesList.get(0).getStartTime(0)
9      Add the highestAvailableNode to nodesReceivedUpdatedReplica list
10     for i ← 0 to replicaAvailabilityTimelineStartTime size do
11         Sort replicaAvailabilityTimelineStartTime
12         Sort replicaAvailabilityTimelineEndTime
13         startTime ← replicaAvailabilityTimelineStartTime.get(i)
14         endTime ← replicaAvailabilityTimelineEndTime.get(i)
15         for j ← 0 to numberOfMirrorNodes do
16             if j does not exist in nodesReceivedUpdatedReplica list then
17                 for k ← 0 to node j's start time list size do
18                     Find interval i1 for startTime and endTime
19                     Find interval i2 for node j's kth instance of startTime and node j's kth instance of endTime
20                     if i1 overlaps with i2 then
21                         Find the overlapping time between the intervals
22                         if the overlapping time is greater than or equal to the startTime then
23                             Remove the overlapping time from the jth node and
24                             Add the remaining non-overlapping time of jth node (if any) to replicaAvailabilityTimelineStartTime
                               and replicaAvailabilityTimelineEndTime
25                             Add node j to nodesReceivedUpdatedReplica list
26                             if node j's received updated replica received time stamp is equal to null then
27                                 Set node j's received updated replica received time stamp to the overlapping time
28                             else if the overlapping time is less than node j's updated replica received time stamp
                               then
29                                 Update node j's received updated replica received time stamp to the overlapping time
30                         if the overlapping time is greater than the node j's start time and node j's end time is less
                           than or equal to the endTime then
31                             Add node j to nodesReceivedUpdatedReplica list
32                             if node j's received updated replica received time stamp is equal to null then
33                                 Set node j's received updated replica received time stamp to the overlapping time
34                             else if the overlapping time is less than node j's updated replica received time stamp
                               then
35                                 Update node j's received updated replica received time stamp to the overlapping time
36                         if the overlapping time is greater than the node j's start time and node j's end time is
                           greater than the endTime then
37                             Remove the overlapping time from the jth node and
38                             Add the non-overlapping time of jth node that is greater than the endTime to
                               replicaAvailabilityTimelineStartTime and replicaAvailabilityTimelineEndTime
39                             Add node j to nodesReceivedUpdatedReplica list
40                             if node j's received updated replica received time stamp is equal to null then
41                                 Set node j's received updated replica received time stamp to the overlapping time
42                             else if the overlapping time is less than node j's updated replica received time stamp
                               then
43                                 Update node j's received updated replica received time stamp to the overlapping time
44                         for l ← k + 1 to node j's start time list size do
45                             Add lth instance of node j's start time to replicaAvailabilityTimelineStartTime list
46                             Add lth instance of node j's end time to replicaAvailabilityTimelineEndTime list
47     for i ← 0 to numberOfMirrorNodes do
48         if ith node's received updated replica time stamp is not equal to null then
49             numberOfMirrorNodesReceivedReplica ← numberOfMirrorNodesReceivedReplica + 1
50             nodesReceivedUpdatedReplicaByHour [hour of node i's received updated replica time stamp] ←
                 nodesReceivedUpdatedReplicaByHour [hour of node i's received updated replica time stamp] + 1
51             Print node i's received updated replica time stamp
52     Print number of mirror nodes received updated replica i.e. numberOfMirrorNodesReceivedReplica
53     Print number of mirror nodes that couldn't receive updated replica i.e. numberOfMirrorNodes −
       numberOfMirrorNodesReceivedReplica
54     for i ← 0 to number of entries in nodesReceivedUpdatedReplicaByHour do
55         if nodesReceivedUpdatedReplicaByHour [i] is not equal to 0 then
56             Print Hour i and number of mirror nodes received updated replica by hour i.e. nodesReceivedUpdatedReplicaByHour[i]
```

Appendix 3 – Replication Degree Algorithm

```
Input  : model (numberOfMirrorNodes, nodes [ ], sessionsDistribution [ ][ ], sessionsDuration [ ][ ], sessionsLowerBound [ ][ ],
         sessionsUpperBound [ ][ ])
Output: 1) IDs' of mirror nodes (in order) that can achieve the same availability as all mirror nodes combined
2) Minimum number of replicas required to achieve the same availability as all mirror nodes combined

Part 1:
begin
    node ← List of mirror nodes
    nodesAvailabilityInMinutes ← Array of mirror node's availability in minutes
    highestAvailableNodesList ← List of highest to lowest available mirror nodes
    chosenReplicaList ← ID's of nodes chosen as mirror nodes
    onlineTimeList ← Entries in online time list indicate how the core node's availability changes on choosing different mirror nodes
    minimumNumberOfReplicasRequired ← int value
    highestAvailableNode ← int value
    duration ← new Duration(startTime, endTime)
    n ← int value // this is a node counter and node ID
    for i ← 0 to numberOfMirrorNodes do
        Add new Node(n) object to node list
        Increment n to +1
    n ← 0
    for x ← 0 to nodes.length do
        for y ← 0 to nodes [x] do
            for z ← 0 to sessionsDistribution[x].length do
                Set sessionsDuration [x][z] for node n
                Set sessionsLowerBound [x][z] for node n
                Set sessionsUpperBound [x][z] for node n
                for l ← 0 to sessionsDistribution[x][z] do
                    Generate random start time t that is between node n's session lower and upper bounds
                    Add start time t to node n's start time list
                    Calculate the duration of node n's start time and end time and
                    Assign it to a variable duration
                    nodesAvailabilityInMinutes [n] ← nodesAvailabilityInMinutes [n] + duration in minutes
            Sort node n's start time and end time list
            Increment n to +1
```

```
Part 2:
begin
    Find the ids of highest to lowest available mirror nodes from nodesAvailabilityInMinutes array
    Add the ids of highest to lowest available mirror nodes to highestAvailableNodesList
    Create a variable minimumNumberOfReplicasRequired and set the value to 1
    Create a variable chosenReplicaList
    Create a variable onlineTimeList
    Add the highestAvailableNode from highestAvailableNodesList to chosenReplicaList
    Create an array mirrorNodesAvailabilityInMinutes that keeps track of each mirror node's availability in minutes
    do
        mirrorNodesAvailabilityInMinutes ← RemoveOverlapingTimesAndFindHighestAvailableMirrorNodes (highestAvailableNode,
          highestAvailableNodesList)
        highestAvailableNodesList ← Sort mirrorNodesAvailabilityInMinutes and return IDs' of most to least available mirror nodes
        Add the 'availability in minutes' of highestAvailableNode to onlineTimeList i.e. onlineTimeList ←
        mirrorNodesAvailabilityInMinutes [highestAvailableNode ]
        if secondMostAvailableMirrorNode ≠ highestAvailableNode then
            Add secondMostAvailableMirrorNode to chosenReplicaList
            Increment minimumNumberOfReplicasRequired to +1
            for i ← 0 to secondMostAvailableMirrorNode's start time list size do
                if secondMostAvailableMirrorNode's start time ≠ secondMostAvailableMirrorNode's end time then
                    Add the secondMostAvailableMirrorNode's start and end times list to highestAvailableNode's start and end
                    times list
    while (secondMostAvailableMirrorNode ≠ highestAvailableNode)
    Print the elements in chosenReplicaList i.e. IDs' of mirror nodes (in order) that can achieve the same availability as all mirror
    nodes combined
    Print the entries in onlineTimeList that indicate how the core node's availability changes on choosing different mirror nodes from
    chosenReplicaList
    Print the value of minimumNumberOfReplicasRequired that can achieve the same availability as all mirror nodes combined
```

