GNUGGIES: A PROPOSAL FOR HOSTING RESILIENT STATELESS SERVICES USING UNTRUSTED NODES

BY

HARSHIT AGARWAL

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Professor Indranil Gupta

ABSTRACT

This thesis outlines a proposal for a serverless cloud compute system hosted on untrusted nodes. We call this proposed system "GNuggies". It is designed to feel instantly familiar to existing serverless offerings such as AWS Lambda or Azure Cloud Functions. The key difference between GNuggies and existing offerings is that GNuggies proposes leveraging spare compute resources by allowing anyone to contribute nodes into the system. These contributed nodes must be treated as untrusted and this is where the bulk this thesis's contributions arise:

1. A proposed architecture that adapts well understood Distributed Systems concepts to situations involving untrusted nodes and to run in the absence of central authorities.
2. A proposed system wherein actors choose to contribute spare compute and are effectively incentivized to do so.
3. An incentive structure that makes actors less willing to behave in a malicious manner.

This thesis discusses the methods to be used and evaluates their strengths and weaknesses. It also argues that *decentralized serverless* is a direction that the internet will potentially benefit from moving towards.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Indranil Gupta for his guidance and support, making sure I had everything that I needed to succeed. I would also like to thank my colleagues, Abhishek Modi and Evan Fabry, who worked with me on this project and helped with its formulation.

Finally, I would like to thank my family and friends for always being there for me and providing me with support whenever I needed it.

Dedicated to the ideals of freedom of speech and information

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) solutions are very popular today as they enable people to host services in a relatively simple manner. Offerings range from the extremely robust like Google Cloud Platform [1], Microsoft Azure [2], and Amazon Web Services [3], to the extremely simple like Heroku [4], Docker Cloud [5], and Parse [6]. On top of this, services like AWS have certain tools, such as AWS Lambda [7], that abstract any sort of infrastructure and scaling tasks from the end user, thus letting the developer focus only on on their application.

These solutions are excellent for most services, however we find that they often are unacceptable for certain kinds of services. Examples of these services include protest websites such as WikiLeaks and content sharing websites such ThePirateBay. This is largely for social reasons, such as disagreements with authoritarian governments or litigation due to negative financial impact on certain industries, rather than technological ones. A high profile example of such a reason can be seen in 2010 when AWS refused [8] to continue hosting Wikileaks at the request of the US government. Another major example is the recent repeal of net neutrality in USA, which allows ISPs to discriminate between different types of content, going against the ideal of freedom of speech and information. We believe that such services have the potential of creating enormous social good. We would therefore like to afford IaaS-like solutions to these services. To solve this, we propose GNuggies, a framework for hosting services in a decentralized manner on untrusted nodes. The name GNuggies is inspired by GNutella [9], one of the first popular public peer to peer

systems, and pays homage to my favorite snack - chicken nuggets. The nodes on which GNuggies is run is sourced from crowds - this is similar to the manner in which GNutella [9], Torrent [10], Tor [11], IPFS [12], BOINC [19], and more recently, Vectordash [51] operate. They all require volunteers to be willing to host data or services and to have an active and reliable network connection in order to make the data and services available to users requesting them.

The **goals** for GNuggies are:

1) The **framework should be resilient.** Due to the inherent nature of many of these services (protest websites, whistleblowing web pages, etc.), there will often be large groups of people or agencies that will attempt to take these services offline. Resilience to various different types of attacks is of utmost importance. We'd like to ensure that every service hosted on GNuggies stays up so long as there are users who want the service to exist.
2) **Reduce the liability** held by creators of the service as well as for anyone who volunteers as a host on GNuggies. This is to help remove factors that may discourage people from creating or hosting services. This point is particularly relevant in context of a global increase in authoritarian governments and censorship, and people's fear of being caught supporting certain targeted services.
3) **Provide volunteers with an incentive** to stay in the network and to maintain truthful behavior for as long as possible. Firstly, to get people to join the network is a challenge, because typically they would be more inclined to mine common

cryptocurrencies or to instead just turn their computers off and save energy costs. GNuggies needs to be a better alternative than both of these. Secondly, many people will want to leave GNuggies after having made some profit from it. Perhaps some ill-intentioned people would also want to maximize their profits without actually doing the tasks assigned to them (not actually run the service as intended, or return malicious responses to requests). Both of these behaviors are things that we want to disincentivize so that GNuggies can remain and efficient and safe platform at all times.

4) **Maintain compatibility with existing interfaces for both, service creators and service consumers.** This includes two parts. The first is to present service creators with an interface that is compatible with IaaS offerings. This is so that creators of a service do not have to make any changes to work with GNuggies. This is important because the failing of many similar systems is that they deviate too much from other existing solutions and create a steep learning curve that deters new users. The second part of this objective is to ensure that the hosted services can serve familiar protocols such as http. This is because many existing solutions such as Tor and IPFS [12] serve using special protocols [11, 12] and thus limit the audience of the service to only those who are technically adept. Designing a platform that can run on public hosts while still maintaining compatibility with common web technologies (without requiring any additional work from clients) is relatively unexplored.

Part of achieving these goals was to propose a novel proof of work protocol for volunteer nodes. The proposed verification system for proof of work is designed with inspiration from other related works that are detailed in the next chapter. A simple gist of the verification mechanism in GNuggies is that it is a adaptation of Byzantine Fault detection using state machine replicas. This proposal also makes simple assumptions about strong node identities and a reliable communication platform, something that is common in literature.

1.1 Contributions of this Thesis

In this thesis, we introduce the concept for GNuggies, and attempt to cover use cases, security guarantees, associated risks, and potential incentives for participation in the proposed system. In particular, the contributions of this thesis are:

1. A proposal for a censorship resistant, **decentralized web service hosting architecture** - GNuggies.
2. The design for a **novel verification mechanism** for proof of work in GNuggies.
3. Establishing potential **attack strategies** for the proposed architecture.
4. A **formal analysis** of resilience, security and reliability claims made throughout the thesis, as well as an analysis of the limitations the proposed system.
5. A detailed **incentive structure** to maximize truthful participation by volunteers on GNuggies.
6. A complete list of potential future improvements to the proposed design.

1.2 Outline of this Thesis

1. Chapter 2 discusses other similar research efforts in this domain and related domains of decentralized information and trusting the untrusted.
2. Chapter 3 details out the attack models that GNuggies provides resilience to, by both, individuals and agencies.
3. Chapter 4 gives a full overview of the architecture and design GNuggies, as well as internal protocols and services.
4. Chapter 5 explains the verification mechanism for proof of work in detail, and lays out available options for verification, as well as their pros, cons, and scenarios for use.
5. Chapter 6 provides a formal analysis for many of the claims made throughout the thesis regarding the capabilities of GNuggies.
6. Chapter 7 analyzes a feasible incentive scheme to attract volunteers for GNuggies, and to make them stay as trusted members of the platform.
7. Chapter 8 goes on to conclude this paper and lists out potential avenues for future work in this domain.

CHAPTER 2: REVIEW OF RELATED RESEARCH

This chapter presents an overview of similar work from related domains that has helped inspire some of the solutions proposed in this paper. There is a large body of work pertaining to systems aimed at reducing censorship, the various ways to utilize volunteer compute, as well as maintaining reliability and security in distributed systems that contain untrusted or undependable machines.

2.1 Censorship Resistant Systems

Our primary motivation with GNuggies was to build a reliable service hosting system that makes services resilient to censorship. With a constant stream of censorship on media of all sorts, whether it was with WikiLeaks, or the censorship that journalists have to face in oppressive regimes, it is critical for social welfare that this content be kept available for the general public to see. Creating content and hosting this content, both become risky tasks in such scenarios, and there has been plenty of work focussed on building systems to limit censorship or in some way aid the persistence of such critical content. Systems like Dispatch[13] and Publius[14] tend to focus on how to build a publishing system that is anonymized and censorship resistant in some ways, but they both have drawbacks and limitations, since they are focussed on a single domain. Tangler takes a very interesting approach to resists censorship using document entanglements, but the focus there is more on data preservation as opposed to data sharing. Services such as Tor have made it possible for websites to be “onionized” [15] in order to combat censorship efforts [16]. Other

similar solutions include Freenet[18], which was based on a system originally proposed in [17], as well as IPFS, which is primarily a decentralized file storage platform. Both of these solutions are entirely peer to peer network based, and they all require specific browsers, as well a fair amount of domain knowledge. This thesis attempts to improve upon this by creating a generalized system that allows for the hosting of anything that the user wishes to host, while making it extremely simple to do so, and making the hosted services available without any extra technology needed.

2.2 Utilizing Volunteer Compute

There are many projects that aim to use volunteer compute or volunteer machines in order to achieve various different tasks, such as anonymity, security of data, calculations and simulations for scientific research, etc. Some such work was discussed in **2.1** as well, since Tor, IPFS, and Freenet all rely on people joining the network and contributing storage space, network capabilities, and compute. BitTorrent also operates in a similar manner, and needs people to remain on the network, constantly making files and data available for download by other machines, in order for the network to remain functional.

There is no doubt that these technologies create a tremendous amount of impact in combatting censorship and causing social good. However, there are also some other volunteer compute technologies with an entirely different objective. Some technologies like BOINC [19] and SETI@Home [21] aim to use publicly provided volunteer compute to carry out massive scientific undertakings [22] and have led to a lot of progress in atmospheric sciences, geological sciences, astronomy, and particularly the search for extraterrestrial life.

A lot of modern blockchain technologies also rely on volunteered machines in the network to provide the basis for their consensus protocols.

Aside from BOINC and SETI@Home, there are is a vast body of work focussed on volunteer computing and grid computing, and the various ways that grid computing infrastructures can be used as clouds and storage infrastructures [20, 23-26]. There are many large systems such as PlanetLab [53] and the Globus Toolkit [54], that pioneered the work on grid computing. An important thing to note however is that barring IPFS (FileCoin [27]) and BitTorrent [28], typically none of the systems mentioned in this section have any sort of strong incentive mechanisms. Even GNutella, one of the first major peer to peer systems, suffered from a free riding problem [29]. BOINC relies on gamification and altruism [19], Tor and Freenet pretty much require people to join the network as volunteers in order to access any of the services, and other systems will usually just rely on altruism. Incentives are certainly important in grid computing [30], and this is something that this proposal aims to combat.

