Membrane - Distributed File Backup - Literature Review

Dominic Hauton

January 25, 2017

Abstract

An application in which users can join a large public network of intelligent storage agents to trade their available file space. This will allow the user to backup and version files safely on potentially untrustworthy computers, increasing availability and redundancy. The hosts will act as intelligent agents, and trade space when they communicate with one another.

0.1 Introduction

Distributed storage is a well studied and explored domain with clear advantages over bare metal. In order to implement a peer-to-peer distributed agent based storage system we must first decompose the problem into several parts and explore the advances made in those fields.

With the advent of cheap high-speed internet users are now able to use monolithic cloud services to backup data. These tend to be expensive for anything but small amounts of storage and many people have expressed security concerns over holding their data in a data centre owned by another company. Especially if their data leaves the users country where data protection laws may be different.

Over time personal storage capacities for users have also increased. It is now common for computers to come with large amounts of hard drive space which is often not filled to capacity. The project proposed promises to swap this free hard drive space to back up other users' data in exchange for their surplus space to backup your data.

To simplify the process, the system will be able to negotiate contracts of varying complexity for space allocation on another machine, in exchange for space on itself. Unlike most distributed databases and cloud storage solutions frequent down-time will be expected across devices and contracts will have to account for this.

The challenges of such a system include a way of only copying parts of a file during updates to reduce bandwidth usage. An effective way of managing where file chunks are stored. A way to encrypt the files on the remote host and a way to authenticate that the host still holds the file.

Part of the required research also includes exploring the more recent technology of intelligent agents, in which we are most interested in trust metrics. Intelligent agents will allow the software to make storage decisions based on the trust and reputation of other nodes in the network.

0.2 History of Problem

A simple file backup can be imagined as simply copying a file to another location. In order to keep the duplicated file in sync they must be compared. The program diff solves this through finding the longest common subsequence of bytes between files. In order to improve performance hashing, presorting into equivalence classes, merging by binary search, and dynamic storage allocation are used. [Hunt and MacIlroy, 1976]. This allows the user to view changes and copy the file over again if required.

0.2.1 Rsync

In a networked scenario, bandwidth from source to destination is at a premium. Rsync, introduced in 1996 presents a much better solution through copying changed file chunks (deltas). [Tridgell et al., 1996]. Rsync splits the file into shards and calculates a weak rolling checksum and strong MD4 checksum for each block that allows quick comparisons of shards along the file. When a discrepancy is found, we assume an extra byte or bytes have been added to the file. The weak checksum can be efficiency recalculated for the next offset and once there is a match, it is confirmed with the strong checksum. The new added chunk can now be transmitted. This results in a lot less data being copied than there would be with a diff file. [Tridgell et al., 1996] This combination of weak and strong checksums has been used across multiple distributed systems including low-bandwidth file systems [Muthitacharoen et al., 2001] and high performance

transactional object stores. [Stephen et al., 2000].

Multiround Rsync improves on the rsync algorithm by allowing for more communication to lower bandwidth. Blocks of smaller and smaller sized are used to find holes in the old file in each round and the file until the minimum block size is reached and a copy occurs. [Langford, 2001] This works better than standard rsync in situations where the source file has been changed in many places distributed around the file.

Rsync requires both old and new copies of a file to exist on the host system during an update. This issue has been addressed by creating an in-place rsync (ip-rsync) that uses techniques used in delta compression and Lempel-Ziv compression to move the areas of the file around. In ip-rsync file sender sends add and move commands to the destination in an order that guarantees no files will be overwritten. [Rasch and Burns, 2003]

0.2.2 Git

Git is an improvement on Rsync as it provides both version history and minimises data transfer. To keep storage simple, a copy of the whole file is stored and a reference is put into the version history. By storing old files locally operations are fast. This is also an important distinction from other version control systems and one of the reasons why Git was chosen as an example versioning system compared to other versioning systems like SVN. The systems can continue to operate without a centralised server. To reduce file duplication all files are referenced using their SHA-1 hash. This means you can be sure the contents of the file hasn't changed since you last read it. [Torvalds and Hamano, 2010]

Git also uses a 3 stage workflow. A working directory, where the current files are stored, a staging area and a .git directory. The staging area prepares your next commit and then it is finally committed. When the staging is complete the change is irreversibly stored. This is a good approach that will be adopted in the final software solution. It will allow incrementally finding changed files, and assessing the need for a new version number to be saved.

0.2.3 Bittorrent

The BitTorrent protocol is a mechanism for sharing files between swarms of hosts. As BitTorrent splits files into parts, users start sharing data even before they have received the full file. Each file has a SHA-1 identifier, similar to Git. [Qiu and Srikant, 2004]

If a user wishes to download a file from the swarm, the user downloads a metadata file from the web and locates users sharing the data using a Tracker Server, Distributed Hash Table (DHT) or Peer Exchange (PEX). [Cohen, 2008]

A Tracker server is a centralised store of all current connected users along with how much of the file they hold. This approach is vulnerable to exploitation by authorities as all of the data about a swarm is strored on a single server and as a result cannot be used for the proposed system.

A DHT contacts other known users for information instead of a centralised server. The Mainline DHT as outlined in BEP No.5 is based on the Kademlia protocol that allows for decentralised peer discovery for a particular piece of content.

PEX is a method for two clients to share a subset of their peer lists. Coupled with DHT, PEX removes a vulnerability from the Bittorrent network by allowing fully distributed bootstrapping, tracking and peer discovery.

A DHT with a form of PEX a tried and tested way of successfully mapping and finding files on a network and will be used within the proposed project.

0.2.4 Resilio

Resilio Sync is an example of a distributed file storage system that utilises the BitTorrent protocol to automatically synchronise folders between a user's systems. It is not a cloud backup solution and not intended as a form of off-site storage. There is no distributed file system and as a result, no redundant data block algorithm adding complexity. [Farina et al., 2014]

As Resilio Sync uses DHT to transfer data, there is no central authority to manage authentication or log data access attempts. This makes it difficult to determine whether a file has been accessed by another user. [Farina et al., 2014] As a result in the project the assumption will be made that everyone in the network has access to all encrypted file chunks. To access and reassemble a file, a user will be required to request all of the file chunks individually and then locally reassemble them.

