Calculating the Availability of Nodes in a Peer-to-Peer Backup System

Amir-Hossein Monshi
Abstract

Calculating the Availability of Nodes in a Peer-to-Peer Backup System

Amir-Hossein Monshi

Dynamic nature of peer-to-peer networks makes implementation of highly reliable backup solutions challenging. Nevertheless, this thesis believes that information about availability of nodes can assist the reliability of peer-to-peer backup solutions. Particularly, data replicas can be placed on subsets of nodes with the highest availabilities to reduce the risk of temporary or permanent data loss. Therefore, the goal here was to investigate an efficient and effective method to predict availabilities of nodes by observing their past behavior.

To reduce communication complexity and other costs, the proposed availability calculation mechanism designated nodes themselves to report the status of their peers through monitoring their ordinary pairwise interactions. However, not all the nodes were constantly communicating and/or were honest in their opinions about their partner nodes. To compensate for this, availability of a node was calculated by aggregating opinions of all of its partners, where each opinion was weighted according to the reputation (i.e. trustworthiness) of the partner. To this end, ideas from some of the most famous reputation systems were studied and adopted to devise and implement an efficient novel reputation calculation mechanism which respected dynamics and connectivity of networks. Simulations were conducted by incorporating the elitist strategy to select nodes with the highest availabilities for replication. The results indicated great success in discerning such nodes. Finally, the algorithm was integrated into an under-development commercial hybrid peer-to-peer backup service.
To my parents.
Acknowledgments

I would like to thank my supervisor Carl Hasselskog for his guidance and help throughout the course of this thesis, without his help this thesis would not be a reality. I would like to appreciate Dr. Arnold N. Pears for his priceless comments.
Table of Contents

Acknowledgments iii

Table of Contents v

List of Figures vii

Chapter 1: Introduction 1

1.1 The Problem 3

1.1.1 Previous Solutions 5

1.1.2 Peer Availability Approach 6

1.2 Organization of the thesis 8

Chapter 2: Background and Methodology 9

2.1 Definition of terms 9

2.1.1 Peer-to-peer 9

2.1.2 Peer-to-peer Backup 10

2.1.3 Availability 11

2.2 Objective and Obstacles 13

Chapter 3: Solution 17

3.1 Reputation Management Systems 18

3.1.1 Trust Transitivity 20

3.1.2 More on Trust and Reputation 21

3.1.2.1 Trust Representation 25

3.2 Availability Calculation 29

3.2.1 The Basic Availability Calculation Algorithm 33

3.2.2 A More Efficient Availability Calculation Algorithm 34

3.2.3 Complexity of the Algorithms 37
Chapter 4: Conclusion

4.1 Evaluation of the Algorithms ........................................ 41
4.1.1 Behaviors of the Nodes in Simulations ...................... 41
4.1.2 Partner Selection .................................................. 42
4.1.3 Evaluation Metric .................................................. 43
4.1.4 Experiments ........................................................... 43
   4.1.4.1 First Experiment: In The Presence of Malicious Providers .......... 44
   4.1.4.2 Second Experiment: In The Presence of Purely Malicious Nodes ........... 46
   4.1.4.3 Third Experiment: In The Presence of Feedback Skewing nodes and Malicious providers . . 49
   4.1.4.4 Forth Experiment: In The Presence of Feedback Skewing nodes and Purely Malicious Nodes 49

4.2 Implementation Notes ................................................ 53
4.3 Summary ............................................................. 54
4.4 Future Work ........................................................ 55

References 57
List of Figures

3.1 Indirect trust paths from peer $i$ to peer $k$ through intermediary peers. ......................................................... 21
3.2 Indirect functional trust (adapted from [14]) ......................... 23
3.3 Beta PDF instances (taken from [12]) ................................. 28
3.4 A beta PDF is a defined on a probability variable (taken from [14]) ................................................................. 29
3.5 Parallel paths of trust between two nodes ............................ 30
3.6 Matrix representation of trust network in EigenTrust ............. 32

4.1 Performance of the algorithms in presence of malicious content providers ................................................................. 47
4.2 Performance of the algorithms in presence of purely malicious nodes ................................................................. 48
4.3 Performance of the algorithms in presence of feedback skewing nodes and malicious content providers nodes ............. 50
4.4 Performance of the algorithms in presence of feedback skewing nodes and purely malicious nodes ................................ 52
Chapter I

Introduction

“Chaos is a name for any order that produces confusion in our minds.”

—George Santayana

PC backup technologies have come a long way since the days of 5.25” and 3.5” floppy disks. Today we are living in the era of remote backup services which not only supply subscribers with gigabytes of storage space but also are globally accessible, convenient, and highly reliable as opposed to the traditional on-site solutions. Nonetheless, still a great percentage of PC users surprisingly do not back up their data, at least not regularly, according to the surveys conducted by service providers themselves[15, 2].

Among a variety of factors which can impede users from picking up the good backup habits, one could point out the yet relatively high price-capacity ratio. For example, at the time of this writing, Dropbox charges the monthly price of $19.99 for 100GB of storage space. An average home PC user, who is unaware of the importance of frequent off-site backup, might consider this to be a significant cost for such an “inessential” task! Unfortunately, significant fractions of prices offered by backup service providers are unavoidable and inflicted not only by the actual storage media costs, but also other inevitable expenses: infrastructure, development, set-up, maintenance, administration, etc.

The aforementioned costs are magnified in backup systems compared to other services due to high-reliability requirements. For instance, a backup service provider might want to launch several backup server clusters at multiple geographic zones to guarantee high availability in the face of technical disruptions or natural catastrophes. Furthermore, service providers who try
to keep up with the demands of their ever-growing number of clients, should expect such capital expenditures to scale as well.

From an architectural perspective, the scalability seems to be an inherent problem of the client-server model in general. This model stems from the conventional notion that strictly designates a set of agents i.e., server machines, to provide a particular service to others who are mere consumers i.e., client machines. In these types of systems, a shift from an equilibrium of supply and demand towards higher rates of demand preludes to bottlenecks at provider’s end, and in this situation, an increase in their number or performance seems to be the inevitable remedy.

On the other hand, in the peer-to-peer model, each peer appears both as a service provider and a consumer to the other peers. In other words, each participant in this model dedicates a part of its resources, hardware in particular, to serve other participants directly. Consequently, a network of demand and supply amongst peers is formed over which they collaborate to achieve desirable levels of service quality. As opposed to the traditional server-based model, when the number of peers grow, the capability of a peer-to-peer network in service provision also scales, moreover it becomes more robust to failures.

These features lead to significant reduction in complexity, service set-up, and maintenance costs. As a result, very popular (and in some cases controversial) peer-to-peer networks have been formed to serve different goals such as file sharing, streaming media, cloud computing, etc. Numerous Internet users have interest to take part in their favorite peer-to-peer networks to share and receive the services they desire.

With the advantages of the peer-to-peer model, the idea of building peer-to-peer backup services becomes attractive. Especially when users have lots of unused sharable spaces lying around on their hard disks. A large-scale study by Douceur and Bolosky, shows that the average space usage of personal computers in a commercial environment is only %53 and this includes virtual memory and file system overheads[9]. An extension of their study over the course of five years has shown that this value has decreased even more to around 40%[42]. This amount of unused storage space can potentially constitute an enormous virtual disk for backup purposes of peers.
Another outcome of the latter study suggests that the amount of locally created and modified contents are decreasing and users tend to copy files more and more from each other. Besides that, significant portions of file systems contain core contents, such as OS, and ubiquitous applications, which users already have in common. As a result of this natural redundancy, many data blocks are identical among groups of nodes. Thus, peer-to-peer backup solutions can detect and coalesce these identical blocks to save the unused storage for replication of dissimilar data blocks if need be. Other distributed file and peer-to-peer backup systems have taken advantage of this feature[6, 43, 44].

Other storage optimizations are also possible. For instance, using erasure coding significantly reduces the number of data blocks required for a successful data restore operation compared to the ordinary replication. Consequently, erasure code replication requires less storage space to provide the same degree of data availability that is achievable by the strict replication[45]. Data compression is another effective method to boost the storage efficiency furthermore.

All in all, it seems that the necessities for building efficient, cost-friendly, and manageable peer-to-peer backup services are satisfiable. However, several attempts to build such a service have been made[6, 3, 18, 17, 22], and none of them have been successful on the Internet like other popular peer-to-peer systems!

While identifying the reasons of their failures and constructing adequate solutions demand thorough analysis of all intrinsic aspects of the those works, but still it is apparent that they lack an important ingredient which can reduce their risk of failure significantly. That is constantly monitoring, learning, and exploiting the dynamics of a network of peers. The next section elaborates on this issue and a different approach to devising peer-to-peer backup solutions.

1.1 The Problem

Server-based architectures usually offer great degrees of reliability, thanks to their centralized control and infrastructure that is hard to achieve in highly
dynamic and decentralized public peer-to-peer systems. Structure of a peer-to-peer network in the Internet is constantly changing and peers which are present in the network at one moment, might be absent later and vice versa. As if this is not enough, open nature of the public peer-to-peer networks welcomes various types of malicious peers which employ different strategies to burden the performance of the system. For instance, they might try to propagate spurious and/or malicious contents. Such degree of uncertainty in the behavior of a system, puts serious challenges in the path of finding solutions that demands extreme reliability such as backup.

It might be possible to approximate high availability of data by increasing the number of replicas by an order of magnitude. This approach alongside proper cryptographic hashing techniques would allow us to eventually spot and verify authentic contents. Obviously, this soon becomes impractical as the number of nodes grow. Because, it is wasteful of nodes’ disk capacities and results in the congestion of the network due to heavy messaging overheads. The problem is intensified in peer-to-peer backup systems compared to other similar services such as peer-to-peer file sharing. Since in the latter, usually great number of peers in the network share same files and therefore there is a high degree of availability regarding shared data blocks. On the other hand, in a backup network, peers naturally bring in distinct files. In the naïve approach to achieving the same availability level of a file sharing network, all data blocks should be replicated with the same degree. While a backup system, by definition, is expected to be more reliable and available than a file sharing system and therefore even higher degrees of replication might be necessary. Thus, the naïve approach if replication is obviously impractical due to the strains that it puts on a network.

---

1 Therefore, in the context of this thesis, a node’s “availability” refers to the probability of a successful data restore from that node at any point in time. This obviously requires the node to be available in the network when requested and capable of providing correct and uncorrupted data.
1.1.1 Previous Solutions

The reviewed works in peer-to-peer backup\cite{6, 3, 18, 17, 22} each use a different approach for tackling this problem\footnote{Stefansson and Thodis have dedicated a chapter of their M.Sc thesis to reviewing some of the more significant works in peer-to-peer backup. Since this thesis tries to focus only on a very specific aspect of peer-to-peer backup systems, namely availability of peers, the interested reader is encouraged to refer to \cite{22} for a more general survey.}. However, generally their actions could be considered to be either “protective” or “corrective” in nature.