2.3 Trusting the Untrusted

There is also a significant body of work pertaining to verification and accountability in distributed systems. PeerReview [31] is one of these, and it's probably the most prominent work in the general study of accountability in distributed systems. PeerReview lays out a log based system to check for accountability in any sort of distributed system with deterministic state machines. Drawing from a lot of other work in accountability that was often more domain or need specific, PeerReview creates a highly generalized mechanism to ensure accountability. And while PeerReview did come as a significant breakthrough and provided a very effective solution, it faces

the problem of eventual accountability. Machines in distributed systems could be detected as faulty after a fair bit of a delay. PeerReview in fact acknowledges this as a limitation as well. Another similar system is LOCKSS [32,33], built to preserve digital informations, in particular articles, on a peer to peer network.

A lot of the previously mentioned systems also have their own means to create trust. Many of them rely on hashing in order to ensure data privacy and secure data transfer. BOINC in particular has a method that runs computations on N replicas and takes the consensus of the same, but interestingly also provides a small scope for error due to hardware differences. There is also other related work that builds up probabilistic models for result checking in global compute systems [34], which has helped inspire some of the work on validation in BOINC as well. All of these validation and verification mechanisms usually boil down to an application of Byzantine fault detection and consensus finding [35, 36], which is also leveraged by GNuggies.

The explored related work has helped point us to a number of systems designed for using volunteered computation, the associated incentives to make people volunteer, the validation and verification mechanisms that make such systems safe from malicious agents, and the various applications for science, data preservation, data sharing, and general social good. This will certainly be very helpful in guiding many of the design choices made for GNuggies.

CHAPTER 3: BASIC OVERVIEW AND DISCUSSION OF IDEAS

This chapter will focus on a general overview of the ideas that will be presented in this thesis. GNuggies is proposed to be a fully decentralized Infrastructure-as-a-Service platform that limits the capability of any agency to monitor, threaten or impede GNuggies and the services hosted on it. It is designed to reduce liability and increase participation. An overview of the various components and design decisions that are needed to make GNuggies successful are as follows:

1. GNuggies is a proposed system that relies on people volunteering their machines to act as hosts for services.
2. To ensure performance and reliability, GNuggies should be optimized for PCs and laptops as opposed to mobile and edge computing devices.
3. Since the machines are volunteered, establishing trust is crucial. This thesis proposes using replicas of every service and relying on consensus in order to identify machines that are malicious. The key idea is that every incoming request can be sent to three different replicas that are at varying degrees of trustedness, and finding a consensus in their responses can help detect malicious machines.
4. Machines that are identified as malicious have to be removed from the system immediately. This means that machines that have been on GNuggies for a longer time are more trusted and will thus be less likely to turn malicious. And the more time they spend on GNuggies, the lower their probability of turning malicious suddenly.

5. All services have to be stateless and deterministic, because verification and truth checking for machines relies on a voting like mechanism, and if the same request returns different responses, then the verification system cannot be depended on. Services will also have to be provided as Docker images in order to make setup and replication easy and quick.
6. There are a few key services that are needed to make GNuggies successful - an introducer and membership keeper, a verification service, and a DNS service. These will not run on volunteer machines. All of these are covered in more detail in Chapter 5.
7. Gnuggies needs a way to take payment from service creators and provide rewards to service hosts. The reward will be assigned after verification of the responses provided by the nodes. The reward will serve to incentivize trust among the volunteer machines. We propose making the reward proportional to the time spent by the machine on GNuggies in order to incentivize the volunteer machines to remain trusted.

Furthermore, we argue several properties of our proposal, such as:

1. Expected time spent by malicious machines on GNuggies (Claim 7.1).
2. Lower bound for trust requirements to ensure no service disruptions (Claim 7.2).
3. Upper bound on resilience to a coordinated flooding of malicious machines on the system (Claim 7.3).

4. Lower bound on number of machines needed to keep all hosted services available (Claim 7.4).
5. Lower bound on trust requirements to guarantee clients' safety from malicious responses (Claim 7.5).
6. Incentive compatibility of the rewards provided by the system (Claims 8.1 and 8.2)

The remainder of this thesis covers these requirements and properties in detail, and provides a better analysis of expected performance and security guarantees that GNuggies can provide.

CHAPTER 4: ATTACK MODEL

This chapter will focus on the two types of attacks that we are attempting to guard our system from. One is the kind initiated by single malicious users who want to potentially harm the clients accessing services hosted on GNuggies. The other is better planned attacks from large institutions that want to take down any of the services that are hosted on GNuggies. We further elaborate on the attack models here.

4.1 Individual Malicious Hosts

GNuggies relies on volunteers to provide compute in order to run services efficiently. However, such a crowdsourced endeavor cannot exist without some troublemakers. These could be people who enjoy testing vulnerabilities of such systems, or want to harm the people accessing services hosted on GNuggies. An obvious way to do this would be to either respond to a visitor's request with malicious data. Another issue pertains privacy of user data, which could be encroached upon by reading the requests received by the service. Reading of private data and man in the middle attacks can be largely prevented by utilizing the HTTPS protocol [37, 38]. As for other issues of malicious intent, we discuss methods to both detect such malicious behavior, and to incentivize against it, in future sections.

4.2 Institutions with Malicious Intent

GNuggies is a completely open platform, which means anybody can sign up to provide compute in the platform. We discussed the concept of malicious users in the previous section, however malicious hosts are usually spread out and easy to detect. The

real risk arises from institutions that have a large pool of resources which they can use to damage the integrity of the GNuggies platform. Instead of just a few malicious nodes, institutions could temporarily insert thousands of malicious nodes in the system. All of these nodes could be coordinated in order harm any single service or the platform as a whole. The next section discusses ways that this can be fought and the limitations that our techniques will face.

These institutions could also cause damage through other methods, such as a denial of service attack [39, 40] or through firewalls [41] and certain directives for internet service providers [42], but those are out of the scope of this thesis. Directives for the IaaS provider itself aren't effective on GNuggies due to its decentralized architecture. Dealing effectively with institutions that have malicious intent is the primary task of the design on this platform, because censorship is usually carried out by large institutions or governments, and protecting from that type of censorship is the core reason for the design and creation of GNuggies.

CHAPTER 5: ARCHITECTURE AND DESIGN

This chapter describes the design of the architecture for GNuggies. We go over the various components and the role of volunteer machines in the platform, as well as how services will be prepared, hosted and served to the end users.

5.1 Assumptions

To build GNuggies, there are certain **assumptions about other common technologies**, that have to be made in order to design our architecture. Other assumptions that pertain to guarantees provided by GNuggies are discussed in Chapter 7.

- The presence of a collision resistant hash function.
- Nodes have a public and private key to sign messages.
- Services hosted are **deterministic** and **stateless.**
- Unique IDs can be generated for all requests received by the system.
- Each node has a unique cryptographic key pair that can be linked to its unique node identifier.

Services hosted on GNuggies have to be deterministic and stateless. The architecture uses an application of state machine replicas for its proof of work mechanism which requires the services to have deterministic responses. As for being stateful, that is as a result of the fact that GNuggies is publicly hosted and services are regularly cycled. This means that the nodes hosting a service are often changed and updated which could lead to data loss in case of stateful services. There will also naturally be a certain amount of churn

in the system, so services have to stay resilient to failure of nodes, which can be ensured by hosting only stateless services.

All our other assumptions are required to ensure guarantees in the proof of work verification as well to maintain accountability on the platform.

5.2 Architecture Overview

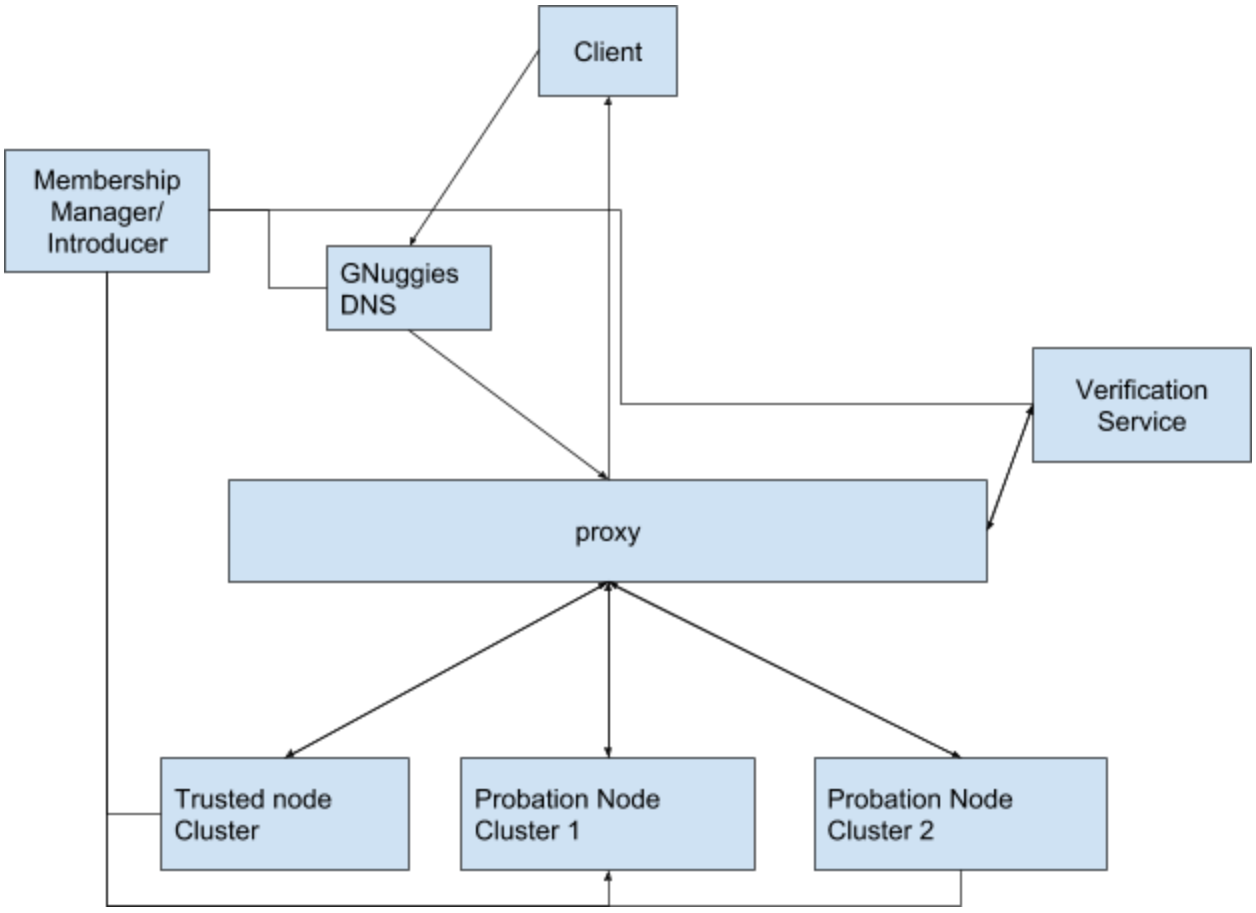

Figure 5.1: Architecture Overview

The different parts and components of the architecture are as follows:

- **Client**: The client is the end user who is sending a request to one of the services hosted on GNuggies. This could be any person making either an API request or

sitting on one end of a computer trying to access a website that is hosted on GNuggies.

- **Membership Manager/ Introducer:** This is the service that keeps a track of all the volunteer machines and their age in the system. It is also the service that is triggered every time a new volunteer machine joins the GNuggies.
- **GNuggies DNS**: This is the service that first receives the request from the client and forwards it to the proxy service with data regarding the appropriate volunteer machines.
- **Proxy:** The proxy service acts as a load balancer and is in charge of forwarding requests from the DNS to the appropriate service clusters.
- **Verification Service**: This service is used to check for any sort of malicious behavior by the machines, and acts the proof of work verifier in this system.
- **Trusted Node Cluster:** This is the cluster of the 1/3rd oldest volunteer machines in the system, which are relatively trusted due to their time spent in the system. These volunteer machines split up into service clusters, each corresponding to a service hosted on GNuggies.
- **Probation Node Cluster 1 and Probation Node Cluster 2:** The middle 1/3rd and the last 1/3rd of volunteer machines. by age in the system, belong in the Probation Node Cluster 1 and Probation Node Cluster 2 respectively. These are machines that are relatively new to GNuggies and aren't completely trusted yet. Each of these clusters is also split up into service clusters, corresponding to every service hosted on GNuggies.

For the purposes of this paper, we will assume that only completely trusted and preset nodes are used for the critical GNuggies services (Introducer, Proxy, Verifier, DNS). These machines will be assumed to be completely reliable and failure proof.

5.3 Volunteered Machines

Host machines provided by the general public are the volunteered machines. There is a specific set of steps for them to enter the platform and become an official part of GNuggies.

Volunteer machine providers will set up their machines to join the GNuggies network by installing the GNuggies Module. This consists of two processes: the first one is the GNuggies Communication Module, and the second is a Docker environment. The communication module is responsible for GNuggies-specific communications such as relaying heartbeats to maintain the global membership list and to maintain the state of clusters for individual services (4.4). Its main purpose however, is receiving a Docker image and deploying it on the volunteer machine. These images are created by people who want to host their service on GNuggies; it contains the code and the necessary environment setup needed to run the service. The Communication Module also provides basic telemetry of the Docker image to the membership keeper.

5.4 Services

As mentioned in the assumptions before, services will have to be deterministic and stateless for GNuggies to work safely and effectively. Moreover, since GNuggies is aimed to

act as a serverless compute system, any databases for services hosted on GNuggies will have to be set up by the service creators on a separate, reliable backend (such as AWS). In order for services to be run and set up on volunteer machines easily, service creators will have to provide access to Docker images that can be replicated on any volunteer machine. These images will be downloaded on and run on volunteer machines, allowing for an easy setup, and security guarantees both, for the volunteer machines and the services. The expectation is that GNuggies will act like a host for the API, and features such as scaling, load balancing, and other infrastructure solutions will be abstracted for the service.

5.5 Service Clusters

Once volunteer machines have been assigned to a service, they receive the Docker image containing the code for the service and deploy it. Multiple volunteer machines will be picked for each service, enabling us to create a cluster for each service. The Docker image of the service is deployed on each node except for on the leader (elected based on node age) which becomes a simple load balancer for the cluster. Each cluster will exist as an all to all network. As a requisite for the detection mechanism detailed in section 3, a total of 3 replicas of the cluster will have to exist, one as part of the trusted node cluster, and the others two as part of the probation and audit clusters.

In case of machines in the cluster leaving, the leader will detect this and report the issue to the membership keeper. This will help us maintain our age tracking and will keep our clusters balanced and in check. In case the leader of the service cluster has a failure of some sort, the remaining nodes will run an election amongst themselves, assigning the

oldest node as their leader, and letting the membership keeper know of this change. This functionality is similar to that provided by ZooKeeper [48].

5.6 Trusted and Probation Node Clusters

Every service cluster is responsible for hosting a service. However, our platform is designed to concurrently host multiple services, so we will have multiple service clusters existing concurrently. In order to tie this in with the detection of malicious behavior that is described in the next chapter, we sort all our volunteer machines in one of three sets, based on their age in the network. The **oldest third** of the machines are put in our trusted node cluster, because these are nodes that have lived in our network for long enough to be relatively trusted. The other machines are then split into two probation clusters.

Each of these sets, the trusted node cluster and the probation node clusters, will host all the services on GNuggies independently. The difference here will be that, as detailed in section 2, every incoming request is forwarded to the relevant service clusters in each of these sets, but only the response from the trusted set is forwarded to the client. This means that all new nodes have to wait for our network to triple in size before they can attain trusted status, allowing us to catch them before they can cause any real damage.

5.7 Membership Keeping and Cycling of Services

GNuggies has a central membership manager that is responsible for keeping track of all machines in the system and information regarding their age and their role (service cluster) in the system. It is also responsible for cycling services regularly to keep the

behavior on every machine constantly changing and to make it tough for malicious users to negatively impact these services.

5.7.1 Fairness and Service Allocation

The GNuggies Allocator uses information about the history of utilization of various services in order to allocate nodes to services. It obtains this information from the DNS service. It then uses this information along with information about the compute power of each node to make an allocation using the **Max-Min Fairness algorithm** [43-47].

Allocations are re-computed across the trusted and the probation clusters every 24 hours, or when over 10% of the nodes from any one of the three primary clusters are lost to failures of different kinds. This ensures that bandwidth usage for the GNuggies hosts is not too high, while keeping the change frequent enough to reduce the impact of malicious nodes. During each allocation refresh, we make sure that only half the nodes are changed for every service. This is done to ensure that the service doesn't have downtime every time that allocation is changed and to ensure that a particular service is difficult to single out and attack as the nodes that are hosting it keep changing.

5.7.2 Introducer

The membership keeper also serves as the introducer for new volunteer nodes as well for the services themselves. New volunteer nodes will have to run a pre-created package that will will ping the membership keeper, which will act as an introducer and add the machine to Probation Cluster 2. When services cycle again, the volunteer machine will get assigned a service as well. Until then the machine will be idle, which could be a good time to run basic testing and telemetry on these machines (this is outside the scope of this

thesis however). Service creators on the other hand will have send a ping to the membership keeper regarding the Docker images that can be used to run their services, and the service will then be accordingly accommodated when services are cycled again.

5.8 GNuggies DNS

One important question to consider is how a client accesses the web service, given that the IP address of the service is likely to change often. This is done by simply accessing a known domain name that has the IP address information in its records. To facilitate this, GNuggies hosts a Name Server. Service owners must purchase a domain name to use with their GNuggies service and make sure that the NS records of this point to the GNUggies Name Server and that the CName record contains the UID of the service. Once this is done, the GNuggies Name Server will be updated every time the machines that run the Service are changed.

5.9 Proxy/ Outgoing load balancer

The proxy service is central to helping abstract a lot of the additional functionality that is needed to make this platform work. The proxy service serves as a middleman to all requests going through the network. When the DNS forwards a request with the appropriate IP addresses for service clusters (trusted and probation) that correspond to the request, the proxy generates a unique ID for the request and then forwards it to the IP addresses provided by the DNS. The cluster leader in each cluster then assigns the request

to a randomly selected node in the cluster, which then sends the response back to the proxy.

The proxy forwards the response from the trusted cluster to the client, while it holds onto the other responses. Once it receives all three responses, it forwards the hashes of each response to the verification service to detect any malicious activity. In case all three responses are not received within time *t*, the proxy will forward the response from the oldest node (age determined as part of the membership protocol) to the client, and then forwards all available responses to the verification service.

5.10 Verification service

The verification service relies on a pub-sub model. Upon being forwarded the responses to unique requests by the proxy, the verification service compares the hashes of the three responses in the manner detailed in the next chapter. The verification service is then able to detect any malicious nodes and inform the membership keeper about those, thus ensuring that these malicious services are kicked out of the network. In case responses from any of the machines are not received, the verification service will ping this machine to get the response to the same request. If the machine is offline, then it is kicked off by the membership keeper, else its response can be evaluated against the other responses received.

5.11 Path taken by a request

Now that we know the different parts of the platform and how they function, so we can now take a look at exactly how a request from the client comes to the GNuggies platform, and is processed by said cloud.