```
1  Part 3:
2  begin
3      int[ ] RemoveOverlapingTimesAndFindHighestAvailableMirrorNodes (highestAvailableNode, highestAvailableNodesList)
4          Create a variable secondMostAvailableMirrorNode and
5          Assign with a value of second node in highestAvailableNodesList
6          Create a variable nodeCounter ← 1
7          do
8              if secondMostAvailableMirrorNode ≠ highestAvailableNode then
9                  for i ← 0 to highestAvailableNode's start time list size do
10                     for j ← 0 to secondMostAvailableMirrorNode's start time list size do
11                         Find highestAvailableNode's interval i1 for ith index of highestAvailableNode's start time and ith index
                           of highestAvailableNode's end time
12                         Find secondMostAvailableMirrorNode's interval i2 for jth index of secondMostAvailableMirrorNode's
                           start time and jth index of secondMostAvailableMirrorNode's end time
13                         if i1 overlaps i2 then
14                             Remove overlapping time from secondMostAvailableMirrorNode
15                         Sort secondMostAvailableMirrorNode start time and end time lists
16                 Increment nodeCounter to +1
17                 if nodeCounter < numberOfMirrorNodes then
18                     secondMostAvailableMirrorNode ← highestAvailableNodesList.get(nodeCounter)
19         while (nodeCounter < numberOfMirrorNodes)
20         Create an array mirrorNodesAvailabilityInMinutes
21         for i ← 0 to numberOfMirrorNodes do
22             for j ← 0 to node i's start time list size do
23                 Find the interval between node i's jth index of start time list and node i's jth index of end time list
24                 Find the duration of the interval in minutes and
25                 Add it to ith index of mirrorNodesAvailabilityInMinutes array
26         return mirrorNodesAvailabilityInMinutes
```

Appendix 4 – Availability on Demand Algorithm

```
Input  : model (numberOfDays, numberOfMirrorNodes, nodes [ ], numberOfSessions [ ][ ], sessionDuration [ ][ ], lowerBound [ ][ ],
         upperBound [ ][ ])
Output: 1)Average daily availability of each of the mirror node
2) Average hourly availability of each of the mirror node and the hour during which mirror node's availability is maximum

1  begin
2      node ← List of mirror nodes
3      availability ← Two dimensional array to store mirror node's availability during each hour for given number of days duration ← new
         Duration(startTime, endTime)
4      for i ← 0 to numberOfMirrorNodes do
5          Add new Node(nodeCounter) object to node list
6          Increment nodeCounter to +1
7      nodeCounter ← 0
8      for n ← 0 to nodes.length do
9          for i ← 0 to nodes [x] do
10             for day ← 1 to ≤ numberOfDays do
11                 Set numberOfSessions[n][day − 1] for node[nodeCounter]
12                 Set sessionDuration[n][day − 1] for node[nodeCounter]
13                 Set lowerBound[n][day − 1] for node[nodeCounter]
14                 Set upperBound[n][day − 1] for node[nodeCounter]
15                 for j ← 0 to node[nodeCounter]'s numberOfSessions do
16                     Generate random start time t that is between node [nodeCounter ]'s session lower and upper bounds
17                     Add start time t to node[nodeCounter]'s start time list for given day
18                 Sort node [nodeCounter ]'s start time and end time list
19             Increment nodeCounter to +1
20     for n ← 0 to numberOfMirrorNodes do
21         availability[][] ← new int[24][numberOfDays ]
22         for i ← 0 to node n's start time list size do
23             duration ← newDuration(ith index of node n's start time, ith index of node n's end time)
24             availability [node n's availability hour][node n's availability day - 1] ← availability [node n's availability hour][node n's
                 availability day - 1] + duration in minutes
25         averageDailyAvailability, totalAvailability, maximumAvailability, maximumAvailabilityHour ← 0
26         for i ← 0 to availability.length do
27             for j ← 0 to numberOfDays + 1 do
28                 if j == numberOfDays − 1 then
29                     Print Node n's average hourly availability is: totalAvailability/numberOfDays
30                     averageDailyAvailability ← averageDailyAvailability + (totalAvailability/numberOfDays)
31                     if totalAvailability/numberOfDays > maximumAvailability then
32                         maximumAvailability ← totalAvailability/numberOfDays
33                         maximumAvailabilityHour ← i
34                 else
35                     totalAvailability ← totalAvailability + availability[i][j]
36         Print Node n's average daily availability is: averageDailyAvailability
37         Print Node n's maximum availability i.e. maximumAvailability and maximum availability hour i.e. maximumAvailabilityHour
```

Appendix 5 – Code: Availability, Update Propagation Delay, Replication Degree

```
package com.adilhassan.mphil.availability;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.SplittableRandom;
import java.util.stream.DoubleStream;
import java.util.stream.IntStream;
import org.joda.time.DateTime;
import org.joda.time.Duration;
import org.joda.time.Interval;

public class App {

    private static int year = 2015;
    private static int day = 1;
    private static int month = 1;


        //Sample Model
        private static int numberOfNodes = 500;
        private static int[] nodes = {320, 80, 50, 30, 15, 5};
        private static int[][] sessionsDistribution = {{10, 5, 5}, {1, 2, 2, 10}, {10,
10, 5}, {20, 20, 5}, {25},{10, 10, 10, 10}};
        private static int[][] sessionsDuration = {{2, 3, 5}, {5, 5, 5, 5}, {5, 5, 5},
{5, 5, 10}, {20}, {30, 30, 15, 15}};
        private static int[][] sessionsLowerBound = {{0, 6, 18},{13, 15, 16, 17},{0, 0,
0},{0, 0, 0},{0},{0, 0, 0, 0}};
        private static int[][] sessionsUpperBound = {{3, 2, 2},{1, 1, 1, 2},{24, 24,
24},{24, 24, 24},{24},{24, 24, 24, 24}};



    private static int[] nodesAvailabilityInMinutes = new int[numberOfNodes];
    private static ArrayList<Node> node = new ArrayList<>(numberOfNodes);
    private static ArrayList<Node> duplicateNode = new ArrayList<>(numberOfNodes);
    private static SplittableRandom random = new SplittableRandom();

    public static void main(String[] args) throws FileNotFoundException
    {
        System.setOut(new PrintStream(new FileOutputStream("MNoRR1.txt")));

        int[] IndicesOfMostOverlappingNodes = IntStream.range(0, nodes.length)
                .boxed().sorted((i, j) -> nodes[i] - nodes[j])
                .mapToInt(ele -> ele).toArray();
        int onlineDuration = 0;
        for(int i =0; i < IndicesOfMostOverlappingNodes.length; i++)
        {
                for(int j=0;
j<sessionsDistribution[IndicesOfMostOverlappingNodes[i]].length; j++)
                {
                        onlineDuration = onlineDuration +
sessionsDistribution[IndicesOfMostOverlappingNodes[i]][j]*sessionsDuration[IndicesOfMo
stOverlappingNodes[i]][j];
                }
            System.out.println("["+nodes[IndicesOfMostOverlappingNodes[i]] + ", " +
onlineDuration+"," + "\"Number of replicas: " +
nodes[IndicesOfMostOverlappingNodes[i]]+ "\\nTime: "+ onlineDuration + " mins each\""
+ "],");
            onlineDuration = 0;
        }
        System.out.println();
        int nodeCounter = 0;

        for(int i=0; i<nodes.length; i++)
        {
            for(int j =0; j<nodes[i]; j++)
            {
                node.add(new Node(nodeCounter));
                duplicateNode.add(new Node(nodeCounter));
```