The protective type of actions involves a peer, wishing to make a backup, to find a set of partners or buddies and exchange quotas with them and replicate chunks of its backup data to them. Should any partner fail to respond with the original un-tampered data to a request or challenge from the original peer, due to unavailability or malice, the data may be recovered from the other partners. In other words, a peer protects itself from the possibility of failure in retrieving the backed up data by replication to a number of peers. The method and criteria of partner selection differ in previous studies and are customized for their specific algorithms\footnote{These criteria are sometimes too restrictive and might remove some useful network configurations that lead to a feasible solutions. (give examples)}.

One question that needs to be asked, however, is whether it is possible to estimate the number of partners/replicas required for high probabilities of successful restore operations, without knowing the availability measure of the partners.

When some partners fail, the first obvious corrective action is an attempt to retrieve the data from other partners. This might be followed by other corrective actions. For instance, by strict punishment of failed partners e.g., by discarding some of their exchanged data to discourage their malicious behavior\cite{17}, or by replacing the unavailable and/or malicious partners with new plausible ones. Although these corrective strategies have been shown to be effective to some extent, they have not escaped situations where there is more uncertainty than expected. For instance the aforementioned punishment mechanism easily results in the loss of an honest partner’s data that has been punished for being chronically unavailable due to reasons other than malice. The partner replacement strategy can result in high levels of data migration from other partners and/or the original peer to the newly elected
plausible alternatives. With such high volumes of stored data nowadays, one can easily imagine the amount of traffic generated as a result of several unsuccessful attempts in finding honest/available partners. Needless to say, none of the corrective mechanisms can guard against the scenario that restoring the data is impossible or deferred unboundedly because partners have become unavailable for a long time or permanently.

These types of problems are in general very difficult to deal with and corrective solutions are not bullet-proof in that regard. Consequently peer-to-peer backup systems may suffer from high traffic rates, high disk consumption, long waiting times due to unavailability of nodes, unreliability issues, and other peculiarities of this sort that may seem daunting to the end user who demands the reliability that a server-based solution can offer.

However, this thesis is optimistic that monitoring dynamicity and nondeterminism of peer-to-peer networks yields valuable information about the network and their nodes which can greatly benefit various strategies that deal with this problem. In particular, useful information about the availability of nodes is attainable which can guide various strategies in their resource investments on peers in a network. All the studies reviewed so far suffer from the fact that they fail to take availability of peers into account, although extensive research in peer-to-peer networks has been carried out on this subject.

1.1.2 Peer Availability Approach

The root of all evil is in the fact that a node in a public peer-to-peer network is generally less consistent in its behavior than a dedicated backup server. That is because there is no centralized control over the peers and thus they are not committed to continuous high quality service provision. The user that controls the peer can shut down the backup application at any moment or uninstall it permanently. In a worse case scenario, the peer might (un)intentionally deliver corrupted or malicious data. Thus, a reasonable solution to deal with these problems is replication of data to more than one partner peer. Although, this approach has been used in previous works, unfortunately, the availability histories of candidate partners were not taken
into account as a selection criterion. This information can help estimate the probability of a successful future restore operation from partner peers. An analogy would be how a seller’s reputation on eBay could estimate the probability of a satisfactory transaction with them.

By knowing the availability measures of candidate partners, a peer can select the ones with the highest availability. As it will be explained later, an availability measure can be represented in different ways and based on that, it can provide different levels of information. For example, availability measure of a peer might only have ordinal quality and serve no more than ranking the peer among the peers in terms of its availability; on the other hand, it can represent the actual availability probability of a peer rather than being a mere ranking. While the former provides a comparative approach to partner selection which allows a strategy to select the highly available partners, the latter additionally allows a strategy to compute the probability of a successful restore operation from a set of candidate partners.

Having availability measures with probabilistic interpretations helps in effectively fine-tuning the set of partners for a successful restore operation. For instance, imagine that a strategy requires that the probability of at least one partner being available all the time to be higher than a certain threshold which the current set of candidate partners falls below. By having availability probabilities of peers, the strategy can replace some of the candidate partners with better ones or maybe add more partners to the set until the probability becomes higher than the threshold.

This opens possibility for designing more resilient strategies that can be effective in more realistic scenarios. For instance, a strategy might prefer replication to a greater number of partners with lower availabilities rather than a few of them having high availabilities. This could be due to some other restrictive factors such as disk and/or bandwidth limitations of highly

---

4 Probability of a satisfactory transaction assigned by a buyer based on the reputation of a seller is a Bayesian probability which is the buyer’s degree of belief in this event. On the other hand, classical probability is bound to physical properties such as that of a fair coin. Bayesian probability encodes uncertainty in probabilities and thus more effectively models the behavior of real world entities such as peers, eBay sellers, etc[12]. Concept of Bayesian probability is a fundamental to computing availabilities of peers in this thesis.
available peers, or for other purposes such as load-balancing, latency reduction, etc. One might consider this as an optimization problem where the objective is to maximize the benefits and minimize the loss with respect to the aforementioned facets of the system. A considerable amount of literature has been published on this problem in distributed and peer-to-peer systems. However, since the scope of this thesis is limited to the calculating the availability of peers, the interested reader is referred to [10, 8, 19, 5, 7, 4] for some instances of research in this field.

1.2 Organization of the thesis

The rest of the thesis is organized as follows. Chapter 2 elaborates on the background and fundamental concepts, requirements and objectives. Chapter 3 illustrates the main solution proposed in this thesis and its variations and enhancements. Chapter 4 explains about experiments and analyzes the results, describes some technical consideration in the integration of the algorithm into the commercial product that this thesis is a part of, and finally summarizes the thesis by giving some hints about unsolved problems and future work.
Chapter II
Background and Methodology

“Speak English!

I don’t know the meaning of half those long words,

and I don’t believe you do either!”

—Eaglet, Alice in Wonderland

Before moving on to the main solution proposed for the problem of calculating availability of nodes in a peer-to-peer backup system, it seems essential to spend some effort in establishing the conceptual foundations for a better understanding of design decisions made throughout the thesis. To this end, an introduction to the fundamental concepts, objectives and requirements of the thesis appears to be inevitable. Thus, this chapter is written to serve the very same purpose by starting to explain some of the basic terminology in section 2.1. Then, section 2.2 elaborates on the objectives and requirements of this work.

2.1 Definition of terms

2.1.1 Peer-to-peer

According to Schollmeier[20], peer-to-peer networks are a subclass of distributed network architectures in which the participants act both as providers and requesters of a particular service. In other words, each participant dedicates a part of its hardware resources to serve other participants directly. He further divides a peer-to-peer network definition into two types, namely, *pure* and *hybrid*. The former refers to a type of network in which there is no central entity e.g., central server, among the peers whereas in the latter type
such entities do exist. This very important architectural difference can have a major effect in design decisions regarding the algorithms and strategies mainly in terms of security and/or distribution.

For instance, in order to select partners for replica placement, a peer can usually trust availability measures which are calculated and published by central authorized servers. However, in the absence of such central authorities, peers must calculate the availability measures of their candidate partners themselves. For candidate partners that a peer has null information about, it might want to rely on opinions of those peers who have had a history with them. To this end, the peer should be able to effectively deal with dishonest information that it might receive from malicious peers regarding the candidate partners[16].

2.1.2 Peer-to-peer Backup

Peer-to-peer backup is a type of distributed system by which a user can copy their data onto other users’ machines in a way that they can be restored later. Landers et al.[17] neatly describe steps involved in a generic peer-to-peer backup process, which are recited here:

- The process starts by a user’s request for a data backup

- Then the system replicates the data to other peers i.e., partners.

- Each node shares some of its storage space with other peers.

The word “system” used subtly to indicate that an originating peer itself may not be the only element that is responsible for the replication of its backup data to its partners. In particular, in a hybrid peer-to-peer backup system, a central server might first receive the backup request of a peer and it might perform some other actions before, during, and after the peer is replicating its data. For instance, it might perform one or more of the following: authentication, authorization, and accounting(AAA [1]); determining the best set of partners for the peer’s backup request; monitoring the replication and data transfer; recording success or failure of replica placements; etc.
2.1.3 Availability

The *Glossary of Telecommunication Terms*\cite{Glossary} defines availability as follows:

"1. The degree to which a system, subsystem, or equipment is operable and in a commitable state at the start of a mission, when the mission is called for at an unknown, i.e., a random, time.

*Note 1:* The conditions determining operability and commitability must be specified. *Note 2:* Expressed mathematically, availability is 1 minus the unavailability.

2. The ratio of (a) the total time a functional unit is capable of being used during a given interval to (b) the length of the interval.

*Note 1:* An example of availability is 100/168 if the unit is capable of being used for 100 hours in a week. *Note 2:* Typical availability objectives are specified in decimal fractions, such as 0.9998."

This is a concrete definition but also a very generic one, and applying it to a peer-to-peer backup system requires removing two ambiguities:

- What is the subject of availability?

- Which of the two definitions should be used?

In order to answer the first question, it should be clarified which part of the system the availability calculation process takes place for. Distributed systems in general and peer-to-peer systems in particular are usually complex systems with various facets and properties. It is therefore desirable to refine the definition of availability by restricting it to specific properties, parts, or granularities of such systems. This approach have been suggested also by others\cite{On et al., On et al.}. In \cite{On et al.}, On et al. give a fine-grained hierarchical definition of availability of a service which relies on the availability of data and system both. Availability of the system is determined by availability of nodes and links. A node is considered to be available when it is online and can allocate resources to provide the request. Service providers can specify to consumers the availabilities of the aforementioned components of the service, in a way that enables consumer to seek a fine-grained customized quality of service. This approach to defining availability is superior to the traditional method
which is merely based on uptime/downtime of the nodes, since it captures the different aspects of functionality of a service.

The quest of the thesis, as its title indicates, is confined to availability calculation of nodes in a peer-to-peer backup system. In other words, the subject of the availability here, is the service that individual nodes in a peer-to-peer backup system can provide. To this end, the effort has been made to have a fine-grained view of the components that contribute the overall service availability of a node (as in [27]). Consequently, a time-specific service availability function has been defined to be the Boolean conjunction of some criteria which determine whether or not a node is providing the correct service at a specific time e.g. at the time when it is requested by another node. These criteria are the online status of the node in the network; consistency and correctness of data requested from the node; and its success in replying with a correct response to a request based on the networks proprietary communication protocol. To put it simply, imagine node $A$ making a service request to node $B$. Upon receiving a response from $B$ or detecting its failure to deliver one, $A$ evaluates the service availability function i.e. the above availability criteria, to a Boolean value. This determines the failure or success of $B$ in presenting itself as a functional node to $A$ at the time of the request.

