Node Selection In Distributed Overlays

Lawrence Latif

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Doctor of Philosophy

of

University College London

Department of Electronic & Electrical Engineering

University College London

September 2012

Examination Committee:

Dr Nicholas Race, Lancaster University

Dr Yiannis Andreopoulos, University College London

Supervised by: Dr Miguel Rio, University College London

I, Lawrence Latif, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Department of Electronic & Electrical Engineering

University College London

Abstract

With the proliferation of latency aware services such as live video streaming, Internet-based financial trading and the popularity of distributed overlays such as BitTorrent there is a growing need for latency-aware distributed overlays. To make such overlays viable, efficient resource discovery services are needed. Anycast is a routing protocol that sends packets to nodes that are a member of a particular group, with the work presented the Anycast protocol in the distributed overlay domain.

Structured and unstructured distributed networks have become a popular way to disseminate data without the need for a fixed infrastructure, however there is a need to provide quality of service (QoS). To meet the demands of applications, an overlay needs to maintain accurate Anycast group membership data, locality information and have minimal protocol overhead.

Three protocols are proposed to meet these goals. The Distributed Overlay Anycast Table (DOAT) brings the notion of locality to a structured overlay, while introducing Bloom filters as an efficient data structure to present an overlay that can accurately return a node that is participating in a particular group.

The Gossip Overlay Anycast Table (GOAT) is a scalable location-aware unstructured overlay that can provide the probabilistic Anycast routing. Through the use of an efficient discovery protocol and the use of Bloom filters, GOAT is able to provide the advantages of a structured overlay, while mitigating the performance issues typically found in unstructured overlays.

The N-casting overlay is an unstructured overlay with the ability to send queries to multiple members of an group, uses a hierarchical decomposition of the Internet and an elegant data structure that offers predictable compression of overlay membership. N-casting shows that unstructured overlays can be scalable and sustain high performance in environments that exhibit realistic membership churn.

DOAT, GOAT and N-casting present viable services that implemented at the application layer provide location aware node discovery in QoS-enabled applications.

Acknowledgements

This work has been entirely possible thanks to the overwhelming support of my colleagues; who became friends, my friends and of course my family. All of you have given me the opportunity to produce this body of work which I hope does justice to your expectations.

In particular I would like to thank my supervisor, Dr Miguel Rio his support, ideas and critique which have made my work wholly better. A side note of extra special thanks goes to Dr Rio as he provided the funding for a water cooler in our lab. This, according to research on hydration [1] increases productivity within the workplace. It is unknown how many extra publications this single watercooler has made possible but its effects should not be underestimated.

Most of the work presented before you would not have been possible without my patient and extremely talented colleagues, Eleni Mykoniati, Raul Landa, Richard Clegg and David Griffin. They provided me with feedback, guidance and advice on how to produce the work that is presented in this thesis while making the laboratory a wonderful place to work.

The contents of this thesis are completely my own work, which has not in its entirety, been presented or published elsewhere. It is being presented as part of my Doctoral Thesis at University College London, 2012.

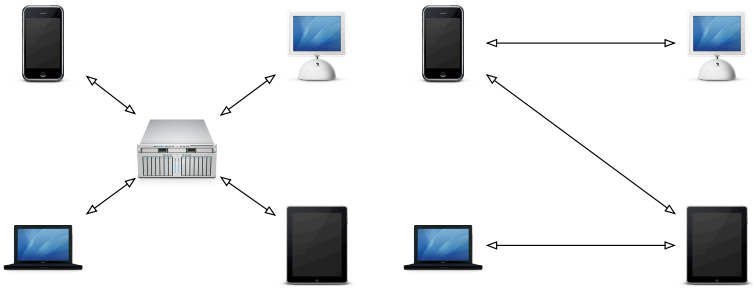
Contents

Chapter 1

Introduction

Mechanisms for searching content over the Internet have traditionally relied on the client-server model with a central repository maintaining an index of content that is available on the network. In the past decade a move away from this model has occurred with peer-to-peer (P2P) networks becoming increasingly popular.

In P2P networks, data is spread over the nodes in the network with no central repository. Advantages of this model include scalability, resilience and lower cost [2]. It is therefore no surprise that P2P networks have garnered a lot of interest in business and research. Throughout this document the terms P2P and distributed network will be used interchangeably.



(a) Client-server topology (b) P2P topology

Figure 1.1: Network topologies of client server and P2P networks

The most visible difference between client-server and P2P networks is presented in Figure 1.1. While the server in Figure 1.1(a) provides the central store of data in the network, in a P2P network, each device can either have part of or all of the data, sharing it to other nodes. In effect all nodes have the opportunity to be a server to other nodes. The aim is to create a network that is dispersed, not centralising a service to a single resource.

By decentralising resources, P2P networks aim to reduce the probability of single point failures

causing disruption in data dissemination. It also tries to lower the requirements of servers both in terms of hardware and connectivity. Instead of a single server having to service 100 percent of requests, a P2P node would only service a small percentage of that, relaxing the requirements of hardware such as CPU and RAM, and network performance such as bandwidth.

Aside from sharing of resources, it is geographic diversity that results in greater resilience. The effect of regional network outages are mitigated by the fact that part or all of the same data is available from other locations. Nodes on a P2P network are, for the most part, treated as equals meaning there is no differentiation based on location, hardware or network connectivity. Later P2P networks introduce bias towards certain node characteristics, however it is geographic diversity, differences in asynchronous bandwidth and uptime that becomes a major challenge in designing latency sensitive distributed overlays, as discussed later in this chapter.

As with traditional client-server networks, the ability to find information is vital in the efficient use of the network, a view presented by Ratnasamy et al. [3]. In the past decade much has been published on search mechanisms for distributed networks. Two main areas of interest have been distributed hash tables [3], [4], [5], [6], [7], [8], [9] and gossip-style unstructured overlays [10], [11], [12], [13]. Both have been deployed in real world P2P networks such as BitTorrent [14], [15] and Gnutella [16] respectively.

The term '*overlay*' is used to describe an application layer network of nodes that is created from heuristics present in the application that runs on the node. This differs from the underlying network where links between routers are defined by the Internet service provider. An overlay is created by nodes running applications that have a defined joining protocol that may not neccessarily follow routing policies present in the underlying network, for instance it may specifically seek to directly peer, at the application layer, to a node that is located in a different country.

Overlay networks gained popularity as protocols that suggested changes to the underlying network failed to gain popularity, such as Multicast and Anycast. Overlay networks that build links between nodes at the application layer do not require changes to the infrastructure and therefore have a lower cost of deployment.

Current research has focused on the ability for such overlays to withstand node failures, known

as churn, and recover from them as fast as possible [17], [18].

Work on node selection based on certain metrics such as latency and hop count [19], [20], [21], [22] have tried to reduce the lookup delay. Lookup times are critical in a system which is going to support real-time services such as streaming audio or video. The focus of this research is to look at the ability of finding nodes which are the closest to the source.

1.1 Problem area

The aim of the work presented is to propose a system where nodes in a distributed system can be found in a resource efficient manner using the Anycast paradigm.

Anycast is a routing protocol that allows for messages to be delivered to nodes that are part of a group. A group is defined as a collection of nodes that serve similar interests, for example a particular television channel or CDN nodes used to distribute a website. While Anycast can return a single group member based on a heuristic, the N-casting overlay presents a protocol that can return multiple group members.

The goal of this work is to make the lookup of relevant nodes participating in an overlay efficient and return high quality results. Quality can be judged by a number of metrics, which are described in the contribution chapters, however the general goal is to provide the requestor node with the location of another node that is as physically close to it as possible. This is a non-trivial problem as distributed networks typically work on the principle that all nodes, regardless of their latency are treated as equal and nodes are selected irrespective of location, node uptime, connection type or quality.

A system which aims to profile nodes based on their latency has to consider the factors which affect latency and what effect that will have on other factors within the system such as availability and resilience. The goal is to have geographic awareness but maintain the resilience that is obtained from geographic diversity.

One of the main advantages of distributed networks is their ability to share resource usage among nodes in the network, alleviating the bottleneck sometimes witnessed in traditional client-server networks.

Routing in distributed networks requires the transfer of routing tables to all nodes in the network. Work has been presented [23], [24] on measuring and creating lightweight node state management systems. An efficient method which minimises the number of messages being sent between nodes in the network is vital in ensuring a system which scales well with the number of nodes in the network.

Distributed networks are characterised by their relative low node lifetimes [25] and [26]. Whereas in the client-server paradigm, the server is expected to have high *uptime* and availability, in distributed networks the server is typically a client (though in some cases distributed networks may incorporate highly available nodes) and therefore cannot be expected to have high availability.

Due to the nodes in distributed overlays comprising of servers, consumer computers and other devices, there is a large spread of connection bandwidths as observed in [27]. It gives rise to the fact that many users are connecting to such networks with connections which typically lack any form of service level agreements guaranteeing their availability or any quality of service aspects such as traffic prioritising. It is for this reason that guaranteed response from nodes cannot be assumed in such a network.

Anycast has the ability to place nodes into related groups, making it particularly suited to applications such as IPTV. The work presented here aims to locate one or more Anycast group members that are physically close to the requesting node, in order to facilitate communication between the two or more nodes at the lowest latency possible.

It is these challenges posed by distributed networks which mean that maintaining knowledge of node availability is important. A balance between correctness of information and maintenance cost has to be met allowing for a system to scale for many tens of thousands of nodes.

1.2 Motivation for low response times

Finding nodes in distributed networks is complicated by the constantly moving membership base of the overlay and the maintenance of subscriber lists, whether they be centrally stored or completely decentralised. There is also typically no discrimination between nodes which are topologically close to a certain node or one that is far away. This homogeneity provides an intrinsic resilience to distributed networks, however it can cause catastrophic results when

trying to find the shortest path or the closest node.

Distributed networks have been deployed as a method of content distribution for static content. That is to say, content which is distributed once it has been produced in its entirety. Harnessing the advantages of distributed networks in order to present *"near live"* content produces a whole new set of challenges [28]. Analysis such as Spoto et al. [29] of modern distributed video streaming systems such as PPLive has shown a spectrum of node bandwidths providing further challenges in ensuring selecting a node that is not only the geographically closest but has enough upload bandwidth to support the requestor node.

The just-in-time nature of this content delivery places higher demands on an overlay requiring it to provide a quality of service. Nodes need to be found in time so that the next frame of video or the next note of a song must be played in the order in which they were broadcast. This differs greatly from the jumbled up contact policies which can be employed in non-live media, quite simply the order doesn't matter until the complete broadcast is received and played back.

Node selection plays a vital role in delivering a quality of service aware distributed network. If a node can eliminate contacting nodes which have higher latency than others then queries can be resolved faster. The latency between a user requesting the data (in the case of television pressing the channel button on the remote control) to the user being presented the data is an important user experience metric. In the area of IPTV over distributed networks, the ultimate goal is to enable the user to select different streams with such a low latency that there is no noticeable delay between delivery of audio and video between channels.

Work presented here is based on the premise that geographical distance is related to network latency. Such a hypothesis has been explored previously with experimental data showing there to be a link between latency and geographical distance. Experimental data is presented in Figure 5.4 using two large datasets, while work presented by Hopper [30], Steiner [31], Sun [32] and Agarwal [33], show a strong link between geographical distance and network latency.

Finding low latency nodes in a distributed system brings forward the ability to provide quality of service not just on a data throughput metric but also a response time metric. It allows for users to experience *"on-demand"* data from distributed networks, something that has previously been serviced by client-server systems. However the challenge is not only finding a node with a low

latency but creating a mechanism which adds as little to the overall query delay as possible.

1.3 Obstacles

These can be summarised as follows:

- Maintaining network redundancy while providing geographical profiling of nodes
- Keeping the number of messages being sent by nodes to a minimum
- The ability to withstand churn and short lived node lifetimes
- Reducing the amount of state information held at each node
- Ensuring low start-up time for low membership networks

These obstacles can really be split into two categories; maintaining the redundancy of distributed networks and making the proposed solution scalable as the network size increases.

The importance of considering these factors is not only from an immediate network performance standpoint but the applications in which such a system can be applied. Nodes may be set-top boxes with limited processing and memory resources or handheld devices such as mobile phones or personal digital assistants (PDAs). Therefore, one should consider the efficiency of any system when placed in environments that are resource limited.

Locating group members that are geographically close is important as measurement data from [34] shows that 95% of any two nodes in a Gnutella network are less than seven hops away nevertheless the majority of those nodes are between 100ms to 400ms as reported in Saroiu et al. [25] with similar findings in Ciullo [35]. The same research also shows that a relationship between node latency and bandwidth. It should be a goal for any node selection strategy that the majority of selections occur at the head of that distribution.

A balanced approach to overcoming these obstacles will manage the requirement of low latency with scalability, redundancy, resilience and low maintenance and messaging protocols.

1.4 Thesis contributions

There are three distinct contributions presented here, with the Distributed Overlay Anycast Table (DOAT), Gossip Overlay Anycast Table (GOAT) and N-casting that uses components of structured and unstructured overlays, described in Chapter 2 and extends its using novel

concepts.

DOAT (Chapter 3) introduces the notion of locality to structured overlays and introduces an efficient Bloom filter data structure that allows for large group sizes without imposing scalability limitations.

GOAT (Chapter 4) introduces probabilistic node selection in an Anycast environment, an environment in which deterministic node selection has been prevelant. Experiments show that the performance of GOAT, even with a probabilistic selection mechanism is high with the added ability to handle unstable overlay membership.

N-casting (Chapter 5) is an variable resolution unstructured overlay that allows nodes to send packets to more than one group member. In order to create a variable resolution overlay, N-casting uses hierarchical clustering to decompose node locations and compress node membership information, something that has yet to be applied in the area of Anycasting. Furthermore, N-casting uses an innovative 'floating point' data structure that allows for predictable compression, ensuring scalability.

The three overlays presented in this thesis further the state-of-the-art by presenting application layer Anycast that is distributed and makes use of underlying network characteristics such as latency to create a latency sensitive Anycast service resiliant to churn. Through the use of innovative datastructures such as Bloom filters and a '*floating-point*' data structure, scalability is achieved.

The hierarchical decomposition of the Internet using agglomerative heirachical clustering is a new way of creating a variable resolution Anycast service that allows nodes to have precise group membership information for those groups that are geographically close to it, while maintaining knowledge of geographically distant nodes. This innovative, low cost method allows for scalability, resiliance and high performance distributed Anycast in unstable networks.

1.5 Table of symbols

The symbols presented in Table 1.1 is used throughout this thesis in descriptions and formulae. In cases where a new symbol is introduced and only used once, it will be defined, otherwise the definition of symbols will retain those listed in Table 1.1.