```
                nodeCounter++;
            }
        }

        int hour = 0;
        int minute = 0;
        int sessionNumber = 0;
        nodeCounter = 0;
        Duration duration = null;
        //Generating data from the Model
        for(int x=0; x<nodes.length; x++)
        {
            for(int y = 0; y<nodes[x]; y++)
            {
               for(int z=0; z<sessionsDistribution[x].length; z++)
                {

       node.get(nodeCounter).setEachSessionDuration(sessionsDuration[x][z]);

duplicateNode.get(nodeCounter).setEachSessionDuration(sessionsDuration[x][z]);

                    node.get(nodeCounter).setLowerBound(sessionsLowerBound[x][z]);
                    node.get(nodeCounter).setUpperBound(sessionsUpperBound[x][z]);


duplicateNode.get(nodeCounter).setLowerBound(sessionsLowerBound[x][z]);

duplicateNode.get(nodeCounter).setUpperBound(sessionsUpperBound[x][z]);

                    for(int l=0; l<sessionsDistribution[x][z]; l++)
                       {
                         hour = (random.nextInt(sessionsUpperBound[x][z]) +
sessionsLowerBound[x][z]);
                                minute = random.nextInt(60);

                       node.get(nodeCounter).setStartTime(year, month, day, hour,
minute, 0, 0);
                       duplicateNode.get(nodeCounter).setStartTime(year, month, day,
node.get(nodeCounter).getStartTime().get(sessionNumber).getHourOfDay(),
node.get(nodeCounter).getStartTime().get(sessionNumber).getMinuteOfHour(),
node.get(nodeCounter).getStartTime().get(sessionNumber).getSecondOfMinute(), 0);
                                duration = new
Duration(duplicateNode.get(nodeCounter).getStartTime().get(sessionNumber),duplicateNod
e.get(nodeCounter).getEndTime().get(sessionNumber));
                                nodesAvailabilityInMinutes[nodeCounter]  =
nodesAvailabilityInMinutes[nodeCounter]  + (int)duration.getStandardMinutes();
                                sessionNumber++;
                                }
                    }
                sessionNumber=0;

                Collections.sort(node.get(nodeCounter).getStartTime());
                Collections.sort(node.get(nodeCounter).getEndTime());

                Collections.sort(duplicateNode.get(nodeCounter).getStartTime());
                Collections.sort(duplicateNode.get(nodeCounter).getEndTime());

                nodeCounter++;
            }
        }

        //Finding most available nodes
        ArrayList<Integer> highestAvailableNodes = sort(nodesAvailabilityInMinutes);
        ArrayList<Integer> highestAvailableNodesDuplicate = new
ArrayList<Integer>(highestAvailableNodes.size());
        for(int i =0; i<highestAvailableNodes.size(); i++)
        {
              highestAvailableNodesDuplicate.add(highestAvailableNodes.get(i));
        }
        //Update propagation delay - Section 3.3.2 - Appendix 2
        findUpdatePropagationDelay(highestAvailableNodes, duplicateNode);

        for(int j = 0; j<node.size(); j++)
        {
            for(int i=0; i<node.get(j).getStartTime().size(); i++)
            {
```

```
                System.out.println("['Replica ID: " + j + "'," +"new Date(" +
node.get(j).getStartTime().get(i).getYear()+"," +
node.get(j).getStartTime().get(i).getMonthOfYear() +"," +
node.get(j).getStartTime().get(i).getDayOfMonth() + "," +
node.get(j).getStartTime().get(i).getHourOfDay()+","+
node.get(j).getStartTime().get(i).getMinuteOfHour() + ",0)," + "new Date(" +
node.get(j).getEndTime().get(i).getYear() +"," +
node.get(j).getEndTime().get(i).getMonthOfYear() +"," +
node.get(j).getEndTime().get(i).getDayOfMonth() + "," +
node.get(j).getEndTime().get(i).getHourOfDay()+","+
node.get(j).getEndTime().get(i).getMinuteOfHour() + ",0)],");

            }
        }

        //Replication degree - Section 3.3.3 - Appendix 3
        //Availability - Section 3.3.1 - Appendix 1
        ArrayList<Integer> sortedHighestAvailableNodes = new
ArrayList<Integer>(numberOfNodes);
        int minimumNumberOfreplicasRequired = 1;
        ArrayList<Integer> chosenReplicas = new  ArrayList<Integer>(numberOfNodes);
        ArrayList<Integer> onlineTime = new ArrayList<Integer>(numberOfNodes);
        chosenReplicas.add(highestAvailableNodes.get(0));
        nodesAvailabilityInMinutes = new int[numberOfNodes];
        do{
              nodesAvailabilityInMinutes =
removeOverlappingTimesAndFindHighestAvailableNodes(highestAvailableNodes.get(0),
highestAvailableNodes);
            sortedHighestAvailableNodes = sort(nodesAvailabilityInMinutes);
            highestAvailableNodes = sortedHighestAvailableNodes;


onlineTime.add(nodesAvailabilityInMinutes[sortedHighestAvailableNodes.get(0)]);

if(!sortedHighestAvailableNodes.get(1).equals(sortedHighestAvailableNodes.get(0)))
            {
               chosenReplicas.add(sortedHighestAvailableNodes.get(1));
               minimumNumberOfreplicasRequired++;
                for(int i=0;
i<node.get(sortedHighestAvailableNodes.get(1)).getStartTime().size(); i++)
                {

       if(!node.get(sortedHighestAvailableNodes.get(1)).getStartTime().get(i).equals(
node.get(sortedHighestAvailableNodes.get(1)).getEndTime().get(i)))
                    {

       node.get(sortedHighestAvailableNodes.get(0)).getStartTime().add(node.get(sorte
dHighestAvailableNodes.get(1)).getStartTime().get(i));

       node.get(sortedHighestAvailableNodes.get(0)).getEndTime().add(node.get(sortedH
ighestAvailableNodes.get(1)).getEndTime().get(i));
                    }
                }
            }

}while(!sortedHighestAvailableNodes.get(1).equals(sortedHighestAvailableNodes.get(0)))
;

        int replicaAvailabilityByhour[] = new int[24];
        int durationLength = 0;
        int startHour = 0;
        int endHour = 0;
        int startHourCounter = 0;
        Interval interval = null;
        for(int i = sortedHighestAvailableNodes.get(0);
i<(sortedHighestAvailableNodes.get(0)+1); i++)
        {
              for(int j=0; j < node.get(i).getStartTime().size(); j++)
              {

       if(!node.get(i).getStartTime().get(j).equals(node.get(i).getEndTime().get(j)))
                    System.out.println("['Core " + "node" + "'," + "new Date(" +
node.get(i).getStartTime().get(j).getYear() + "," +
node.get(i).getStartTime().get(j).getMonthOfYear() + "," +
node.get(i).getStartTime().get(j).getDayOfMonth() + "," +
node.get(i).getStartTime().get(j).getHourOfDay() + "," +
node.get(i).getStartTime().get(j).getMinuteOfHour() + ",0)," + "new Date(" +
```