0.2.5 Storj

Storj is a peer-to-peer cloud storage network which aims to allow users to store files on a decentralised platform. The platform takes special care to provide protection against Sybil attacks and other forms of fraud. [Wilkinson et al., 2014]. To store files it stores encrypted hashed shards on the network. In order to provide proof of storage it uses Merkle Audits and pre-generated audits with hash-challenges to determine whether the client still holds the required data. By adding a seed to the hash-calculation the client can enforce the workers are still in possession of the data. It prevents the client cheating a farmer through using blockchain proof-of-existence to keep both parties honest.

The most efficient form for proof of storage is through using a deterministic heartbeat. Using Feistel permutations data can be verified with $n + 2\sqrt{n}$ challenges. Erasure encoding is added to shards to detect any minor changes to the data. This is less I/O intensive than a full heartbeat, but still allows an attacker to complete heartbeats with only a data integrity of 1/n, where n is the number of nodes holding the data.

In order to add extra protection to files, we can use erasure encoding to allow file recovery if one of our shard types is lost. This can be investigated in our software, however, as the shards are expected to change on a regular basis because of versioning, this may not be possible.

To prevent Sybil based attacks, Storj encrypts each shard segment with a different salt. This stops workers completing proof of storage on behalf of another node.

0.3 Peer Admission

The first step in designing the distributed file system is locating other peers within the swarm. This is accomplished through Peer Admission. Once the first peer is found data within the swarm can be located using a DHT which guarantees content can always be found.

0.3.1 Bootstrapping

There are two types of PTP networks which must be examined:

- *Asynchronous*
- *Synchronous*

Within Synchronous networks the number of nodes on the network is constant and all of the nodes are aware of each others existence. This does not allow storage networks to scale but it does allow data to be kept private. [Saxena et al., 2003] This is the simplest and first approach that will be taken in locating nodes within Membrane.

In most current P2P systems such as Gnutella [Klingberg and Manfredi, 2002], Chord and Tapestry as well cryptocurrencies such as in Bitcoin and Litecoin a bootstrapping node is contacted, which provides information about what clients are currently online. Once a bootstrapping node allows the client to find the edge of the swarm, more information can be found using peer exchange.

Within a local network we can also use Universal Plug and Play to find other nodes within the local network. This prevents an external call to a bootstrapping node and as a result is less prone to attack.

Through looking at availability metrics within Bittorrent systems Neglia et al. [2007] determined that both trackers and DHT should be used in creating a highly available distributed storage system such as BitTorrent. DHT tends to be slower at finding new data, however it is much more reliable.

Within Membrane, I plan to use a combination of Asynchronous and Synchronous techniques. Users will try to bootstrap from their last known neighbour nodes on the network, this takes advantage of the static nature of the backup data. Only if this fails, and with the user's permission will they contact a centralised bootstrapping node. This should only happen during a first install and if the user has no referrals. Throughout the lifetime of the application the centralised bootstrapping node would ideally be replaced with a referral system.

A further extension of this, would be to allow hosts to provide a DNS name along with their IP. Users that have setup Dynamic DNS (DDNS) [Bound and Rekhter, 1997] would be able to locate each other without the help of a bootstrapping server.

0.3.2 Peer Exchange

When bootstrapping is complete new Peers can increase their knowledge of the network through Peer Exchange. This is used by Bittorrent to help share swarm information with other nodes. As soon as a client connects to the swarm, peer information is collected using DHT or PEX.

There are two common extension protocols called AZMP and LTEP, which send at most one message a minute when a client leaves or exits the swarm. To reduce congestion at most 50 peers can be added or removed in one PEX message. [Vuze, 2010]

Shard Discovery

Bittorrent also uses the Mainline DHT to find other hosts in the network. This is a Kademlia DHT which now according to Jones [2015], now supports 25M users. It works through assigning each node and file a 160-bit string as an ID. We can work out which node is meant to store a file metadata and crawl in the direction of the node using a hill climb algorithm. Once the metadata is stored on the host, if a host host wants to download a file, it can take the metadata on the known host to find the IP of hosts with the file.

Hosts in Membrane will store a metadata file with all of the information required for the specific account to run, including friends, encryption keys as well as local mappings for which host owns which shard and which IP belongs to each host. As a result a DHT will only be required if this initial metadata is lost and needs to be recovered. The proposed version of Membrane will make use of this metadata, but recovery can be added in further iterations of the project.

Dynamic IP Address

Dynamic IP addresses have proven to be problematic in distributed computing, as ISP typically charge more for users to have a static, unchanging IP. Bittorrent tackles this issue through using a DHT to dynamically find the IP address of the user that owns a file. This approach is robust, however, it is a complex solution to IP address resolution.

Another widely used approach is using Dynamic DNS (DynDNS) as described by Bound and Rekhter [1997] in RFC 2136. This allows a client to automatically update a nameserver with a new IP or other information, this allows clients to have a persistent addressing method for devices that change their location. This approach requires initial configuration by the user, however, it provides a reliable way to connect with a user when their IP is lost. There are several tools such as MintDNS, cURL and Iandyn that could be used to ease the development of a built in DynDNS. When setting up a relationship with another host, both an A/AAAA Address and CNAME could be provided, where the CNAME is a backup if the A/AAAA address does not work.

To resolve IP Address resolution within Membrane, I would like to take advantage of small-world networks. [Porter, 2012], in which the mean shortest-path between two nodes increases slowly compared to the number of notes in a network. Within a group of users in Membrane hosts are likely to share multiple first, second and third degree connections. By storing a list of IP addresses from all of the hosts. It is highly unlikely that all connections within three hops will have changed IP address. We take inspiration from ARP [Plummer, 1982], and send broadcasts with a limited hop count in the network to see if anyone is aware of the current address of a host. The downside of this approach is that it relies on a node within your social network to be online. Measures will need to be put in place to reduce broadcast spam.