Symbol	Definition
n	node
n_t	number of nodes
t	time
h	number of hops
d	distance
l	latency
g	Anycast group
m	length of address space
e	number of routing table entries
c	cluster
v	tier

Table 1.1: Table of commonly used symbols

1.6 Organisation of thesis

Thesis chapters are structured as follows:

Chapter 2 is a literature review of structured and unstructured overlays. This chapter looks at the structure and routing policies of existing structured and unstructured overlays, and reviews previously published work on locality awareness in distributed overlays and methods used to deal with membership churn. A literature review of the Anycast protocol both at IP and application layer is presented, along with manycasting, a multi-recipient routing protocol that is an extension to traditional application layer Anycast.

Chapter 3 presents the Distributed Anycast Overlay Table, a locality aware structured overlay that uses space-filling curves and Bloom filters to provide a high-performance resource discovery overlay.

Chapter 4 presents the Gossip Overlay Anycast Table, a locality aware unstructured overlay featuring an efficient node discovery protocol, which shows that locality-aware unstructured overlays can be scalable and provide latency sensitive service discovery.

Chapter 5 presents the N-casting overlay, a multi-recipient routing paradigm that allows nodes to send messages to multiple members of an Anycast group. The overlay uses an innovative hierarchical decomposition of the Internet and an efficient floating point data structure to bring locality and scalability for an Internet-scale overlay.

Chapter 6 summarises the contributions made in this thesis and presents future work that can be conducted on the basis of research presented in previous chapters.

Appendix A presents early work done on virtual worlds, an area where the N-casting overlay presented in Chapter 5 could be deployed.

Chapter 2

Distributed overlays

Structured overlays employ a preset heuristic that every node joining the overlay follows to be part of a preset structure. Such an overlay provides predictable results as nodes follow a preset pattern that has little variance, however due to the rigid joining procedure, overlay performance can be negatively impacted when the overlay has volatile membership.

Unstructured overlays are built using probablistic node selection protocols that results in every node that is taking part in the overlay having a different view of the overlay. The notion of a unstructured overlay is to avoid a rigid set of overlay joining rules that may compromise an overlay's functionality in unreliable networks, such as the Internet.

Anycast is routing protocol that delivers packets to nodes that are part of groups. Nodes are grouped by mutual interest, for example web servers that serve news web pages, and packet routing is determined by a preset heuristic. The heuristic used to determine where the packet is routed to can be varied, from IP-layer characteristics such as latency to the application-layer such as server load.

This chapter provides a literature review of structured and unstructured networks and Anycast, looking at existing overlays and the key principles behind the overlays and the Anycast protocol.

2.1 Unstructured overlays

Originally proposed for the replication of databases [36], epidemic algorithms now are commonly referred to as a gossip or flooding protocol and forms the basis of an unstructured overlay. Gossip has been cited as a mechanism for resource location [37], [38], [39], [40] load balancing [41], network management [42], aggregation [43] and search in P2P networks with its use in Gnutella widely documented [44], [45], [46].

Two common implementations of gossip protocols are messages being sent to all nodes in the network given a certain radius and a more directed gossip as defined by nodes kept in each nodes membership table. The gossip radius is defined through a time-to-live (TTL) or another overlay proximity metric and the query message is *'flooded'* to all nodes within a given constriction.

Demers et al. [36] proposed two policies for their epidemic algorithm, anti-entropy and rumour mongering [sic]. Anti-entropy randomly selects another node and through transfer of the whole data set resolves any differences between the two nodes. Work done later shows that this is a very reliable policy, however the message overhead is too great for this policy to be viable in large scale systems. This fact was borne out from tests carried out in the Gnutella overlay [25].

Since anti-entropy selects nodes at random from a small group of other known nodes it is with high probability that all messages sent to nodes within that small group will be delivered successfully, assuming that the knowledge held by nodes in the overlay is accurate. As the number of nodes in the overlay increases, the cost of maintaining an accurate view of the overlay membership increases. This is because nodes engage in exchanging messages to keep an accurate view of the overlay and it is possible that the maintenance cost encroaches any '*quality of service*' requirements that may be required.

As an chorally to anti-entropy, rumour mongering aims to decrease the number of messages being sent through the network by scheduling periodic updates of *'updated'* data with the possibility of bounding the update rate. However, updates could be lost in highly dynamic systems where the number of updates may exceed that set by the updates/time parameter. In this case there may be a period of time when no updates are registered or some updates are ignored in order to create a standard distribution of updates throughout the time period. As with anti-entropy, only the updates are sent.

Depending on the application, Demers claims that a system with a high enough update rate should result in a push-like system for updates being sent to other nodes. It should be noted that if the update rate was set too high it would result in an anti-entropy-*esque* system. A pull system, one where the nodes periodically message other nodes for updates, produces traffic which may not lead to knowledge of updated data.

In an effort to get the advantages of both gossip schemes, Karp et al. [47] propose that typically a system will be use both anti-entropy and rumour mongering methods, with the latter being used more frequently while anti-entropy being used for improved robustness in overlays that have dynamic membership.

2.1.1 Unstructured overlay challenges

A number of challenges are presented when creating and maintaining an unstructured overlay. Unlike more traditional structured overlays, the *'loose'* knowledge of nodes within the overlay has both advantages and disadvantages.

- Cost of routing queries which may not lead to desired destination
- Time taken to reach nodes *'far away'* on the overlay
- Response times to queries can be high due to no notion of locality
- The cost of keeping the overlay membership knowledge accurate

The last point presents the most researched challenge; to stem the number of messages being relayed in a large scale overlay. Nodes become overwhelmed by maintaining the infrastructure resulting in a failure to participate in their own activities such as answer queries. Modifications to initial unstructured overlays have tried to mitigate the message relay problem to varying degrees of success and are discussed further in this chapter.

2.1.2 Gossip types

Uniform gossip is when a node is chosen at random to forward all messages. Because the random choice is from a uniform distribution, it has been published [47] that with high probability all nodes will receive a copy of the message in $\mathcal{O}(\log N)$ steps from its first appearance.

Neighbour flooding is when a node picks a neighbour from within distance *'d'* of itself (typically defined by latency) and forwards all messages to that neighbour. All nodes within *d* will receive the message within $\mathcal{O}(d)$ from its initial appearance. However for nodes outside of *d*, the time is $\mathcal{O}(\sqrt{n_t})$. It could be such that all or n_t within *d* will receive the message.

Neighbour flooding exhibits the favourable characteristics that the overlay size has no affect on the propagation delay of messages. In uniform gossip, nodes close to the message originator may only be notified after the message has traversed far-flung corners of the network, a trait

that would be unacceptable for applications that need locality awareness, in order to exploit underlying network latency.

Spatial gossip [37] aims to guarantee that nodes at d which are spread uniformly throughout a Euclidean space will receive information from the source in $\mathcal{O}(\log 1 + d)$ time steps. Spatial gossip can be seen as taking the desirable fast propagation of uniform gossip with the fact that propagation time depends on the distance from the message originator rather than the size of the network, found in neighbour flooding.

2.1.3 Overlay sampling

Since unstructured overlays do not offer nodes a central library of overlay node membership data, a procedure must be present to probe the overlay in order to acquire membership information. In order to do this, there are a number of ways to '*sample the graph*'.

To illustrate the importance of a sampling algorithm in unstructured overlays, it is best to look at the difference between itself and a structured overlay. In structured overlays, discussed in Chapter 2.2, a deterministic overlay sampling heuristic is applied. This rigid selection mechanism provides a way of acquiring neighbours in a preset area of the overlay, however in a dynamic overlay, it may provide an inaccurate view, with certain nodes being repeatedly sampled. This may result in a two-tier system, whereby those nodes frequently sampled are favoured or worse, burden them with both routing and message updates, diminishing the performance of the overlay.

A poor sampling heuristic also lacks load balancing properties, meaning nodes could be selected a disproportionate number of times, utilising a similarly disproportionate amount of resources. Not only does this serve as a motivation for utilising unstructured networks but the design of an unbiased sampling algorithm that does not prescribe points at which joining nodes must look for overlay neighbours.

2.1.4 Random Walks

The most commonly used sampling technique is a random walk [48], [49], [50] with the Metropolis-Hastings algorithm [51] cited as a candidate algorithm. The challenge is not applying the random walk technique per-se, but estimation of the minimum length of the walk and making sure it provides an unbiased sample of the overlay where membership is unstable.

To measure the performance of a random walk algorithm a number of metrics can be employed, examples of this include; the error between the sampled cumulative distribution function and the expected cumulative distribution function, time taken - in hops - for the error to reach a particular point and the probability of selection.

As random walks can be modelled to produce sampling of a particular distribution, it is important to see how far off the designed '*optimal*', the returned sample is. The shorter the time taken for a random walk to attain its optimal (lowest) error or standard deviation in selection probability signifies how efficient the algorithm is. A lower number of hops results in fewer messages and a shorter time for the walk to occur.

The probability of selection should have low standard deviation among nodes. The aim is to show that the probability doesn't sway wildly providing unpredictable results. A high standard deviation here would suggest a bias in the random walk algorithm.

Stutzbach et al. [50] claim there are three "*fundamental properties*" which interact with a random walks which are; node degree, session length and query latency.

A node's degree is the number of connections it maintains with other nodes on the overlay. A high degree node is more likely to be visited as it appears on a greater number of other nodes' neighbour lists.

Node session time come into effect due to the nature of sampling the overlay, where nodes are picked only once. This means that if the node property changes, that change would not be taken into account. With P2P overlays more conducive to dynamic membership, nodes with low session times may result in a long term or aged view of an overlay, an overlay where certain nodes may not be present. Nodes with high session times may provide a more stable view of the overlay.

Figures presented by Stutzbach et al. [50] show that the longer an overlay is measured, the opportunity of an incorrect measurement increases as the percentage of transient nodes over the total nodes increases. It is suggested that any monitoring system must take into account the notion that the same node may re-appear on the overlay and therefore shouldn't be counted as a unique node, saying that it "*must be possible to sample from the same peer more than once,*

at different points in time".

The time it takes to query a node in an environment where overlay membership is static is merely measured as the length of the walk. As different nodes can take a different amount of time to respond it is important that the standard deviation of response times between nodes does not cause a bias in sampling. According to Stutzbach, sampling bias can only occur if the desired property has a correlation with the fundamental properties mentioned previously. Stutzbach shows sampling is unbiased for any property it is sufficient to show that it is unbiased for the fundamental properties that interact with the sampling technique.

An unbiased sample is one that does not favour a particular area of the overlay, that is to say, the sampling process produces a set of results which are a representative sample of the overlay's membership. Stutzbach observe that in certain circumstances, it is favourable to introduce bias into random walks in order to overcome initial bias in the walk. The aim is to introduce a controlled bias, one that can be designed and tweaked to particular problems with random walk algorithms, such as Metropolis-Hastings. The authors introduce such a bias with positive results, discussed later in this chapter.

The maximum-degree algorithm has been often used as a benchmark. In the distributed implementation of the algorithm, nodes generate a transition matrix locally by knowing the maximum degree of every node in the overlay. The knowledge of all nodes' maximum degree in a network is a non-trivial task in a distributed network, especially one that has a dynamic membership. Another challenge is the ability for the overlay's membership to alter during the sampling process, meaning that nodes which are returned may no longer exist or visited nodes leave the overlay.

Stutzbach [50] refers to churn as the temporal dynamics of the system. Churn is a problem in any sampling algorithm as it takes time to accrue the sample during which the nodes sampled may have left the overlay. The second cause of bias refers to the increased probability of nodes which are well connected being sampled during a random walk.

While Stutzbach points to bias towards high degree nodes, any node degree bias would result in performance issues. A bias towards high degree nodes would allow for nodes which have low connectivity in the overlay to remain such, whereas the '*rich club*' would not only get

preferential service but also be burdened with a greater percentage of overlay maintenance.

In order to avoid bias towards high-degree nodes, which inherently would have a greater chance of being visited due to the higher number of connections they have, the author put forward a method termed '*Metropolized Random Walk*' (MRW) [50]. MRW picks the next node to visit after producing a stationary distribution which creates the same probability for all nodes to be at the end of the walk, therefore eliminating a bias towards visiting high degree nodes. The algorithm introduces two further steps to the standard Metropolis-Hastings algorithm which occur after the queried node returns a list of neighbours to determine its degree.

After the queried node *n* returns its list of neighbours allowing the querying node *n'* to determine the node degree, the node generates a random value *p* uniformly between 0 and 1. If $p \leq \frac{degree(n)}{degree(n')}$ then *n to n'* is the next step of the walk. If not, the next step is to remain at *n*.

Sampling from a '*static*' overlay, that is one where churn is not present, was evaluated by Awan et al. [52] utilising a number of different random walk algorithms including the *maximum degree* and *Metropolis-Hastings* algorithms. It should be noted that the simplest random walk is merely a sequence of nodes visited, the order of which is determined by a probability matrix. It can also be shown that the probability of the following node is not affected by any previous node that has been visited. This property is known as '*memoryless*' allowing a random walk to be modelled as a Markov chain.

The memoryless feature of a random walk algorithm is important as in some cases a node may not be able to store the previous steps in a random walk due to physical node characteristics. The implementation of a random walk by Craswell and Szummer [53] "*forgets*" previous nodes that have been visited due to limited memory on the walking node. Therefore the walk inherently has to be memoryless, rather than relying on a store of previous nodes to aid in taking the next step of the walk.

Determining the required length of a random walk is vital. Too short and it may not visit enough of the overlay to create an accurate representation, too long and it'll waste resources and time. Given that P2P overlays are not of a fixed size, there has to exist a method in which overlay size can be estimated, prior to the random walk.

Awan's [52] propose a random weight algorithm (RWD), a decentralised algorithm in which nodes set their transition probabilities based on the maximum connection properties of the overlay. The maximum connection property is one that is common to a number of overlays and the algorithm's performance does not deteriorate if the maximum number of connections is incorrectly estimated. However during the initialisation phase, the authors claim RWD "*results in high self-transition probabilities for low degree nodes.*"

After initialisation, the node distributes its self-transition probability randomly and symmetrically to other nodes on the overlay, all of which run the algorithm. In the RWD algorithm a *n* receives an *INCREASE* message from *n'*, which then reduces the self-transition probability of *n* by the weight of *n* held at *n'*. It also increases its transition probability of *n* by the weight held at node *n'*. After this an acknowledgement message is sent to node *n'* when it does the same to make the probability on both nodes equal to one. If the node does not receive the acknowledgement message or receives a *NACK* it then removes node *n* from its set of known nodes and does not change its weight.

Awan et al. compared RWD against maximum degree and Metropolis-Hastings algorithms noting that the maximum degree algorithm requires knowledge of the maximum degree of every node in the overlay. Given this can change during the lifetime of the overlay, maintenance of this information can be costly, something that is borne out in results presented which show RWD and Metropolis-Hastings algorithms had significantly lower messages. The number of messages is an important performance metric in unstructured overlay performance, as stated in Section 2.1.1, making the maximum degree algorithm particularly unfavourable.

Of greater importance than ruling out the maximum degree algorithms are the results presented by Awan showing that self-transitioning probability is high on both maximum degree and Metropolis-Hastings algorithms. Self-transitioning in random walks is the occurrence of revisiting a node in one cycle, resulting in a loop. A high self-transitioning probability infers a bias towards low degree nodes, as it is more likely that where there are fewer options, the chance for a self-transition is higher. A basis such as this would, as mentioned previously lead to a preference for nodes that had just arrived on the overlay.