Challenging a node and evaluating its availability i.e., sampling, with high frequency over an interval of time should result in an estimation of the average availability that is equivalent to the part two of the standard definition above, hence, addressing the second ambiguity regarding this definition. It should be noted that to the best of author’s understanding, applying the first availability definition above seems absurd when the functionality of a service is being represented by a Boolean value, such as here. On the other hand, if time-specific availability can be indicated with values between zero and one i.e., “degree”, then the first part of the definition is preferred. Examples of this type can be the storage capacity of a node or bandwidth of a link at an instance of time. The average of those values can represent availability of the system which is meant by the first definition above.

At this point, it should be mentioned that there are major difficulties in gathering large and reliable sets of time-specific availability values for
calculation of the average availability of a node in a public and dynamic peer-to-peer network. They will be discussed in the next section.

2.2 Objective and Obstacles

This section proceeds with elaboration on thesis objectives and obstacles in the path of achieving them. In this regard, first a brief description of the development platform is presented. Then the problem of availability calculation is redefined according to the current platform and its intrinsic boundaries. Consequently, considerations and decisions which are made to approach the problem are justified.

This work attempts to extend an under development proprietary peer-to-peer backup system¹ with an effective peer availability calculation mechanism. The goal of the product is to enable a node to share a part of their storage space in exchange for access to a massive storage space of a virtual disk that is comprised of shared spaces of all nodes in the network.

When a user makes a backup request, system replicates their data among other nodes in the network. Similarly, upon a request for retrieval of their backed up data, system queries the nodes which are available and hold the data. For this purpose, the system is assisted by the nodes availability estimations in replica placement and resource management (ref section 1.1.2).

The backup system is designed to follow the hybrid peer-to-peer network architecture. In other words, there are central servers that are responsible for management of the network. Critical tasks such as authentication, authorization, and accounting (AAA) regarding users/nodes; calculation of nodes availabilities; finding replica placement configurations with low risk of failure; and directing nodes accordingly in their replications; are all done by central servers to maximize the reliability of backup service.

Designating central servers to calculate the availability of nodes, they require viable sample sets of time-specific availabilities of nodes. Traditionally in distributed systems, servers acquire this data by frequently probing the nodes. However, if this were a feasible choice here, the problem of availability

¹ This system is a continuation of CollabBackup which was developed by Carl Hasselskog for his Master’s thesis[11].
calculation would become trivial. Probing nodes by the server is not possible due to some inherent design considerations.

The system follows Representational State Transfer (REST) architecture \cite{26} for its well-known benefits (mainly scalability) and thus embeds a stateless request-response protocol (e.g. HTTP) for communication between nodes and servers. As a result of this protocol, servers can not actively probe the availability of nodes due to inability in initiating requests. Even if nodes probation by servers were possible e.g., by having them constantly reporting to the servers, still it would be practically impossible to track their availabilities completely. Recall from the previous section that the availability of a node at request-time depends on delivering correct response to the request of another node. For a server to confirm availability of a node, it should be able to verify the correct delivery of node’s responses to requests made by others. This involves verifying that those responses do not contain corrupted data. Should this be feasible, it puts a lot of strain on servers and also on the network due to extra probation traffic overheads.

A method used mostly in some empirical research in order to gather data regarding availability of the nodes at different times is to designate some special nodes i.e., crawlers/probers, to probe others frequently for their availabilities\cite{29, 28, 30}. In order for this method to be effective in capturing all of the dynamics of a large network in a timely fashion, it requires a great number of active crawlers, which indeed impose a lot of traffic overhead on the network. Existence of these and other issues are confirmed by \cite{31} and \cite{32}.

While relying on central servers and crawlers to probe nodes in a network for their availabilities seems cumbersome, using nodes themselves to monitor each other sounds promising. Simply put, a node can assess the availability of its partners based on the correctness of their responses. Assessment of results from various nodes regarding their partners then can be aggregated to estimate average availabilities of partners. Having nodes sampling availabilities of one another, comes with much less overhead as opposed to the previous techniques. Because, no redundant probing is necessary when nodes

\footnote{This architecture is best explained by its most famous conforming implementation that is World Wide Web.}
can monitor each others’ availabilities by checking only the correctness and consistency of their normal interactions. Unfortunately, however, nodes can not often be considered to be reliable samplers.

The unreliability can arise at sampling and reporting stages. A node needs to frequently probe its partner in order to acquire a statistical sample that is sufficient for a close availability estimation. Obviously, in a peer-to-peer network, nodes are not to be delegated with such a task. Because, not only frequent sampling causes traffic overheads, but also it is naïve to assume that a node is constantly present and sampling. Even if some nodes constantly probe each other, they are not always reliable in reporting these data. Misbehavior of this kind could be unintentional e.g., corruption of data, or with malice aforethought. The latter case is quite possible due to open and decentralized nature of peer-to-peer networks.

In general, therefore, it seems that the measurements done by nodes in a peer-to-peer network come with a degree of uncertainty. This is while reliability-critical peer-to-peer backup strategies surely can benefit from certainty in nodes availability data which helps them in making perceptive decisions in replica placement and resource management. Therefore, it is advantageous to be able to reduce the degree of uncertainty as much as possible and furthermore to be able to somehow estimate it so that system can cautiously deal with it. This thesis also aims at these very improvements.

In summary this work tries to fulfill the following criteria in devising a method for peer availability calculation:

- Using nodes themselves to monitor each others’ availabilities through their ordinary interactions in order to be resourceful.

- Enduring the existence of malicious nodes which pass dishonest judgments on other nodes.

- Estimation of uncertainty in data regarding the availability on nodes.

The following chapter introduces a novel approach for calculating the availability of nodes by respecting the above criteria. The method is extensively based upon studies on another very popular area of research particularly in
peer-to-peer systems, namely, *Reputation Management Systems*. Then, in the chapter 4 effectiveness of this method is tested by simulations.
Chapter III

Solution

"So far as the laws of mathematics refer to reality,

they are not certain.

And so far as they are certain,

they do not refer to reality."

–Albert Einstein

This chapter explains a new approach\textsuperscript{1} for nodes availability calculation in a peer-to-peer backup system. Recall from section 2.2 that there is a weak notion of centralized control in the target peer-to-peer system. Therefore we are left to the opinions of nodes on each other in order to estimate their availabilities. In other words, degree of availability of a node should somehow be calculated using the opinions of other nodes about that node based on success in their past transactions with the node.

This approach is prone to errors as nodes opinions are not completely reliable due to inconsistent sampling or malice. The situation becomes even more complex when malicious peers build collectives of high degrees of complexity to subvert the system. Identifying these collectives and analyzing their effects and perhaps nullifying them are all extremely difficult goals to achieve in a decentralized peer-to-peer system where normally no peer can be trusted.

\textsuperscript{1} Although this work is heavily based on previous work in the field of Reputation Management Systems, to the best of the author’s knowledge, the approach that is taken in this thesis has not been used before directly toward availability calculation in peer-to-peer networks. However, since a large and growing body of literature has investigated this very problem which is impossible to thoroughly cover in the limited scope of this thesis, the author apologizes in advance for unintentional omission of any overlapping previous research.
To reduce the impact of these anomalies, techniques from area of Reputation Management (RM) have been utilized extensively to devise a feasible availability calculation method. Therefore, in section 3.1 a brief review of some of the most influential work in RM is provided. Then section 3.2 delves into the details regarding the basic form of our availability calculation method and considerations that were inspired by studies in RM systems. The chapter that follows, conducts a series of simulations that shows the effectiveness of the method proposed here.

3.1 Reputation Management Systems

Many societies are formed around the notion of reputation. A familiar example is eBay where users usually prefer to trade with highly reputed merchants. In these types of systems, the trust that users put in each other is usually in direct relation with their reputation\(^2\). Therefore, if a trader is to make their business flourish, they should strive to gain a high reputation in their respective society. When users first join the system they have no reputation but as they trade with other users, they gain or lose reputation based on the quality of the service they offer and consequently the feedbacks that they receive from their business partners\(^3\). eBay manages the reputation of users by collecting and publishing the feedbacks that they receive from other users. This way for example when someone wants to purchase a merchandise, she can assess the kind of service that she can expect from the candidate merchants by knowing their reputations.

A network of peers is not unlike the eBay society in terms anonymity of joining peers and their freedom in providing various degrees of quality of service (QoS). Therefore, a reputation management system can also benefit peer-to-peer networks (although the absence centralized control e.g., eBay servers, might make this task more of a challenge particularly in pure peer-to-peer architectures). An effective reputation management system can contribute significantly to the information about the expected QoS from peers.

---

\(^2\) In fact trust and reputation are used sometimes interchangeably in the literature.

\(^3\) Obviously, dishonest partners might give false negative/positive feedbacks due to antipathy toward the other user or the system.
Not surprisingly, a large and growing body of literature has investigated the role of reputation management systems in peer-to-peer networks. In fact, the generalisability of such large volume of published research on this matter is problematic. This thesis, exploits ideas from some of the most influential work in this field in order to approach the peer availability calculation problem.

Interactions between peers helps them form opinions about QoS of each other. However, if there is no mechanism to make these opinions “global”, obviously peers stay confined only to their own “local” opinions which might be incomplete due to infrequent sampling, unreliability... and thus insufficient for decision making. Moreover, a peer often might want to find a partner from a set of candidates which it have had no interactions with and thus have no local opinions about. In this situation, it is impossible for the peer to discern the best partner from the others unless it can learn their global reputation.

A reputation management system helps reducing the risk of such abominable experience in a network of peers where interacting members anticipate quality service from their counterparts, but simultaneously are allowed to provide arbitrary levels of QoS. In such systems, a reputation management system usually computes members’ reputations by aggregation of other members’ opinions about them. However, aggregation of peers’ opinions is not as trivial as it appears at the first sight. This is because peers’ opinions about each other are not always totally reliable, and since there is no single central authority in the system that can confidently confirm (un)reliability of such opinions, therefore trust of peers on each others’ opinions also should be taken into account as a determinant factor. As a result, a network of trust [13] is formed over the actual network of peers. Existing reputation management mechanisms generally exploit the information that is scattered over a network of trust in order to arrive at a consensus on opinions of everyone and thereby estimate reputations.

The trust relationship among peers is transitive, a property that the most effective reputation management systems respect in their analysis of trust

4Interested reader is encouraged to refer to [33, 34] for surveys on reputation/trust systems
networks. Next parts proceed from elaborations on trust transitivity[14] and the definition of trust itself to the actual techniques of reputation estimation.