```
node.get(i).getEndTime().get(j).getYear() + "," +
node.get(i).getEndTime().get(j).getMonthOfYear() + "," +
node.get(i).getEndTime().get(j).getDayOfMonth() + "," +
node.get(i).getEndTime().get(j).getHourOfDay() + "," +
node.get(i).getEndTime().get(j).getMinuteOfHour() + ",0)],");

       if(node.get(sortedHighestAvailableNodes.get(0)).getEndTime().get(j).getDayOfMo
nth()== (day+1)
&&node.get(sortedHighestAvailableNodes.get(0)).getEndTime().get(j).getHourOfDay()==0)
                {

       node.get(sortedHighestAvailableNodes.get(0)).getEndTime().set(j, new
DateTime(year, month, day, 23, 59, 59, 0));
                }
                }
        }

        for(int j = sortedHighestAvailableNodes.get(0);
j<(int)(sortedHighestAvailableNodes.get(0)+1); j++)
        {
               startHourCounter = 0;
            for(int i=0; i<node.get(j).getStartTime().size(); i++)
            {
               startHour = node.get(j).getStartTime().get(i).getHourOfDay();
                startHourCounter = startHour;
                endHour = node.get(j).getEndTime().get(i).getHourOfDay();
                for (int ju = 0; ju <= (endHour - startHour); ju++)
                {
                   if(ju == 0 && ((endHour - startHour)!=0))
                   {
                           interval = new Interval(node.get(j).getStartTime().get(i),
new DateTime(2015, 1, 1, (startHourCounter + 1), 0, 0, 0));
                           durationLength = (int)
interval.toDuration().getStandardMinutes();
                       replicaAvailabilityByhour[startHourCounter] =
replicaAvailabilityByhour[startHourCounter] + durationLength;
                       startHourCounter++;
                   }
                   else if(ju != (endHour - startHour))
                   {
                           interval = new Interval(new DateTime(2015, 1, 1,
(startHourCounter), 0, 0, 0), new DateTime(2015, 1, 1, (startHourCounter + 1), 0, 0,
0));
                           durationLength = (int)
interval.toDuration().getStandardMinutes();
                       replicaAvailabilityByhour[startHourCounter] =
replicaAvailabilityByhour[startHourCounter] + durationLength;
                       startHourCounter++;
                   }
                   else if(ju !=0 && ju == (endHour - startHour))
                   {
                           interval = new Interval(new DateTime(2015, 1, 1,
(startHourCounter), 0, 0, 0), node.get(j).getEndTime().get(i));
                           durationLength = (int)
interval.toDuration().getStandardMinutes();
                           if(node.get(j).getEndTime().get(i).getMinuteOfHour()==59)
                           {
                                   durationLength = durationLength + 1;
                           }
                       replicaAvailabilityByhour[startHourCounter] =
replicaAvailabilityByhour[startHourCounter] + durationLength;
                       startHourCounter++;
                   }
                   else  if((endHour - startHour)==0)
                   {
                           interval = new Interval(node.get(j).getStartTime().get(i),
node.get(j).getEndTime().get(i));
                           durationLength = (int)
interval.toDuration().getStandardMinutes();
                           if(node.get(j).getEndTime().get(i).getMinuteOfHour()==59 &&
(node.get(j).getEndTime().get(i).getSecondOfMinute()==59))
                           {
                                durationLength = durationLength + 1;
                                }

replicaAvailabilityByhour[node.get(j).getStartTime().get(i).getHourOfDay()] =
replicaAvailabilityByhour[node.get(j).getStartTime().get(i).getHourOfDay()] +
```

```
durationLength;
                    }
                }
            }
        }
        for(int h=0; h<replicaAvailabilityByhour.length; h++)
        {
               if(h<10)
               {
                System.out.println("[["+ h +", 0, 0]" + "," +
replicaAvailabilityByhour[h]+",\"Time: 0" + h +":00\\nMinutes Online: " +
replicaAvailabilityByhour[h] +"\"],");
               }
               else
               {
                System.out.println("[["+ h +", 0, 0]" + "," +
replicaAvailabilityByhour[h]+",\"Time: " + h +":00\\nMinutes Online: " +
replicaAvailabilityByhour[h] +"\"],");
               }
        }
        System.out.println("Minimum number of replicas required = " +
minimumNumberOfreplicasRequired + " to achieve the same availability as " +
numberOfNodes + " and the list is: " + chosenReplicas);

        ArrayList<Integer> nodesNotChosenAsReplicas = new
ArrayList<Integer>(numberOfNodes+2);
        for(int i=0; i<numberOfNodes; i++)
        {
            if(!chosenReplicas.contains(i))
            {
               nodesNotChosenAsReplicas.add(i);
            }
        }
        System.out.println();
        int counter = 0;
        for (int i = 0; i < numberOfNodes; i++) {
            if(i<onlineTime.size())
            {
               System.out.println("[\t" +  (i+1) + "\t,\t" + onlineTime.get(i) + "\t,"
+ "\""+ "Replica: " + (i+1) + "\\n"+ "Replica ID: " + chosenReplicas.get(i) +"\\n"+
"Time: " +onlineTime.get(i) +" mins"+ "\"" + ", null, " + "\""+ "Replica: " + (i+1) +
"\\n"+ "Replica ID: " + chosenReplicas.get(i)+ "\\n"+ "Time: " +onlineTime.get(i) +"
mins" + "\"" + "],");
            }
            else
            {
                System.out.println("[\t" + (i+1) + "\t,\t" +
onlineTime.get(onlineTime.size()-1)  + "\t," + "\""+ "Replica: " + (i+1) + "\\n"+
"Replica ID: " + nodesNotChosenAsReplicas.get(counter) + "\\n"+ "Time: " +
onlineTime.get(onlineTime.size()-1) +" mins" + "\"" + ", "
+onlineTime.get(onlineTime.size()-1) + ", " + "\""+ "Replica: " + (i+1) +
"\\n"+"Replica ID: " + nodesNotChosenAsReplicas.get(counter) + "\\n"+  "Time: "
+onlineTime.get(onlineTime.size()-1) +" mins" + "\"" + "],");
                counter ++;
            }
        }
        System.out.println();

        double onlineMinutes = onlineTime.get(onlineTime.size()-1);
        double offlineMinutes = 1440 - onlineMinutes;
        double Onlinehours = onlineMinutes/60;
        String onlineHoursAndMinutesString = "" + Onlinehours;
        int onlineDecimalIndex = onlineHoursAndMinutesString.indexOf(".");
        String onlineHourString = onlineHoursAndMinutesString.substring(0,
onlineDecimalIndex);
        String onlineMinuteString =
onlineHoursAndMinutesString.substring(onlineDecimalIndex+1, onlineDecimalIndex+2);
        double offlineHours = offlineMinutes/60;
        String offlineHoursAndMinutesString = "" + offlineHours;
        int offlineDecimalIndex = offlineHoursAndMinutesString.indexOf(".");
        String offlineHourString = offlineHoursAndMinutesString.substring(0,
offlineDecimalIndex);
        String offlineMinuteString =
offlineHoursAndMinutesString.substring(offlineDecimalIndex+1, offlineDecimalIndex+2);
        System.out.println("['Status',  'Time (hh:mm)'],\n"
                + "['Online',    " +
((onlineMinutes/60)>24?24:Integer.parseInt(onlineHourString)+"."+onlineMinuteString)+"
```

```
],\n"
                + "['Offline',  " +
Integer.parseInt(offlineHourString)+"."+offlineMinuteString+"]");
    }
    public static int[] removeOverlappingTimesAndFindHighestAvailableNodes(int
mostAvailableNode, ArrayList<Integer> nodes)
    {
        int nodeCounter = 1;
        int secondMostAvailableNode = nodes.get(nodeCounter);
        DateTime secondMostAvailableNodeEndTime = null;
        DateTime secondMostAvailableNodeStartTime = null;
        Interval mostAvailableNodeInterval = null;
        Interval secondMostAvailableNodeInterval = null;
        int[] nodesAvailabilityInMinutes = new int[numberOfNodes];
        Interval interval = null;