A broadcast storms runaway broadcast events, common in networks that used broadcasting for communication, particularly when areas of the network overlap. [Tseng et al., 2002] These can be mitigated by reducing broadcast traffic, however, Membrane will rely on broadcasts to find hosts. The first step to limiting these broadcasts, is implementing a hop count on broadcasts. This is commonly seen in routing protocols such as IPv6 [Deering, 1998]. The Spanning Tree Protocol (STP) as seen in IEEE 802.1d [Group et al., n.d.; Sharma, Gopalan, Nanda and Chiueh, 2004] provides loop-free routing in LANs by pruning redundant links. Topology changes are dealt with by rebuilding the tree.

Within Membrane rebuilding a Spanning Tree would be an expensive operation. If broadcasts become a problem a 'block request' system can be used, similar to that of ICMP redirects. [Postel et al., 1981] If a node receives a duplicate broadcast message it sends a request back one hop to not send broadcasts from that source toward it for some time. The time limit would allow allow for corrections if the network topology changes. This preventative approach and could be improved by using the full Spanning Tree implementation. As a further step it could be improved by using a DHT 'closest jump' approach if required.

0.4 Data Allocation on External Nodes

In order to store data on another node Membrane must first have permission to store files on another node. In order to make a choice we must be able to look at trust information about other nodes on the network, and negotiate and trade space once a suitable candidate has been found. These two areas have been explored in the context of Multiagent Systems (MAS) in

the past. [Wooldridge, 2009]

0.4.1 Negotiation

Negotiation aims to reach a level of resource allocation that is acceptable for all involved parties. [Rahwan, 2005] It allows two or more parties that value each others service, to participate in a mutually beneficial exchange of services, however, as there are multiple beneficial outcomes it can be defined as "distributed search through a potential space of agreements" [Jennings et al., 2001] Within Membrane, this service is storage space, that is physically separated from the current user. We now take a look at negotiation and how we can build a negotiation framework that our agents can use to exchange storage.

There are three main areas that are important for negotiation. Negotiation protocols, negotiation objects and the node's reasoning models. [Beer et al., 1999] The sophistication of the negotiation is determined by each of these and can take different forms such as auctions, argumentation and protocols in the style of a contract net. The simplest negotiation uses fixed size items, which Membrane shall be initially using for this reason. More complex negotiation allows for counter offers and stronger guarantees.

A simple negotiation protocol issues a call for proposals to a number of nodes and waits for their bids. To formalise this a Agent Communication Language (ACL) such as KQML (Knowledge Query and Manipulation Language) [Finin et al., 1992] or the more modern FIPA (Foundation for Intelligent Physical Agents) [Fipa, 2002]. [Rahwan, 2005] Beer et al. [1999] tells us that within KQML the agent sending the query is expected to decide how the receiver will handle it, which places limits on negotiation. On the other hand, FIPA is newer and as a result can be more error prone.

Negotiation Logic

The form of negotiation within Membrane also needs to be decided. We must first decide on a reasoning mechanism to use within the agent. The distinction between monotonic and non-monotonic logic is important in the study of AI and multiagent systems. In non-monotonic logic new axioms (or knowledge) can invalidate old theorems. [McDermott and Doyle, 1980; Antonelli, 2008] This is important in the real world as we need to be able to make assumptions on facts and retain flexibility in our understanding of the world. Within a MAS a non-monotonic logic can be more difficult to implement as theorems need to be constantly asserted, and as a result they often result to first-order (or monotonic) logic. Within Membrane we need to implement a monotonic logic to assert trust in our contracts and negotiations. In the context of a negotiated contract, throughout it's duration, we cannot effort to re-evaluate our trust of the agent.

Negotiation tactics

In order to exchange storage space we must find the most suitable node in the network. There are multiple negotiation tactics between agents for collaboration and coming to an agreement. [Beer et al., 1999] We shall explore the advantages and disadvantages of game-theoretic approaches [Rosenschein and Zlotkin, 1994; Kraus, 2001; Sandholm, 2002], heuristic-based approaches [Faratin, 2000; Fatima, Wooldridge and Jennings, 2002] and argumentation-based approaches [Kraus, Sycara and Evenchik, 1998; Jennings, Parsons, Noriega and Sierra, 1998] as well as exploring practical implementations negotiation. Ideally a negotiation mechanism is computationally cheap, produces good outcomes, distributed, fair to all participants, compatible with fixed strategies and is able to function without com-

plete information. [Rahwan, 2005]

Using a *game theory* approach we assume all agents are self-interested and allows agents to analyse optimal behaviour. [Osborne and Rubinstein, 1994] We can apply this in agent reasoning by giving each combination of collaborations a utility. Doing this an optimal set of interactions can be calculated. This can even be used to help agents interact in a certain way. [Varian, 1995]. The main downsides include the assumption of unbounded computational resources, complete knowledge of the outcome space. [Rahwan, 2005] This makes a game theory approach unusable in Membrane as the network is not fully understood by each agent.

A *heuristic* approach produces 'good enough' negotiations. Instead of exploring the full extent of possibilities they focus on the subset most likely to lead to positive interactions. In the context of Membrane, this may be hosts that you have had successful interactions with before. The downsides of this approach is that it becomes difficult to predict the negotiation actions of other agents and as the full search space is not explored the result may not be optimal. [Jennings et al., 2001]

A *argumentation* approach is beneficial when a flexible negotiation is required and the agents have limited knowledge of the world. It's commonly used in the human world by advertisers to convince humans to try products. [Slade, 2002]. Instead of simply rejecting an offer a agent can say why or offer a counter-proposal, which can result in more successful negotiations. Although this approach offers far better negotiation, it is much more complex to implement and is not required within initial implementations of Membrane.