3.1.1 Trust Transitivity

Transitivity of trust is usually best explained by tangible examples from our trust-based relationships as humans. For instance, imagine Alice is an employer who wants to hire Charlie, whose performance she has no opinion about. However, Alice knows Bob who has worked directly with Charlie and can inform her of Charlie’s performance at work. In this scenario, Alice can form a level trust on how Charlie might perform at work indirectly, by trusting Bob’s words. Obviously the level of trust that Alice has in Bob affects her final trust in Charlie’s performance.

This notion is used in trust and reputation management of particularly peer-to-peer systems to calculate indirect trust between peers. For instance, in EigenTrust, a competent reputation management system, proposed by Kamvar et al. [16], indirect trust of a peer $i$ in peer $k$ is:

$$t_{ik} = \sum_{j \in J} c_{ij} \times c_{jk}$$

where $c_{ij}$ is the direct trust of peer $i$ in peer $j$ and similarly $c_{jk}$ is the opinion of peer $j$ in peer $k$, where $0 \leq c_{ij}, c_{jk} \leq 1$. In other words, peer $i$ makes a weighted average of opinions of other peers about peer $k$ where weights are the trust that peer $i$ has in them as a result of direct interactions. Each multiplication term in the above summation indicates an indirect trust relation between peers $i$ and $k$ through peer $j$ as it is shown in figure 3.1.

The above formula contains one level of indirection which is sometimes not enough to establish trust between two peers because in a trust network there might be multiple intermediary peers between subject and object of the trust relation. For instance, Alice knows Bob who knows Carol who has an opinion about the performance of a candidate employee called David. Obviously, the length of trust paths can be arbitrary.

Indirect trust is useful when sufficient or reliable direct experience with a party is lacking. In peer-to-peer systems because generally a peer is neither
assumed to be consistent in its sampling nor always completely reliable in reporting its true opinions. Therefore, generally direct opinions of no single peer can be considered as sufficient and reliable, rather this information is scattered in the network. Thereby, reputation management systems usually traverse indirect trust paths which are formed amongst peers, and aggregate their “local trusts” into “global trust” values regarding peers i.e. their reputations. As it will be illustrated later in section 3.2, one of the more significant factors of trust and reputation management systems is their approach in taking indirect trust paths into analysis.

3.1.2 More on Trust and Reputation

Before proceeding to analysis of a network of trust, it is essential to elaborate more on trust and reputation and their differences to prevent confusion.\(^5\) Trust and reputation are closely related notions, however Josang et al. in [33] point out a subtle distinction between trust and reputation (in the senses that they are used in this thesis and many reputation systems). Simply put, reputation can be considered as an objective global measure that is formed by aggregation of opinions of all peers, but (direct) trust is subjective and local to peers themselves.

For instance, when an entity $A$ starts interacting with $B$ for the first time,
she does not have a personal sense of trust in him due to lack of previous
direct experience. However, she might want to put her trust in B based
on its reputation that can be considered as an objective measure of trust
in B which is estimated by reputation system. Later when A gains enough
experience regarding B and forms a level of trust in him, this trust might
be close to B’s global reputation or very different based on A’s individual
experience of B, or a special relationship between them.

Josang et al. in [14] bring up the concept of **scope** of trust which is informally equivalent to the quality, service... that trust is based on. For instance
when Alice trusts in Bob’s recommendation regarding Charlie’s work ethics,
the scope of trust is “work ethics”. Authors also have made an interesting distinc-
tion between two variants of trust regarding a scope, namely functional
trust and referral trust:

**Functional trust** is a type of trust that is result of a trusting party’s eva-
uation of a trusted party regarding the scope of trust e.g. Bob’s trust
in Charlie regarding his work ethics is functional trust, since its the
outcome of Bob working directly with Charlie.

**Referral trust** is the trust a party has in the ability of a trusted party
to precisely reflect the trustworthiness of a third party regarding the
scope of trust e.g. Alice’s trust in Bob’s ability to correctly recommend
Charlie as a good worker is Alice’s referral trust in Bob.

It should be noted that direct functional trust regarding a scope can be
extended to indirect functional trust as a result of trust transitivity\(^6\). For
instance Alice will have an indirect functional trust in Charlie by considering
Bob’s recommendation about him. The indirect functional trust that Alice
has in Charlie might be weaker than Bob’s direct trust in him due to Alice’s
partial trust in Bob’s recommendation. As it is illustrated in figure 3.2, an
indirect trust path can be of arbitrary length but it is necessary that every
path ends with a direct functional trust which guarantees the “functional”
 nature of the indirect functional trust.

\(^6\)Trust transitivity is possible for a trust scope when both direct functional and referral
trusts are in place.
In a reputation system that takes indirect trust paths into calculation, reputation of a peer then can be considered as global indirect functional trust of the reputation calculating entity e.g., a central server, another peer..., in that peer. Therefore, if the direct functional trust of peers in each other is defined to be based on the scope of availability, then the reputation of peers is the global measure of their availability i.e. the availability that is aggregated from various sources taking into consideration the trustworthiness of sources.

The distinction between functional trust and referral trust is important because it allows mounting special functionalities on top of already existing reputation systems. For instance, EigenTrust is a reputation algorithm that does not distinguish between referral trust and functional trust, perhaps because both referral trust and functional trust have the same definition in this system which is based on the reliability of node in data provision. That might be fine as long as the purpose of the reputation algorithm is to rank peers based on their reliability. However, if the definition of a reputation should indicate another aspect of the system e.g., availability, then the definition of functional trust should be changed accordingly so that peers could also reflect on that aspect.

Thereby, before modifying the actual reputation algorithm to calculate peer availability, the elemental components, being the referral and functional trusts, should first be redefined accordingly. Since the availability calculation algorithm proposed relies on subjective logic[35], we follow the same notation.

**Referral trust:** The referral trust of peer $A$ in peer $B$ is shown by $\omega_{A}^{B}$, as $A$'s degree of belief in $B$'s honesty, and is estimated by $A$ based on its previous observance of $B$'s ability in providing correct and authentic
Functional trust: The functional trust of peer A in peer B is shown by $\omega_{x_B}^A$, as A’s degree of belief in B’s ability to respond correctly to its requests. Obviously, this requires B not only to provide authentic data, in the sense defined above, but also be available and behave correctly regarding the communication protocol between the two. Functional trust is obtained by A based on its observance of B’s past behavior.

The role of referral trust, as previously mentioned, is to provide the trusting party with an estimation of the reliability of the trusted party’s opinions regarding third parties. To this end, a peer’s capability to correctly return the data blocks, which it has been assigned to, can be considered as a measure of it’s reliability and consequently trustworthiness in giving correct feedbacks. This assumption for instance has been incorporated in EigenTrust[16].

However, the above definition of referral trust might be subject to criticism as its does not include the correctness of the feedbacks of a trusted party about other nodes. In fact, some malicious peers might provide authentic data to gain high levels of referral trust but when requested for feedback, they return false information to discredit honest peers or glorify other malicious peers.

Dealing with this type of malicious behavior is non-trivial in this system (and many other reputation systems). Because, peers do not directly query other peers for highly trusted peers. If this were the case, then the referral trust of a trusted peer could have been directly penalized upon a bad recommendation. Rather, here, a peer asks the central reputation managers e.g., central servers, which in turn rely on peers feedbacks to calculate reputations and in this regard they run a complicated aggregation algorithm that makes average values out of the feedbacks from all the peers. In this situation it is very difficult to trace back the source of incorrect feedback. But still based on an assumption that most of the peers in the network are honest in their feedbacks, the system should survive by outnumbering the malicious peers.\[7\]

\[7\]Please recall that although the central servers are not responsible for actively probing nodes for their availabilities, however still peers send their feedbacks to them and also
A point worth mentioning about the definition of functional trust is that it can be changed independently of the reputation algorithm that is presented later in this chapter. At this point the roots of unavailability which fulfill the requirements of the application are included in the above definition. However, if need be, this definition can be modified without requiring any modification in the main reputation algorithm. This is a positive side-effect of discerning between trust variants.

Also note that the above definition of functional trust is based on combination of different metrics. Merging different metrics into one has also been suggested in other reputation systems such as FuzzyTrust[21]. In the next part, the trust representation is explained according to subjective logic and Bayesian statistics.

3.1.2.1 Trust Representation

Trust can be represented in various forms the simplest of which is by Boolean values i.e. an entity is either trusted or not. Another way is the discrete representation where a trust value is chosen from a finite set of ordinal discrete values[36]. However, often reputation systems use more sophisticated continuous representations which are capable of conveying higher levels of information which at least allow a more fine-grained comparison between rankings. For instance, EigenTrust uses normalized local trust values which are calculated as is shown below:

\[ c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_k \max(s_{ik}, 0)} \]

where \( s_{ij} = \text{sat}(i,j) - \text{unsat}(i,j) \) and \( \text{sat}(i,j) \) and \( \text{unsat}(i,j) \) are respectively the number of past successes and failures of peer \( i \) in having satisfactory transactions with peer \( j \). Each peer can keep track of the number of its satisfactory and unsatisfactory transactions with other peers and therefore calculate the normalized local trust values as mentioned. These local trust values are then converged into global trust values by EigenTrust algorithm which are also positive real numbers in \([0, 1]\).
The global and local trust values are effective in ranking the peers (globally and locally) based on their trustworthiness. However, as the authors of EigenTrust also confirm[16], these values do not have any absolute probabilistic interpretation and are only useful for relative comparison of peers with each other in terms of trustworthiness. Recall from section 2.2 that, the objective of this thesis is to estimate the availability of peers in a probabilistic form that captures and expresses the uncertainty existing in availability values. Obviously, the representation of trust that is proposed in EigenTrust does not fulfill this requirement. Two different methods of expressing uncertainty, namely, fuzzy logic and subjective logic, have been already exploited in trust management systems.

FuzzyTrust[21] is an effective reputation system that is based on fuzzy logic. It calculates reputation by aggregating local scores which are derived from the result of a set of fuzzy inference rules. For instance, availability might be defined by the following rules:

- If a partner’s rate of communication failures is low and the rate of inauthentic content download is high, then the availability is very low.

- If a partner’s rate of communication failures is medium and the rate of inauthentic content download is medium, then the availability is low.

The results of these rules are superimposed i.e., aggregated, and defuzzified by each peer, to produce the local trust score values. A set of fuzzy membership functions are used to map the actual observation numbers into a set of predefined fuzzy levels e.g. very low, low, medium, etc. by which, the fuzzy inference rules are definable.