Results from Stutzbach et al. [50] give some insight into this occurrence. They found that when a node starts on a walk which has a single neighbour, it causes a "*large fraction*" of walks to

end on the low degree node. Due to the probability of a transition from *n* to *n'* is $\frac{degree(n)}{degree(n')}$ this causes the selection of the same peer to occur many times, causing bias in the results. It should be noted that if a node visits a low degree node later in its walk, it is simply a re-weighting to account for a random walk's tendency to pick high degree nodes.

Losavz [54] stated that the length of walk would be of the order $\mathcal{O}(logn_t)$ however the bound is dependant on the second largest eigenvalue modulus. It is possible to estimate network size from local information using techniques presented by Horowitz and Malkhi [55]. As the experiments carried out by Awan et al. were on overlays with static membership it would be fair to assume that the length of a random walk, once worked out by using techniques presented in [55], would not change during the lifetime of the overlay. This would not be a logical assumption to make on a P2P overlay deployed on the Internet, exposed to churn.

While a longer random walk provides more samples, and one may intuitively assume that such a walk would render a more accurate snapshot of the overlay, walking an overlay has costs associated with it. Stutzbach [56] shows that total walk time rises close to linearly with walk length. In one experiment conducted by Stutzbach et al. [50], the longer execution times led to a decrease in accuracy of the sample with other experiments showing that the sampling error decreased with walk lengths of up to 30 hops, after which sampling error and standard deviation increased.

Results by Awan et al. and Stutzbach et al. show the maximum depth and Metropolis-Hastings random walk algorithms require longer random walks in order to achieve uniformity in the probability of selection. The goal is to have a probability that is equal among all nodes, therefore lowering the standard deviation and thus increasing the performance of the random walk. Having a shorter random walk is favourable as it lowers the number of messages being sent on the overlay and the time taken for the walk to occur.

In considering churn, Stutzbach et al. [50] propose Metropolized Random Walk with Backtracking (MRWB). The method applies a heuristic to deal with visiting a node that has disconnected from the overlay. The original Metropolis-Hastings algorithm, being designed for static overlays does not place any heuristics for when this occurs. To that end, Stutzbach et al. propose that the node maintain a stack of visited nodes. Every new node chosen to query is pushed onto the stack with the node popped if it fails to respond. A new node is chosen from

the stack, which now does not contain the recently discovered unresponsive node. If every node in the stack fails to respond, the walking node re-queries the last responsive node for a new set of nodes to fill its stack. The procedure is repeated until either a responsive node is reached or the stack underflows. If an underflow occurs, the random walk is classed as a failure.

Results from Stutzbach displaying sampling error in dynamic overlays show that when node session times are lower than 90 seconds, there is a significant increase in error. The authors test the error when forcing a 10,000 hop walk which takes one hour to complete. The walk remained unbiased and the error was similar to that found in experiments with average session times prior to the large error increase. Stutzbach use these experiments to show MRWB is capable of providing accurate, unbiased sampling in dynamic overlays.

2.1.5 Locality aware unstructured overlays

Early optimisations of unstructured overlays focused on reducing node resource usage, and taking into account the underlying network characteristics was first tackled through the notion of *'directional gossip'*. The motivation for the introduction of locality awareness in unstructured overlays stems from large-scale overlays where the cost of complete overlay knowledge is too great.

The idea of using random walks to locate a node was improved upon by Adamic [57], with the heuristic that high-degree nodes should be favoured. The rationale behind this heuristic is high degree nodes are likely to *'know'* more information and therefore a query is likely to be answered in fewer hops. While this bias towards knowledgeable nodes does reduce the query path for the majority of messages, the problem is that by favouring such nodes, these become overloaded unable to service the volume of queries being directed to them. It can also lead to isolation of processes[1] should a major gossip '*hub*' fail.

Ning et al [58] proposed the use of network coordinates in order to have location aware gossip. Through the use of Vivaldi [59], nodes are placed on the overlay with nodes differentiating others through the use of a *"clique radius"*. The clique radius is a continually adjusting area on the overlay, nodes in which are deemed to be '*nearby*' with those outside the clique are *'long range'*. Further exchanges with nodes within its clique results in a reducing of clique size with the aim accurately representing the underlying topology. The number of long range contacts grow as clusters become smaller, reducing the duplicity of a node's routing table. Results show

[1] Processes are the nodes which handle queries in the overlay

that the number of cliques discovered against the number of nodes traversed followed a Pareto curve. The authors suggest a traversal that visits 35 nodes produces a complete view of the system with a near proportional growth in number of cliques discovered against the number of nodes traversed to that point.

Since 1999, the Internet Assigned Numbers Authority (IANA) has assigned IP ranges to geographical regions, while previously it was a free-for-all. Since IANA's move towards a more structured allocation of IP address ranges, Zhao and Lu [60] propose the use of IP address matching as a method of grouping nodes on Gnutella into groups that are geographically close to each other. Motivation for such a system can be provided by figures cited by Liu et al. [61] where the authors state that more than 40 percent of all Gnutella nodes are located within the top 10 ASes. Being within an AS, nodes are generally regarded as being geographically close to each other, however as ASes could span large distances, it isn't a perfect solution.

Zhao and Lu's Adjacency Measurement (AM) system works by matching nodes' IP addresses in left to right segments, where each segment is an octet. For each matching segment, the algorithm increments the AM counter for the nodes. The AM algorithm stops when it finds a segment that doesn't match, leaving each node pairing with an AM. If a node pair has an AM of 0, then it is checked against a list of IP assignment tables provided by a regional Internet registry (RIR) such as RIPE, ARIN or APNIC as for instance, a node with the IP address of 84.x.x.x and 202.x.x.x are two IP blocks that are assigned by RIPE to UK networks but would have an AM of zero.

In 500 node experiments conducted by Zhao and Lu showed the AM algorithm managed to create groups that represented regions controlled by RIRs such as ARIN, APNIC and RIPE. However the authors conceded that due to pre-1999 IP addresses being allocated on a random first-come-first-serve basis, inaccuracies exist. As the AM system does not increase the number of messages on the overlay, the authors say that it provides an improvement in locality awareness in unstructured overlays such as Gnutella, without any increase in cost for the overlay itself. The AM system is part of the inspiration for the locality awareness in N-casting Overlay, proposed in Chapter 5.

Wong et al. [62] propose Meridian, an unstructured overlay that utilises 'multi-resolution' rings to characterise node distances. The ring closest to the node contains nodes that are close to it

based on RTT measurements, with each ring expanding exponentially. Nodes have a bias for nearby nodes, that is ones in the closest ring but maintain knowledge of nodes in other rings. The number of nodes in each ring is, according to the authors, a trade off between accuracy and overhead, as it is the frequency at which nodes recalculate their distances from each other increases overlay maintenance. The knowledge of nodes on the overlay is done by a periodic polling of the overlay, with the option to send node discovery requests that include finding the closest node to a latency target. Such an operation requires the node to measure its RTT to the particular point, or a landmarking system to determine which node in the ring that fall within the measured latency is the closest.

In simulations, Wong et al. found their active measurement and multi-resolution rings produced significantly less error, classified as the difference between the optimal and the returned result, than systems that used Vivaldi's landmarking system on a CAN DHT implementation. It also showed that the average query latency dropped by over 100ms when the number of nodes in each ring was doubled from four to eight. Further addition to the rings led to incremental improvements in query latency, with 16 nodes only lowering latency by 10 percent.

The use of random walks are described in this chapter and other schemes such as AM by Zhao and Lu provide the ability to increase the efficiency of unstructured networks by not wasting resources passing messages to nodes that may provide less value to the querying node. Other tools such as IP geolocation can be used to furnish locality information, something that is proposed by N-casting Overlay in Chapter 5.

2.1.6 Membership and routing

Voulgaris and Van Steen [42] proposed gossip as a means of managing routing tables, even within structured search systems. Voulgaris suggests the use of such techniques in Newscast [63] as a means of keeping a node's routing tables up-to-date.

Newscast is based on the uniform gossip principle, with each node having knowledge of a certain number of nodes, held in a cache that is changing frequently thus, *n* picks *n'* at random from its cache and exchanges information with that node. The newest entries, as designated by the time-stamp present in each *"news item"* are kept by both nodes while older ones are discarded. This is similar to the anti-entropy policy proposed by Demers [36].

The viability of Newscast as a system for maintaining routing tables was tested by Voulgaris

[42] using 50,000 nodes, and with small cache sizes the overlay remains connected. Nodes in a Newscast overlay store a fixed sized cache of news items, with each news item containing an ID of the news item originator, network address of the originator, time-stamp of generation and application specific data. In other experiments the average path length on overlays of 128,000 nodes each having a cache size of 40 reaches 4 within the first 30 cycles. Further experiments show that should up to 50% of the nodes be removed a Newscast overlay will remain fully connected. Voulgaris' work shows that more than 75% of nodes will need to be removed before the overlay becomes disjoint, providing evidence that unstructured overlays fare well in environments with churn.

Voulgaris notes the main downside to using Newscast as a system is the bandwidth usage. To illustrate, a typical node with 20 cached entries will transfer a total of 640 bytes in an exchange with another node. If a node runs four agents it works out to be around 4Kbps, with each agent responsible for updating a single row in the node's routing table.

Membership services are typically ones that maintain the knowledge of the network. In a client-server model, the membership server would know about all the nodes either within a group or the network. A subset of this known information would then be transferred to joining nodes and a random selection of current nodes would be sent an update of joining nodes. Initial work on membership services typically focused on the replication of complete lists of network or group membership, however as the number of nodes in a network grows the cost of this quickly becomes prohibitive. Work carried out by Kermarrec et al. [64] present the size of a subset of membership data required by nodes in order to successfully maintain a connected graph and Ripeanu et al. [34] measured the amount of traffic required to maintain membership in the Gnutella system being the majority of a node's total traffic.

Kermarrec et al. [64] found that overlay membership knowledge required by a single node creates a sharp drop-off between success and failure. Formally, in a system with n_t nodes, if each node gossips to $\log(n + n')$ nodes the probability that every node receives the notification is $e^{-e^{n'}}$. The success rate also varies on overlay conditions such as node failures and link failures.

In further work presented by the same authors [24], they propose membership schemes which allow for the number of nodes known by a node to vary in real time as opposed to being set prior to joining and therefore pose an inability to adapt to changing overlay size, such as those

found in Lpbcast [65]. Another proposed solution [66] utilise properties of Harary graphs [67] in order to provide a balance between the selection of nodes and the interactions between them, to reduce the probability of isolation. Kermarrec's proposed membership system, SCAMP, provided equal performance to those which use complete system view even where overlay failures occur.

This work uses terminology from graph theory, with terms such as *fanout*, *edge* and *node*. A node, also known as a vertex, is a device that is connected to the overlay, it can also be referred to as a peer. Edges are the links that connect nodes within the overlay and *'fanout'* is the number of links going out of each node or cluster. In hierarchical memberships there can be intra and inter-cluster fanouts. A remote list contains a number of random nodes which maintain inter-cluster links and can be thought of as cluster gateways.

In a flat membership, the fanout required to achieve 99.9% message delivery rate grows with the size of the network and the probability of node failure. Exhibiting exponential growth, it is clear that the information on the number of links needing to be maintained is high and therefore constitutes a high cost to participating nodes. However, resilience to churn is high with close to 100% of nodes reachable even with 50% node failure in the network. In a 50,000 node network with each node having a fanout of 15 and 50% node failure rate, 99.94% nodes were still able to be reached.

Although reliability remains high in a hierarchical network, SCAMP's performance falls behind that of flat gossip. The variations of inter and intra-node fanouts has considerable effect on reliability and an increase in average latency in message delivery between nodes was observed by Kermarrec et al. [64]. However, the decrease in link stress is considerable, especially as clusters become smaller. Lin and Marzullo [68] have shown that when there is a hierarchical structure of network connections then a hierarchy on the overlay may exist. The preference of certain network characteristics such as low latency will tend to not only attract a certain type of node but one that is possibly determined geographically. One must consider the requirements placed on gateway nodes between hierarchies and the repercussions should these fail, therefore the selection strategies for the nomination and appointment of gateway nodes has a significant bearing on overlay reliability.

2.1.7 Gossip broadcast strategies

Search and membership mechanisms based on gossip protocols may seem primitive, however it can in some cases lead to low query response times should a node containing the requested information be within the query originator's area. Research has shown that the number of messages required to be sent by querying nodes quickly becomes prohibitive and causes gossip-style searching to be unscalable with network size. The load on each node grows at a linear rate alongside the size of the network. On systems with tens of thousands of nodes such as Gnutella, those on lower bandwidth connections have problems sustaining traffic required in order to maintain a good level of service to other nodes on the overlay.

Ripeanu [44] suggests that the flooding mechanism used within Gnutella should be replaced by a system which is *"less expensive in terms of communication costs"*. Ripneau charts the number of messages being handled by a single node showing query messages forming a significant percentage of all traffic within that node. Castro et al. [69] propose the replacement of an unstructured overlay, with results showing that their *'Structella'* overlay had a lower average message per node cost.

LV et al. [11] found that 95% of nodes on the Gnutella overlay were less than seven hops apart. This means that 95% of the time a message has to travel seven or less links between nodes. Further work by Ripeanu et al. show that as the Gnutella network scales up, the number of links per node remains constant at 3.4 connections per node.

Gnutella crawling results presented by Ripeanu et al. [34] found that 55% of all node traffic was simply used to maintain group membership. They estimate around 330TB of data is transferred by Gnutella nodes simply to maintain the network. However improvements later on brought the percentage of traffic due to membership maintenance down to 8%.

Improvements proposed to Gnutella's flooding approach include random walks [11], [48], [13] and [70], expanding ring [11] and breadth first search [71], [72]. Gkantsidis et al. [48] found that for popular objects, random walks exhibited better performance. However it is the '*randomness*' characteristic which becomes a downside to this technique, coupled to the fact that if a node which is already overloaded is randomly chosen then the request may get queued, delaying response.

Work presented by Karp et al. [47] tries to move towards the performance benefits of a deterministic gossip protocol while retaining the robustness of a randomised selection gossip protocol. They concede that it is the randomness which leads not only to greater propagation time but more messages being sent (similar to the problem faced in Gnutella). To mitigate the costs of gossip style algorithms, heuristics were proposed to allow for targeted query routing.

Setting the TTL for search queries is a compromise between query success rate, the number of messages within a system and the time one can reasonably wait for an answer. As the TTL is increased, the probability of a successful result increases, with measurements done by Lv et al. [11] show that a plateau on probability query success rate is quickly reached. This is similar to the issue of increasing the query routing performance of DHTs discussed in Section 2.2.10.