Practical Negotiation

We now look at practical real-world negotiation. Real world approaches take into consideration the practical implications of reasoning systems and prove that concepts work. Within cloud computing an agent is often required to request a service. The use of these resources is based on service-level agreements (SLAs) which are designed to provide users with a service when requested. [Paletta and Herrero, 2009] One critical issue in SLAs determining the Quality of Service (QoS) constraints of the offered service.

Yan et al. [2007] explores creating a SLA negotiation system, splitting SLA negotiation into three parts. Defining *Negotiation Protocol* that allows participants to send offers to each other, *Negotiation Coordination* which ensures the final result of the negotiation fulfils the QoS requirements of the agent and a *Decision Making Model* which allows the parties to decide if they are satisfied with the deal.

Within Membrane, hosts are unable to know if they will be compatible with another agent. It might be the case that they both have 50% uptime, however they are never online at the same time. As a result a system will have to be created to allow agents to find if they are compatible with each other and storage decisions will have to be based on this.

Paletta and Herrero [2009] presents an negotiation mechanism which makes use of mean algorithm and protocol, using key awareness concepts first proposed by Herrero et al. [2007]. It uses an quantitative metric in the range [0, 1] to decide how much the agent should be collaborating with another agent. The agents can then communicate using the messages:

1. REQUEST - Can you perform A?
2. CONFIRM - Yes I will do A.
3. DISCONFIRM - No I will not do A.

Collaboration decisions decided using an Artificial Neural Network (ANN). Three metrics are used, physical resource availability, if the node is available for collaboration and the number of previous collaborations of the same type. The ANN used was a Multi-Layer Perceptrons (MLPs) using one hidden layer and two units.

When evaluated the mechanism was able to deal with 93% of situations and negotiation with the first node was successful 68% of times. [Herrero et al., 2007]

With Membrane we take this model and make it more practical by introducing reputation to allow third party ratings to be shared.

Service Level Agreements

In order to create a negotiation system we need to explore what makes up an successful software SLA. Keller and Ludwig [2002] defines the WSLA framework which sets out to create a SLA based negotiation frame work. They describes 3 key areas that must be present in an SLA.

1. Parties Involved
2. Service Description
3. Obligation

Within Membrane the parties involved will always be the two nodes exchanging information. The service description will be a promise to store a block of data on another host. The obligation will include various parts such as proof of storage and the ability to update and retrieve the data a set number of times. These will be explored when the SLAs for membrane is designed.

Another key area Keller and Ludwig [2002] discusses is the 5 stages of an SLA lifecycle.

1. SLA Negotiation and Establishment
2. SLA Deployment
3. Measurement and Reporting
4. Corrective Management Actions (in case of violation)
5. SLA Termination (in case of violation)

These are the backbone of every real SLA and will be explored in more detail while designing the negotiation system for Membrane.

0.4.2 Trust and Reputation

Agents in a distributed system need to be able to protect themselves from 'bad' agents. Pinyol and Sabater-Mir [2013] describes three main approaches to control the acts of agents. The *Security Approach* which guarantees the authenticity and integrity of interactions. The *Institutional approach* which relies on a central authority to enforce good behaviour and finally the *Social approach* which allows agents themselves to punish other agent for 'bad' behaviour. This final approach is where trust and reputation is used.

Trust can be used to predict the behaviour of an agent. [Wooldridge, 2009] A classification presented by Balke and Eymann [2009] defines 5 stages that exist in reputation and trust models. Co-operative behaviour is first stored and then rated using a utility function. Within Membrane it is easy to see the utility being rated as shared transaction time. This cooperative behaviour is then stored in an image of the other agent at a predetermined level of detail. This image can now be recalled to infer trust. Finally the agent can use this trust to learn and adapt it's strategy.

The ReGreT model [Sabater and Sierra, 2001] is "one of the most complete reputation and trust models" [Pinyol and Sabater-Mir, 2013] so we shall use it to see what a good trust system includes. It uses direct experience, third party information and social structures to calculate trust, reputation and credibility of another agent. An important note for

this model is that trust is contextual. Agent a will trust b while certain conditions are met. In the context of Membrane, perhaps another agent will refuse to return our data, if we cannot prove that we still have their data, which would mean that they would be useless if the client experienced completed data loss. We must therefore consider simulating a situation in which complete data loss was experienced.

Trust and Reputation Attacks

Attempts to misrepresent reliability and manipulate reputation are common in traditional communities and have been exploited by con artists for centuries. In a distributed system agents must be able to protect themselves against common trust attacks. Jøsang and Golbeck [2009] describes 9 potential attacks on reputation systems.

- Playbooks - Gain high reputation and burn it quickly with low quality actions
- Unfair ratings - Give incorrect reputation or image
- Discrimination - Give high quality service to one set of users and low quality to another set
- Collusion - A coordinated reputation attack.
- Proliferation - Offer a service through multiple channels
- Reputation Lag Exploitation - Provide a large number of low-quality services quickly
- Reentry - Change identity to recover reputation
- Value Imbalance Exploitation - Gain reputation with easy actions
- Sybil Attack - Inflate reputation with fake accounts.

When implementing a trust system in Membrane these attacks should be considered and special attention should be paid to prevent a new user being exploited for their storage space when joining the swarm.

0.5 Communication

As Membrane is a distributed system communication is key for nodes on the network to interact. It is traditional to form a protocol that agents can use to share information with each other. These protocols are often very rigid and do not allow for expansion. Formalising communication using ACLs is a more flexible approach that aims to let agent share a common understanding of the domain, and allows hosts to reason about communication themselves. Instead of using a strict protocol we shall instead take an agent-based approach to communication using ACLs.

0.5.1 Agent Communication Language

On balance, we shall be using FIPA for communication. This is an ACL based on Speech Act Theory [Labrou et al., 1999]. It splits communication into a communicative act (CA), an ACL message structure and a set of communication protocols such as XML or OWL (Web Ontology Language). In FIPA communication should be rational, in that when sending a message:

- The sender believes the proposition.
- The recipient does not already believe the proposition.
- The recipient will believe the proposition after the proposition.