An advantage of fuzzy inference rules is that they are easily human-understandable and thus provide the means for designers to define or extend the rule sets based on application requirements. Also, being defined as fuzzy propositions, these rules deal with uncertainty. However, the precision is sacrificed in the process of mapping values to fuzzy levels; rule-based inference; and aggregation i.e., superimposition, of the results of rules. For instance, the degree of uncertainty directly depends on the number of fuzzy levels and
the shape of membership functions which are defined by the designer. In
other words, uncertainty is “artificially” embedded in membership curves to
approximate the real uncertainty in data[38]. This is while in our system it
is essential to estimate the ‘real’ uncertainty that is inherent in the sampled
data so that replica placement subsystem can take it into calculations.

Subjective logic is another approach for expressing uncertainty explicitly
in trust values[13]. Trust in subjective logic is shown in the form of an
opinion tuple which is of the form \( \langle b, d, u, a \rangle \) where \( b \) stands for belief, \( d \)
for disbelief, \( u \) for uncertainty, and \( a \) is atomicity which is a fixed \( a \) priori degree
of belief i.e., a way to embed the foreknown success probability of a trusted
party into an opinion. Also, the equation \( b + d + u = 1 \) always holds. As it
is obvious from the equation, as the degree of belief or disbelief grows, the
uncertainty level shrinks. Intuitively, as a trusting party gains more positive
and/or negative experience regarding a trusted party, her belief and/or dis-
belief in that party’s trustworthiness becomes stronger and consequently the
uncertainty in her opinion about him is reduced. This relationship is shown
in the the following definitions between \( r \) and \( s \), which are the number of
successful and unsuccessful experiences regarding the trusted party, and \( b, d, \)
and \( u \):

\[
\begin{align*}
  b &= \frac{r}{r+s+2} \\
  d &= \frac{s}{r+s+2} \\
  u &= \frac{2}{r+s+2}
\end{align*}
\]

\[\iff\]

\[
\begin{align*}
  r &= 2b/u \\
  s &= 2d/u
\end{align*}
\]

In [13], Jøsang justifies a bijection between opinion tuples and beta prob-
ability distribution functions (PDF) which are also a way to represent \( a \)
posteriori probabilities of binary events in Bayesian statistics[12]. A beta
distribution can be denoted by \( \text{Beta}(\alpha, \beta) \) where \( \alpha \) and \( \beta \) are shape param-
eters of the distribution. Figure 3.3 shows some of the instances of beta
distribution. As it can be observed, for the greater \( \alpha \) and \( \beta \), the certainty
around a particular probability value also increases. \( \alpha \) and \( \beta \) are in direct re-
lationship with number of successful and unsuccessful experiments i.e. \( r \) and
\( s \), and hence a mapping between them and opinion tuples exists which is
shown below:
Informally, a beta PDF approximates the uncertain probability of future successful experiments by combining prior knowledge with evidence that is acquired from past observations. It has been reminded in [14] that a beta PDF is defined on a probability variable (as it is depicted in Fig. 3.4). Therefore, a beta PDF is a second order probability. For instance, if we assume that the variable $p$ in figure 3.4 shows the probability of a successful interaction with a node in future, then the beta PDF indicates that 0.8 is about the average probable value for this probability ($p$).

Since we have adopted opinion tuples to be the representative form of trust values, therefore both functional trust and referral trust have corresponding beta PDFs. This facilitates the replica placement decision subsystem with probabilistic analysis capability. For instance, $\int_{p_1}^{p_2} \text{beta}(\alpha, \beta)$ can show the likelihood that a node responds correctly with probability that falls in $[p_1, p_2]$. As a result, this representation helps achieve an important goal of this thesis which is to provide replica placement subsystem with a representation for a node’s availability that not only have probabilistic interpretation but also quantify uncertainty explicitly in a way that in can be taken into
calculations by replica placement subsystem.

Subjective logic includes a set of operators to manipulate the opinions for trust network analysis. Fortunately, since there is a mapping between opinion tuples and beta PDFs, the subjective logic operators provide the means to implicitly modify the parameters of beta curves consistently. Next section uses a related subset of subjective logic operators to approach the trust network analysis and availability calculation problems.

3.2 Availability Calculation

Recall from the previous section that trust transitivity allows calculation of indirect trust of one node in another along arbitrarily long trust paths. However, usually trust networks consist of parallel paths as it is shown in Fig. 3.5 and therefore a rather comprehensive mechanism should include the information from all of these paths in trust analysis and possibly aggregate them into one trust value between two nodes. In this regard, two famous methods are compared here and one that seems more suited to meet the requirements is chosen to inspire the approach to trust network analysis in the thesis.

In [14], Jøsang et al. propose a thorough method of calculating the trust of
Figure 3.5: Parallel paths of trust between two nodes

one node in another according to the paths from the source to the destination in order to offer a precise trust network analysis based on subjective logic opinion tuples and operators. To this end, first all the paths between the source and the target are discovered. Then these paths are combined to construct simplified graphs, called directed series-parallel graphs (DSPG), which are resulted from removing the cycles and dependencies. A DSPG then can be traversed to calculate the trust value of the source in the target. However, there are $2^n - 1$ possible DSPGs construct-able for $n$ different paths, of which the one resulting in the least amount of uncertainty in the final trust value is desirable. Although there are algorithms for recognition of DSPGs in linear time\cite{37, 39} which can be used for finding such DSPGs (and thereby for estimation of trust values), however still there are prohibitive computational burdens in using this technique for large networks.

As the system ages the number of edges increase in a trust network, and thereby number of paths grow exponentially. In this situation, even a linearly complex DSPG recognition algorithm is hindered by the great number of paths\cite{24}. Note that this is only for recognition of a DSPG between two nodes. When global reputation is to be calculated, the opinion of each node on any directly/indirectly connected node must be taken into account which results in DSPG recognition algorithm to be called $O(n^2)$ which obviously increases the computational costs even more.

Therefore, the more efficient trust network analysis technique of EigenTrust\cite{16} is adopted here, in order to combine it with subjective logic opinion tuples and operators to exploit both the efficiency of EigenTrust algorithm and features of subjective logic opinion tuples.

In EigenTrust, the network of trust relationships is expressed as a matrix
Algorithm 3.1 The basic centralized version of EigenTrust algorithm

1 $\overrightarrow{t}(0) = \overrightarrow{c}_i$
2 repeat
3 \[ \overrightarrow{t}(m+1) \leftarrow C^T \overrightarrow{t}(m) \]
4 \[ \delta \leftarrow ||\overrightarrow{t}(m+1) - \overrightarrow{t}(m)|| \]
5 until $\delta < \epsilon$

$C$ (illustrated in Fig. 3.6a) where element $c_{ij}$ is the local trust of node $i$ in node $j$. The trust values represented by elements of the matrix are the normalized local trust values (ref 3.1.2.1). Let $t_i = C^T c_i$ where $c_i$ is the normalized local trust vector of peer $i$ in other peers. If looked closely each element of $t_{ik}$ of vector $t_i$ is calculated according to the formula $t_{ik} = \sum_{j \in J} c_{ij} \times c_{jk}$. This is the same formula that was mentioned in section 3.1.1, and recall that it calculates one level of indirect trust by aggregating opinions of other peers regarding peer $i$. If the above definition of $t_i$ is now extended to $t_i = (C^T)^m c_i$, where $m$ is a positive integer, then the effect will be that paths of trust of length $m$ among nodes are taken into calculation i.e., $m$ level of indirection will be included. The expression $t_i = (C^T)^m c_i$ can be reformulated as a recursive expression of the form $t_i^{(m+1)} = C^T t_i^{(m)}$, an illustration of which is shown in figure 3.6b.

The very basic centralized version of EigenTrust algorithm that uses $t_i^{(m+1)} = C^T t_i^{(m)}$ is shown in algorithm 3.1. For a big enough value of $m$, $t_i$ converges to the left principal eigenvector of $C$ and therefore algorithm terminates. The $t_i$ vector will contain the indirect trust of node $i$ in all other nodes, resulted from aggregating multiple paths of indirect trust until convergence. Results show that this technique is very effective in calculation of nodes reputations. It is also more efficient than the previous technique of trust network analysis as the algorithm converges fast according to[16].

Therefore we rely on EigenTrust trust calculation model to compute the reputation of the nodes. However, instead of using normalized local trust values, local opinions tuples of peers are incorporated. Moreover, instead of using multiplication and addition of elements in matrix/vector products,
\[ C = \begin{bmatrix}
  c_{11} & c_{12} & \cdots & c_{1n} \\
  \vdots & \vdots & \ddots & \vdots \\
  c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix} \]

(a) Trust network matrix representation

\[ \begin{bmatrix}
  t^{(m+1)}_{i1} \\
  \vdots \\
  t^{(m+1)}_{ij} \\
  \vdots \\
  t^{(m+1)}_{in}
\end{bmatrix} = \begin{bmatrix}
  c^{(0)}_{i1} & \cdots & c^{(0)}_{ik} & \cdots & c^{(0)}_{in} \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  c^{(0)}_{j1} & \cdots & c^{(0)}_{jk} & \cdots & c^{(0)}_{jn} \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  c^{(0)}_{n1} & \cdots & c^{(0)}_{nk} & \cdots & c^{(0)}_{nn}
\end{bmatrix}^T \times \begin{bmatrix}
  l^{(m)}_{i1} \\
  \vdots \\
  l^{(m)}_{ij} \\
  \vdots \\
  l^{(m)}_{in}
\end{bmatrix} \]

where

\[ \begin{bmatrix}
  c^{(m+1)}_{i1} = c^{(m)}_{i1} \times c^{(0)}_{i1} + \cdots + c^{(m)}_{ij} \times c^{(0)}_{jk} + \cdots + c^{(m)}_{in} \times c^{(0)}_{in} \\
  \vdots \\
  c^{(m+1)}_{ik} = c^{(m)}_{i1} \times c^{(0)}_{k1} + \cdots + c^{(m)}_{ij} \times c^{(0)}_{jk} + \cdots + c^{(m)}_{in} \times c^{(0)}_{nk} \\
  \vdots \\
  c^{(m+1)}_{in} = c^{(m)}_{i1} \times c^{(0)}_{in} + \cdots + c^{(m)}_{ij} \times c^{(0)}_{jn} + \cdots + c^{(m)}_{ij} \times c^{(0)}_{jn}
\end{bmatrix} \]

(b) Recursive expression of global trust calculation based on matrix of normalized local trust values.

Figure 3.6: Matrix representation of trust network in EigenTrust
Algorithm 3.2 Simple availability calculation algorithm.