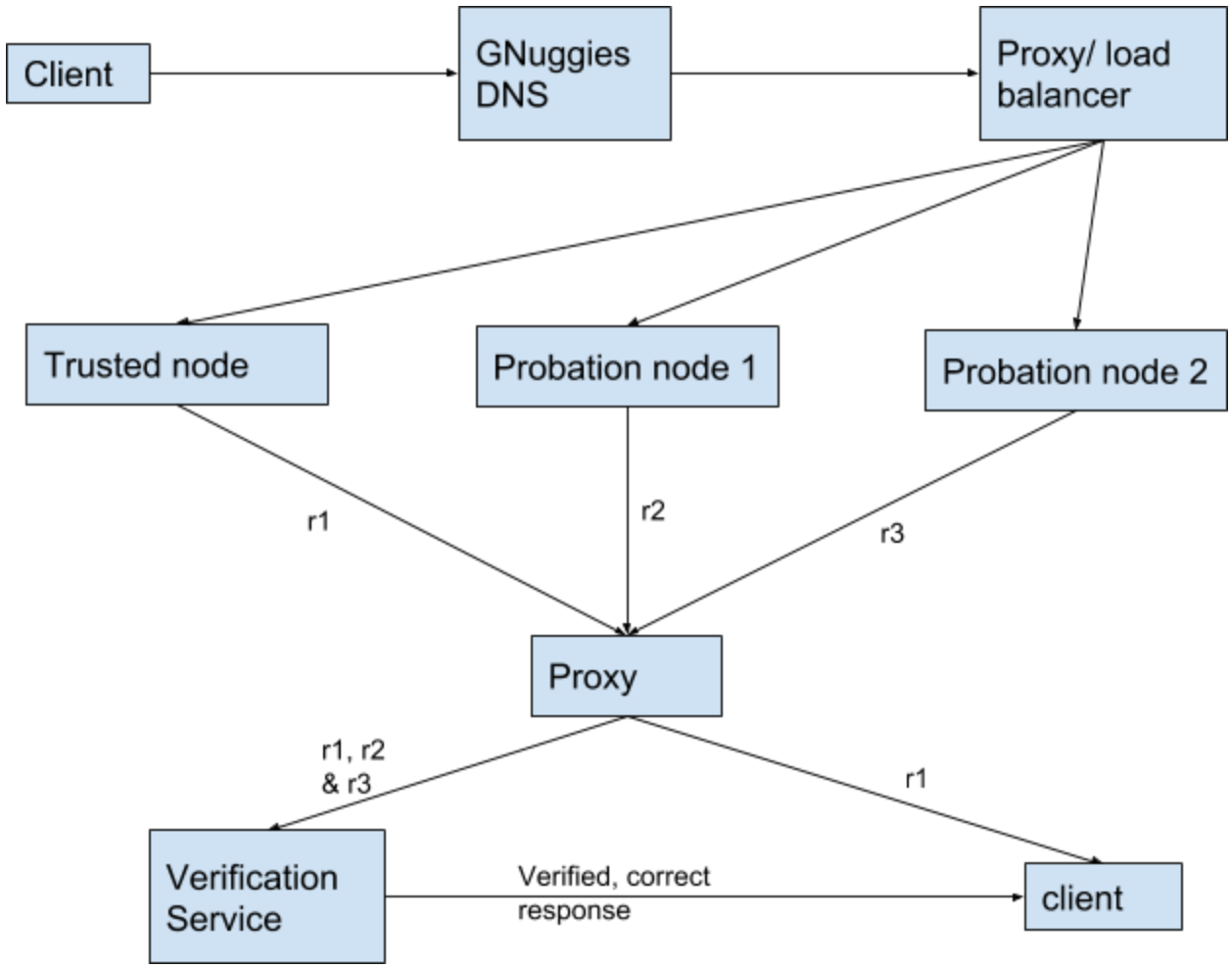

Figure 5.2 Path taken by a request to services hosted on GNuggies

As can be seen in the diagram above, the request goes from the the client to the GNuggies DNS service, which forwards it to the proxy with the details of the three clusters

that host the service in question. These three clusters then have a machine in each of them process the request and forward their responses to the proxy, which then takes the response from the oldest machine (from the trusted node cluster), and forwards it to the client. All three responses are then forwarded to the verification service, which compares the responses to find a consensus, and update what the client is seeing if the consensus does not match the oldest node's response. The verification service is also able to detect the responses from malicious machines and deal with those machines accordingly.

CHAPTER 6: VERIFICATION MECHANISM

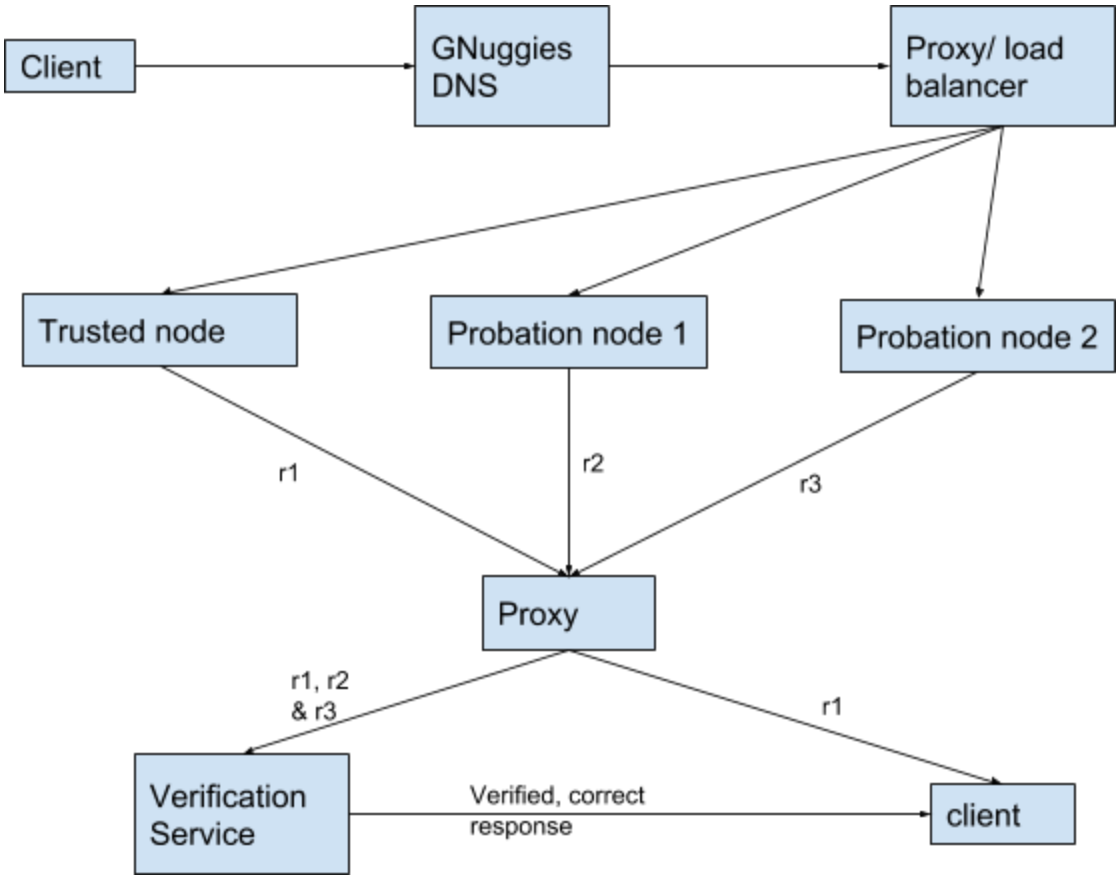

Figure 6.1 Path taken by a request to a service hosted on GNuggies

This chapter discusses the proposal for the workings of the verification service in GNuggies. Efficient detection of malicious behavior by untrusted nodes is a tough challenge. However, by restricting services hosted on GNuggies to be deterministic and stateless, we can model the problem for malicious behavior detection as byzantine fault detection. This lets us use state machine replication as an effective strategy, and we add to that the concept of trusted and probation machines, allowing us to catch potentially malicious machines

before they can become a central part of the GNuggies platform and cause any sort of network disruptions.

6.1 The Verification Process

The verification and accountability mechanism in GNuggies involves a few components - the verification service, the load balancer (proxy layer), the trusted service replica, and two verification replicas (the machines on probation). The verification service and load balancer are assumed to be run by us so they are trustable and failure proof at all times. As for the three replicas (trusted replica, two probation replicas), they are three state machine replicas of the same hosted service, on three separate nodes. The oldest of these three nodes is considered trusted, and in case of a situation where no consensus can be achieved, this node's response is picked as our winner. The way that this detection works is that any time we receive a request from a client, the request is relayed to all three replicas, which then forward their responses to the proxy service.

The load balancer then has two options. The first option is to relay the response from the oldest ("trusted") node directly to the client, while also relaying hashes of all the responses to the verification service, which can then detect any node which is malicious. The second option is to relay all the hashes to the verification service first, and then based on the decision of the verification service, only relay the correct response to the client. It is clear that while the second option provides better guarantees of security for the client, it does this at the cost of extra processing time taken to actually get a result to the client.

For the purposes of GNuggies, we currently choose to go with the first choice, but that is a design decision that can be conveniently changed. The reason for this being that we want any client to have as short of a wait time for the response as possible, which is tough to do already due to the potentially large geographical or network distances between the machines that form the various layers of our service. Also, the presence of the "probation period" for the verification replicas ensures that in a much larger version of this system, any node that has reached the "trusted" state has a very low likelihood to have a spurious malicious intent. This is because any malicious node would have potentially been caught and removed from GNuggies during the probation period itself. Also, any new node joining the system will have to wait for an exponential amount of time before it becomes trusted.

Our mechanism does however face the same limitations that state machine replication for byzantine fault tolerance does - if more than one third of the nodes in our system are unreliable or malicious, then our verification system cannot provide any guarantees regarding the identification of malicious nodes. This is a drawback that can be exploited by larger institutions which can potentially flood the system with enough nodes to wreak havoc. This however will obviously be a smaller issue once the network reaches a large enough size, where it would be infeasible to flood the network with that many malicious nodes. The guarantees that the system design provides, are further explored in Chapter 6.