        Collections.sort(node.get(mostAvailableNode).getStartTime());
        Collections.sort(node.get(mostAvailableNode).getEndTime());

        do
        {
            if(secondMostAvailableNode!=mostAvailableNode)
            for(int i=0; i<node.get(mostAvailableNode).getStartTime().size(); i++)
            {
                for(int j=0;
j<node.get(secondMostAvailableNode).getStartTime().size(); j++)
                {
                    mostAvailableNodeInterval = new
Interval(node.get(mostAvailableNode).getStartTime().get(i),
node.get(mostAvailableNode).getEndTime().get(i));
                    secondMostAvailableNodeInterval = new
Interval(node.get(secondMostAvailableNode).getStartTime().get(j),
node.get(secondMostAvailableNode).getEndTime().get(j));

if(mostAvailableNodeInterval.overlaps(secondMostAvailableNodeInterval))
                    {

       if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(second
MostAvailableNode).getStartTime().get(j))>0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))>0)
                        {

       node.get(secondMostAvailableNode).getStartTime().set(j,
node.get(secondMostAvailableNode).getStartTime().get(j));

       node.get(secondMostAvailableNode).getEndTime().set(j,
node.get(mostAvailableNode).getStartTime().get(i));
                        }
                                else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))<0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))>0)
                                                {

       node.get(secondMostAvailableNode).getStartTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));

       node.get(secondMostAvailableNode).getEndTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));
                                                }
                                        else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))<0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))<0)
                                                {

       node.get(secondMostAvailableNode).getStartTime().set(j,
node.get(mostAvailableNode).getEndTime().get(i));

       node.get(secondMostAvailableNode).getEndTime().set(j,
node.get(secondMostAvailableNode).getEndTime().get(j));
                                                }
                                        else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
```

```
lableNode).getStartTime().get(j))==0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))>0)
                                                {

        node.get(secondMostAvailableNode).getStartTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));

        node.get(secondMostAvailableNode).getEndTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));
                                                }
                                            else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))==0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))<0)
                                                {

        node.get(secondMostAvailableNode).getStartTime().set(j,
node.get(mostAvailableNode).getEndTime().get(i));

        node.get(secondMostAvailableNode).getEndTime().set(j,
node.get(secondMostAvailableNode).getEndTime().get(j));

        Collections.sort(node.get(secondMostAvailableNode).getStartTime());

Collections.sort(node.get(secondMostAvailableNode).getEndTime());
                                                }
                                        else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))==0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))==0)
                                                {

        node.get(secondMostAvailableNode).getStartTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));

        node.get(secondMostAvailableNode).getEndTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));
                                                }
                                            else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))>0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))==0)
                                                {

        node.get(secondMostAvailableNode).getStartTime().set(j,
node.get(secondMostAvailableNode).getStartTime().get(j));

        node.get(secondMostAvailableNode).getEndTime().set(j,
node.get(mostAvailableNode).getStartTime().get(i));
                                                }
                                            else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))<0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))==0)
                                                {

        node.get(secondMostAvailableNode).getStartTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));

        node.get(secondMostAvailableNode).getEndTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));
                                                }
                                            else
if(node.get(mostAvailableNode).getStartTime().get(i).compareTo(node.get(secondMostAvai
lableNode).getStartTime().get(j))>0 &&
node.get(mostAvailableNode).getEndTime().get(i).compareTo(node.get(secondMostAvailable
Node).getEndTime().get(j))<0)
                                                {
                                                        secondMostAvailableNodeEndTime =
node.get(secondMostAvailableNode).getEndTime().get(j);
                                                        secondMostAvailableNodeStartTime
= node.get(secondMostAvailableNode).getStartTime().get(j);
```

```
        node.get(secondMostAvailableNode).getStartTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));

        node.get(secondMostAvailableNode).getEndTime().set(j, new DateTime(year,
month, day, 0, 0, 0, 0));


        node.get(secondMostAvailableNode).getStartTime().set(j,
secondMostAvailableNodeStartTime);

        node.get(secondMostAvailableNode).getEndTime().set(j,
node.get(mostAvailableNode).getStartTime().get(i));


        node.get(secondMostAvailableNode).getStartTime().add(node.get(mostAvailableNod
e).getEndTime().get(i));

        node.get(secondMostAvailableNode).getEndTime().add(secondMostAvailableNodeEndT
ime);
                                                }
                    }

Collections.sort(node.get(secondMostAvailableNode).getStartTime());
                    Collections.sort(node.get(secondMostAvailableNode).getEndTime());
                }
            }
            nodeCounter++;
            if(nodeCounter<nodes.size())
            {
                secondMostAvailableNode = nodes.get(nodeCounter);
            }
        }while(nodeCounter<nodes.size());

       for(int i=0; i<numberOfNodes;i++)
        {
            for(int j=0; j<node.get(i).getStartTime().size(); j++)
            {
                interval = new
Interval(node.get(i).getStartTime().get(j),node.get(i).getEndTime().get(j));
                nodesAvailabilityInMinutes[i]  = nodesAvailabilityInMinutes[i]  +
(int)interval.toDuration().getStandardMinutes();
            }
        }
        return nodesAvailabilityInMinutes;
    }

    public static ArrayList<Integer> sort(int[] nodesAvailabilityInMinutes)
    {
        ArrayList<Integer> sortedHighestAvailableNodes = new
ArrayList<Integer>(numberOfNodes);

        int nodeId = 0;
        for(int i=0; i<numberOfNodes; i++)
        {
            int large = 0;
            for(int j=0; j<numberOfNodes; j++)
            {
                if(nodesAvailabilityInMinutes[j]>large &&
!sortedHighestAvailableNodes.contains(j))
                {
                    large =nodesAvailabilityInMinutes[j];
                    nodeId = j;

                }
            }
            sortedHighestAvailableNodes.add(nodeId);
        }
        return sortedHighestAvailableNodes;
    }

    //Update propagation delay - Section 3.3.2 - Appendix 2
    public static void findUpdatePropagationDelay(ArrayList<Integer>
highestAvailableNodesDuplicate , ArrayList<Node> node)
    {
        ArrayList<DateTime> replicaAvailabilityTimeLineStartTime = null;
```

```
        ArrayList<DateTime> replicaAvailabilityTimeLineEndTime = null;
        Interval replicaAvailabilityTimeLineInterval = null;
        Interval nodeInterval = null;
        DateTime startTime = null;
        DateTime endTime = null;
        DateTime replicaReceivedStartTime = null;
        DateTime replicaReceivedEndTime = null;
        DateTime replicaReceivedTimeStamp = null;
        ArrayList<Integer> nodesReceivedUpdatedReplica = new
ArrayList<>(numberOfNodes);
        int numberOfNodesReceivedUpdatedReplica = 0;
        double[] nodesReceivedUpdatedReplicaByHour = new double[24];

        node.get(highestAvailableNodesDuplicate.get(0)).setUpdatedReplica(true);

node.get(highestAvailableNodesDuplicate.get(0)).setUpdatedReplicaReceivedTimeStamp(nod
e.get(highestAvailableNodesDuplicate.get(0)).getStartTime().get(0));