1. Let $\vec{c}_i$ be a vector where $c_{ij}$ is the direct referral trust of $i$ in node $j$.
2. Let $C$ be a matrix where element $c_{ij} =$ the direct referral trust of node $i$ in $j$.
3. Let $F$ be a matrix where element $f_{ij} =$ the direct functional trust of node $i$ in $j$.
4. $\vec{t}^{(0)} = \vec{c}_i$
5. for $i \leftarrow 0$ to $N$
   6. do
5. $\vec{t}^{(i+1)} \leftarrow C^T \times \vec{t}^{(i)}$
   7. \(\triangleright\) where $\times$ is an overload of matrix product where multiplication is replaced by subjective logic discount operator $\otimes$ and addition is replaced by average fusion operator $\oplus$
8. for $k \leftarrow 1$ to $\#$ of nodes
10. do
11. $a_{ik} \leftarrow t_1 \otimes f_{1k} \oplus t_2 \otimes f_{2k} \ldots t_n \otimes f_{nk}$
   \(\triangleright\) $a_i$ now contains indirect functional trust i.e. availabilities of all nodes from point of view of node $i$.

Two subjective logic operators are used, which will be explained in the next subsection.

3.2.1 The Basic Availability Calculation Algorithm

The simple version of the availability calculation mechanism is shown in the algorithm 3.2. From line 1 to 6, the algorithm calculates the indirect referral trust of node $i$ in all others. Then from line 7 to 10, the availability of a peer $k$ is calculated as the weighted average of functional trusts of all other nodes in $k$. The weights are the trust values in peers that judge $k$ which were calculated in previous lines. In other words, trustworthiness of the judging nodes weighs the influence of their judgments regarding other nodes' availabilities.

Note that using opinion tuples instead of normalized local trust values, entails usage of subjective logic operators as mentioned earlier. To this end,
two operators from subjective logic are used to replace multiplication and addition operators in matrix product. Namely, \( \text{discount operator} (\otimes) \) [13] for multiplication and \( \text{average fusion operator} (\oplus) \) [40] for addition. The discount operator is used when moving along a trust path to compute the effect of trust transitivity similar to multiplication in EigenTrust (Ref. 3.1.1), while the average fusion operator is used to combine parallel paths of trust in order to come to a consensus of opinions regarding the scope of trust.

More specifically, average fusion operator is applicable when dependent opinions regarding a single object are to be combined e.g. availability of a node at a relatively short period of time, when observed by other nodes, can yield several dependent opinions regarding the node which can be combined using averaging fusion operator\(^8\).

Eventually, lines 7 to 10 calculate the availability by first discounting and then averaging the opinions, in order to compute the weighted average of the opinions of all nodes regarding a node’s availability. Note that opinions of nodes are weighted based on their trustworthiness which is intuitive since opinions of the more trustworthy nodes should be of more effect.

An important matter to note is that the convergence detection condition in line 5 of algorithm 3.1 has been replaced with a bound checking condition in the for-loop of line 3 in algorithm 3.2. This is because here we can not expect the algorithm to converge due to not normalizing trust values as it is done in EigenTrust [24]. However, still the matrix multiplication scheme can be exploited as an efficient mechanism to calculate trust along chains of nodes. Effectiveness of this method is demonstrated empirically in the next chapter.

### 3.2.2 A More Efficient Availability Calculation Algorithm

The algorithm proposed in the previous subsection can be enhanced in terms of efficiency in some aspects. Algorithm 3.3 presented here takes into account the concepts of isolated sub-graphs, and intervals to provide a more efficient

\[\text{cumulative fusion operator} (\oplus)\]

which is applicable to independent events such as observations of a node’s availability in different time periods. We will be assisted by this operator to construct a more efficient version of the algorithm in the next subsection.

\(^8\) There is another fusion operator called cumulative fusion operator (\(\oplus\)) which is applicable to independent events such as observations of a node’s availability in different time periods. We will be assisted by this operator to construct a more efficient version of the algorithm in the next subsection.
Algorithm 3.3 The more efficient version of the availability calculation algorithm

\textbf{AVAILABILITYCALC}(t : \text{current interval})

1. Let \( \vec{r} \) be a vector containing the all-time reputations of the nodes.
2. Let \( C \) be a matrix where element \( c_{ij} = \) is the direct referral trust of node \( i \) in \( j \).
3. Let \( F \) be a matrix where element \( f_{ij} = \) is the direct functional trust of node \( i \) in \( j \).
4. Let \( P \) be the set of isolated sub-graphs containing only nodes that have interacted in the network during the current interval \( t \).
   \( \triangleright \) The following loop computes the temporal reputation of nodes in current interval \( t \) and updates their all-time reputions in vector \( \vec{r} \) using cumulative fusion operator \( \oplus \).
5. \textbf{for} each isolated sub-graph \( p \in P \) \textbf{do}
6. \hspace{1em} \( \triangleright \) The following loop initializes trust vector \( q^{(0)} \) with average trust values of nodes in \( p \) about each other. This is done because here we assume the algorithm is run in a central server which does not have any initial estimation of the trust of nodes thus it has to rely on the average opinion of the society that is \( p \).
7. \hspace{2em} \textbf{for} \( k \leftarrow 1 \) to \# of nodes in \( p \) \textbf{do}
8. \hspace{3em} \( q^{(0)}_k \leftarrow c_{1k} \oplus c_{2k} \oplus \ldots \oplus c_{nk} \)
9. \hspace{4em} \( \triangleright \) where \( c_{ik} \) is the direct referral trust of node \( i \) in node \( k \).
10. \hspace{2em} \textbf{for} \( i \leftarrow 1 \) to \( N \) \textbf{do}
11. \hspace{3em} \( q^{(m+1)} \leftarrow C^T_p \times q^{(m)} \)
12. \hspace{4em} \( \triangleright \) where \( \times \) is an overload of matrix product where multiplication is replaced by subjective logic discount operator \( \otimes \) and addition is replaced by averaging fusion operator \( \oplus \).
13. \hspace{2em} \( \triangleright \) Now the all-time reputations of nodes in \( p \) are updated with their corresponding reputations calculated for the current interval \( t \).
14. \hspace{3em} \( r^*_p \leftarrow r^*_p \oplus q \)
15. \hspace{2em} \( \triangleright \) The availability is calculated by averaging opinions of all nodes weighted by their reputations.
16. \hspace{3em} \textbf{for} \( k \leftarrow 1 \) to \# of nodes \textbf{do}
17. \hspace{4em} \( a_k \leftarrow r_1 \otimes f_{1k} \oplus r_2 \otimes f_{2k} \ldots r_n \otimes f_{nk} \)
alternative to using algorithm 3.2.

One can observe that the algorithm 3.2 computes a chain of matrix products for every node that requires an estimation of its partners' availabilities. Since this algorithm is to be run on central servers, it can put a lot of burden on them since it is run for every peer separately. Therefore, it is of convenience if an algorithm can compute the availabilities globally, for all the nodes, at one run.

Another notable matter is the invocation frequency of the algorithm. The algorithm 3.2 is invoked at central servers instantly after any node reports its local opinions. Obviously, the more frequently the algorithm is invoked the more precise the results will be. However, there is a trade-off between invocation frequency and efficiency. In other words, sometimes it might be more desirable to sacrifice precision a bit to gain much higher efficiency. As a result, the previous algorithm is modified here to be invoked only whenever a certain time interval has elapsed. In other words, the life cycle of the system is divided into intervals and the availability calculation takes place at the end of each interval.

The algorithm 3.2 also lacks consideration for isolated sub-graphs i.e., connected components in a network of trust. Put simply, as a network of trust is being formed among the nodes, not all the nodes interact with each other. Thus, the network might be decomposable into isolated sub-graphs where nodes in each sub-graph have interacted only with the nodes in their own sub-graphs. Therefore, it is possible to reduce the complexity by running the algorithm for each sub-graph separately because there are no two nodes from separate sub-graphs that have effective opinions about each other.

However, as the system ages sub-graphs start merging into bigger sub-graphs and eventually the whole network. As a result, the complexity resurges. To handle this situation, the history of interactions is reset at the end of each interval. This way new sub-graphs emerge during each interval and disappear at the end of them and thus an increase in complexity is prevented on average. Since availabilities are calculated for a different set of sub-graphs and at different times, therefore, the result of the availability calculation for distinct intervals can be considered as independent observations of node’s availabilities. Hence, the cumulative fusion operator (⊕) that has been suggested for
accumulating independent opinions by [40] can be exploited here to combine the availabilities calculated at different intervals into all-time availabilities of the nodes.

Note also that every time the first algorithm 3.2 is run it calculates the ‘personal’ trust of one particular node such as $i$ in other nodes. Accordingly, the initial trust vector in that algorithm is composed of local trusts of $i$ in others. However, the algorithm 3.3 is run once every interval for all the nodes. Therefore, the ‘global’ trust values are being calculated here instead. Consequently, the initial trust vector i.e., $q^{(0)}$ in matrix multiplication, is composed of the average of opinions of nodes, within their sub-graphs, about each other. Next section compares the complexities of the algorithms.

### 3.2.3 Complexity of the Algorithms

The following are the main parameters that affect the complexities of the previous algorithms:

- $n$ is the total number of nodes in the network which we assume to be fixed throughout the life of the system.

- $f$ is the total number of the feedbacks reported from all $n$ nodes in the lifespan of the network.

Both of the aforementioned algorithms have $O(n^2)$ memory complexity since they both store direct local trusts of nodes in the form a two-dimensional matrix. However, depending on the transaction rate and number of nodes such matrices can be more sparse since not all the nodes interact with each other during the life-time of the network.

Since algorithms are run on central servers the only communication overhead is due to nodes reporting their feedbacks to servers. Therefore, communication complexity is linear with regards to the total number of feedbacks reported by the nodes i.e. $O(f)$.

Before calculating and comparing the computational complexity of the algorithms, it should be noted that the more efficient algorithm 3.3 exploits some implicit characteristics of the input, which should be made explicit in order to correctly calculate the complexity and compare it with the rather
inefficient version (algorithm 3.2). Hence, the main input parameters are redefined here in terms of more fine-grained parameters which can effectively express the aforementioned properties of the input:

- \( n \), the number of nodes, can be defined as \( n = \sum_{i=1}^{\psi(t)} v_i(t) \) where \( \psi(t) \) is the number of isolated sub-graphs at interval \( t \), and \( v_i(t) \) is the number nodes in sub-graph \( i \) which is formed at interval \( t \). Clearly, we have \( n \leq \psi(t) \times v_{max} \) where \( v_{max} = \max_{1 \leq i \leq \psi(t)} v_i(t) \).