6.2 Analysis

We can look at certain example cases for how both the choices would work, especially in the eyes of the end user, under various different circumstances. Assume that we have a service *X* that is hosted on nodes *n1, n2* and *n3*; *n1* being the oldest of the three. Also assume that sending the response from *n1* directly to the client (first strategy) takes time *t1*, and taking the longer route by sending the response after verification (second strategy), will take *t2*. Under standard conditions, we can say for a fact that $t1<t2$. Also, since each one of *n1, n2* and *n3* can independently be either malicious or non-malicious, we have a total of 8 possible scenarios. For all of these scenarios, we will assume that a node is more likely to join GNuggies as a malicious node, or to very quickly turn malicious after joining, as opposed to turning malicious after spending a significant amount of time as part of GNuggies. The different possible scenarios are as follows:

1. None of *n1, n2* and *n3* are malicious. In this scenario, the first strategy works better, and the client sees the correct response generated by the service. This is a relatively likely scenario for our system, because malicious nodes get removed from GNuggies as soon as they are detected. Also most common volunteers are not skilled enough to perform the tasks needed to send a malicious response. This is great for GNuggies because it means that almost always, the client will receive the correct and expected

response, and no machines will have to be removed from GNuggies either.

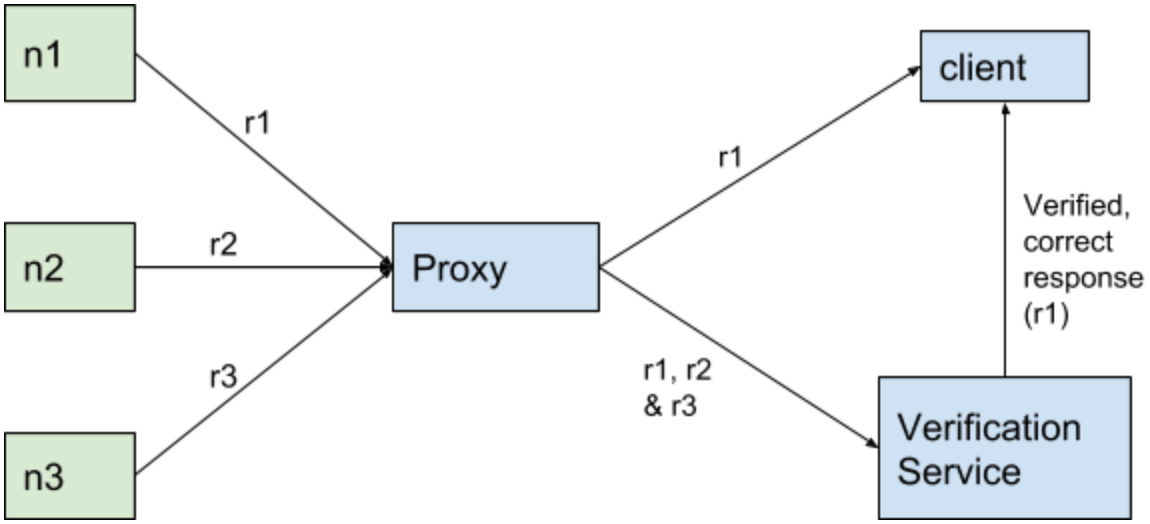

Figure 6.2 If no nodes are malicious. Green (not shaded) represents non malicious

2. Node *n1* is not malicious, but **one of** *n2* and *n3* is. In these two scenarios, we still have the same result as before, i.e., the client sees the correct response generated by the service. The first strategy will still provide the correct response to the client, as will the second strategy, but the first strategy will do so quicker. Also, the malicious node will be identified due to consensus in the results of the other two nodes, and it will be successfully removed from the platform in this scenario. Typically, it is a lot more likely that *n3* is the malicious node in this scenario, because nodes becoming malicious spuriously is uncommon, and it is a lot more likely that nodes that have been in the network and have served a number of requests correctly, will continue remaining trustworthy. This also implies that malicious nodes that are added to the system will be detected and removed as soon as they serve their first malicious response, assuming that the two nodes that they are paired with are both trustworthy.

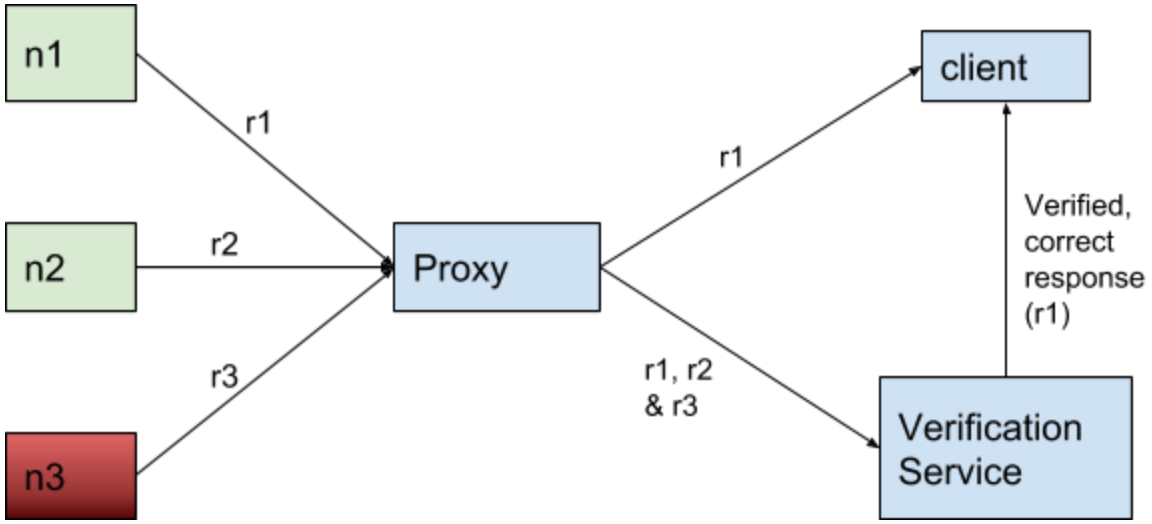

Figure 6.3 If one node (*n2* **or** *n3*) is malicious. Green (not shaded) represents non malicious, red (shaded) represents malicious.

3. Node *n1* is not malicious, but both *n2* and *n3* are. In this scenario, the first strategy guarantees that the client will not see a malicious response, but the verifier will remove *n1* from the platform **if** *n2* and *n3* are collaborating their malicious reponse. This is certainly not ideal, but due to GNuggies' reliance on simple consensus of the three, it is unavoidable. The second strategy on the other hand would give the client the malicious response if *n2* and *n3* are collaborating, but the correct response otherwise. If *n2* and *n3* were not collaborating and had different malicious responses, then both the strategies will remove them from the platform.

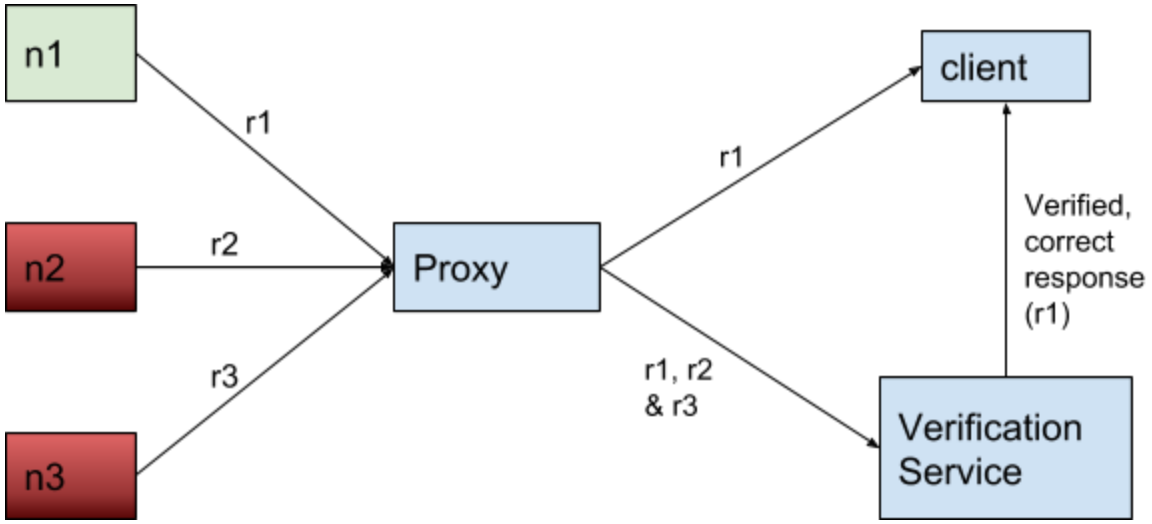

Figure 6.4 If two nodes (*n2* **and** *n3*) are malicious. Green (not shaded) represents non malicious, red (shaded) represents malicious.

4. Node *n1* is malicious, but *n2* and *n3* are not. This is a highly unlikely scenario because it requires *n1* to have suddenly turned malicious after having spent a significant amount of time on the platform as a non-malicious machine. In this case, the first strategy (directly deliver n1's response to client) will provide the client with the malicious response but will then successfully remove *n1* from the platform. The second strategy on the other hand, will serve the client the correct response, albeit slower, and will remove *n1* from the platform as well. The first strategy would also be able to force the client side to reload with the correct response, so the amount of time the user spends viewing malicious content is reduced.

Another thing to note is that in case the malicious response does get to the user, and it is some sort of unexpected file or something similar, then browsers today are usually able to handle such scenarios easily, and block this malicious response

before it causes any problems to the client. This is particularly helpful in making sure that certain extremely harmful malicious responses, such as a virus or malware, are unable to impact the client. This can certainly help deter volunteers from spuriously turning their machines malicious at any point, since their efforts would typically be in vain.