        System.out.println("['Time', ''],");
        replicaAvailabilityTimeLineStartTime =
node.get(highestAvailableNodesDuplicate.get(0)).getStartTime();
        replicaAvailabilityTimeLineEndTime =
node.get(highestAvailableNodesDuplicate.get(0)).getEndTime();
        Collections.sort(replicaAvailabilityTimeLineStartTime);
        Collections.sort(replicaAvailabilityTimeLineEndTime);
        nodesReceivedUpdatedReplica.add(highestAvailableNodesDuplicate.get(0));

        for(int i=0; i<replicaAvailabilityTimeLineStartTime.size(); i++)
        {
            Collections.sort(replicaAvailabilityTimeLineStartTime);
            Collections.sort(replicaAvailabilityTimeLineEndTime);

            startTime = replicaAvailabilityTimeLineStartTime.get(i);
            endTime = replicaAvailabilityTimeLineEndTime.get(i);

            for( int j=0; j<node.size(); j++)
            {

                if(!(nodesReceivedUpdatedReplica.contains(j)))
                    for(int k=0; k<node.get(j).getStartTime().size(); k++)
                    {
                        replicaAvailabilityTimeLineInterval = new Interval(startTime,
endTime);
                        nodeInterval = new Interval(node.get(j).getStartTime().get(k),
node.get(j).getEndTime().get(k));
                        if(replicaAvailabilityTimeLineInterval.overlaps(nodeInterval))
                        {

       if(node.get(j).getStartTime().get(k).compareTo(startTime)>0 &&
(node.get(j).getEndTime().get(k).compareTo(endTime)>0))
                                {
                                                                replicaReceivedStartTime =
endTime;
                                                                replicaReceivedEndTime =
node.get(j).getEndTime().get(k);


        replicaAvailabilityTimeLineStartTime.add(replicaReceivedStartTime);

        replicaAvailabilityTimeLineEndTime.add(replicaReceivedEndTime);


        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(node.get(j).getStartTime().get(k));


        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {
```

```
        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                }
                                                        else
if(node.get(j).getStartTime().get(k).compareTo(startTime)>0 &&
(node.get(j).getEndTime().get(k).compareTo(endTime)<0))
                                                                {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(node.get(j).getStartTime().get(k));


        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(node.get(j).getStartTime().get(k).compareTo(startTime)>0 &&
(node.get(j).getEndTime().get(k).compareTo(endTime)==0))
                                                                {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(node.get(j).getStartTime().get(k));

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(startTime.compareTo(node.get(j).getStartTime().get(k))==0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)<0)
                                                                {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(startTime);

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(startTime.compareTo(node.get(j).getStartTime().get(k))==0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)>0)
                                                                {
                                                                replicaReceivedStartTime =
endTime;
                                                                replicaReceivedEndTime =
```

```
node.get(j).getEndTime().get(k);

        replicaAvailabilityTimeLineStartTime.add(replicaReceivedStartTime);

        replicaAvailabilityTimeLineEndTime.add(replicaReceivedEndTime);


        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(node.get(j).getStartTime().get(k));

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(node.get(j).getStartTime().get(k).compareTo(startTime)==0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)==0)
                                                        {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(node.get(j).getStartTime().get(k));

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(node.get(j).getStartTime().get(k).compareTo(startTime)<0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)==0)
                                                        {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(startTime);

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
                                                else
if(node.get(j).getStartTime().get(k).compareTo(startTime)<0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)<0)
                                                        {

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(startTime);
```

```
        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }
        /* 7*/                                  else
if(node.get(j).getStartTime().get(k).compareTo(startTime)<0 &&
node.get(j).getEndTime().get(k).compareTo(endTime)>0)
                                                        {
                                                                replicaReceivedStartTime =
endTime;
                                                                replicaReceivedEndTime=
node.get(j).getEndTime().get(k);


        replicaAvailabilityTimeLineStartTime.add(replicaReceivedStartTime);

        replicaAvailabilityTimeLineEndTime.add(replicaReceivedEndTime);

        nodesReceivedUpdatedReplica.add(j);
                                                                replicaReceivedTimeStamp =
new DateTime(startTime);

        if(node.get(j).getUpdatedReplicaReceivedTimeStamp()==null)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                                else
if(replicaReceivedTimeStamp.compareTo(node.get(j).getUpdatedReplicaReceivedTimeStamp()
)<0)
                                                                {

        node.get(j).setUpdatedReplicaReceivedTimeStamp(replicaReceivedTimeStamp);
                                                                }
                                                        }

                            for(int l=(k+1); l<node.get(j).getStartTime().size(); l++)
                            {

        replicaAvailabilityTimeLineStartTime.add(node.get(j).getStartTime().get(l));

        replicaAvailabilityTimeLineEndTime.add(node.get(j).getEndTime().get(l));
                            }
                        }
                    }
                }
                }

        for(int i=0; i<node.size(); i++)
        {
            if(node.get(i).getUpdatedReplicaReceivedTimeStamp()!=null)
            {
                numberOfNodesReceivedUpdatedReplica++;
                System.out.println("[new Date(" + year + "," + month + "," + day + ","
+ node.get(i).getUpdatedReplicaReceivedTimeStamp().getHourOfDay()+","+
node.get(i).getUpdatedReplicaReceivedTimeStamp().getMinuteOfHour() + ",0),"+i+"],");

nodesReceivedUpdatedReplicaByHour[node.get(i).getUpdatedReplicaReceivedTimeStamp().get
HourOfDay()] =
nodesReceivedUpdatedReplicaByHour[node.get(i).getUpdatedReplicaReceivedTimeStamp().get
HourOfDay()] + 1;
            }
        }
        System.out.println(numberOfNodesReceivedUpdatedReplica + " nodes received core
node\'s replica.");
        System.out.println();
        double total = 0;
```

```
		for(int i=0; i<nodesReceivedUpdatedReplicaByHour.length; i++)
		{
			total = DoubleStream.of(nodesReceivedUpdatedReplicaByHour).sum();

			if(nodesReceivedUpdatedReplicaByHour[i]!=0)
				System.out.println("[\'Hour: \t"+ i +"\t\'" + ",\t" +
nodesReceivedUpdatedReplicaByHour[i]+"\t],");

		}
		if(total < numberOfNodes)
		{
				System.out.println("[\'Number of replicas unable to receive
core node\'s updated profile: " +"\'" + "," + (numberOfNodes-total)+"],");
		}
	}
}

package com.adilhassan.mphil.availability;

import java.util.ArrayList;
import java.util.Random;
import org.joda.time.DateTime;
import org.joda.time.Interval;

public class Node {

		private int id;
		private ArrayList<DateTime> startTime = null;
		private ArrayList<DateTime> endTime = null;
		long uniqueTime = 0;
		private int numberOfSessions;
		private int eachSessionDuration;
		private int lowerBound;
		private int upperBound;
		private DateTime dateTime;
		private boolean updatedReplica = false;
		private DateTime updatedReplicaReceivedTimeStamp;
		private Random random = new Random();
		private boolean geographicalDistribution;

		Node(int id)
		{
			this.id = id;
			startTime = new ArrayList<>(1440);
			endTime = new ArrayList<>(1440);

		}
		public void setLowerBound(int lowerBound)
		{
			this.lowerBound = lowerBound;
		}
		public int getLowerBound()
		{
			return this.lowerBound;
		}
		public void setUpperBound(int upperBound)
		{
			this.upperBound = upperBound;
		}
		public int getUpperBound()
		{
			return this.upperBound;
		}
		public void setGeographicalDistribution(boolean b)
		{
			this.geographicalDistribution = b;
		}
		public boolean getGeographicalDistribution()
		{
			return geographicalDistribution;
		}
		public int getId() {
			return id;
		}
		public void setId(int id) {
			this.id = id;
```