- \( f \), the total number of reported feedbacks, can be reformulated as \( f = \sum_{t=1}^{T} \phi_t \) where \( T \) is the total number of intervals in the lifespan of a network and \( \phi_t \) is the number of feedbacks that are reported during interval \( t \). Also, note that \( f \leq T \times \phi_{max} \) where \( \phi_{max} = \max_{1 \leq t \leq T} \phi_t \).

Without loss of generality, we analyze the computational complexity of algorithms in a single interval by assuming \( \psi(t) = \psi \), and \( v_{max} = v_{max} \) for all \( t \). The results are then trivially extensible to the whole lifespan of the system. Also, we assume \( \phi_{max} = c \times \psi \times (v_{max})^2 \) to take into account the worst-case scenario where all the nodes in each sub-graph give feedback about each other.

For the basic version of the algorithm, each invocation costs almost \( N \times n^2 \) where \( N \) is the constant number of loop iterations, and \( n^2 \) is inflicted by the product of matrix and vector in the loop. But the algorithm is invoked each time a feedback report is received from any node. Therefore, the total processing time required by algorithm 3.2 during an interval is:

\[
T_1 = \phi \times N \times n^2 \leq \phi_{max} \times (\psi \times v_{max})^2 \implies T_1 = c \times \psi^3 \times v_{max}^4 = O(\psi^3 \times v_{max}^4)
\]

The more efficient algorithm 3.3 conducts the loop of \( N \) matrix and vector product iterations only for each isolated sub-graph. Also, it is run once every interval and thus the loop invocations number does not depend on \( \phi \). However, still to keep track of creation and consolidation of sub-graphs during the interval all the \( \phi \) feedbacks are processed. Therefore, the computation
time required by algorithm 3.3 is:

\[ T_2 = N \times \psi \times (v_{\text{max}})^2 + O(\phi_{\text{max}} \times \alpha(v_{\text{max}})) = N \times \psi \times (v_{\text{max}})^2 + O(\psi \times (v_{\text{max}})^2 \times \alpha(v_{\text{max}})) \]

The term \( O(\phi_{\text{max}} \times \alpha(v_{\text{max}})) \) is for incrementally updating the data structure that keeps track of the isolated sub-graphs as feedbacks are being received. Isolated sub-graphs can be maintained with a disjoint-set data structure (Ref. chapter 21 in [46]) which is a collection of disjoint sets. Each such set can represent an isolated sub-graph in the network. There are operations defined for this data structure to update or merge disjoint sets incrementally at a very low cost of \( O(\alpha(v_{\text{max}})) \). It can be shown that \( \phi_{\text{max}} \) invocation of such operations for input of size \( v_{\text{max}} \) costs \( O(\phi_{\text{max}} \times \alpha(v_{\text{max}})) \) which practically is equivalent to \( O(\phi_{\text{max}}) \) that is negligible in our algorithm since it is of the same complexity as the main loop of the algorithm. Therefore, the complexity of algorithm 3.3 is:

\[ T_2 = O(\psi \times v_{\text{max}}^2) \]

Note that the above complexities hold for instances of networks where on average in each interval the network is decomposable into \( \psi \) separate subgraphs where \( v_{\text{max}} \) is asymptotically smaller than \( n \). This assumption is likely to be valid for large enough networks and short intervals. In other words, it seems more likely that the interactions of nodes, within a large network and during a short interval, to be confined to groups of partners smaller than the whole network. However, in a pessimistic worst-case scenario in which all the nodes interact during an interval, we have \( \psi = 1 \) and \( v_{\text{max}} = n \) therefore:

\[ T_1 = O(n^4) \]

\[ T_2 = O(n^2) \]

---

\( \alpha() \) is a very slowly growing function which is the inverse of the Ackermann function that on the other hand grows very rapidly.
The computational complexity of the improved algorithm is still superior. Because, it is invoked once per interval as opposed to the first version of the algorithm which is invoked $O(n^2)$ at worst.

Next chapter concludes the thesis by empirically demonstrating the effectiveness of the aforementioned availability calculation algorithms. Also, it will briefly describe some of the more important aspect of the actual implementation of the system. Finally, it ends by pointing out problems that can be addressed in the future.
Chapter IV

Conclusion

This chapter first starts by empirical evaluation of the availability calculation algorithms by comparing them to EigenTrust algorithm, and the naïve algorithm that assumes all the nodes are equally available. After that, section 4.2 briefly explains about the actual implementation of the algorithm in the product that this thesis has been a part of. Last sections of this chapter summarize the paper and provides some pointers for possible future works.

4.1 Evaluation of the Algorithms

To evaluate the algorithms suggested in the thesis, an open source simulator from [24] has been used which originally was targeted at reputation management systems. The original version of the simulator had already supported the data authenticity notion for the purpose of reputation and trust evaluation. It further has been modified here, in order to simulate various degrees of availabilities for nodes also.

4.1.1 Behaviors of the Nodes in Simulations

In our experiments each node is one of the following types: honest; malicious content provider; purely malicious; or feedback skewing[24]. Honest nodes, when requested, provide authentic data with high probability.

On the other hand, both types of malicious content provider and purely malicious nodes most likely deliver malicious/corrupted data. Purely malicious nodes are even more evil than malicious content providers, because they also usually give dishonest feedbacks regarding availability or reliability of other nodes, while malicious content providers are honest in their feedbacks about other nodes. In other words, malicious content providers can
resemble faulty nodes that corrupt the data.

Feedback skewing peers provide authentic data but they are also dishonest in their opinions about other nodes. Feedback skewing peers gain high reputations by providing authentic data, but they betray the system by damaging the reputation of honest nodes and praising the other malicious nodes. As it was mentioned previously in section 3.1.2, disposing of feedback skewing nodes directly, is not trivial. But, presence of a large enough set of honest nodes in the network should reduce their nuisance effect.

To completely model our definition of availability in the simulator, nodes have been modified to fail to commit their transactions once in a while. Nodes of each type are divided to equal proportions with low, medium, and high levels of availabilities in each simulation. This can simulate the situations in which a node can not respond or behave correctly due to technical failures or absence from the network.

4.1.2 Partner Selection

An effective way to demonstrate the precision of the algorithms in calculation of the availabilities of the nodes is their ability in finding the nodes with the highest availabilities. Therefore, experiments have been conducted by incorporating an elitist partner selection strategy. In other words, at each step of a simulation, only the node with the highest availability is selected by the simulator for a new transaction. For the simulator to select the best node, it relies on the outcome of an availability calculation algorithm. As a result, the algorithm that calculates the availability of nodes more closely to their “real” values, provides a better estimation of who the best node is among all others.

The algorithms 3.2, and 3.3 have been compared to EigenTrust\(^1\), and the naïve algorithm that assumes all the nodes are equally available all the time.

The elitist strategy requires a ranking among the nodes according to their

---

\(^1\)Please note that a modified version of EigenTrust is used here to adopt the notion of availability as it is defined in the context of this thesis. More precisely, here, EigenTrust computes the reputations of nodes which are then used as weights in computing the average opinions of nodes about each others’ availabilities. Similar to the approach in our availability calculation algorithms.
availabilities. The trust values in EigenTrust already provide this ranking. However, since the outcome of our availability calculation algorithms are in the form of opinion tuples (or equivalently beta PDFs), therefore, probability expectation of opinions are used as $a$. That is:

$$E(op) = b + au$$

Where \( b \) is belief, \( a \) is atomicity i.e. a priori trust, and \( u \) is uncertainty of the opinion tuple \( op \) (Ref. section 3.1.2.1). Note that when there is little belief in a node and too much uncertainty about it due to lack of previous interactions with that node, the a priori trust value of \( a \) determines how much that node can be trusted. This explains an interesting phenomenon in the first experiment that we shall study shortly.

### 4.1.3 Evaluation Metric

The experiments compare the performance of naïve algorithm, EigenTrust, and our availability calculation algorithms according to the following evaluation metric:

$$\text{Metric} = \frac{s}{t}$$

Where \( s \) is the number of successful transactions by honest nodes, and \( t \) is the total number of transactions attempted by honest nodes. Success in a transaction requires that the partner of the honest node to be available and provide authentic data. Since the elitist partner selection strategy relies on the estimations of an availability calculation algorithm to select the most highly available partner for a transaction, therefore, the performance of the algorithm is in direct relation with the value of the metric. In other words, the better an algorithm performs the higher the value of the metric will be.

### 4.1.4 Experiments

Experiments are conducted for low, medium, and high ratios of transactions to nodes. Obviously, the higher the ratios the more feedbacks from nodes are gathered. Different transaction-node ratios can demonstrate the performance
of algorithms in various stages of a network life-cycle. For instance, low ratios can resemble the early stages of the network, when still not too many interactions have taken place between the nodes and so forth.

A requirement of our backup system is to have equal \textit{a priori} trust values for all the peers i.e. $a = 0.5$ in all the opinion tuples about all nodes. In other words, our algorithm should do reasonably well in networks without any pre-trusted nodes present in them, although existence of pre-trusted nodes might enhance the precision of our availability calculation algorithms. It is worth mentioning that the dependence on pre-trusted nodes introduces some vulnerabilities to the system which are mentioned in [41]. Also, it demands computational and administrative resources to have a set of pre-trusted nodes available all the time in the network which is not convenient.

Nonetheless, performance of EigenTrust depends heavily on existence of pre-trusted nodes in the network. Therefore, some of the experiments bring a reasonable number of pre-trusted nodes\footnote{In opinion tuples regarding pre-trusted nodes, $a$ equals 1 all the time.} into the network in order to provide the fair grounds for comparison of the algorithms.

Accordingly, four major experiments have been performed for different types of malicious configurations. Each experiment consists of several simulations for the aforementioned availability calculation algorithms executed on the same set of transaction traces and using the same random seeds in order to have fair comparisons. Simulations in all experiments are categorized into nine different configurations according to transaction-node ratio levels and the percentage of pre-trusted nodes in the network.

4.1.4.1 First Experiment: In The Presence of Malicious Providers

In the first experiment nodes of the network are either honest or malicious providers. Figure 4.1 shows result of test runs for different simulation configurations. In all of the graphs the percentage of malicious providers grow from zero to eighty percent of the total number of nodes in the network. From the trends of the graphs, it can be clearly seen that the performance of all of the algorithms deteriorate when the percentage of malicious nodes
increases which is inevitable. However, we can see that both of our algorithms and EigenTrust outperform the naïve algorithm by a large margin. This confirms that the idea of estimating the availability of nodes based on their own opinions is promising.

It is worth noting that at the point where there are zero malicious providers in the network, few transactions may fail due to inauthentic data exchanges. Nonetheless, the naïve approach still performs poorly compared to other three algorithms according to the evaluation metric. This is because even when there are almost no inauthentic data downloads, still honest nodes might fail to stay available in the network. Therefore, the gap between performance of the naïve approach and the other algorithms at point zero, shows how a network can benefit from such availability calculation algorithms even under the unrealistic assumption that no node corrupts data.