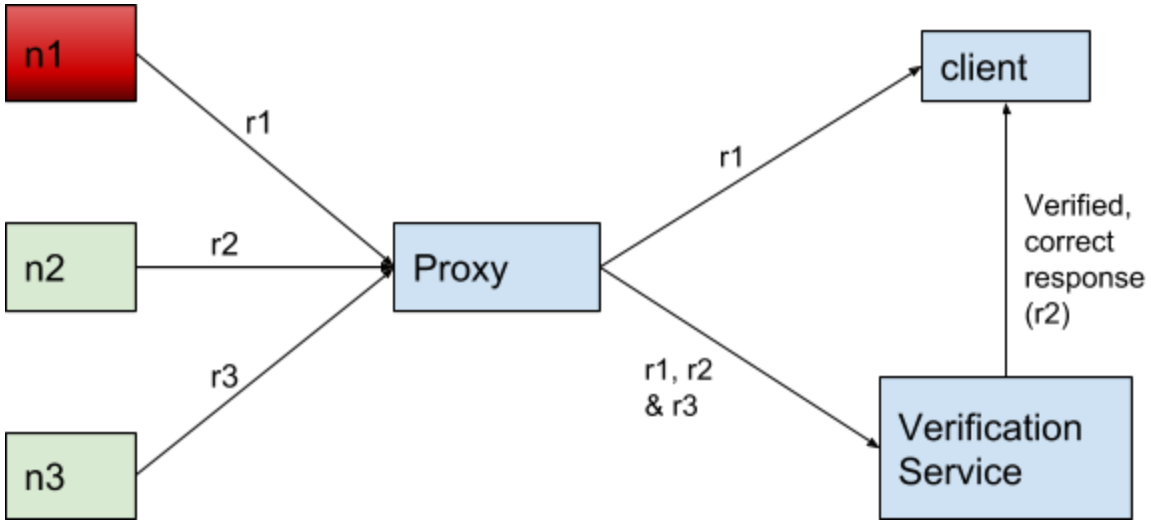

Figure 6.5 If *n1* is malicious. Green (not shaded) represents non malicious, red (shaded) represents malicious.

5. Node *n1* and one or both of *n2* and *n3* are malicious, but there is no collaboration. In this scenario, the user will get a malicious response no matter what, and both *n2* and *n3* will be removed from the platform. This is however extremely unlikely, for reasons discussed previously.

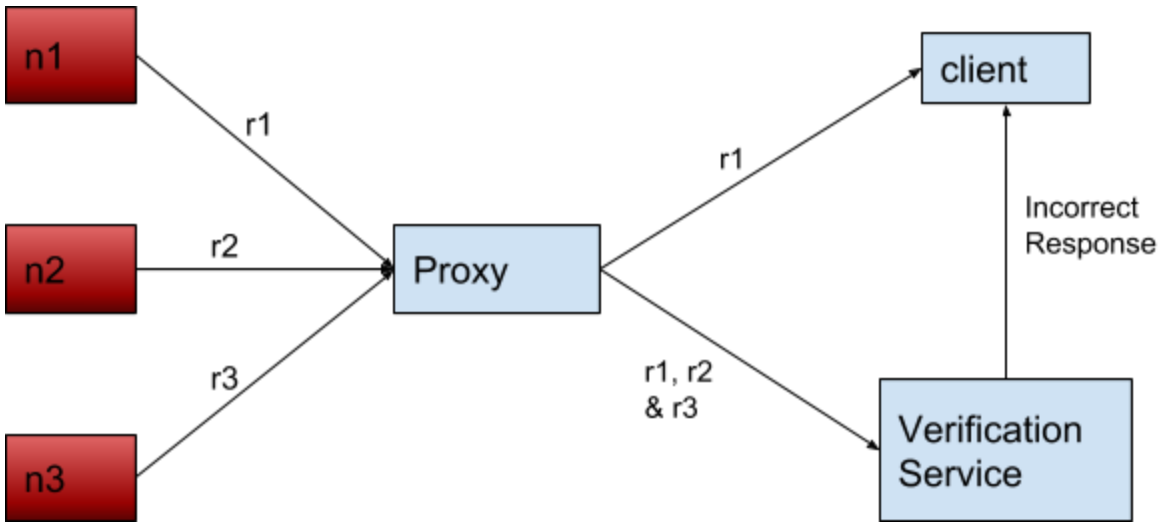

Figure 6.6 If *n1* and one or both of *n3* and *n2* are malicious. Red/shaded represents malicious.

6. Node *n1* and one or both of *n2* and *n3* are malicious, and there is collaboration between at least two of the malicious nodes. In this scenario, the user will get a malicious response no matter what, and the node that wasn't involved in the collaboration will be removed from the platform. If all three nodes were malicious and collaborating, then they would all continue to remain in the system. This again, is extremely unlikely, for reasons discussed previously.

This covers all the possible scenarios and the expected result in each scenario as well. And as can be seen, both strategies of delivering content to the client have the benefits and drawbacks, but either way, in almost all common situations, the client's safety is maintained.

CHAPTER 7: FORMAL ANALYSIS

A **malicious machine** is defined as one that responds to requests with data other than what is generated by the actual hosted service itself. A **malicious response** is a response sent to a request by a malicious machine, which will be any data that is not exactly what was generated by the service itself in response to the request. A **trustworthy or trusted machine** is one that is not malicious. A **fully trustworthy machine** is one that is not malicious and can be assumed to not turn malicious spuriously at any point.

7.1 Time Spent by a Malicious Machine on the Platform

Claim 7.1 *- Assuming that otherwise all our machines are trustworthy, any one machine that sends a malicious response will be detected and removed from the network soon (as soon as the verification service receives and processes the malicious response), thus keeping the number of malicious machines on the platform close to 0.*

Assuming that our platform is made entirely of fully trustworthy machines, if a malicious machine joins GNuggies, it will be placed in the second probation node cluster. It will be assigned a service for which it will be part of one of the three replicas. As explained before in the description of the verification service, the response to any request made to the malicious machine will be checked against the responses to the same request by machines in the replicas. Since the other machines are assumed to be trustworthy, their responses will be the same, and thus consensus will be achieved, leading to the malicious

machine getting flagged by the verification service and removed from the platform by the membership keeper.

Alternatively, if we cannot make an assumption about fully trustworthy machines, because it is certainly possible that some machines turn malicious spuriously over a longer term. Due to the verification and its implications, we can assume that this will be relatively infrequent and uncoordinated. Since turning a machine malicious after a long period of time on GNuggies would require technical adeptness that few people have, and coordinating two malicious machines would require somehow fooling the https protocol, our assumption is valid. Thus if a machine *m* were to in fact turn malicious spuriously, we could optimistically assume that none of the machines that are replicas for the service hosted on *m* are also malicious. Thus the machine *m* would be flagged and removed by the verification service right after the machine makes its first malicious response (immediate removal discussed in Section 5.10 and in Chapter 6).

7.2 Reliability on GNuggies

Claim 7.2 *- If the oldest 2/3 machines are fully trustworthy, then in the absence of any growth or churn in the platform, and as long as these machines remain fully functional and there are no network interruptions, it is impossible for any service to be disrupted or made unavailable, or for any end user (client) to receive a malicious response.*

Because of how the verification mechanism works, and from Claim 7.1, we assert that the assumption that the oldest two-thirds of the machines on GNuggies will be trustworthy will hold true. This is barring situations when these machines turn malicious

spuriously, or have been trusted for a long time and respond maliciously suddenly. Given that, we can ensure that any malicious activity by new machines will be detected very soon (from Chapter 6) and will be guaranteed to be detected (since the other two replicas for the services will be in the 2/3rd oldest, and thus trusted machines). This further implies that any network disruption or unintended impact on the end-client is completely avoided. Also, due to how the verification mechanism is designed, the response from the oldest replica will be directly delivered to the client, as was discussed in the Section 5.11, thus making it impossible for the client to notice any disruption caused by the malicious machine.

7.3 Resilience to a Coordinated Flooding of Malicious Machines

Claim 7.3 *- Assuming there are currently n machines in the network, all of which are fully trustworthy, an outside agency will have to submit more than n/2 malicious machines as volunteer machines on the platform (assuming no other growth or churn on the platform in the time that these new machines are added), in order to disrupt or make unavailable any service hosted on GNuggies, or in order to make the verification service falsely kick a trusted machine off the platform.*

This claim draws from the Claim 7.2. Since there is no network disruption and services continue to remain available if the oldest 2/3rd of the machines are trustworthy, then in such a situation, upon the addition of *n/2* new nodes, the oldest *n* will still continue to remain trustworthy. Thus the oldest 2/3rd machines are still fully trustworthy, and the machines added by the agency will not cause any service disruption. However if more than

n/2 malicious machines are added, then our guarantees no longer hold, and there could be a disruption in the services.

7.4 Minimum Requirements to Host Services

Claim 7.4 *- Assuming m services are hosted on GNuggies and that all machines on GNuggies are trusted, as long as there are at least 6m volunteer machines in the platform, all services will be hosted and will stay available.*

As described in the architecture design, in every service cluster, there has to be a leader, and at least one other machine that hosts the service. And since three replica clusters have to (see Section 5.5) exist, that means we need 6 machines per service, at the very least. This implies that if we had *6m* machines, we could successfully run upto *m* services on GNuggies, barring any sort of network disruptions.

7.5 Minimum Requirement to Protect Every Client from Malicious Responses

Claim 7.5 *- If at any given point of time, the oldest 1/3 of machines on the platform are fully trustworthy, then no client will receive a malicious response. And as long as the oldest 1/3 machines of the platform continue to remain trustworthy, this will continue to remain true.*

This derives directly from the design of the verification service. All volunteer machines are put into 3 different node clusters - trusted, probation 1, and probation 2. The oldest 1/3 machines are placed in the trusted cluster. For every service, there are three service clusters, one in each of the node clusters. Thus the oldest replica that responds to each request for this service, will always be from the trusted cluster, i.e., it is a part of the

oldest 1/3 machines in GNuggies. Since responses from the oldest replica for each service are directly passed to the user, the user will not receive any malicious responses as long as the oldest 1/3rd of the machines are trustworthy.

However, in the verification step, these machines can potentially be falsely identified as malicious if the machines on the other two replicas were collaborating their malicious response. This would cause the loss of machines from the trusted node cluster (oldest 1/3rd), which would imply that other potentially untrustworthy machines could now be in the updated oldest 1/3rd of all machines. But if the situation of collaboration between the malicious machines does not take place, then the trusted machine will remain on the platform, ensuring that the clients only receive responses generated by the service that they are trying to access. This claim essentially says that - if all machines in the trusted node cluster are in fact fully trustworthy at any given point of time, then no client will receive a malicious response at that point of time.