In the case of Membrane this could be put into the context of asking another node to store data, it would only be reasonable to send another node data to be stored, if the two nodes had negotiated storage of that block between them previously.

To keep communication simple for nodes, we shall use two CAs

- REQUEST
- INFORM

where *REQUEST* expects a reply of some sort and *INFORM* does not. This will enable easier implementation and can be expanded if required.

0.5.2 Ontology

An ontology is a way of defining basic terms and relations comprising the vocabulary of a topic area. Sugumaran and Storey [2002] tells us the the three most commonly used relationships in an ontology are *is-a*, *synonym* and *related-to* (which is a generic association between entities). So why should we create an ontology for membrane? Noy et al. [2001] lists the benefits of ontology within distributed systems:

- Enable common understanding of the structure of information
- Enable reuse of domain knowledge
- Make domain assumptions explicit
- Separate domain and operational knowledge
- Analyse domain knowledge

Within Membrane we could benefit from an ontology during negotiation for storage space. Obligations could be predefined between agents and an agent could request a set of obligations, instead of explaining an obligation during negotiation. This allows agents to be more concise and expressive. Noy et al. [2001] gives us 7 steps for ontology creation.

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumerate important terms
4. Define classes and their hierarchy
5. Define class properties (slots)
6. Define slot facets
7. Create Instance

It is important to ensure the ontology goes into the right amount of detail for the requirements. An overly detailed ontology can make reasoning difficult, and an ontology that is too simple will limit expressiveness and reasoning. [Wooldridge, 2009; Sugumaran and Storey, 2002].

When considering ontologies for reuse Noy et al. [2001] points us towards the DAML ontology library (http://www.daml.org/ontologies/). Using this and Google I was able to locate 2 ontologies, the datastore schema provided by purl.org [2001] and an OWL based ontology for SLA services provided by Pande Joshi and Finin [2012]. Both of these will be examined further during the ontology creation process.

0.6 Peer-to-Peer Connection

Network Address Translation (NAT), defined in RFC 2663 by Srisuresh and Holdrege [1999] is used extensively to solve the IP exhaustion problem, however, it also creates a lot of well documented problems for peer-to-peer communication as the IP address a hosts has set, may

be a private and only useful for peers on the same network. Multiple Membrane hosts may be using the same IP address and the TCP listening port used by a node will change between connections. In order to transfer data over NAT a NAT-Traversal Technique must be employed. There are four major techniques described by Ford et al. [2005] that we shall explore.

Relaying is a method of communicating through a well known server, that has a static IP address and port. This has the advantage of being simple, however, it has the disadvantage of using the server's processing power, network bandwidth and increases latency between peers. It is a good fallback strategy but also requires both clients to initiate the connection, which makes listening for incoming connections difficult. The TURN protocol [Rosenberg et al., 2005] describes a fairly secure method for relaying.

Connection Reversal is the use of an external server to request a 'call back' to a host. The connecting host contacts an external server with a target host and leaves it's public IP and port. The relaying host can then choose to send a request to host behind NAT and the host behind NAT can choose to forward the request. It has similar limitations to relaying, however, it is easy to imagine an implementation that uses connection reversal for bootstrapping a P2P connection.

UDP Hole Punching is a technique defined in RFC 3027 5.1 [Holdrege and Srisuresh, 2001] that allows two clients behind NAT to connect with the use of a rendezvous (RV) server. Hosts contact the RV and leave their UDP port and address. You should be weary of poor NAT implementations translating IP addresses within data when packets are coming in and out. [Ford et al., 2005] When clients are both behind the same NAT, packets between them do not need to go through NAT. If Hairpin NAT is correctly configured they will be bounced back within the internal network, however, it could result in overheads otherwise. If this problem is detected during the implementation of Membrane private IP address could also be sent to the RV server. It's important to consider that the ports behind a NAT are dynamically allocated, so connections must be authenticated between connections, to ensure the IP address and port still direct towards the same host. The initial outbound packet from a host to a the RV server is key in 'punching a hole' through the NAT. The timeout of a hole punched port can be as low as 20 seconds, so the host must maintain the hole by sending single packets to the RV on a regular basis.

TCP Hole Punching is more complex than with UDP as clients need to establish sessions across the NAT. For hole punching to work the same port needs to be used for listening and establishing a connection. This can be done using *SO_REUSEPORT* option, which allows an application to claim multiple sockets from the same endpoint. Within the application an new port needs to used for handling the connection.

There are two practical solutions that I shall explore for implementation within Membrane. TURN [Wing et al., 2010] provides a set of methods for NAT hole punching. UPnP [Boucadair et al., 2013] is a method through which a host can request a NAT gateways to open a port for the application. There have been security concerns with UPnP which has resulted in UPnP being disabled on many routers. This will need to be explored during implementation of Membrane.

0.7 Distributed File System

Membrane shards files and stores copies on multiple nodes for redundancy. This is a technique that is commonly found in distributed file systems (DFS) such as Hadoop File Sys-

tem (HDFS) [Shvachko et al., 2010], Google File System (GFS) [Ghemawat et al., 2003] and Parallel Virtual File System (PVFS) [Ross et al., 2000]. This section aims to explore the reasoning and decisions taken with these file systems.

HDFS has proved to be highly successful in storing large quantities of data over thousands of nodes and is used by over 100 organisations world wide. It stores file system metadata on a central dedicated server called a NameNode [Shvachko et al., 2010]. This is a similar approach to GFS [McKusick and Quinlan, 2010]. Membrane takes this approach by storing file metadata on the client. Ceph takes this approach further, storing file meta on a distributed cluster of NameNodes, using hash function to spread metadata over the name nodes. [Weil et al., 2006] This allows for larger, more distributed system, this could be used within Membrane if meta-data proves too large to store on the client.