In order to overcome the randomness of random walks, a bias towards high degree nodes was suggested by Stoica et al. [57]. These highly connected nodes are more likely to know nodes that have the requested object. They are also more likely to be longer-lived nodes which can be relied upon. However such a bias does not solve the problem of overloading certain nodes, in fact it could make the problem worse as highly connected nodes get a greater percentage of queries. This returns to the problem of whether creating controllable bias in random walks is favourable, as discussed in section 2.1.3.

Yang [72] proposes two modifications to breadth first search, iterative deepening and directed breadth first search. Iterative deepening utilises the low cost of running *'shallow'* queries by running a tiered query resolution strategy. A system wide policy determines the number of increments and the depth level at each increment.

Once a node on the deepest level of a particular run is reached the whole search is suspended with all nodes at that depth being labelled *"frontier"* nodes. The next iteration commences after a particular wait time (another system parameter). In order to reduce overhead and duplication, when the next, deeper run is started nodes that have already been traversed do not respond to queries, merely forwarding them on. The messages are marked with unique identifiers which the nodes evaluate, a system which is used in Gnutella, and only when the query hits a frontier node will the message be dropped and the search will unfreeze. This procedure continues until the depth specified in this iteration is reached with the process repeating until the final round in the strategy is completed. It should be noted that the round can be completed with an

unsuccessful resolution to the query.

Yang's other proposition was *"directed breadth first search"*, a more pinpoint and possibly latency reducing strategy which aims to use data collected on neighbours to make judgements based on a particular metric. Instead of sending messages to all its neighbours, a node would pick a subset of them based on some metric. This is similar to anycasting which is investigated in Chapter 2.4.

Metrics suggested include response time, the number of queries returned, the number of forwarded queries (not necessarily answered) and the message queue size. Previously cited research has shown that metrics that assume past performance is a reliable indicator of future node performance. Stutzbach and Rejaie [73] have shown this to a be reasonable assumption, however with the fluid nature of P2P overlay membership it should not be relied upon solely. Other metrics such as message queue size and latency can be used to make more immediate judgements on a node's viability to answer a query.

The combination of one or more of these metrics would need to be incorporated in any node selection strategy. For example, selecting nodes based on latency alone may lead to a drop in the number of successfully forwarded queries as the node is unable to cope with the number of messages it has to process. Results published by Yang [72] shows that using a subset of nodes provides a *"surprisingly"* small decrease in quality of results but only if neighbours are selected *"intelligently"*.

Another strategy proposed is the expanding ring. The expanding ring strategy is to start a flood with a low TTL and successively increases it should the object not be found. For the majority of nodes within a system will be able to reach highly replicated content with low TTL as borne out by results published by Lv et al. [11]. Results published by Lv et al. [11] show the message overhead and the number of nodes visited compared to flooding is significantly lower. While expanding ring is favourable in these characteristics, it does increase hop count, typically doubling it from 3 to 6. Unlike random walks, expanding ring does not tackle the issue of duplication of messages something which is addressed by random walks, leaving an inefficiency in the system. Expanding rings are used in GOAT, presented in Chapter 4.

Chawathe et al. [13] argue that distributed hash tables are less suited to networks which have

more *"transient"* node behaviour. Prior research [25], [74] has shown that sessions on P2P overlays follow a traditional Pareto cumulative distribution curve. Measurements done by Saroiu et al. in 2002 show that around 90% of user sessions lasted for 300 minutes with 65% of nodes measured stating they were on broadband connections. The median session duration is about 60 minutes.

The cost of users leaving the network without proper disconnection procedures is one aspect where Chawathe argues DHTs have greater cost in '*healing*' the overlay in order to maintain correct routing and lookups.

As broadband, always-on Internet connections become more prevalent and within P2P systems average download speed increase as a factor of average upload speed, the average download size may increase as will the session time. In the Gnutella network, measured by Saroiu, the top 20% of nodes had more than 45% uptime.

There is evidence that while the pinpoint accuracy of DHT like systems provides the ability to find even the least replicated data, the majority of users are seeking a small percentage of the content on a P2P network. Although flooding algorithms can manage this, the $\mathcal{O}\ (n_t)$ lookups required in order to do so mean it is not feasible. There is evidence to show that poorly replicated content is rarely searched for. While obscure objects may tend to remain obscure this may produce an acceptable real world experience in systems.

Work presented by Chawathe show that their tests on a live Gnutella system, data which had more replicas were requested more often. This tends to suggest that *'hot'* - popular - content is what the majority of users are looking for, and presents a challenge for P2P overlays to deal with flash crowd scenarios.

Heuristics and a replication strategy can result in objects which are only available at a single node being replicated on on nodes that are geographically far away from the original copy. Such a strategy could be used to mitigate the shortcomings and forecast when an object is to become popular, relieving the burden on a single node and reducing latency of lookups. Conversely, picking a piece of content and replicating it on the basis that it *may* become popular is a hit and miss affair. Incorrect choices would result in content being replicated with no gain, adding cost.

Gnutella's flooding algorithm can theoretically find any data within its network, however it needs to flood all nodes to guarantee do so. However data which is popular will tend to have more replicas within the network, with Chawathe et al. [13] indicating that it is a common phenomenon in the Gnutella system. Therefore, one can argue that for the majority of users, flooding can provide adequate search results for popular data while not having such a high viability for poorly replicated data.

Although investigations centred around the use of gossip for searching for data, the same principles can be used for the building and transfer of routing tables. Although unstructured networks, by their nature are not based around a rigid topology like a ring or tree, nodes still need to maintain some knowledge of other nodes otherwise it won't be able to set respectable TTL times for flooding or to pick a node to forward a query.

This chapter shows unstructured overlays require a number of supporting methods in order to construct an efficient overlay that is representative of the underlying network. Discovering nodes can be done in a random manner or through using a random walk, and it is this part of the overlay joining process that is crucial to the overlay's performance.

Overlay sampling methods despite the heuristics employed, can only provide part of the solution when designing a low-latency unstructured overlay and message routing policies also provide an integral role in ensuring low-latency message passing between nodes. A number of schemes have been described in this chapter, highlighting the need for both overlay sampling and routing policies.

Unstructured overlays provide a means of distributing resources through a probabilistic mechanism. To create an overlay that does not have preset heuristics introduces a randomness that has both positive and negative effects as discussed in this chapter. While past unstructured overlays such as Gnutella have been shown to exhibit attributes that question its scalability, changes such as creating hierarchies, has shown that unstructured overlays can be efficient.

Perhaps the biggest motivation for the development of unstructured networks is the way they can handle churn. Churn is investigated more closely in section 2.3 and presents a major challenge to P2P overlays. Due to a lack of pre-defined heuristics, the ability for churn to result in islands is mitigated and the recovery time from mass node departure.

Further work on unstructured overlays is presented in GOAT, Chapter 3, and N-casting Overlay, Chapter 5.

2.2 Structured overlays

Structured overlays is the general term given to overlays which identify and place nodes through a function. The most common function is a hash that is applied on a node's IP address to place it on a predefined structure, such as a ring or tree. This leads to the use of a hash table, and in the case of a distributed network, a distributed hash table (DHT).

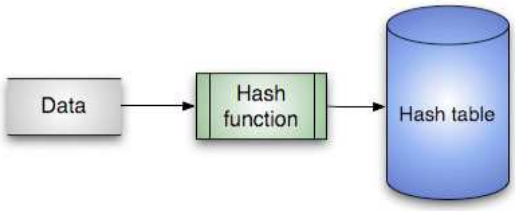


Figure 2.1: Hashing into tables

Hash tables provide a randomisation scheme by operating a hash function on the data supplied to provide an index or address producing an address where the data is stored. The hash function, or key, can be a relatively simple calculation such as subtracting a number from the data or a more complex translation such as applying SHA-1 [75] or MD5 [76] hash functions. When a query takes place, the hash key is applied to the query and the contents of that position in the table is returned.

Hash tables generally have two functions, get and put, and to delete an entry one can simply put zeros. Te risk of collisions can be mitigated by the use of complex hashing algorithms such as SHA-1 or MD5. The methods by which hash functions such as SHA-1 and MD5 reduce the risk of collisions are outside the scope of work presented here. There are a number of types of hash tables including perfect, probabilistic, Robin Hood and schemes on addressing such as chained and open addressing [77].

In a DHT the data structure is split and stored on many nodes. There are keyspace partitioning schemes controlling parts of the hash table each node has knowledge of and an overlay network is formed to connect these nodes. The keyspace is large represented by 128-bit or 160-bit string

by virtue of the aforementioned hash functions, however this can be adjusted by the application. In order to maintain the overlay, DHTs have many more functions than simply get and put, however from a data structure viewpoint these functions are still the two most commonly used for data and membership manipulation.

The maintenance of a structured overlay is non-trivial task, especially on a network that is susceptible to churn. Challenges arise when DHTs are typically used in unreliable networks, such as the Internet and in particular P2P systems where transient nodes may make up a large percentage of overlay membership of a system. The system has to be able to scale up to hundreds of thousands of nodes without placing too great a burden on any specific node in the join and departure of nodes. One major obstacle is the maintenance of the hash table when nodes leave the DHT when the membership incurs churn. Strategies for dealing with graceful and abrupt failures are discussed in Section 2.3.

The motivation for designing DHTs come from P2P networks. Ratsanamy et al. [3] propose that the then popular (2000-2001) Napster P2P service which used a central server to index all the files available on the network would be the *"best example of [then] current Internet systems"* to be improved by their proposed DHT implementation, Content Addressable Network (CAN).

DHT systems such as CAN, Chord [4] and Pastry [5] provide protocols for routing and functions. However function sets can be limited such as in Chord, where the base protocol only provides a mapping between keys and nodes. A node's key is assigned upon joining the DHT, typically it is a hash of the node's IP:PORT address. As proposed in [4] the hash function used is SHA-1 though the general methodology is that of consistent hashing [78].

Most DHTs offer a limited number of operations apart from the ones required to maintain the overlay. Commonly a lookup query of some sort is provided. Mapping the query to an IP or nodeID for the application. If the query has a valid match, that is to say the value is stored somewhere in the hash table, a DHT will return a result; the nodeID of the answer harbouring node. A number of DHTs have been proposed and provide different representations of the address space, membership, routing and strategies to deal with churn.

Queries are typically for nodeIDs of nodes, providing consistent results on content location.

Searching for keywords such as those conducted on Web search engines has been proposed [79] to provide a more user friendly, interactive ways of using DHTs. These utilise current DHT implementations and modify them so keywords can be used in queries. Joung et al. [80] propose a keyword search overlay for DHTs which utilise reverse indexing. Content searching is not covered in the scope of this work, however represents one of the major research areas in the field of structured distributed overlays. This work focuses on how quickly a DHT can locate nodes that will return the result in the lowest possible time.

2.2.1 Structured overlay fallacies

Before delving deeper into the intricacies of structured overlays there are a few common misconceptions about their operational behaviour.

- DHTs cannot by default be used for keyword searching
- Routing on DHTs isn't, by default to the node which is topologically closest to the original node
- The structure of nodeID space in different DHTs are the same
- NodeIDs are a reliable indicator of geographical distance between nodes

DHTs garnered a lot of attention in recent years however they shouldn't be seen as the Holy Grail in distributed data storage and retrieval especially when churn is taken into consideration. They do however provide some advantages over gossip style systems such as provide an overlay which can scale to large P2P networks.

2.2.2 Identifier space

An area where there is significant differences between DHT implementations is the way they represent identifier/address space. Traditionally indexes have been held in tree-like structures such as binary trees, however with DHT implementations such as CAN, x-dimensional address spaces allow for greater segmentation while systems such as Chord, maintain a single dimensional space yet both have the same lookup complexity $\mathcal{O}(\log n_t)$. A number of DHTs utilise tree based structures for organising nodes such as Pastry [5] and Kademlia [7], with recent work being done on how to incorporate load balancing in DHTs [81].

Ratnasamy et al. [3] propose a d-torus as their coordinate space split into zones with nodes owning particular zones. While further divisions are present in OneHop's hierarchy, including fine grain divisions such as units. Similar to a slice, a unit leader is also present, and is the

mid-point of the unit key space. The slice leaders act as a gateway to other slices. The number of levels in a OneHop hierarchy can be varied however Gupta et al. [82] propose a three level system to achieve lower message overhead for the higher tier nodes (slice leaders) and keep the number of messages, and consequently the bandwidth, low.

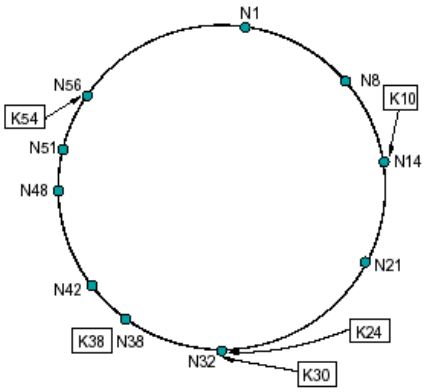


Figure 2.2: Chord ID space from Stoica [4]

In Chord, nodes are arranged in an *"identifier circle"* of modulo 2*m*, typically 160bits. The hashing function is chosen by the application using Chord and should be large enough so that the probability of two nodes or keys hashing to the same function is negligible. Chord is not the only DHT to utilise a ring structure with Pastry, D1HT [83] and OneHop [82] also arranging nodes in a ring. Though unlike Chord which uses an SHA to hash IP:PORT addresses, in Pastry nodeIDs are MD5 hashes of the node's IP address. Although work has been presented on collisions of the MD5 algorithm [84], [85] where for the purposes of nodeID assignment it is deemed that MD5 provides a *'random enough'* hash function.

OneHop places nodes into an address space represented as a circle with *m=128*. Chord and Pastry [5] utilise consistent hashing [78] in order to minimise the number of updates required when nodes join and leave the network. The implementation of address space in OneHop [82] is arranged in a ring, like Chord, it employs the use of slices which divide the circular space into a hierarchy. The *i*th slice contains nodes which have the nodeID in the range $[i \cdot 2^{128}/k, (i+1) \cdot 2^{128}/k]$. Due to the uniform assignment of identifiers each slice has roughly the same number of nodes. Joining nodes know their slice leader from their neighbours, the slice leader is the node which is in the middle of a slice's address space, similar to the zone controller in CAN.

Keys in Chord are stored in nodes which are equal or have *'higher'* nodeIDs further clockwise in the ring than the key itself. For example, key 10 would be stored at nodeID 14 if there is no node with the nodeID of 10 and 14 is the nearest clockwise node in the ring as shown in Figure 2.2. Like CAN, when a node joins a Chord network it needs to populate its routing table or successor list. Nodes do this by calling a *stab()* function which allows it to locate the next node in the network and that node's predecessor (before the new node joined).

All DHT systems assign IDs randomly in order to maintain geography diversity among neighbouring nodes. Rowstron et al. [5] specifically mention the advantages of this, saying nodes can have diverse geography, jurisdiction and ownership due to this. These attributes are utilised by both Chord and Pastry to replicate objects on neighbouring nodes.