```
	}
	public boolean hasUpdatedReplica() {
		return updatedReplica;
	}
	public void setUpdatedReplica(boolean updatedReplica) {
		this.updatedReplica = updatedReplica;
	}
	public DateTime getUpdatedReplicaReceivedTimeStamp() {
		return updatedReplicaReceivedTimeStamp;
	}
	public void setUpdatedReplicaReceivedTimeStamp(DateTime
updatedReplicateReceivedTimeStamp) {
		this.updatedReplicaReceivedTimeStamp =
updatedReplicateReceivedTimeStamp;
	}
	public ArrayList<DateTime> getStartTime() {
		return startTime;
	}
	public int getNumberOfSessions() {
		return numberOfSessions;
	}
	public void setNumberOfSessions(int numberOfSessions) {
		this.numberOfSessions = numberOfSessions;
	}

	public int getEachSessionDuration() {
		return eachSessionDuration;
	}
	public void setEachSessionDuration(int eachSessionDuration) {
		this.eachSessionDuration = eachSessionDuration;
	}

	public void addUniqueTime(long time)
	{
		uniqueTime = uniqueTime + time;
	}
	public void setUniqueTime(long time)
	{
		uniqueTime = time;
	}
	public long getUniqueTime()
	{
		return uniqueTime;
	}
	public ArrayList<DateTime> getEndTime() {
		return endTime;
	}
	public void setStartTime(int year, int month, int date, int hour, int min, int
sec, int milliSecond) {
		dateTime = new DateTime(year, month, date, hour, min, sec,
milliSecond);
		boolean overlappingFlag = false;
		Interval i1 = new
Interval(dateTime,dateTime.plusMinutes(this.eachSessionDuration));
		for(int i =0; i<startTime.size(); i++)
		{
			Interval i2 = new Interval(startTime.get(i), endTime.get(i));
			if(i1.overlaps(i2))
			{
				overlappingFlag = true;
			}
		}
		if(!overlappingFlag)
		{
			if(!(this.startTime.contains(dateTime)))
			{
				this.startTime.add(dateTime);
				setEndTime(year, month, date, hour,
(min+this.eachSessionDuration), sec, milliSecond);
			}
		}
		else
		{
			int sh =  random.nextInt(this.upperBound) + this.lowerBound;
			setStartTime(year, month, date, sh, random.nextInt(60), 0, 0);
		}
	}
```

```
        public void setEndTime(int year, int month, int date, int hour, int min, int
sec, int milliSecond) {
                if(min<60){
                        dateTime = new DateTime(year, month, date, hour, min, sec,
milliSecond);
                }
                else
                {
                        int h = min/60;
                        min = min % 60;
                        hour = hour + h;
                        if(hour >= 24)
                        {
                                dateTime = new DateTime(year, month, (date+1), (0), 0,
sec, milliSecond);
                        }
                        else
                        {
                                dateTime = new DateTime(year, month, date, (hour), min,
sec, milliSecond);
                        }
                }
                this.endTime.add(dateTime);
        }
}
```

Appendix 6 – Code: Availability on Demand

```
package com.adilhassan.mphil.availability.history;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import org.joda.time.DateTime;
import org.joda.time.Duration;

public class App {

        //Availability on demand - Section 3.3.4 - Appendix 4
        private static int year = 2015;
        private static int month = 1;
        private static ArrayList<Node> node = new ArrayList<>();
        private static int numberOfDays = 3;
        private static int numberOfNodes = 6;
        private static int[] nodes = {3, 2, 1};
        private static int[][] numberOfSessions = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        private static int[][] sessionDuration = {{2, 4, 6}, {8, 10, 12}, {14, 16,
18}};
        private static int[][] lowerBound = {{0, 0, 0}, {4, 5, 6}, {7, 10, 13}};
//from - Time
        private static int[][] upperBound = {{24, 24, 24}, {4, 4, 4}, {5, 5, 5}};
//to - Time
        private static Random random = new Random();

        public static void main(String[] args) throws FileNotFoundException
        {
                System.setOut(new PrintStream(new
FileOutputStream("availabilityHistoryThisIsBinIt1001010.txt")));

         int nodeCounter = 0;
         for(int i=0; i<nodes.length; i++)
         {
             for(int j =0; j<nodes[i]; j++)
             {
                 node.add(new Node(nodeCounter));
                 nodeCounter++;
             }
         }
                generateDaysOfData();
```

```
	}

	public static void generateDaysOfData()
	{
		int counter = 0;
		for(int n=0; n<nodes.length; n++)
		{
			for(int i=0; i<nodes[n]; i++)
			{
				for(int day=1; day<=numberOfDays; day++)
				{

node.get(counter).setEachSessionDuration(sessionDuration[n][((day-1))]);

node.get(counter).setNumberOfSessions(numberOfSessions[n][((day-1))]);
                                node.get(counter).setLowerBound(lowerBound[n][((day-1))]);
                                node.get(counter).setUpperBound(upperBound[n][((day-1))]);

                                        for(int j=0;
j<node.get(counter).getNumberOfSessions(); j++)
                                        {
                                                node.get(counter).setStartTime(year,
month, day, (random.nextInt(upperBound[n][((day-1))])+lowerBound[n][((day-1))]),
random.nextInt(60), 0, 0);
                                        }

	Collections.sort(node.get(counter).getStartTime());

	Collections.sort(node.get(counter).getEndTime());
                                }
                                counter++;
                        }
                }

                for(int i=0; i<node.size(); i++)
                {
                        nodesOnlineTimeByHour(i);
                }
        }

        public static void nodesOnlineTimeByHour(int n)
        {
                int[][] availibility = new int[24][numberOfDays];
                System.out.println();
                int startYear, startMonth, startDay, startHour, startMinute,
startSecond = 0;
                int endYear, endMonth, endDay, endHour, endMinute, endSecond = 0;
                DateTime start, end, start2, end2 = null;
                Duration duration = null;

                for(int i=0; i<node.get(n).getStartTime().size(); i++)
                {
                        startYear = node.get(n).getStartTime().get(i).getYear();
                        startMonth =
node.get(n).getStartTime().get(i).getMonthOfYear();
                        startDay = node.get(n).getStartTime().get(i).getDayOfMonth();
                        startHour = node.get(n).getStartTime().get(i).getHourOfDay();
                        startMinute =
node.get(n).getStartTime().get(i).getMinuteOfHour();
                        startSecond = 0;

                        endYear = node.get(n).getEndTime().get(i).getYear();
                        endMonth = node.get(n).getEndTime().get(i).getMonthOfYear();
                        endDay = node.get(n).getEndTime().get(i).getDayOfMonth();
                        endHour = node.get(n).getEndTime().get(i).getHourOfDay();
                        endMinute = node.get(n).getEndTime().get(i).getMinuteOfHour();
                        endSecond = 0;

                        start = new DateTime(startYear, startMonth, startDay,
startHour, startMinute, startSecond);
                        end = new DateTime(endYear, endMonth, endDay, endHour,
endMinute, endSecond);

                        start2 = new DateTime(startYear, startMonth, startDay,
startHour, 0, startSecond);
                        end2 = new DateTime(endYear, endMonth, endDay, endHour, 59,
endSecond);
```