Files and directories are represented my inodes, these record attributes such as permissions and location. Within Membrane this will need to store information such as checksums, encryption keys and SLA for each file, to allow each file to be verified on a regular basis. When writing data to the cluster the NameNode chooses the targets for shards. The client in Membrane will be able to negotiate shard targets on the network using mechanisims described previously.

The NameNode stores the metadata as a combination of a checkpoint and journal (a write-ahead commmit log). If there is a fault on the NameNode, such as a power outage, the journal can be replayed over the latest checkpoint. In HDFS a new snapshot is created during a node restart, however, in Membrane the journal is kept to be able to restore the metadata during any point in the journal. By not removing any shard that is still in the journal we are able to rollback to any previous state in ahead of the oldest checkpoint.

In HDFS, during startup a DataNode connects to a NameNode, and performs a handshake with a unique storage ID that can identify it. In addition a DataNode will also send heartbeats and reports on a regular basis. Shvachko et al. [2010] This is a great approach as it makes the DataNode responsible for notifiying the NameNode that it still holds the data. Not the other way around. A similar approach will be used in Membrane, however, the NameNode will also request some proof that the DataNode still holds the file, as in Storj.

In both GFS and HDFS data is streamed directly from the client to the DataNode in 64k chunks and a checksum is sent with the original file to assert the transfer was successful. Both systems also have a default of 3 replicas per shard, which Membrane shall adopt. TCP is used for communication as it provides reliability and a session for transfers. Chunks are given a 64bit chunk handle. Membrane will use a hash to compute this identifier. This should help nodes prevent attackers predicting a chunk handle.

0.8 Authentication

In order to maintain a relationship a host needs to be able to verify it's identity when it connects to a host a second time. When visiting a website there is often a need to map a virtual identity to a real identity of a service user [Hericourt and Le Pennec, 2001] a bank for example, this is typically done using a certificate authority. Membrane requires authentication in two situations:

- Is this the same node I was talking to before?
- Have I found the correct node when adding a friend?

Session authentication is a common occurrence while using the web and it typically happens with every https site visit. A session cookie is stored on both clients to allow for authorisation and authentication without the use of a password in further interaction. [Mayo et al., 2008]

SSL/TLS, defined in RFC 5246 [Dierks, 2008] is often used for this authentication. It uses asymmetric public-key cryptography for initial connection, which uses a separate key is used for encryption and decryption. A client generates a public SSL certificate that it sends for any users that want to communicate with it. They encrypt their communication with it, and the server can decrypt it. This is called a SSL Handshake.

1. Server sends the client it's asymmetric public keys
2. Client creates a symmetric session key and encrypts it using the public key.
3. Server decrypts the encrypted session keys
4. Client and Server can now communicate using the symmetric key.

The client can ensure it is talking to the same server by ensuring the public key is the same. RSA is the commonly used algorithm for this.

Pre-shared key encryption uses algorithms like Twofish, AES or Blowfish to create keys. These come in two flavours; stream ciphers and block ciphers. Stream ciphers encrypt binary digit by binary digit in a stream, and block ciphers encrypt a block of data at a time. We shall be using pre-shared key encryption to encrypt data blocks using a password saved on the host, that never under any circumstances leaves the Membrane host.

Studies have shown that Twofish (and Blowfish which it is derived from) has the best performance and has no know security flaws. AES showed poorest performance. [Thakur and Kumar, 2011; Rizvi, Hussain and Wadhwa, 2011; Mushtaque, Dhiman, Hussain and Maheshwari, 2014] An interview with the creator of Blowfish suggested "I'm amazed it's still being used. If people ask, I recommend Twofish instead." Schneier [2007]. One thing to note is that encrypted data is expanded, testing has shown a 240KB file encrypted with Blowfish expands to 955KB which is almost a 4 times increase. [Mushtaque et al., 2014]

0.9 Proof of Ownership

A Membrane client needs a method to ensure the storage host still holds the data it says it has. To prevent Sybil attacks where agents collude to store a shard between each other, a shard will be salted before encryption and transfer. This has proven effective in Storj [Wilkinson et al., 2014].

In order to determine consistency of what the storage agent has stored a mix of different hash verification is used.

- Full Heartbeat - Expensive and complete
- Cyclic Check - This checks a sections of the file in sequence, wrapping to the start when the end is reached. Cheap but could be exposed to attacks if the storage agent wants to remove part of the file over time.
- Deterministic - Audits shards in a deterministic order known only to the client. This stops the storage agent being able to predict the next shard.

To stop the client pregenerating hashes the shard or shard chunk is seeded before hashing.

filecoin.io [2014] is another cryptocurrency operated file storage network. Transactions are stored in a ledger to assist with trust. At any point a client can issues a challenge to the server to clients who must the calculate the corresponding proof. These challenges are then

confirmed by another node on the network who takes the file and recomputes the challenge itself.

Within Membrane we can build challenge issues into the SLA agreed upon by both users. If a client has no trust of another user, then they value their data less. The initial cost of building a relationship with a user stops a storage node defecting by deleting their stored shard early. It is important to also run file retrieval spot checks, to simulate a situation in which a host loses all of their files.

0.10 Conclusion

In this literature review we discussed the key important areas of creating a distributed storage system. We first explored the history of the problem starting a basic manual copying of files to another host, looking a improvements of that such as Rsync, and a comprehensive solution like Git that requires paid hosting and manual setup. We saw how to exchange file and swarm information in distributed systems like bitcoin and studied file storage systems like Resilio and Storj that achieved distributed file storage.

Then we explored different challenges in creating the system, namely Peer Admission, Data Allocation, Communication, the challenges of connecting to other peers, lessons learnt from distributed files systems and the challenges of Authentication and Proving ownership of a file.

Using the lessons from exploring other systems we can implement our own distributed file storage system and improve on techniques that previous systems have used.

Bibliography

Antonelli, G. A. [2008], 'Non-monotonic logic', *Stanford Encyclopedia of Philosophy* .