The random allocation of nodeIDs based on IP addresses results in unpredictable distances and response times between nodes and is what this work is trying to mitigate. The random assignment of nodeIDs is preferential for resilience, it allows for nodes to be close to each other in the Pastry system which have wildly varying latencies. Rowstron et al. proposes a metric in order to overcome this and is discussed in Section 2.2.4.

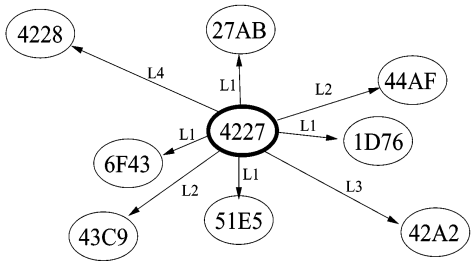


Figure 2.3: Node placement in Tapestry taken from Zhao [6]

Tapestry [6] is another DHT system proposed in 2001 along with CAN, Chord and Pastry. Nodes are given a 160-bit nodeID and referenced with a 40 HEX characters. The fundamentals are very similar to a Plaxton Mesh [86]. Figure 2.3 illustrates links between *node 4227* and its neighbours. These are primary neighbour links which are then backed up with two other links which share the same prefix as nodeID 4227.

Both Pastry and Tapestry uses longest prefix matching of entries in the node's routing table. In Tapestry, the closest node, *n* is the primary entry on the *n'*th level is represented by *prefix(n, n'-1)+n* An example of this would be the 4th entry on the 4th level for node 648E9 would be 6484.

The Kademlia DHT [7] has been widely implemented in P2P systems such as BitTorrent[2], eMule[3] and eDonkey[4]. Based on nodes being placed into a binary tree overlay, using a XOR metric (described in Section 2.2.7) in order to bring the notion of locality in routing. Kademlia nodes are identified by 160-bit nodeIDs and keys are also of the same length and achieves $\mathcal{O}(\log n_t)$ lookups. Nodes maintain state in each sub-tree where it does not preside in order to guarantee lookups.

The structure of the overlay found in DHT systems can allow for quicker generation of routing tables when a node joins the overlay. Unlike unstructured overlays, as the system already knows where nodes are, by following a heuristic it doesn't have to send messages blindly or walk the graph awaiting responses. This allows a node to fill its routing tables, finger tables or successor lists without the costs associated with overlay discovery mechanisms in unstructured overlays.

2.2.3 Routing

Routing to overlay nodes which holds the information required can be done in a number of different ways, with policies varying depending on the DHT implementation. Pastry maintains routing tables on each node akin to IP routers, while Chord nodes keep successor lists of nodes which include *'shortcuts'* to nodes along the overlying topology.

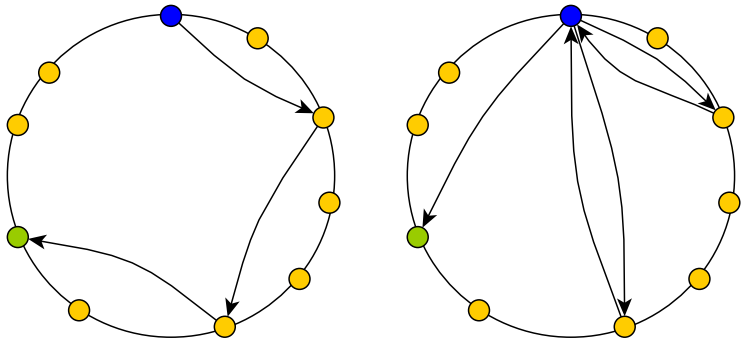
Routing within DHTs can be generalised into two types, recursive and iterative. Most DHTs utilise recursive routing, a method which causes hops on the route to assume responsibility for forwarding the original request.

Using the example presented in Figure 2.4(a), with recursive routing that uses longest prefix matching, a node with ID 111*x* requesting to find nodeID 011*x* will first find the closest node with ID 0*xxx*. This node will then forward the request to its closest known node with ID 01*xx* which in turn will forward the request to the node 011*x*. No messages are sent back to the

[2]BitTorrent Inc: www.bittorrent.com

[3]The eMule Project: www.emule-project.net

[4]Network shut down in September 2006: http://articles.latimes.com/2006/sep/13/business/fi-donkey13



(a) Recursive routing (b) Iterative routing

Figure 2.4: Recursive & iterative routing taken from Rhea [17]

original requestor node, 111x, until the lookup has been completed. Iterative routing conversely puts the requestor node at the centre of the routing, with all nodes on the hop sending back their results to the query originator.

Although the nodes involved in the query remains the same, the number of messages sent on the network is doubled. While the originator node is now in control of the process, query latency has increased due to the response cycle of each hop sending the nodeID of corresponding hop. It also could be argued that this process doesn't help nodes on the route as the amount of state information won't be reduced nor does the routing overhead alters; one request and one query per lookup.

Both Chord and Kademlia utilise iterative lookups in their original proposed implementations. Dabek et al. [21] shows that query latencies can be lowered by up to 30% through the use of recursive routing schemes in place of iterative schemes. Further improvements explored by Dabek et al. [21] include the use of proximity routing and server selection, something that can be implemented in using Anycast, investigated in Section 2.4.

2.2.4 Policies

Routing policies vary among DHTs. CAN utilises a greedy forwarding policy which forwards the query to a known node that has the highest nodeID. This process is iterated until the query is forwarded to a node which has the requested nodeID containing the key in its routing table. It then returns the location of the node to the query originator.

It has been shown that greedy routing, although seemingly simplistic, does provide the shortest

path in a regular overlay, where the nodes are equidistant from each other and their density is even throughout the overlay. Greedy routing is still the simplest and most reliable routing heuristic as it requires only one link to be present towards the targeted node.

Improvements on greedy routing include neighbour on neighbour (NoN) routing [87] and collective routing representing routing tables as bitmaps [88]. NoN routing leverages increased knowledge of neighbouring nodes routing tables to reduce the average hops to $\mathcal{O}(\frac{\log n_t}{\log \log n_t})$ whereas greedy routing has an average hop count of $\mathcal{O}(\log_2 n_t)$. However the cost to the routing table size held at each node is orders of magnitude greater than when greedy routing is deployed.

NoN routing can be looked upon as very similar to greedy routing. It will select the node which is closest to the destination node from a list of a node and its neighbours routing tables. Formally, let n_i be the set of neighbours of *n* and n'_i be the set of neighbours for node *n'*. If n'_2 is the closest destination node then the message will route to *n'* from *n* via n_2. Strictly speaking, NoN is not greedy routing as n_2 may not be the closest to the destination node.

Simulations conducted by Naor and Wieder [87] show improvement over traditional greedy routing in hop count in situations where routing tables are accurate. This means that in high churn environments either performance will be poor or routing tables need to be kept up-to-date, adding to the cost of overlay maintenance.

Wang et al. [88] propose the use of bitmaps to store DHT routing information. The advantage of using bitmaps is that they are efficient, allowing bitwise operations to occur on the data structure. Perhaps most crucially the size of the bitmap does not depend on the number of nodes in the DHT meaning that even in a large DHT overlay, the routing table at each node is relatively small. Not only is this favourable to the nodes themselves but reduces the amount of bandwidth consumed during routing table updates.

The RASTER routing algorithm proposed by Wang requires similar knowledge to that of NoN, a node's knowledge of neighbouring nodes routing tables within a certain radius of itself. Routing proceeds by identifying the bin in the system's hash space along with the associated bit in the stored bitmaps which correspond to the hash value of the key. While RASTER requires similar knowledge to that of NoN due to the use of bitmaps, operations on routing table aggre-

gation and the amount of data being sent along with memory being used is significantly less than NoN. In simulations with churn enabled, RASTER was able to consistently produce lower hop counts compared to greedy and NoN routing.

Greedy routing can be used in multi-dimensional spaces such as CAN. Nodes keep track of the IP address and the coordinate zone of its immediate neighbours in the coordinate space defining neighbours as nodes whose zones overlap along *d-1* dimensions and joined in one dimension. It is with this information that CAN nodes use a greedy forward routing policy.

In order to improve on greedy forwarding, Ratnasamy et al. [3] propose a latency based approach. By measuring the round trip time (RTT) to its neighbours a node can forward requests to neighbours who are closer on the IP overlay. This policy doesn't reduce the number of hops but it may reduce the latency for a query to be returned. Ratnasamy reported [3] latencies with RTT awareness was found to be similar to that of IP routing policies. With RTT awareness enabled, a 24-40% drop in per hop latency was recorded.

Further routing strategies include utilising successor lists that are maintained by nodes in a Chord network. Nodes push requests to the next node in the ring as defined by a node's finger table resulting in a linear relationship between number of messages and number of nodes in the system.

To aid this Chord nodes maintains a *"finger table"* which lowers the latency and hop count of some messages. The nodes stored in a finger table are determined by the formula *n+2e-1* where *e* is the entry in finger table, $1 \leq e \leq m$. It is through the use of finger tables that Chord is able to provide $\mathcal{O}(\log n_t)$ lookups to find any node with proof presented in Stoica [4].

Pastry nodes maintain multi-tiered routing tables which consist of the neighbour set, leaf set and a routing table. The neighbourhood set is not typically used for routing queries but rather to maintain locality properties. This set contains nodes which are close to the node using a proximity metric, such as latency. The leaf set contains nodes which have numerically close nodeIDs both greater than and less than that of the node itself. This is the first port of call when a Pastry node needs to route a message. So, if the requested node is in the leaf set, it is at most one hop away.

Similar to CAN, Pastry proposes the use of a proximity metric in order to bring a notion of locality to the system. The proposal calls for the routing table with nodes that are *'close'* by the application defined proximity metric. This system is more generally known as proximity neighbour selection (PNS), although in the case of Pastry, the overlay provides the heuristics for routing tables to be populated.

Castro et al. [89] argue that this method can cause problems with replication and correlated failures. These failings are attributed to the use of one-dimensional ID spaces and therefore Chord and Tapestry suffer from the same problem. In other research Castro et al. [90] propose enhancements both the proximity awareness of Pastry and removing the neighbour set from each node's state. This is discussed in Section 2.2.5.

OneHop [82] creates a hierarchy within a Chord-like ring by dividing the key space using slices and units. Both coarser grained slices and finer grained units have *"leaders"*. Nodes cannot contact nodes in other slices directly but have to do so through their slice leader, creating a tiered or hierarchical overlay.

The slice leader collects all notifications from its slice and holds them for a period of time t, at which it sends a message to all other slice leaders in the network. This broadcast is not synchronised to distribute bandwidth utilisation. The slice leaders then collect all messages received from other slice leaders forwarding the message along to unit leaders which then notify their successor and predecessor accordingly. Nodes only notify their successor using keep-alive messages. This continues until nodes at the boundaries of their unit are reached. The number of hierarchies can be increased or decreased however this has a bearing on the resource usage of the slice and unit leaders.

Kelips [8] does not maintain an overlay like that of CAN or Chord and is therefore loosely structured, instead nodes are assigned affinity groups by a constant hashing function based on the node's IP:PORT. Members in different affinity groups can message each other directly (unlike OneHop). Each node has knowledge of a subset of the affinity group's membership along with a number of nodes in foreign affinity groups. Within an affinity group, messages are gossiped which the authors say constitutes the background communication within the group and is used to disseminate views, contact and filetuples. In Kelips, contacts contain information on nodes in other affinity groups on the overlay, while the filetuples contain a partial set of

tuples that have file names and IP addresses of nodes. Nodes only store filetuples of files where its 'homenode' is in its own affinity group.

In D1HT [83] routing tables contain the IP addresses of all other nodes in the system. All changes in node membership of a D1HT network must be propagated in order to maintain correctness of tables. D1HT does away with the slice and unit leader strategy therefore trying to spread the routing traffic load more evenly among peers in the network. It forwards the message to the relevant node present in its lookup table.

As Kademlia arranges nodes in a binary tree, with nodes needing to know at least one node in the sub-tree where it is not present in order to be able to route to all nodes in the system. The node 0011 must know at least one node in the sub-tree 11, 01, 0010 and 000, denoted by the grey circles, and is a policy used in the N-casting overlay presented in Chapter 5.

Routing in Kelips [8] is done on a preferential RTT basis, similar to OneHop's routing policy. Nodes maintain information about a small number of nodes within their affinity group and those from foreign affinity groups. Each entry in the routing table contains information such as RTT to that node.

For routing schemes which perform worse than $\mathcal{O}(\log n_t)$ Xu et al. propose expressways [91] as a way of improving performance to at least $\mathcal{O}(\log n_t)$ without any negative impact on maintenance costs. Xu et al. proposed expressways built upon CAN because it provides worse than $\mathcal{O}(\log n_t)$ routing found in that DHT system. Using similar zonal splitting of the Cartesian address space, each expressway zone ranges from the smallest being a CAN zone to the biggest being multiple zones. Expressways offer nodes the chance to contact nodes in other layer zones; zones which are not adjacent to each other; in a single query. Expressways build a tree-like structure onto CAN's address space, where four CAN zones make up a Level 2 expressway zone and four Level 2 zones make up a Level 1 zone. Each node's routing table contains the default CAN routing table, plus entries for one node in each neighbouring Level 2 and Level 1 expressway zones.

2.2.5 Locality

The notion of locality within a DHT system goes against a number of the original design goals of such a system. Due to nodeIDs being randomly generated a DHT should offer resilience should a region of the World incur network disruption. The geographical diversity of nodes

regardless of their nodeID also offers a level of physical security to the system. For instance targeting a range of nodeIDs would not result in all nodes in a particular country being knocked offline.

DHTs such as CAN, Chord and Pastry all require nodes to maintain information about neighbouring nodes and while they may be neighbours on the DHT ID space, geographically they could be on different continents. This inevitably leads to lookup times being higher than optimal. Xiang et al. [92] state that *"in order to provide QoS-aware multimedia distribution service, a topology-aware network is necessary"*. A number of methods have been proposed to reduce the latency of lookups.

A number of schemes [92], [93], [94], [95] and [96] have been proposed as topologically aware overlays, including expanding ring search, landmarking and heuristics. They have been applied to both structured and unstructured overlays displaying better adaptation to the underlying IP level topology.

The use of oracles to create these overlays was outlined by Xiang et al. [92] where it is called dynamic landmarking with a *"global host cache"* which would be queried by all nodes joining the network. The cache would randomly pick a host and the querying node would then measure its RTT from that of a randomly picked node. Depending on thresholds set by the application, the node would then be placed in the closest group to the randomly queried host or the other groups which exist on the overlay. The node will then conduct measurements to find which group contains nodes that are closest to itself.

Using this method nodes can find the closest group to them in $\mathcal{O}(\log n_t)$ steps. The presence of an oracle means that the dynamic landmarking system still has a single point of entry and also failure and the maintenance of this is vital in the proper operation of this system. One could envisage the global host cache server being replaced with a DHT itself.