```
                        if(startHour > endHour)
                        {
                                while(startHour > endHour && startHour < 24)
                                {
                                        start = new DateTime(startYear, startMonth,
startDay, startHour, startMinute, startSecond);
                                        end = new DateTime(endYear, endMonth, startDay,
startHour, 59, endSecond);
                                        duration = new Duration(start, end);
                                        availibility[startHour][startDay-1] =
availibility[startHour][startDay-1] + ((int)duration.getStandardMinutes()+1);
                                        startMinute = 0;
                                        startHour++;
                                }
                        }
                        if(startHour < endHour)
                        {
                                while(startHour <= endHour)
                                {
                                        if(startHour < endHour)
                                        {
                                                start = new DateTime(startYear,
startMonth, startDay, startHour, startMinute, startSecond);
                                                end = new DateTime(endYear, endMonth,
endDay, startHour, 59, endSecond);
                                                duration = new Duration(start, end);
                                                availibility[startHour][startDay-1] =
availibility[startHour][startDay-1] + ((int)duration.getStandardMinutes()+1);
                                                startMinute = 0;
                                        }
                                        else if(startHour == endHour)
                                        {
                                                startMinute = 0;
                                                start = new DateTime(startYear,
startMonth, startDay, startHour, startMinute, startSecond);
                                                end2 = new DateTime(endYear, endMonth,
endDay, endHour, endMinute, endSecond);
                                                duration = new Duration(start, end2);
                                                availibility[startHour][startDay-1] =
availibility[startHour][startDay-1] + (int)duration.getStandardMinutes();
                                        }
                                        startHour++;
                                }
                        }
                        else if (startHour == endHour)
                        {
                                duration = new Duration(start, end);
                                availibility[startHour][startDay-1] =
availibility[startHour][startDay-1] + (int)duration.getStandardMinutes();
                        }
                }
                        print(numberOfDays+1,availibility, n);
        }

        public static void print(int b, int[][] availibility, int node)
        {
                System.out.print("[\'Day\',");
                double totalDailyAvailability=0;
                double maxMinutes=0;
                int hour = 0;
                for(int d=0; d<numberOfDays; d++)
                {
                        System.out.print("\t\t\t\'Day "+ (d+1) + "\',");
                }
                System.out.println("\t\t\'Average\'],");

                int sum = 0;
                for(int i=0; i<availibility.length; i++)
                {
                        System.out.print("[["+i+", 0, 0],");

                        for(int j=0; j<b; j++)
                        {
                                if(j==b-1)
                                {
```

```
        System.out.print("\t\t"+sum/(double)numberOfDays);
                                        totalDailyAvailability = totalDailyAvailability
+ (sum/(double)numberOfDays);
                                        if(sum/(double)numberOfDays > maxMinutes)
                                        {
                                                maxMinutes = sum/(double)numberOfDays;
                                                hour = i;
                                        }
                                }
                                else
                                {
                                        sum = sum +  availibility[i][j];

        System.out.print("\t\t\t"+availibility[i][j]+",");
                                }
                        }
                        sum = 0;
                        System.out.println("],");
                }
                System.out.println("On average node " + node + "'s daily availability
is " + totalDailyAvailability + " mins");
                System.out.println("Node " + node + "'s maximum availability is during
" + hour + ":00 - " + (hour+1) + ":00 " +  "for " + maxMinutes + " mins");
                System.out.println("[{row: " + (hour) + ", column: " + (numberOfDays+1)
+"}]");
        }
}
```

```
package com.adilhassan.mphil.availability.history;

import java.util.ArrayList;
import java.util.Random;
import org.joda.time.DateTime;
import org.joda.time.Interval;

public class Node {

        private int id;
        private ArrayList<DateTime> startTime = null;
        private ArrayList<DateTime> endTime = null;
        long uniqueTime = 0;
        private int numberOfSessions;
        private int eachSessionDuration;
        private int lowerBound;
        private int upperBound;
        private DateTime dateTime;
        private boolean updatedReplica = false;
        private DateTime updatedReplicaReceivedTimeStamp;
        private Random random = new Random();
        private boolean geographicalDistribution;

        Node(int id)
        {
                this.id = id;
                startTime = new ArrayList<>(1440);
                endTime = new ArrayList<>(1440);
        }
        public void setLowerBound(int lowerBound)
        {
                this.lowerBound = lowerBound;
        }
        public int getLowerBound()
        {
                return this.lowerBound;
        }
        public void setUpperBound(int upperBound)
        {
                this.upperBound = upperBound;
        }
        public int getUpperBound()
        {
                return this.upperBound;
        }
        public void setGeographicalDistribution(boolean b)
        {
                this.geographicalDistribution = b;
        }
```

```
	public boolean getGeographicalDistribution()
	{
		return geographicalDistribution;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public boolean hasUpdatedReplica() {
		return updatedReplica;
	}
	public void setUpdatedReplica(boolean updatedReplica) {
		this.updatedReplica = updatedReplica;
	}
	public DateTime getUpdatedReplicaReceivedTimeStamp() {
		return updatedReplicaReceivedTimeStamp;
	}
	public void setUpdatedReplicaReceivedTimeStamp(DateTime
updatedReplicateReceivedTimeStamp) {
		this.updatedReplicaReceivedTimeStamp =
updatedReplicateReceivedTimeStamp;
	}
	public ArrayList<DateTime> getStartTime() {
		return startTime;
	}
	public int getNumberOfSessions() {
		return numberOfSessions;
	}
	public void setNumberOfSessions(int numberOfSessions) {
		this.numberOfSessions = numberOfSessions;
	}

	public int getEachSessionDuration() {
		return eachSessionDuration;
	}
	public void setEachSessionDuration(int eachSessionDuration) {
		this.eachSessionDuration = eachSessionDuration;
	}

	public void addUniqueTime(long time)
	{
		uniqueTime = uniqueTime + time;
	}
	public void setUniqueTime(long time)
	{
		uniqueTime = time;
	}
	public long getUniqueTime()
	{
		return uniqueTime;
	}
	public ArrayList<DateTime> getEndTime() {
		return endTime;
	}
	public void setStartTime(int year, int month, int date, int hour, int min, int
sec, int milliSecond) {
		dateTime = new DateTime(year, month, date, hour, min, sec,
milliSecond);
		boolean overlappingFlag = false;
		Interval i1 = new
Interval(dateTime,dateTime.plusMinutes(this.eachSessionDuration));
		for(int i =0; i<startTime.size(); i++)
		{
			Interval i2 = new Interval(startTime.get(i), endTime.get(i));
			if(i1.overlaps(i2))
			{
				overlappingFlag = true;
			}
		}
		if(!overlappingFlag)
		{
			if(!(this.startTime.contains(dateTime)))
			{
				this.startTime.add(dateTime);
				setEndTime(year, month, date, hour,
```

```
(min+this.eachSessionDuration), sec, milliSecond);
                        }
                }
                else
                {
                        int sh =  random.nextInt(this.upperBound) + this.lowerBound;
                        setStartTime(year, month, date, sh, random.nextInt(60), 0, 0);
                }
        }

        public void setEndTime(int year, int month, int date, int hour, int min, int
sec, int milliSecond) {
                if(min<60){
                        dateTime = new DateTime(year, month, date, hour, min, sec,
milliSecond);
                }
                else
                {
                        int h = min/60;
                        min = min % 60;
                        hour = hour + h;
                        if(hour >= 24)
                        {
                                dateTime = new DateTime(year, month, (date+1), (0), 0,
sec, milliSecond);
                        }
                        else
                        {
                                dateTime = new DateTime(year, month, date, (hour), min,
sec, milliSecond);
                        }
                }
                this.endTime.add(dateTime);
        }
}
```