Balke, T. and Eymann, T. [2009], 'Using institutions to bridge the trust-gap in utility computing markets–an extended "trust-game"'.

Beer, M., D'inverno, M., Luck, M., Jennings, N., Preist, C. and Schroeder, M. [1999], 'Negotiation in multi-agent systems', *The Knowledge Engineering Review* **14**(03), 285–289.

Boucadair, M., Penno, R. and Wing, D. [2013], 'Universal plug and play (upnp) internet gateway device-port control protocol interworking function (igd-pcp iwf)'.

Bound, J. and Rekhter, Y. [1997], 'Dynamic updates in the domain name system (dns update)'.

Cohen, B. [2008], 'The bittorrent protocol specification'.

Deering, S. E. [1998], 'Internet protocol, version 6 (ipv6) specification'.

Dierks, T. [2008], 'The transport layer security (tls) protocol version 1.2'.

Faratin, P. [2000], Automated service negotiation between autonomous computational agents, PhD thesis, University of London.

Farina, J., Scanlon, M. and Kechadi, M.-T. [2014], 'Bittorrent sync: First impressions and digital forensic implications', *Digital Investigation* **11**, S77–S86.

Fatima, S. S., Wooldridge, M. and Jennings, N. R. [2002], Multi-issue negotiation under time constraints, *in* 'Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1', ACM, pp. 143–150.

filecoin.io [2014], 'Filecoin: A cryptocurrency operated file storage network', *URL http://filecoin.io/filecoin.pdf* .

Finin, T., Weber, J. et al. [1992], 'Specification of the kqml agent-communication language'.

Fipa, A. [2002], 'Fipa acl message structure specification', *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)* .

Ford, B., Srisuresh, P. and Kegel, D. [2005], Peer-to-peer communication across network address translators., *in* 'USENIX Annual Technical Conference, General Track', pp. 179–192.

Ghemawat, S., Gobioff, H. and Leung, S.-T. [2003], 'The google file system', *SIGOPS Oper. Syst. Rev.* **37**(5), 29–43.
URL: *http://doi.acm.org/10.1145/1165389.945450*

Group, I. . S. W. et al. [n.d.], 'Ieee standard for local and metropolitan area networks: media access control (mac) bridges', *IEEE Std* **802**.

Hericourt, O. and Le Pennec, J.-F. [2001], 'Method and system for using with confidence certificates issued from certificate authorities'. US Patent App. 10/007,750.

Herrero, P., Bosque, J. L. and Pérez, M. S. [2007], An agents-based cooperative awareness model to cover load balancing delivery in grid environments, *in* 'OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"', Springer, pp. 64–74.

Holdrege, M. and Srisuresh, P. [2001], 'Rfc 3027', *Protocol Complications with the IP Network Address Translator* .

Hunt, J. W. and MacIlroy, M. [1976], *An algorithm for differential file comparison*, Citeseer.

Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Wooldridge, M. J. and Sierra, C. [2001], 'Automated negotiation: prospects, methods and challenges', *Group Decision and Negotiation* **10**(2), 199–215.

Jennings, N. R., Parsons, S., Noriega, P. and Sierra, C. [1998], On argumentation-based negotiation, *in* 'Proceedings of the International Workshop on Multi-Agent Systems', pp. 1–7.

Jones, B. [2015], 'Bittorrent's dht turns 10 years old', *URL https://torrentfreak.com/bittorrents-dht-turns-10-years-old-150607/* .

Jøsang, A. and Golbeck, J. [2009], Challenges for robust trust and reputation systems, *in* 'Proceedings of the 5th International Workshop on Security and Trust Management (SMT 2009), Saint Malo, France'.

Keller, A. and Ludwig, H. [2002], Defining and monitoring service-level agreements for dynamic e-business., *in* 'Lisa', Vol. 2, pp. 189–204.

Klingberg, T. and Manfredi, R. [2002], 'The gnutella protocol specification v0. 6', *Technical specification of the Protocol* .

Kraus, S. [2001], *Strategic negotiation in multiagent environments*, MIT press.

Kraus, S., Sycara, K. and Evenchik, A. [1998], 'Reaching agreements through argumentation: a logical model and implementation', *Artificial Intelligence* **104**(1-2), 1–69.

Labrou, Y., Finin, T. and Peng, Y. [1999], 'Agent communication languages: The current landscape', *IEEE Intelligent Systems and Their Applications* **14**(2), 45–52.

Langford, J. [2001], 'Multiround rsync'.

Mayo, M. A., Neemann, T., Pearson, H., Sekhar, C. C. and Toraason, D. [2008], 'Security session authentication system and method'. US Patent 7,356,694.

McDermott, D. and Doyle, J. [1980], 'Non-monotonic logic i', *Artificial intelligence* **13**(1-2), 41–72.

McKusick, K. and Quinlan, S. [2010], 'Gfs: evolution on fast-forward', *Communications of the ACM* **53**(3), 42–49.

Mushtaque, M. A., Dhiman, H., Hussain, S. and Maheshwari, S. [2014], 'Evaluation of des, tdes, aes, blowfish and two fish encryption algorithm: based on space complexity', *International Journal of Engineering Research & Technology (IJERT)* **3**(4).

Muthitacharoen, A., Chen, B. and Mazieres, D. [2001], A low-bandwidth network file system, *in* 'ACM SIGOPS Operating Systems Review', Vol. 35, ACM, pp. 174–187.

Neglia, G., Reina, G., Zhang, H., Towsley, D., Venkataramani, A. and Danaher, J. [2007], Availability in bittorrent systems, *in* 'IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications', IEEE, pp. 2216–2224.

Noy, N. F., McGuinness, D. L. et al. [2001], 'Ontology development 101: A guide to creating your first ontology'.

Osborne, M. J. and Rubinstein, A. [1994], *A course in game theory*, MIT press.