The ratio between the average inter-node latency on the overlay and the average inter-node latency at the IP level is represented by the latency stretch. Simulations conducted by Xu et al. [96] show that expanding ring is the least favourable of the three schemes with a very large number of nodes needing to be tested before reliably the nearest neighbour is contacted, further experiments with expanding rings are conducted in Chapter 4. Landmarking, when using RTT

provides much better results. Experiments conducted by Xu et al. show that as the number of RTT measurements increase, the latency stretch decreases when expanding ring search and a hybrid landmarking plus RTT is used. The effectiveness of landmarking is highlighted by Xu et al. [97], where their hierarchical DHT, Hieras, exhibits lower routing latency with an increase in landmarking nodes.

Ratnasamy et al. [95] propose the use of landmarking to create bins of nodes which have a similar distance from specific landmarks. With both relative and absolute distances being able to be calculated, distributed binning aims to have nodes which are topologically close to each other in the same bin. Using exact RTT values would provide too high a resolution and therefore binning levels offer a range of latencies to profile nodes. For instance a system may have three levels where level 0 would contain nodes with RTTs between 0-100ms, level 1 100-200ms and level 2 $>$ 200ms. Using a level vector, nodes could be identified by their levels in increasing order of their landmarks.

Multi-tiered overlays are also present in unstructured networks with Gnutella utilising *'ultra'* and *'leaf'* peers. The ultra peers are high bandwidth peers which communicate with other ultra peers. These peers would route messages among other ultra peers and those from leaf nodes. Leaf nodes are not permitted to route messages as they are considered to be too unstable.

The introduction of this two-tier hierarchy was a response to the poor scalability of the original flooding algorithms used in the Gnutella system. Since its conception, performance has improved significantly with measurements conducted by Stutzbach et al. [56] showing that around 10-12% of nodes become ultra peers with the greatest percentage of them having between 25 and 30 leaf nodes, while the majority of leaf nodes had 1 and 5 ultra peers. Although a small percentage of ultra peers were very highly connected (having around 3500 links) some of these were found not to be routing any traffic. Generally the number of links an ultra peer maintains effects the bandwidth utilisation of that peer.

2.2.6 Proximity neighbour selection

Proximity neighbour selection (PNS) has been proposed as a promising area of research to minimise delays in a number of works including Plaxton [86] and more recent work by Castro et al. [90]. Both highlight the advantages PNS has over simple routing protocols without proximity awareness. It is the work presented by Castro et al. that provides motivation for DOAT, the structured overlay presented in Chapter 3 that has proximity awareness at its heart.

PNS populates a node's routing table with nodes close at the IP layer. The definition of close is dependant on the implementation however the most common interpretation is round trip time.

Castro et al. propose constrained gossiping, a method which alters Pastry's join and maintenance protocols which they claim reduce the overhead over those found in Pastry using an oracle, though the authors propose an alternative method where nodes pick a random set. When *n* joins the overlay it contacts an existing node, *n'*. *n'* then routes a message back to the newly joined node using *n*'s key. The *r*th row of *n*'s routing table is updated with the node that has a nodeID which matches *n*'s nodeID in the *m-1* digits (typical Pastry behaviour). Castro et al. argue that the resulting routing table of *n* is *"nearly perfect"* provided that *n'* is the closest node on the overlay given a proximity metric. They provide their own implementation of such a proximity metric or suggest using expanding ring IP multicast.

2.2.7 XOR metric

Kademlia [7] utilises a XOR function which is applied to either the two nodeIDs or the {key, value} pairing in order to assign them to a particular node. Formally described, the validity of XOR in Kademlia is because

$$d(n, n') = 0, d(n, n') > 0 \text{ if } n \neq n', \forall n, n' : d(n, n') = d(n', n) \tag{2.1}$$

This unidirectional metric works because it holds the triangle property, that is the distance between point *n* and point *n"* is shorter than the distance between point *n* and point *n'* plus the distance between point *n'* and point *n"*. Formally,

$$d(n, n') + d(n', n'') \geq d(n, n'') \tag{2.2}$$

Uni-directionality results in all requests for the same key converging along the same path. This property allows for caching along the route, reducing the work done at each hop. The notion of distance in Kademlia through the use of the XOR metric is the overlay distance and not geographical distance. So while distance in the node space may be small, the nodes could be separated by thousands of miles. Although this can be seen as a shortfall in Kademlia, the job of assigning node IDs becomes an important one. A modification takes into account locality against a node which has its location known or through a coordinate system such as Vivaldi [59]. This allows nodeIDs to be assigned which are geographically close to each other. However it should be noted that such approaches could have an impact on the system's resilience should there be a network problem within a particular region.

2.2.8 Single hop routing

Work has been carried out on reducing the latency of queries by using greater resources on each node to achieve lower hops per lookup. The greatest reduction of hops, down to one, was proposed by Gupta et al. [82] in the OneHop DHT system. By sacrificing memory usage and bandwidth, nodes maintain information on all other nodes in the network. OneHop classifies queries as successful only if they utilise one hop from source to destination. The system can use more hops for lookups if required.

Both OneHop and D1HT require nodes in the overlay to know all other nodes. Through the use of ERDA, an event detection and reporting system, D1HT nodes maintain knowledge of node churn. This global knowledge of nodes is seen to be a barrier to scalability. Not only would routing table size become a considerable burden on nodes but the time taken to know a significant percentage of nodes needs to be taken into consideration. However the idea of single hop routing may be feasible for low membership overlays found in multi-tier structured overlays. Churn also presents systems with large routing tables greater problems, as updates would be required more frequently to maintain overlay membership knowledge.

2.2.9 Hierarchical DHTs

The DHT overlays described so far have all been flat structures. This presents some advantages such as a uniform distribution of functions and load throughout the system. However Ganesan et al. [98] present the notion of hierarchical DHTs which have greater fault isolation, better bandwidth utilisation, hierarchical storage and access control. Garces-Erice et al. [99] state that hierarchical DHTs can *"significantly"* reduce the number of peer hops in a lookup and latency.

Although hierarchical DHTs can mimic the underlying network topology by using root and leaf nodes this can place more strain on root nodes. These nodes may act as a gateway for the leaf set and can, in the case of OneHop, cause significant load to be placed on them. The lack of uniformity across the ID space is highlighted by Artigas et al. [100] that could lead to non-uniform load areas.

The system they propose, Cyclone, provides an ID assignment scheme which introduces a tree-like structure to multi-level Chord rings and uses a XOR metric to route across levels. Its goal is to remove the idea of a single gateway to other groups in the same hierarchy and thus remove the notion of specially selected nodes which are presented with this burden.

Zoels et al. [101] uses a multi-Chord structure where rings are connected by a single node to a super ring. They use it as the basis of their cost analysis of hierarchical DHT systems. Superpeers (SP) are root nodes with leafnodes hanging off them. The leafnodes use simple PING/PONG messages to check the availability of their SP and know of other SPs should the one they are attached to disconnect. This architecture is similar to that of Cyclone.

HIERAS [97] uses distributed binning [95] to create rings of nodes. Depending on the number of hierarchies the layer vector, identical to that described in Section 2.2.5 by Ratnasamy's distributed binning scheme. The number of layers offers a level of granularity in the latency filtering of nodes with the highest layer encompassing all nodes on the overlay.

Further results from HIERAS shows the balance that needs to be attained between the number of hierarchy levels and the cost of maintenance. HIERAS latency performance improved with the number of layer that were present in the overlay, however the cost of maintaining the routing table also increased in relation to the number of layers. The authors suggest a HIERAS system with four layers has the optimal cost/performance characteristics.

Aside from the resource demands placed on super nodes, the choice of super node is critical in the performance of DHTs. Artigas et al. [102] illustrate that as a possible weakness the root/super node design of hierarchical DHTs. The assignment of these nodes typically depend on their uptime however with no global knowledge of the system Artigas argues that it is difficult to determine what uptime is long enough for the node to be bestowed super node status. Results presented by Stutzbach [73] shows while historical node uptime can be used as a reliable indicator of future node uptime the degree to which it can forecast ahead varies between P2P networks. Nevertheless past performance cannot be taken for granted as a perfect indicator of future performance, especially given external factors which may affect the overlay.

For systems approaching anywhere near critical use it would be foolhardy to choose a single node to be upgraded to supernode status based on a forecast. It should also be noted that measuring node uptime does not measure a node's capability to handle the extra load on resources such as memory, processing power and link utilisation that comes with being a supernode. This could lead to the node failing at the most critical time in the system; a time when it is relaying or resolving the most queries. Prudent selection would be to have more than a single supernode per cluster with some form of *'hot spare'* being assigned.

The performance of hierarchical DHTs in both super node and CORAL's homogeneous design presented by Artigas et al. [102] showed the hierarchical design produced a lower average path length given the number of nodes within a cluster. From a latency perspective the path which has the biggest effect on latency is the inter-group paths meaning traversing links to nodes which are far topologically far away. Results from [102] show that when the probability of inter-group paths being used increases both super node and homogeneous converge to the same result. For systems with small groups both designs perform similarly.

Hierarchical structures have been applied to 'flat' DHTs such as CAN by the use of Expressways has been presented throughout this chapter, however there is a set of commonalities among hierarchical DHTs that pose challenges, especially in overlays that have unstable membership. The ability to go between hierarchy layers is handled by a subset of nodes, creating a single point of failure. This can be mitigated by having multiple 'layer gateways', however this would increase the amount of membership data that is transferred to maintain an accurate view of the overlay.

2.2.10 The cost of DHTs

Like any system, a cost exists with DHTs that prohibit them from exhibiting optimal performance in real world systems. Traditionally the cost of a DHT has been the amount of state data each node keeps, however Li et al. [18] suggest other metrics should be taken into consideration, such as maintaining the state data and exploring for neighbours to populate routing tables. The cost of finding new neighbours is an important aspect of the bootstrapping within any DHT system, especially one which is minimising lookup time from first joining the overlay.

While building and maintaining state is an important metric, it is important to consider the cost of a lookup query. Metrics for queries have borrowed heavily from traditional network metrics. Typically DHT systems provide lookup cost in terms of hop count, latency, success rate and probability of timeout. It is a reasonable assumption that latency is affected by hop count given that hops in a DHT system may not be geographically close to each other, only one of these needs to be considered. Rhea et al. [17] showed the possibility that near real-time distribution systems will set lower timeouts for responses from nodes in order to minimise any possible delay at a single hop or node and the time taken for the previous node to try another route. This has been also mentioned in Lv et al. [103] and is typical of the performance versus cost decisions that have to be made when designing DHTs and provides an insight into measuring

the performance of DHTs. Those that are optimised for low latency may report higher failure rates in the pursuit of lower latency. It has also been argued [103] that latency is not a reliable performance metric for a DHT that experiences churn.

Metrics such as CPU time and memory have mitigated the impact of cost with the proliferation of low cost hardware available to the public. The availability of memory has made systems such as OneHop [104], where each node has complete overlay knowledge allowing for each message to be routed directly to the destination node in one hop. While node memory availability has increased, an on unstable network and large network such as the Internet it produces too great protocol overhead.

Work has been presented [18], [103] on the performance cost for establishing a framework to evaluate the cost of DHT systems. It should be noted that this work does not model the amount of data being transferred within the DHT in order to replicate information. Li et al. [103] claim that this *"dwarfs"* the amount of data typically transferred in maintaining routing tables and present the argument that the storage cost of keeping routing tables within a node is far lower than cost of maintaining *"correctness"*. The position, maintenance of replicas within any such overlay is important from the aspect of redundancy but also from a node's knowledge; exactly how many replicas should a single node know about in its routing table?

Li et al. measure the *"live node bandwidth"* that is, the amount of data required to join, lookup, fill and maintain the correct node state, for five DHT implementations. Their results for failed lookups provide very similar results among DHT implementations. While the amount of data transferred between nodes has an impact on the success rate of queries, the failure rate quickly plateaus for almost all DHT implementations simulated. The cost of bandwidth to failure rates follows a Pareto curve with the biggest gain achieved for a relatively small increase in bandwidth, after which subsequent increases fail to register much of an improvement. This could be attributed to the unstable nature of the overlay or a percentage of nodes that are unable to cope with the increased live bandwidth and therefore unable to service a percentage of queries.

A similar trend was recorded when measuring successful lookup latency compared to the amount of data transferred between nodes. Except for the OneHop implementation, which exhibited constant latency despite increasing data transfer, all Tapestry, Chord, Kelips and Kademlia showed a rapid drop in latency by up to 20 bytes/nodes/sec after which any decreases

were minimal. Only Kademlia required more bandwidth (40-50 bytes/nodes/sec) before latency stabilised.

OneHop presents favourable results using the Performance versus Cost framework (PVC), though Li et al. suggest that the fact that all nodes knowing each other would not be viable for very large networks. The most visible problem is the amount of bandwidth that slice and unit leaders over standard nodes. Li et al. found the overhead for such nodes is *"eight to 10 times"* greater than that of normal nodes. Gupta et al. [82] say this can be mitigated by having more slices, consequently creating more units with more leaders. The downside of this is greater propagation delay and more bandwidth being transferred in order to maintain routing tables. This leads to Li et al. suggesting that OneHop is only viable for smaller networks and nodes which can handle greater than 20 bytes per node per second.

Work presented by Monnerat [83] attempts to curtail the amount of bandwidth particular nodes in the D1HT one hop DHT network by removing slice and unit nodes. From their simulation, OneHop is seen to place unreasonable bandwidth demands on slice leaders. Instead, D1HT attains its reduction from nodes notifying a subset of overlay nodes of membership changes. Both OneHop and D1HT have parameters which allow for the percentage of queries *f*, which take a single hop. D1HT shows that the amount of node bandwidth utilised varies little when *f* is between 1 and 10. This is used as the motivation for work presented in Chapter 5.

Kelips [8] increases the use of resources on each node to achieve $\mathcal{O}(1)$ lookups. By utilising $\mathcal{O}(\sqrt{n_t})$ space per node, it presents a usage of 1.93MB per node in a 10 million object, 100,000 node Kelips DHT. Not only does the routing table have a cost of $\mathcal{O}(\sqrt{n_t})$ but also replication of the file index also presents the same cost.

The cost of DHTs are derived from the willingness to accept a certain failure rate or by setting a threshold of latency which is deemed acceptable. Systems can leverage resources of a node in order to lower the number of hops a query has to take or to abstract the overlay into hierarchies so that a subset of nodes on the overlay are reachable in a lower number of hops.

Acceptable query failure rate must be considered against the definition of failure in a given system. Those systems which require a sense of *'liveness'* can set timeouts to be lower for queries thus increasing query failure rates, in systems where nodes can afford a longer delay for

their queries to be answered this may be lower. Nevertheless, evidence has been presented that relying on high resource usage, whether they be node-side in the form of bandwidth or memory usage or in overlay construction can only positively affect performance to a certain degree. The cost of decentralised, structured overlay networks will remain.

2.3 Churn

Structured overlays deployed in P2P networks have to deal with nodes which do not exhibit uptime characteristics associated with servers. The transient nature of nodes in a P2P system means that overlays such as DHT systems have to cope with node join and departs. The arrival and departure of nodes is called churn.