Comparing Performance of our algorithms with EigenTrust in figure 4.1, we can observe that EigenTrust is outperforming our algorithms by a narrow margin. However, when there are pre-trusted nodes in the network, or when the transaction-node ratio grows, the gap fades away.

First thing to note is that EigenTrust can perform well without pre-trusted nodes only when there are no dishonest feedbacks. In fact, the pre-trusted nodes are beneficial because they can effectively neutralize the dishonest feedbacks. Therefore, it is not surprising that EigenTrust in this experiment is performing so well without existence of pre-trusted nodes. Later experiments show that when there are no pre-trusted nodes in the network, dishonest feedbacks severely damage the performance of EigenTrust.

The reason that our algorithms in this experiment are not performing as well as EigenTrust is that when there are low numbers of transactions the uncertainty($u$) values in opinions are rather big and accordingly belief($b$) and disbelief($d$) values are small. In the extreme case $u \approx 1, b \approx 0, d \approx 0$. In addition, when there are no pre-trusted nodes, it means that $a \text{ priori}$ trust value $a$ is 0.5 for all the nodes. Therefore, the probability expectation of opinion tuples which is used for ranking the nodes is close to 0.5 and thus the distinction between good and bad nodes is blurred. On the other hand,

\[ \text{In the graphs, } A vlCalc\text{Intlvl represents the more efficient algorithm 3.3, } A vlCalc \text{ is the basic algorithm 3.2, } Eigen \text{ is EigenTrust, and None represents the naïve algorithm.} \]
for EigenTrust the trust value is the proportion of successful transactions to all transactions which can sharply distinguish the difference between two nodes even with very low number of transactions.

Also, note that the probability expectation of opinions regarding a pre-trusted node in the lack of evidence is \((b = 0) + (a = 1) \times (u = 1) = 1\). In other words, when the evidence is sparse, pre-trusted nodes are ranked as the best nodes by our algorithms and thus can successfully provide some of the authentic data that is required by other nodes. This explains the reason behind enhancement in performance of our algorithms when pre-trusted nodes percentage increases.

However, this shortcoming is trivial because it happens at very early stages of the system and when the number of transactions are very low and as the system ages and number of transactions increase, our algorithms catch up to EigenTrust. Furthermore, in real systems it might be desirable not to trust nodes that have had low number of transactions e.g., new comers or inactive nodes, even if they have shown to be highly available in those few transactions.

4.1.4.2 Second Experiment: In The Presence of Purely Malicious Nodes

When dishonest feedbacks exist in the system due to purely malicious nodes and there are no pre-trusted nodes in the network, the performance of EigenTrust is deteriorated and our algorithms overtakes. As the system ages and the number of transactions increases, EigenTrust loses its ability to distinguish between good and bad nodes more and more. However, our algorithms seem to act more robustly in this situation. Although when pretrusted peers are added, at lower numbers of transactions EigenTrust wins by a small margin. Again it can be observed that gap is closed when the number of transactions increases.

Before proceeding to the next experiments, note that the more efficient algorithm 3.3 while boosting the efficiency, has not sacrificed the precision compared to algorithm 3.2 and their perform very closely according to the evaluation metric. Therefore, in the next two experiments, the performance of the more efficient algorithm is shown only.
Figure 4.1: Performance of the algorithms in presence of malicious content providers
Figure 4.2: Performance of the algorithms in presence of purely malicious nodes
4.1.4.3 Third Experiment: In The Presence of Feedback Skewing nodes and Malicious providers

When there are malicious collectives that are composed of feedback skewing nodes and malicious providers, the performance of EigenTrust is punished severely as it can be seen from the graphs. Note that our algorithm is performing very robustly in this situation compared to EigenTrust. Although as the number of pretrusted peers increases performance of EigenTrust also enhances, but still it is not performing as well as our availability calculation algorithm.

4.1.4.4 Forth Experiment: In The Presence of Feedback Skewing nodes and Purely Malicious Nodes

As we can observe from the graph the problem is escalated even more for EigenTrust when the malicious provider nodes are replaced with purely malicious nodes. When pre-trusted peers are absent the sheer number of dishonest feedbacks from both groups of feedback skewing and purely malicious nodes has inverted EigenTrust’s perception of nodes’ availabilities to the point that it performs worse than the naïve algorithm.

On the other hand, note that in this and previous experiment, it seems that the performance of our algorithm is even better than the first two experiments. In the last two experiments the number of malicious nodes is divided equally between group of feedback skewing nodes and purely malicious/malicious provider nodes. As a result, there are half the number of nodes that provide inauthentic data as in the first two experiments. It shows that our algorithm and EigenTrust are harmed almost equally from inauthentic data, but the former is more robust to dishonest feedbacks.

The reason is that the discount operator of subjective logic generally tends to increase the uncertainty. In other words direct trust has more certainty than transitive trust. When trust paths are combined using fusion averaging operator, the certain paths have more weight due to the nature of the operator. As a result, honest nodes’ direct opinions are more persistent.

---

4 Recall that feedback skewing nodes always propagate authentic data.
Figure 4.3: Performance of the algorithms in presence of feedback skewing nodes and malicious content providers nodes
in trust analysis of our algorithm than in EigenTrust. Therefore, the former performs well despite numerous of dishonest feedbacks.
Figure 4.4: Performance of the algorithms in presence of feedback skewing nodes and purely malicious nodes.
4.2 Implementation Notes

The availability calculation algorithm is integrated into an underdevelopment closed-source commercial peer-to-peer system. There are central servers in the system which are responsible for authentication, authorization, and accounting. Moreover, servers have the responsibility of replying to the queries of peers regarding to/from which other peers they can backup/restore their data.

A peer starts a backup operation by requesting a central server for a set of partner nodes that can share their storage for the backup process. The responding server relies on a replica placement decision subsystem to select a set of peers for a backup operation. The selection is based on various criteria including quota of the requesting peer and properties of partner peers such as their bandwidth, idle storage space, availability, etc.

As it was mentioned earlier, availabilities of a set of candidate partner peers, for a backup operation, provides the necessary input for calculating the risk of their failure for a successful restore operation. To this end, the availability calculation algorithm is implemented such that it is invoked periodically after a certain interval e.g., one hour, is passed. During an interval peers aggregate their interactions outcomes to form functional or referral opinion tuples about each other. At the end of an interval, nodes send their opinions to the servers so they can be used for availability calculation.

The network delays are unbounded and also it is difficult to precisely synchronize to a global time by common consent among peers specially in a RESTful client/server architecture. Therefore, synchronization between peers and servers is loosely based on interval “tickets”. More precisely, a server publishes a unique token that is only valid for one interval throughout the whole lifespan of the system. When a node needs to send its opinions about other nodes to a server during an interval, it includes the token for that interval in its message, so the server can distinguish which interval the feedbacks belong to. Then half way through the new interval, the servers run the availability calculation algorithms for the last interval. The delay is there to count for the possibility of latent messages from peers.

The system allows a user to run arbitrary number of peers (possibly on
different machines). In some peer-to-peer systems, this feature allows an adversary to launch many peers to lead various types attacks e.g. by large scale propagation of inauthentic data and/or dishonest opinions. Furthermore, the adversary can replace old instances which have lost reputation in the system, due to misbehavior, with new ones. This types of attack is commonly referred to as Sybil attack in literature.

To make this attack nearly impossible in our system, creating new user identities is accompanied with the computationally challenging task of decoding a text from an image. Also, each peer that a user runs in the network is identified by a public key certificate that is issued by one of central servers in the network so that other peers can verify its validity. Each certificate includes an anonymous identity that is used as a reference in the interactions of peers and also when giving feedbacks to the central servers about other peers. The user’s original credentials are bounded to the certificate IDs of user’s peers in a one-to-many relation at the serves. As a result, a user can be rewarded or punished, based on the behavior of their peers in the network., by increasing or decreasing their quota, banning them from the network, etc.

4.3 Summary

This thesis attempted to concoct a novel method for calculating the availability of nodes in a peer-to-peer backup system. To reduce the communication complexity, availability calculation mechanism relied only on feedbacks of nodes themselves regarding each others’ availabilities, resulted from nodes’ ordinary interactions. This required the algorithm to sustain not only the uncertainty in the data resulted from inconsistency in nodes samplings, but also, the dishonest opinions of malicious nodes which tried to subvert the system.

To quantify the uncertainty in sampled data, the Beta probability distribution functions were chosen to express availability of nodes in terms of uncertain probability estimates. The explicit representation of uncertainty in availabilities can enhance the failure risk calculation for a set of nodes, which in turn assists in strategizing more flexible and/or reliable replica placement configurations.
To deal with dishonesty in feedbacks from malicious nodes, concepts from famous reputation/trust management systems were studied, compared, and adopted in order to devise and implement a novel and efficient availability calculation mechanism. In particular, the mapping between Beta probability distribution functions and opinion tuples of subjective logic allowed using the operators of that logic on top of the efficient vector-matrix multiplication model from EigenTrust to devise the basic version of our availability calculation algorithm. Then, an efficiency were introduced into the algorithm based on notions of time intervals, and graph connectivity.

Simulations were conducted for comparing the new method with EigenTrust by incorporating the elitist strategy to select the nodes with the highest availabilities. The results indicated great success in spotting nodes with highest availabilities in presence of various types of malicious nodes. In particular, the new algorithm showed to be considerably more robust to dishonest feedbacks compared to EigenTrust. Moreover, having used Beta PDFs i.e., subjective logic opinion tuples, the outcome from the new algorithm represent a probabilistic measure for node's availabilities which can be used for probabilistic analysis for replica placements. Finally, the algorithm was integrated into an under-development commercial hybrid peer-to-peer application.

4.4 Future Work

There are interesting problems that can be addressed in the current system. For instance, the algorithm proposed in this thesis has not been tested in a real network (because the software product is not still in a functional stage). It is of great importance to observe the performance of the algorithm in a real network and see how much the availability metric is effective for finding the best nodes.

The problem of correlated failures is another issue that has not been studied in this thesis. For instance, a set of machines might fail together due to propagation of virus contaminated files in the network, or a natural catastrophe. A more commonplace scenario is that users residing in a particular timezone shut down their computers at night. The algorithm proposed in this thesis calculates availabilities of nodes once every interval e.g. every hour.
Therefore, the output of our algorithm not only can be used for discovering nodes’ failures association patterns but it also contains temporal information about availabilities of nodes which can be exploited by sequential patterns discovery to predict future behaviors of nodes.
Bibliography