7.6 Requirements for Alternative Design Decisions for GNuggies

Claim 7.6 - *For each service, to handle failures (Byzantine Faults or otherwise) in* f *replicas of the service, we will need to have* (2f+1) *replicas of the service*

Since one of the primary features in GNuggies is the novel use of state machine replicas for Byzantine Fault detection, we need to adhere to the requirements laid out for the same. This implies that for every service, we can handle a certain number of failures and Byzantine faults, if we still have more perfectly working machines than failed machines. This means that if we had to handle *f* failures, we need to have at least *f+1* other

replicas that were perfectly functional and trustworthy, in order to guarantee no network disruptions.

The implication of this is that, if we wanted to go with more than 2 probation clusters, we could implement a similar design, where instead of 3 total clusters (1 trusted, 2 probation), we would have *2f+1* total clusters (1 trusted, *2f* probation), and that design would guarantee resistance to failures in upto *f* of these clusters. Using a number other than 3 however, does come with potential issues regarding needing more nodes to run the same number of services and a greater waste of resources.

CHAPTER 8: INCENTIVES FOR TRUTHFUL PARTICIPATION

One of the bigger challenges on GNuggies has to do with individual volunteers and the first attack model - malicious individual users. Recruiting users to volunteer their machines for the network is certainly a difficult task, and altruism or gamification are rarely, if ever, good incentives for such volunteer systems. It has worked for SETI@Home and BOINC [49], but is unlikely to work for GNuggies. Moreover, incentivizing these individual users to not play around or interfere with GNuggies in any way is also important in order to maintain a fully functioning platform. For this we design an additional incentive mechanism, which relies on the assumption that for every request served, the creator of the service would be charged a certain amount of money. For simplicity's sake, we will assume that the charge per request is 1 unit of money, and the money can be either actual currency, or some type of cryptocurrency (which helps maintain relative anonymity [50]). The value of the unit could also be fluctuating with time, but that is a discussion outside of the scope of this paper. Charging service creators to host their services is a model similar to AWS Lambda, or other common cloud offerings.

There are **two goals** for our incentive mechanism - 1) users should want to contribute compute and 2) users should not want to behave maliciously. Since we will be charging service creators for hosting, the benefit from that can be distributed to all the machines that responded to the request in consideration. This would be the three machines from the three different replicas for the service. Payoffs could be distributed proportional to the inverse of the machines' age on the platform.

8.1 Payoff Distribution

Assuming *n* total volunteer machines on the platform, every machine could be associated with an age rank, ranging from *1* to *n*, 1 being the oldest, and *n* being the youngest. Our proposed payoff scheme, for a request processed by three machines with age ranks *n1, n2* and *n3* will be to provide each with a chunk of the **1 unit of currency** (either virtual - like a cryptocurrency, or real - like USD), proportional to the inverse of their age rank. So for each the three machines, their payoffs will be:

$$\text{Payoff of } n_i = \frac{\frac{1}{n_i}}{\frac{1}{n_1}+\frac{1}{n_2}+\frac{1}{n_3}} \quad \forall\ i\ \epsilon\ (1,2,3) \qquad (8.1)$$

If the machine with age *n1* is the oldest, then its payoff will be the largest. Since services are commonly cycled and requests are randomly assigned inside service clusters, it would be good to analyze average payoff for machines in every third of the age range (the three larger clusters of service clusters - trusted node cluster, the two probation node clusters). This would be done opposed to analyzing specific cases, and would provide a better understanding for the incentive mechanism.

8.2 Analyzing Payoffs

We can assume that truth telling in this system is equivalent to remaining a trustworthy machine. We have to show that any volunteer strictly benefits from staying in the system for longer, in order to prove that it is incentive compatible for the volunteer to stay trustworthy. We can assume that in almost every scenario, there is no tangible benefit

for an individual for being malicious. A tangible benefit is any sort of monetary or other measurable payoff. This can be assumed because if the volunteer machine is in the younger 2/3rd of the platform, then any malicious response will just result in removal from the network, and nothing else. Since volunteer machines don't actually know whether they are in a trusted service cluster or not, a malicious host has no way of saying that their malicious response (perhaps some sort of malware or unsafe content) will even get to a client.

If the volunteer machine were in a trusted cluster however, with the amount of security existing in web technologies such as browsers today, it is near impossible for a simple attack through a malicious response will even get to the client. Thus it is reasonable to assume that in almost every case, there is no tangible benefit to being malicious, and the only possible benefit for the host would be some sort of enjoyment in testing the limits of GNuggies. Now we just need to show that it will be in the host's benefit to stay trustworthy as opposed to getting removed from the platform just for enjoyment. This can be shown by showing that the older a machine is on the platform, the higher its total payoff will be for every request served.

8.2.1 Average Payoff For Machines in Every Cluster

The average age rank in every cluster is as follows- $n/6$ for the trusted node cluster, $n/2$ for the probation node cluster 1, and $5n/6$ for the probation node cluster 2. Now taking each of these average age ranks as representative of any node on the cluster (since we did say that over a period of time, because of cycling of services and randomness, payoffs tend to average out to a consistent value), we will evaluate the average payoff in each cluster.

$$\text{Cluster 1 (trusted):} \quad \frac{\frac{6}{n}}{\frac{6}{n}+\frac{2}{n}+\frac{6}{5n}} = \frac{30}{46}\text{units}$$

$$\text{Cluster 2 (Probation 1):} \quad \frac{\frac{2}{n}}{\frac{6}{n}+\frac{2}{n}+\frac{6}{5n}} = \frac{10}{46}\text{units} \qquad (8.2)$$

$$\text{Cluster 3 (Probation 2):} \quad \frac{\frac{6}{5n}}{\frac{6}{n}+\frac{2}{n}+\frac{6}{5n}} = \frac{6}{46}\text{units}$$

An important thing to keep in mind here is that the currency used can be a virtual or cryptocurrency, or it can be a real currency. In keeping with the objectives of reduced liability and sidestepping censorship, a cryptocurrency would make more sense. IPFS uses a cryptocurrency (Filecoin [27]) in order to incentivize people to volunteer their machines for storage. More traditional IaaS systems like AWS Lambda on the other hand, use real currency at a pre-established rate.

8.2.2 Payoff for Machines in Specific Scenarios

We can see that on average, it is strictly better for a machine to be in an older cluster. Now that we know that on average, older clusters always make a larger portion of money, we need to show that for within the same cluster, it pays more to be older. Assuming that the machine's age is *i*, we can still take the average age ranks from the other two clusters, which for now we can assume to be *n1* and *n2.* We can take the average ages for the other two clusters, because as before, for the machine in question, each of its

requests served will typically be verified against randomly picked machines from the other two replica clusters for the same service. This implies that over a large number of these requests, their age ranks would average out, but for our machine *i* in question, its own age will stay the same. The payoff will be as follows:

$$\frac{\frac{1}{i}}{\frac{1}{n_1}+\frac{1}{n_2}+\frac{1}{i}} = \frac{n_1 \times n_2}{i \times (n_1+n_2)+(n_1 \times n_2)} \quad (8.3)$$

Thus within a cluster, a machine's payoff is inversely proportional to its age rank, and thus the older it is, the smaller its age rank, implying a higher payoff for being older. Thus we have shown that it is always better to be in an older cluster, and within your current cluster, it is always better to be the oldest machine possible. This implies that it is incentive compatible to remain a trustworthy machine on the GNuggies platform, and being malicious, getting detected and then rejoining the system as the youngest volunteer is not beneficial for the volunteer machine provider. And assuming this payoff is higher than the cost of providing compute to the GNuggies platform, it is also strictly in a person's interests to volunteer their machine for the GNuggies network.

8.3 Formal Analysis and Claims

Through this incentive mechanism, everyone gets paid for their compute from day 1 of joining. And *the longer you stay, the more payoff you get, and as more machines join the network, the payout also keeps growing.* This also means that if an outside agency were to

try and pay longtime GNuggies volunteers to turn malicious in a coordinated fashion, these volunteers would have an incentive to turn that down and continue staying in that network, as their payoff from remaining in the network is likely higher than what they will get paid to turn malicious. If a node leaves GNuggies, it will never earn as much as it did when it was still part of GNuggies, even if it rejoins GNuggies at a later point.

Claim 8.1 *: Volunteers are strictly better off upon joining GNuggies, if the expected payoff of a new node in Probation Cluster 2 is greater than the operational costs of providing and running the volunteer machine and the potential profits from any other endeavors for the machine (e.g., mining cryptocurrency). If this condition is met, volunteers will have a strong incentive to join GNuggies.*

In any game, every rational player aims to maximize their payoff. In order to incentivize people to volunteer for GNuggies, they should have a higher payoff upon joining the system, than the "payoff" received from the saved energy costs. If this can be ensured, rational volunteers will join GNuggies instead of leaving their machines idle.

Claim 8.2 *: Volunteers are strictly better off staying trustworthy on GNuggies, if their ongoing payoff is greater than the expected payoff from any malicious activity (e.g., sending malware to the client) or from any activity that they may engage in upon leaving GNuggies. If this condition is met, volunteers will have a strong incentive to remain trustworthy volunteers on GNuggies.*

In any game, every rational player aims to maximize their payoff. If GNuggies can ensure a payoff greater than volunteers' expected payoff from malicious activities, then volunteers are strictly better off if they remain trustworthy. Moreover, if the expected payoff from remaining volunteers on GNuggies is greater than the immediate payoff of malicious activities, followed by the payoff from re-entering GNuggies, then volunteers have no incentive to be malicious or to leave GNuggies for any reason.

CHAPTER 9: SUMMARY AND FUTURE WORK

In this thesis we presented a proposal for GNuggies, a novel provider of infrastructure- as-a-service that uses volunteered and untrusted machines to host web services, in an effort to create a more censorship resistant internet. We then discussed the various attack models that GNuggies needs to be resilient to, the design of GNuggies itself, the mechanism for verifying the work done by volunteer machines, a formal analysis of the claims made regarding GNuggies' features, and a detailed incentive mechanism for maximizing truthful participation by volunteers.