The measurement of churn boils down to two metrics, the node's session and lifetime. These represent two different views of the system, with the lifetime typically exceeding the session time. The difference between the two is stark as the node is likely to join the overlay for a period of time, disconnect from the overlay and then reconnect to the same overlay at a different time. This particular profile of churn, discussed later, is represented by the metric of node availability.

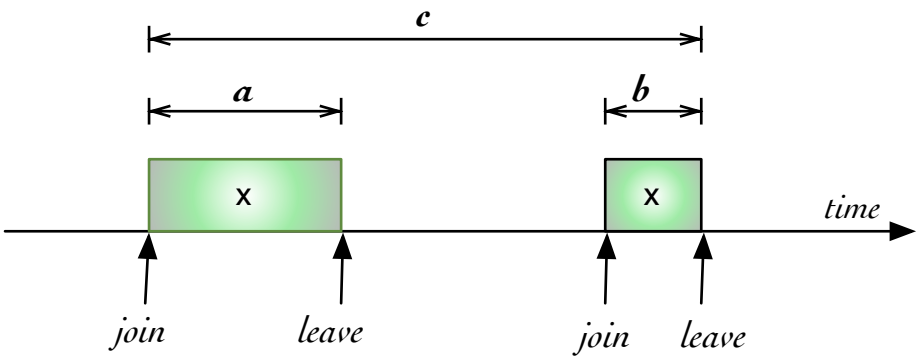


Figure 2.5: A node's session time and overall lifetime

The notion of node availability is calculated from the percentage of a node's lifetime that it is part of a session. To illustrate the difference between node availability and lifetime, Figure 2.5 shows how a node's session time differs to the total lifetime of the node. The calculation of a node's availability on a particular overlay is presented in Equation 2.3.

Nodes may have lifetimes of days, however it is more likely the session uptimes during that period may only span a couple of hours, causing the node's availability on the overlay to be

low. In Sariou et al. [25], the terminology for node availability differs somewhat. By using the terms '*offline*', '*inactive*' and '*active*' to mean departure from the Internet, departure from the overlay and active participation on the overlay, respectively, the same metric of availability can be calculated.

The distribution of the gaps between *a* and *b* in Figure 2.5 could be affected by factors such as the time taken to join the overlay. For example, if the time *t* to receive a set of nodes from a BitTorrent tracker is close to or greater than *c - (a + b)* then it is unlikely such short departures will occur. Long and complex node join protocols and high *t* can have a detrimental effect on performance should certain churn profiles which are discussed later in this section occur.

$$\beta = \frac{a+b}{c} \tag{2.3}$$

A node is likely to return to the overlay and some point during its lifetime by overcoming hardware or network failure or or user behavioural patterns, such as switching off the computer at the end of the day. The availability distribution may allow for prediction in node behaviour, similar to that used in electricity provisioning [105] allowing for overlays to brace for mass, geo-localised node departure.

Bhagwan et al. [106] present measurements based on Overnet, a P2P file sharing system which is based on the Kademlia DHT found that as the measurement duration increased, the availability of nodes decreased. This isn't a particularly surprising conclusion as not only are nodes more likely to fail for a number of reasons but that their own need to take part in the overlay diminishes. In the same research, it was found that over a nine day trace, it was observed that there was a 32 hosts/day decrease. The authors suggest periodic refreshes of files to introduce 'interest' for nodes to stay available on the overlay. A multitude of incentive mechanisms exist to increase node involvement in overlays, however they are outside the scope of this research.

While findings in Bhagwan, Sariou et al. [25] and Cha et al. [107] exhibit diurnal patterns of node availability, other research found that the availability of a particular node had little to no dependence of another node. This independence displays a positive characteristic of distributed overlays, that is to say, a domino effect of node availability is unlikely to occur.

Churn encompasses two types of node departures, graceful and disgraceful. While node joins are included in the definition of churn, the majority of research has concentrated on the effect

of node departure as it provides the DHT with an immediate problem; a *'hole'* in the overlay. A graceful departure is when nodes leave the overlay giving prior notice, allowing for the DHT to reallocate the ownership of the keyspace and for nodes to remove the departing node from their routing table. This could be likened to a Bittorrent user closing their client. Disgraceful departure is one with little to no prior notice given to other nodes in DHT. Possible reasons for this could be underlying network or hardware failure.

Graceful departures occur naturally in the life-cycle of a P2P system and as such doesn't place too much stress on the efficient running of a structured overlay. However mass disgraceful departures can cause significant problems for nodes which remain on the overlay as their routing tables may forward queries to nodes that are no longer on the overlay. The recovery time after mass departure is one of the challenges in DHT design.

Node departures due to network or hardware failure are simply unavoidable and guaranteeing the reliability of the underlying network is out of the scope of any overlay, however the methods in which the overlay deals with them is within the scope. Therefore nodes that depart due to those circumstances must be treated as a lifetime hazard within a DHT. Similarly, nodes that leave the overlay once having their query resolved is something the DHT has to live with. The job of keeping nodes connected to the overlay is really the job of an incentives mechanism which promotes those who contribute to the functioning of the overlay. A DHT must provide a mechanism to handle the flow of nodes entering and leaving the system, maintaining lookup success rate and latency.

Accurate measurement of churn on P2P networks is a non-trivial task as outlined by Stutzback [73]. The problem lies with the requirement of continuous data. Any loss in network connectivity or hardware failure results in a gap in monitored results. Later if this gap in data is seen as a single, continuous set of results, significant error may be reported. Other factors such as nodes behind NAT connections may result in inaccurate measurements as many nodes could be viewed as a single node. Conversely, dynamic addresses, while not posing problems for uptime measurement can induce problems in measuring the gap between sessions or correlations between sessions.

Two ways of keeping a node's routing tables up-to-date with information is presented by Rhea et al. [17]. Reactive recovery occurs after a node is aware that an entry in its routing table is no

longer active. The node then sends its updated routing table to neighbours. To reduce the size of data being transferred, only those entries which have changed are transferred, not the whole routing table. This scheme allows for shorter update cycles at the cost of message numbers which still is $\mathcal{O}(e^2)$ and is used in MSPastry [89].

The second method is periodic recovery, which results in nodes sending their routing table to a randomly selected node already known to the node. The update takes place periodically regardless of whether any known nodes disconnect in between routing table updates. The update time is a system parameter, however in the interests of minimising maintenance costs a balance between bandwidth cost and routing performance should be considered.

Convergence is not as fast as the reactive recovery method taking $\mathcal{O}(\log r)$ rounds for a routing table with e however total messages sent is lower. This system is used by Chord [4] to maintain node finger tables. Results presented by Li et al. [103] show that stabilisation intervals in Tapestry [108] have a great effect on average lookup latency. Although lower latency comes at a cost of bandwidth, in results published by Li et al. show that the delta is an acceptable cost given the performance increase.

Li et al. proposed the notion of routing table *"freshness"* with the intent of finding the optimal freshness whereby entries don't result in timeouts over a certain threshold. The importance of this value is that it can reduce the number of times routing table updates occur, saving bandwidth and reducing link stress. In simulations, Li et al. found that not only does the routing table size decrease the latency of lookups but that using the freshness heuristic their modified DHT, Accordion, based on Chord, can get close to OneHop's lookup latency. Other desired behaviour is the bandwidth usage remaining at roughly the same levels regardless of the number of nodes which isn't the case with OneHop or Chord.

Simulations conducted by Rhea [17] show that under low churn conditions, reactive recovery works well, with updates being sent sparsely bandwidth utilisation was low. However for high churn environments it was visible that periodic recovery presented better performance with significantly lower bandwidth utilisation and lower latency. This behaviour could create a vicious cycle in systems which utilise reactive recovery. A node may mistakenly report a node in its routing table as being down when in fact its link has been saturated by sending other nodes' routing table updates. In bandwidth usage terms, overlays which are larger cause

reactive recovery systems to be almost prohibitive. Not only is this due to the size of the routing tables held at each node but the number of node state changes. Modifications such as extending the requirements in order to classify a node as being down helps make reactive recovery more feasible but only for nodes which maintain small routing tables and, at most, environments that exhibit moderate churn. It is for this reason that periodic recovery is favoured in many DHT systems.

In order to minimise the effects felt by churn a system could enforce policy whereby nodes only maintain routes to other nodes which meet an uptime threshold. Strategies for selecting nodes with high uptimes include random selection (measurements done by Stutzbach [73] show that the distribution of uptimes in nodes are weighted towards high uptime) or through measurement. The problem with employing such a strategy is that newer nodes will have higher lookup times as routing would have to resort to simplistic *'neighbour hopping'* until the message reached a neighbour which had a high enough uptime to be part of the uptime *'rich club'*.

Other strategies for node selection include selecting from a certain percentile and the highest predicted uptime based on prior knowledge. Simulations conducted by Godfrey et al. [109] show picking nodes with the highest expected uptime provided closest to the optimal strategy for estimating the future lifetime of each node. Due to the dynamic nature of P2P networks, fixed selection strategies such as picking from a certain percentile is outperformed by strategies that predict based on prior knowledge. This concurs with prior work done which show that peer lifetimes is often heavy tailed. That is to say, if a peer has a long lifetime it is more likely to stay alive for longer.

When selecting neighbours in Chord using a randomised topology where neighbours were chosen at random, Godfrey et al. found that failed node requests fell when selection policy was biased to nodes that are close on the overlay's address space. This result gives further credence to the fact that random node selection can pick nodes which have high availability.

2.3.1 Replication

The recovery from mass churn provides more of a challenge for structured overlay systems. While routing is affected with churn when a large percentage of nodes leave or fail, replication of data also becomes an issue. Systems have been evaluated when as much as 90% of nodes fail in order to provide some insight into their capabilities when mass node departures occur.

One such system is Bubblestorm [110]. Replica maintenance forms the basis of handling churn in Bubblestorm systems and while this creates an overhead for the creation and maintenance of replicas it allows for the system to withstand severe churn. In simulations with 90% of nodes departing, query times increase but within 90 seconds drop down to pre-departure levels. Bubblestorm utilises a form of reactive recovery to resize *"bubbles"* which hold the query and data.

Although Bubblestorm aims to mitigate the problem of churn by creating multi copies and therefore not relying on a single node to answer a particular query the notion of churn still remains within the domain of DHTs. The problem of churn is fundamentally one of how to maintain an accurate distributed membership system, without the need for a central server with a God-like view of the overlay. The dynamic nature of P2P overlay membership means that maintenance of routing tables may utilise a significant percentage of a node's bandwidth. The cost of maintaining routing tables populated with nodes which are able to answer queries is a balance between resources placed on the node and the percentage of delayed and failed queries one is willing to absorb.

Replication strategies include uniform, proportional and square root, which are compared in [111]. Uniform replication is the placing of replicas regardless of their demand, whereas proportional replication varies the number of replicas to the query distribution. So more requested objects are replicated more often. Square root replication r, is similar to that of proportional however rather than being $r \propto q$ it is proportional to the square root of the query probability (q), $r \propto \sqrt{q}$.

Replication can be both reactive and proactive. Gnutella replication is reactive, the node which requests an object is also the node which replicates it. However in Freenet [112] objects may be replicated without any prior request. With proactive replication the fundamental question of how many replicas should be kept needs to be answered. The greater the number of replicas, the easier an object can be found, query times will be lower and the number of messages being sent on the overlay decreases. However, the cost of replication is two fold. Not only is there an increase in memory usage on the node itself but the updating of routing tables in order to acknowledge the existence of the new replica. The routing table problem is less of a concern in flooding systems where replication can in some ways reduce the search time as there is a greater probability of finding a *'hit'* to a query.

Working on the assumption that replicating every object on every node is unfeasible, even uniform replication can be deemed as being inefficient. The same number of replicas would be present for the least queried object as the most queried object. The logical progression is proportional replication however calculations by Lv et al. [111] show that the average search size, the inverse of the number of replicas of a particular object, remains the same as uniform replication. The optimal replication strategy is square root producing far lower variance in search size than uniform and proportional replication.

Replication of objects is important not only in lowering search times but for withstanding churn. Through the use of replicas, objects can still remain accessible should a particular node fail or leave the overlay. The question of when, where and how many replicas is important for the scalability, resilience and feasibility of any distributed overlay.

Structured overlays provide a reliable data retrieval by distributing a relatively simple hash data structure. There are a number of different implementations of DHTs that have been explored in this chapter, however all of them boil down to providing *key,value* lookups.

By having a rigid structure in which nodes are organised, it can allow for assumptions to be made in nodes' routing table that results in minimising the resource demands placed on nodes and shortcuts to decrease message routing time.

The ability for DHTs to cope with churn is discussed and the issue of replication has been proposed as a way of mitigating the negative performance effects of churn. It could be argued that by having a rigid structure, DHTs are more susceptible to the effects of churn, however section 2.3.1 shows that with an effective replication strategy, performance recovery times during churn can be low.

As this chapter has shown, structured overlays should be considered when looking at P2P systems. They provide a means of distributing resources and can, with some tweaks, provide low latency node lookup. Further work on structured overlays is presented in Chapter 3.

2.4 Anycast

Anycast is a service that routes messages to a node in a group best matched by a predefined metric and can be implemented at both the IP and application level.

Anycast allows one network address to represent multiple nodes on a network, which are grouped typically by the service they offer. For example, Web sites offering news on sports events could be placed into an Anycast group. DNS [113] has incorporated anycasting as a means to distribute load and maintain resilience.

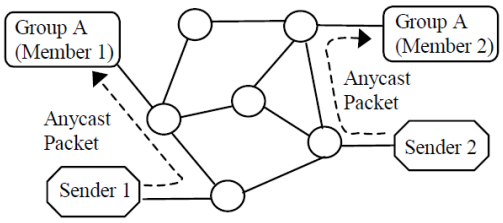


Figure 2.6: IP anycasting

IP level anycasting sends the Anycast packet to the closest member of the Anycast group as defined at the IP level, for example hops. Figure 2.6 taken from Katabi [114], illustrates that even though both senders send packets to the same Anycast group, the packet gets routed to the server which is closest to the sender. Anycast has long been suggested as a means of efficient service discovery [115], load balancing [116], [117], voice over IP networks [118], CDNs [119] and for use in QoS scenarios [120], [121] including unstructured networks [122], yet due to limitations it has yet to achieve widespread deployment.

According to Castro et al. [123], Anycast is a "*powerful building block*" in P2P overlays. The system can be used as part of a resource discovery scheme to manage hardware or software resources such as processor usage, storage, bandwidth and files. However there exists a number of challenges when adapting Anycast for use in P2P overlays. Due to the variation in group size and the dynamic membership found in these overlays, the maintenance of routing tables becomes an important aspect to the efficiency of an Anycast system.

The motivation behind implementing Anycast is large despite its associated costs, because of the favourable results it has shown. Analysis of results are presented in Section 2.4.2, however results presented in Battacharjee et al. [124] clearly show that as the percentage of nodes utilise

Anycast to select servers as opposed to random selection, the average response time to the servers recorded across the overlay decreases.

2.4.1 Anycast limitations

Anycast was proposed by Partridge et al. [125] in 1993, yet aside from root DNS servers and AS-112 servers, where there has been recent work to improve security through Anycast development [126], there has been few major deployments due to limitations which have hindered its uptake.