Paletta, M. and Herrero, P. [2009], A mas-based negotiation mechanism to deal with service collaboration in cloud computing, *in* 'Intelligent Networking and Collaborative Systems, 2009. INCOS'09. International Conference on', IEEE, pp. 147–153.

Pande Joshi, K. and Finin, T. [2012], 'Ontology for cloud services sla (service level agreement)', *URL http://ebiquity.umbc.edu/resource/html/id/344/Ontology-for-Cloud-Services-SLA-Service-Level-Agreement-* .

Pinyol, I. and Sabater-Mir, J. [2013], 'Computational trust and reputation models for open multi-agent systems: a review', *Artificial Intelligence Review* **40**(1), 1–25.

Plummer, D. [1982], 'Ethernet address resolution protocol: Or converting network protocol addresses to 48. bit ethernet address for transmission on ethernet hardware'.

Porter, M. A. [2012], 'Small-world network', *Scholarpedia* **7**(2), 1739.

Postel, J. et al. [1981], 'Rfc 792: Internet control message protocol', *InterNet Network Working Group* .

purl.org [2001], 'Datastore schema', *URL https://www.w3.org/2001/05/rdf-ds/datastore-schema* .

Qiu, D. and Srikant, R. [2004], Modeling and performance analysis of bittorrent-like peer-to-peer networks, *in* 'ACM SIGCOMM computer communication review', Vol. 34, ACM, pp. 367–378.

Rahwan, I. [2005], *Interest-based negotiation in multi-agent systems*, Citeseer.

Rasch, D. and Burns, R. C. [2003], In-place rsync: File synchronization for mobile and wireless devices., *in* 'USENIX Annual Technical Conference, FREENIX Track', Vol. 100.

Rizvi, S., Hussain, S. Z. and Wadhwa, N. [2011], Performance analysis of aes and twofish encryption schemes, *in* 'Communication Systems and Network Technologies (CSNT), 2011 International Conference on', IEEE, pp. 76–79.

Rosenberg, J., Mahy, R. and Huitema, C. [2005], Traversal using relay nat (turn), Technical report, October 2003. Internet-Draft (Work in Progress).

Rosenschein, J. S. and Zlotkin, G. [1994], *Rules of encounter: designing conventions for automated negotiation among computers*, MIT press.

Ross, R. B., Thakur, R. et al. [2000], Pvfs: A parallel file system for linux clusters, *in* 'Proceedings of the 4th annual Linux showcase and conference', pp. 391–430.

Sabater, J. and Sierra, C. [2001], Regret: reputation in gregarious societies, *in* 'Proceedings of the fifth international conference on Autonomous agents', ACM, pp. 194–195.

Sandholm, T. [2002], 'Algorithm for optimal winner determination in combinatorial auctions', *Artificial intelligence* **135**(1-2), 1–54.

Saxena, N., Tsudik, G. and Yi, J. H. [2003], Admission control in peer-to-peer: design and performance evaluation, *in* 'Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks', ACM, pp. 104–113.

Schneier, B. [2007], 'Bruce almighty: Schneier preaches security to linux faithful', *URL https://www.computerworld.com.au/article/46254/bruce_almighty_schneier_preaches_security_linux_faithful/* .

Sharma, S., Gopalan, K., Nanda, S. and Chiueh, T.-c. [2004], Viking: A multi-spanning-tree ethernet architecture for metropolitan area and cluster networks, *in* 'INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies', Vol. 4, IEEE, pp. 2283–2294.

Shvachko, K., Kuang, H., Radia, S. and Chansler, R. [2010], The hadoop distributed file system, *in* '2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)', pp. 1–10.

Slade, C. [2002], 'Reasons to buy: The logic of advertisements', *Argumentation* **16**(2), 157–178.

Srisuresh, P. and Holdrege, M. [1999], 'Ip network address translator (nat) terminology and considerations'.

Stephen, Z. H., Blackburn, S. M., Kirby, L. and Zigman, J. [2000], Platypus: Design and implementation of a flexible high performance object store., *in* 'In Proceedings of the Ninth International Workshop on Persistent Object Systems', Citeseer.

Sugumaran, V. and Storey, V. C. [2002], 'Ontologies for conceptual modeling: their creation, use, and management', *Data & knowledge engineering* **42**(3), 251–271.

Thakur, J. and Kumar, N. [2011], 'Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis', *International journal of emerging technology and advanced engineering* **1**(2), 6–12.

Torvalds, L. and Hamano, J. [2010], 'Git: Fast version control system', *URL http://git-scm.com* .

Tridgell, A., Mackerras, P. et al. [1996], 'The rsync algorithm'.

Tseng, Y.-C., Ni, S.-Y., Chen, Y.-S. and Sheu, J.-P. [2002], 'The broadcast storm problem in a mobile ad hoc network', *Wirel. Netw.* **8**(2/3), 153–167.
URL: *http://dx.doi.org/10.1023/A:1013763825347*

Varian, H. R. [1995], Economic mechanism design for computerized agents., *in* 'USENIX workshop on Electronic Commerce', New York, NY, pp. 13–21.

Vuze [2010], 'Peer exchange', *URL http://wiki.vuze.com/w/Peer_Exchange* .

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. and Maltzahn, C. [2006], Ceph: A scalable, high-performance distributed file system, *in* 'Proceedings of the 7th symposium on Operating systems design and implementation', USENIX Association, pp. 307–320.

Wilkinson, S., Boshevski, T., Brandoff, J. and Buterin, V. [2014], 'Storj a peer-to-peer cloud storage network'.

Wing, D., Matthews, P., Rosenberg, J. and Mahy, R. [2010], 'Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)'.

Wooldridge, M. [2009], *An introduction to multiagent systems*, John Wiley & Sons.

Yan, J., Kowalczyk, R., Lin, J., Chhetri, M. B., Goh, S. K. and Zhang, J. [2007], 'Autonomous service level agreement negotiation for service composition provision', *Future Generation Computer Systems* **23**(6), 748–759.