We proved several different claims regarding the security guarantees and incentives that were detailed in this thesis. We showed that in a mostly trustworthy system, the number of malicious machines on GNuggies will remain close to zero. Also since GNuggies relies on a voting system that uses 3 replicas for every service, if the oldest 2/3rd of the machines are fully trustworthy, then there will be no service disruptions. We also showed that if all n volunteer machines on GNuggies were fully trustworthy, then any outside agency would have to add more than $n/2$ machines to GNuggies in order to disrupt any service. This is an important claim as it provides the limitations faced by GNuggies in combating the primary attack model. We also discussed the minimum number of volunteer machines needed to keep services running ($6m$), as a function of the number of services (m), as well as the claim that the oldest 1/3rd of machines on GNuggies should be fully trustworthy in order to protect clients from malicious responses.

We also looked at how making rewards for machines inversely proportional to their age on the system would act as an incentive to make machines join the system as well to continue staying on the system without giving any malicious responses.

The thesis discussed the various strengths and weaknesses of the proposed architecture, and attempted to analyze the many use cases and limitations. The biggest limitation perhaps is the need for services to be deterministic and stateless. This does make it challenging to have particularly large and popular services hosted on GNuggies, but if GNuggies were to be used entirely like AWS Lambda, only providing an endpoint to host services but no actual data, then there should be no issue with the current design.

Future Work

There are many things that can be done to improve upon the proposal presented in this thesis. Firstly, an actual implementation of the system would certainly be very helpful to better evaluate and test the proposed design. It would provide an even better idea of the strengths and limitations of the system, especially in scenarios where volunteer machines are highly geographically distributed. Other major avenues for future work would be based around the core GNuggies services. Currently the design assumes that they are hosted on fully trustworthy and reliable machines, however the only real way to ensure that is making those services centralized, creating a single attack point for the entire architecture. This can be circumvented by improving the mechanism for cycling of services, and creating a few extra verification checks in the trusted node cluster. A full design and analysis of the same would be a remarkable stride towards a completely decentralized GNuggies.

It would also be very interesting to explore the possibility of creating a distributed database on GNuggies, similar to the one created for IPFS (OrbitDB [52]). That would certainly help to move away from the requirement that services be completely stateless. This would likely need a large overhaul of the current design however, and will need more time to fully explore and implement.

REFERENCES

[1] Google Cloud Platform. https://cloud.google.com

[2] Microsoft Azure. https://azure.microsoft.com

[3] Amazon Web Services. https://aws.amazon.com/

[4] Heroku. https://www.heroku.com/

[5] Docker Cloud. https://cloud.docker.com/

[6] Parse. http://parseplatform.org/

[7] AWS Lambda. https://aws.amazon.com/lambda/

[8] Pelofsky, Jeremy. "Amazon stops hosting WikiLeaks website" Reuters

[9] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," Proceedings First International Conference on Peer-to-Peer Computing, Linkoping, 2001, pp. 99-100.

[10] Cohen, Bram. "BitTorrent-a new P2P app." Yahoo eGroups (2001).

[11] Dingledine, Roger, Nick Mathewson, and Paul Syverson. "Tor: The second-generation onion router." Naval Research Lab Washington DC, 2004.

[12] Benet, Juan. "IPFS-content addressed, versioned, P2P file system." arXiv preprint arXiv:1407.3561 (2014).

[13] Biscuitwala, Kanak, Willem Bult, Mathias Lécuyer, T. J. Purtell, Madeline KB Ross, Augustin Chaintreau, Chris Haseman, Monica S. Lam, and Susan E. McGregor. "Dispatch:

Secure, resilient mobile reporting." In ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 459-460. ACM, 2013.

[14] Waldman, Marc, Aviel D. Rubin, and Lorrie Faith Cranor. "Publius: A Robust, Tamper-Evident Censorship-Resistant Web Publishing System." In 9th USENIX Security Symposium, pp. 59-72. 2000.

[15] "Tor: Onion Service Protocol", Tor Project, torproject.org/docs/onion-services.html.en

[16] "News Orgs & Activists: Onionize Your Sites Against Censorship" Tor Project Blog blog.torproject.org/news-orgs-activists-onionize-your-sites-against-censorship. January 24, 2018

[17] Clarke, Ian, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. "Freenet: A distributed anonymous information storage and retrieval system." In Designing privacy enhancing technologies, pp. 46-66. Springer, Berlin, Heidelberg, 2001.

[18] Freenet. freenetproject.org

[19] Anderson, David P. "Boinc: A system for public-resource computing and storage." In proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 4-10. IEEE Computer Society, 2004.

[20] Marosi, Attila, József Kovács, and Peter Kacsuk. "Towards a volunteer cloud system." Future Generation Computer Systems 29, no. 6 (2013): 1442-1451.

[21] Anderson, David P., Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. "SETI@ home: an experiment in public-resource computing." Communications of the ACM 45, no. 11 (2002): 56-61.

[22] "Project list". BOINC. http://boinc.berkeley.edu/wiki/Project_list

[23] Kacsuk, Péter, József Kovács, Zoltán Farkas, A. Cs Marosi, and Zoltán Balaton. "Towards a Powerful European DCI Based on Desktop Grids." Journal of Grid Computing 9, no. 2 (2011): 219-239.

[24] Marosi, Attila Csaba, Zoltan Balaton, and Peter Kacsuk. "GenWrapper: A generic wrapper for running legacy applications on desktop grids." In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pp. 1-6. IEEE, 2009.

[25] A. Csaba Marosi, P. Kacsuk, G. Fedak, O. Lodygensky, Using virtual machines in desktop grid clients for application sandboxing, Technical Report TR-0140, Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID—Network of Excellence, August 2008.

[26] Marosi, A. Csaba, Peter Kacsuk, Gilles Fedak, and Oleg Lodygensky. "Using virtual machines in desktop grid clients for application sandboxing." In 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP 2010). 2010.

[27] FileCoin. https://filecoin.io/

[28] Cohen, Bram. "Incentives build robustness in BitTorrent." In Workshop on Economics of Peer-to-Peer systems, vol. 6, pp. 68-72. 2003.

[29] Adar, Eytan, and Bernardo A. Huberman. "Free riding on Gnutella." First monday 5, no. 10 (2000).

[30] Sun, Hailong, and Jinpeng Huai. "Combining Reliability and Economic Incentives in Peer-to-Peer Grids." ICN'09. Eighth International Conference on Networks. IEEE, 2009.

[31] Haeberlen, Andreas, Petr Kouznetsov, and Peter Druschel. "PeerReview: Practical accountability for distributed systems." ACM SIGOPS operating systems review 41, no. 6 (2007): 175-188.

[32] Maniatis, Petros, Mema Roussopoulos, Thomas J. Giuli, David SH Rosenthal, and Mary Baker. "The LOCKSS peer-to-peer digital preservation system." ACM Transactions on Computer Systems (TOCS) 23, no. 1 (2005): 2-50.

[33] Reich, Vicky, and David SH Rosenthal. "LOCKSS (lots of copies keep stuff safe)." New Review of Academic Librarianship 6, no. 1 (2000): 155-161.

[34] Germain-Renaud, Cécile, and Nathalie Playez. "Result checking in global computing systems." In Proceedings of the 17th annual international conference on Supercomputing, pp. 226-233. ACM, 2003.

[35] Alvisi, Lorenzo, Dahlia Malkhi, Evelyn Pierce, and Michael K. Reiter. "Fault detection for Byzantine quorum systems." IEEE Transactions on Parallel and Distributed Systems 12, no. 9 (2001): 996-1007.

[36] Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." In OSDI, vol. 99, pp. 173-186. 1999.

[37] Rescorla, Eric. "HTTP over TLS." (2000).; www.ietf.org/rfc/rfc2818.txt.

[38] Callegati, Franco, Walter Cerroni, and Marco Ramilli. "Man-in-the-Middle Attack to the HTTPS Protocol." IEEE Security & Privacy 7, no. 1 (2009): 78-81.

[39] Mirkovic, Jelena, Sven Dietrich, David Dittrich, and Peter Reiher. "Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)." (2004).

[40] Sanders, James. “Chinese government linked to largest DDoS attack in GitHub history”. TechRepublic. April 3, 2015.

[41] Great Firewall of China. www.greatfirewallofchina.org

[42] Dikshit, Sandeep. “Bid to block anti-India website affects users”. The Hindu. Sep 23, 2003

[43] Huang, Xiao Long, and Brahim Bensaou. "On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation." Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing. ACM, 2001.

[44] Bonald, Thomas, Laurent Massoulié, Alexandre Proutiere, and Jorma Virtamo. "A queueing analysis of max-min fairness, proportional fairness and balanced fairness." Queueing systems 53, no. 1-2 (2006): 65-84.

[45] Nace, Dritan, and Michal Pióro. "Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial." IEEE Communications Surveys & Tutorials 10, no. 4 (2008).

[46] Wang, Wei, Baochun Li, and Ben Liang. "Dominant resource fairness in cloud computing systems with heterogeneous servers." In INFOCOM, 2014 Proceedings IEEE, pp. 583-591. IEEE, 2014.

[47] Shue, David, Michael J. Freedman, and Anees Shaikh. "Performance Isolation and Fairness for Multi-Tenant Cloud Storage." In OSDI, vol. 12, pp. 349-362. 2012.

[48] Apache ZooKeeper. https://zookeeper.apache.org/

[49] Shahri, Alimohammad, Mahmood Hosseini, Raian Ali, and Fabiano Dalpiaz. "Gamification for volunteer cloud computing." In Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on, pp. 616-617. IEEE, 2014.

[50] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).

[51] Vectordash. https://vectordash.com

[52] OrbitDB. github.com/orbitdb/orbit-db

[53] Chun, Brent, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. "Planetlab: an overlay testbed for broad-coverage services." ACM SIGCOMM Computer Communication Review 33, no. 3 (2003): 3-12.

[54] Foster, Ian T. "The Globus Toolkit for Grid Computing." In ccgrid, vol. 1, p. 2. 2001.