IP level Anycast has problems scaling when group numbers increase. Each group requires a specific route and they cannot be aggregated, similar to IP multicast, which too has had less than expected deployment in production environments. This causes tables to be excessively large and in routers which already have memory utilisation, it is seen as a tremendous operational problem. Another limitation is the use of IP routing as a metric for host selection. Other metrics such as server load are not taken into account. General barriers to entry include routers needing support for Anycast addressing and interaction between routers to ensure that only one host gets the packet.

Furthermore, IP level Anycast requires some part of the IPv4 address space to contain Anycast addresses, so for example a /28 from a /24 would be allocated Anycast addresses. It is addressing which also leads to scalability issues due to Anycast not being able to aggregate routes like unicast though the use of classless interdomain routing (CIDR), every unicast address must be broadcast separately. This in turn causes the size of routing tables to grow exponentially with the number of Anycast groups.

Work has been presented on making IP level Anycast viable through managing the routing of Anycast groups. The classification of groups as being "*local*" and their frequency of use allows Katabi et al. [114] to cache popular routes and not waste resources on rarely used routes. The premise is that from a certain location certain services simply are not going to see the same level of usage. For instance users of a UK university network may access *news.bbc.co.uk* far more regularly than *peruviantimes.com* and therefore equal resources should not be placed for the routing to both those locations on the Internet.

While network layer Anycast has in the past proved difficult to sell, application layer anycasting offers many of the benefits with few of the drawbacks. Nevertheless there are still challenges

in application layer Anycast such as efficient group membership, especially in high churn environments such as P2P networks. Even when considering application level anycasting, there are associated costs. The solutions detailed in Section 2.4.2 have costs attached to them, such as network messages and system load. The goal is to mitigate the cost by improving the performance benefits.

2.4.2 Proposed Anycast solutions

Limitations with IP level Anycast have provided the motivation for application level Anycast. Operating at a higher level, application level Anycast overcomes some of the shortcomings and allows for much easier large scale deployment. Bhattacharjee et al. [127] suggest that while the network layer is good at finding the shortest path (hops), the application layer allows selection on a wider range of metrics, opening up opportunities to use more fine grain heuristics in service discovery.

Proposals to utilise Anycast addressing similar to the widely used unicast addressing scheme include Anycast Domain Names [128] and PIAS [129]. Such services utilise domain names similar to those used in unicast addressing and resolve them to IP addresses.

In order to overcome the inefficiencies of IP address usage in IP level anycasting mentioned in Section 2.4.1, Ballani [129] propose PIAS, an Anycast architecture using proxies that tunnel Anycast packets to hosts in groups represented by their unicast addresses. PIAS combines IP level Anycast - the proxies can be contacted by native IP Anycast - and tunnelling to providing not only IP address identification for a group but IP:PORT identification, allowing thousands of Anycast groups to be represented by a single IP address, an important feature with the growing usage of NATs. PIAS doesn't solve the address scalability problem as such, it moves it from the IP level to the overlay however Ballani argues that this is much easier to deal with.

Zegura et al. [128] propose the Anycast resolution service, which returns an IP address when supplied with a Anycast domain name. It applies a filter to control the selection, even on an empty set, with the returned set can then have another filter applied to it by the original client. This could be a proximity filter or just a random selection from the returned set.

The resolver maintains a database of metrics for every server, which could include system or network level performance such as server CPU load or network throughput, Bhattacharjee et al. [124] term these as "*metrics of interest*". Clearly the accuracy of these metrics has a large

impact on the effectiveness of such a system and maintaining such a database is not a trivial task. Anycast domain names are utilised in work presented by Bhattacharjee.

Techniques proposed for measuring performance include probing, server push, parsing server logs and reputation. All these methods have positive and negative points with some requiring modification of servers, and not all metrics being measured or high variance in accuracy.

Bhattacharjee et al. [124] propose that any metric collection scheme should have the following properties

- Metric updates should only occur when there are changes to that metric. This is done through the monitoring/polling of the metric
- Nodes should be able to control the monitoring/polling according to system load
- Multiple domain resolvers to achieve load balancing, with a suitable selection scheme
- A balance between optimal returned result and overhead of maintaining the resolver's routing table must to be maintained
- Minimal modifications to underlying infrastructure

The collection of metrics is further complicated due to transparent web proxies, NATs, firewalls and other '*middle boxes*' which can distort performance figures. The acquisition of data can in itself affect performance as nodes have to deal with not only serving requests but reporting particular performance data to resolvers. One should also note that when concerned with deployment among P2P nodes, a reasonable assumption would be that the majority of nodes are standard, general purpose computers, not servers. This brings up the possibility, as demonstrated by Freedman et al. [130] of intrusion detection systems or firewalls causing probe messages to be answered incorrectly.

The selection of a node based solely on performance in a particular metric can lead to uneven load balancing. As nodes are being served by a single server node in turn that node may notice a deterioration in performance. The "*best*" server at any one time may actually be the best of an overloaded Anycast group, where all the other group members have had their performance levels compromised by servicing requests. To mitigate this effect Bhattacharjee propose an algorithm called "*equivalent servers*".

Equivalent servers are a subset of replicated servers which are within a performance threshold of the highest performing node. The Anycast resolver randomly picks a node from that subset. The threshold is a system parameter, though determining a suitable value represents a further challenge as too low may result in few, if any, nodes being part of the equivalent server subset and too high may include nodes which do not posses the performance characteristics required.

Probing or polling the server constitutes of sending the node legitimate requests and recording the time taken for a response. The probe client then acts as a judge of performance for clients within a certain distance of the probe. The further the node is on the network overlay, the probe's accuracy deteriorates. The problem associated with probing include heavy load on networks and nodes. As probing occurs regardless of server and network load, it can contribute to worsening results.

Work by Balakrishnan [131] shows results from clients (or probes) remain similar in the timescale of several minutes and are similar for nodes which are close-by. Further results published by Battacharjee [124] found such an anycasting metric collection technique resulted in lower response times to the servers returned when compared against *nearest server* and random selection. The standard deviation of the response times were lowest when utilising this Anycast method, by an order of magnitude when compared against random server selection.

Perhaps the most impressive aspect of the results in Battacharjee [124] is that anycasting using the probing method offers significant performance advantages over shortest number of hops, referred to as "*nearest server*". The results puts forward the notion that while fewer hops may instinctively seem a good metric to base latency assumptions upon, the actual outcome, in some cases could be very different. One reason for this could be geographical boundaries such as oceans. For instance it could be possible that from London to New York is one hop however London to Frankfurt could be four hops yet due to the significantly shorter distance, irrespective of the three extra hops to reach the server in Frankfurt, it may be significantly lower latency.

Server pushing proposed by Gwertzman [132] and log parsing are similar in that the server records its own performance. With 'pushing' the server sends data out to the resolvers when there is a major change. Unlike probing, server push requires modifications on the server to enable the pushing of data, however push techniques are unable to provide round trip times as the data is only travelling towards the resolver.

Server pushing can mitigate the problem found in probing of contributing to server and network overload. The server can schedule monitoring and pushing of data during non-peak periods. The downside to such a policy is that the latest data on servers under heavy load will not be available in the resolver database and therefore the resolver can provide less than optimal results to the requestor.

Experiments conducted by Bhattacharjee [124] found higher push frequencies do help in lowering the average response time of selected servers. This means that the quality of the result attained through an Anycast query is higher. However a stagnation of performance improvement was observed as the push frequency reached six times per minute. The experiment also varied the probe frequency and recorded that higher frequency resulted in higher performance.

It is the combination of the lightweight but less accurate server pushing technique combined with the more expensive real-time probing method which Zegura [128] proposes as a means of collecting metrics from servers. The Push-Probe technique consists of less frequent probes while the server pushing of metrics is bound so as to provide a forecast of the amount of network load with the aim being for the server not to send updates unless a major change has occurred. Results obtained by Zegura shows that the response times obtained through the push-probe technique is relatively accurate to those done through constant polling.

Log parsing is both similar to probing and server push. The server monitors and logs performance allowing for remote probing nodes to parse the log files. This differs from probing in that the server isn't presented with a live request and the parsing, which can be computationally heavy, is left to the probe node. However monitoring and recording activities still take place on the server and therefore a less than trivial load still exists. As many services already maintain such logs (web servers log usage for third party statistics programs) this may already be an acceptable cost for some users.

A heuristic based on user experience or reputation has also been proposed. The primary advantage of this is that it places no load on the server, probe or network. By utilising multiple clients and sharing information among them the quality of the information can be improved and accuracy increased.

Stoica et al. [133] propose a "*rendezvous*" system onto which Anycast services can be mapped. This system aims to break the point-to-point communication paradigm that exists by tagging packets which are then sent to the receivers which have triggers that match the ID of the sender's packet. For Anycast, the k most significant bits in both the packet identifier and the trigger must be identical and the least significant bits on the trigger matches the least significant bits of the identifier. The value of k is set so that the probability of the packet being delivered to an incorrect receiver is low. The k most significant bits represent the Anycast group identifier.

Anycast performance in overlays with dynamic membership was also recorded by Bhattacharjee. The system, which utilises push and probe/pull the system featuring *equivalent servers* to avoid oscillation, favoured poorly when the leave threshold increased. To understand what this means, it is important to understand how the equivalent server list works. The list of servers tries to balance queries so that queries are not repeatedly sent to a single server, which may lower its performance.

The join and leave threshold, which was varied in experiments conducted by Bhattacharjee [124] is the threshold at which nodes are inserted and removed into the equivalent server list. Therefore setting the threshold too high would result in more nodes being put into the list sooner, however it would be harder to get removed from it, resulting in stale nodes being returned when queried. A low threshold would be the reverse, though that could result in equivalent server lists emptying under certain circumstances.

Therefore it is little surprise that a high leave threshold would cause a decrease in performance when keeping the join threshold constant. Performance when varying the join threshold was less consistent in Bhattacharjee's results though the majority of recordings taken show that a threshold of 0.1 produced the lowest average response time. The author shows a trace with equal join and leave thresholds to present the occurrence of server oscillation, suggesting to increasing the threshold, therefore incorporating more servers into the list while dropping servers quicker. Again it is important to note that this is a balance, so the question is how to attain a value for the join and leave thresholds in a particular system?

Bhattacharjee did not propose a method for determining the join and leave thresholds in a particular system in [124], however there was clear evidence to support that setting too low a threshold would result in poor load distribution. Thresholds could be set based on an as-

sumption of the number of servers, the average load they can contend with without perceptible performance degradation. For certain types of overlays where it can be safely assumed that the nodes are machines with considerable hardware and connectivity, such as desktops on a broadband connection, the threshold is likely to be far higher than for mobile phones on wireless connectivity.

In further experiments conducted by Bhattacharjee [124] where the threshold was varied, it was shown both join and leave thresholds had an effect on average response times. It was also shown that server load was more evenly spread when the join and leave thresholds were increased. After the leave threshold increases above 1.6, there is no performance gain recorded due to server selection becoming a random selection as no server would leave the equivalent server group.

Having to set the join and leave thresholds could be seen as a problem for the equivalent servers technique. However combining it with a monitoring system that has knowledge of loads on nodes in the equiv lent server list, could lead to a flexible system that adopts to the prevailing usage conditions.

Anycast services, whether implemented at the IP or the application level, allows for intelligent server or node selection based on a set of metrics. The grouping of nodes based on a common characteristic, such as their content and ranked by some metric can provide the basis of an intelligent heuristic based node selection process.

2.4.3 N-casting

Anycast sends messages to a single member of an Anycast group, however there may be instances when a user wants to notify a larger subset of the Anycast group. Routing messages to more than one node in an Anycast group is typically called *n*-casting or manycasting.

N-casting has been described by Carter et al. [134] a something that "*fills the spectrum of network communication space between Anycast and multicast*". Other work in the area of sending packets to multiple group members have been classed as K-Anycast [135], [136] and [137]. The ability to send to multiple members of the group is particularly inviting as it allows multicast-esque abilities with the notion that a particular subset of nodes meeting a selection criteria will be contacted. For example if a node wants to receive concurrent streams of the same video from similarly performing nodes in order to build up layers of the video, or to

replicate data on servers with particular storage characteristics.

Castro et al. [123] implemented an N-casting system utilising Scribe [138], an overlay built using the Pastry DHT [5]. For standard anycasting where each group has been allocated a *groupID*, a node sends its message which has the groupID as the key. The message is routed to the node which is acting as the root of the group tree (Pastry organises nodes in a tree structure). At each hop the node handling the message checks whether it is a member of the group which has been referenced in the message. It there is a match, then the message is not forwarded, rather a depth-first search of the group tree is initiated. In the case of Anycast, the message would stop routing after a single group member had handled it, while in the case of N-casting, the message would continue, keeping a count of how many nodes within the group had handled the message, until the desired number of nodes had been visited.

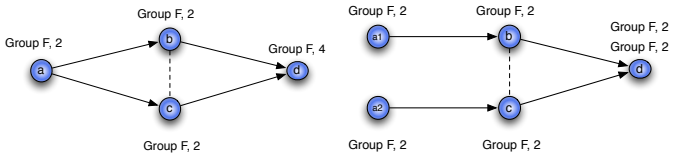
Other methods to enable N-casting include Beaconing as proposed by Kommareddy et al. [139]. They propose a system where reference points, known as beacons, are assigned throughout the network. Each group has a beacon, with all nodes within that group periodically polling its distance from it and the beacon, reporting this distance to the beacon. The beacon acts as a repository of distances, which is queried by external nodes when it wishes to find the closest node in a group.

When a query is issued the querying node measures its distance between itself and the beacon and the beacon returns a list of nodes that are within a distance parameter, for example, hops. This procedure of polling beacons for nodes within a pre-determined distance is repeated, with the query node performing an intersection of the node lists it has accumulated. As more beacons are queried the intersection reduces and the node stops querying beacons until the numbers of nodes after the intersection is down to a reasonable amount, or it has queried all the beacons. The node then chooses a closest node from that list. It is possible, with modification that such a system could be used to send messages to more than one node by picking the X closest nodes from the final list, to facilitate N-casting.

However, Beaconing has potential problems in distributed overlays. By assuming a deterministic approach to the location of beacons, performance could deteriorate should there be churn in two ways. The deterministic approach assumes the beacons are connected to a stable overlay, ensuring that both the beacons and the node membership data they contain are

accurate. However in experiments carried out by Kommareddy et al., beaconing showed significant performance degradation when tested on the Internet. Results presented show 25 per cent of runs had more than 10 percent accuracy error with 10 percent of runs having close to, or greater than 100 percent accuracy error. So, the assumption that the beacon itself will be alive at the time of query and the data is accurate has an effect on the performance of beaconing.

N-casting introduces a new problem to Anycast, the accurate knowledge of group membership. It is possible that as nodes report the membership of Anycast groups, the receiving node attains duplicates and therefore has an '*inflated*' view of group membership. While this problem exists is standard Anycast, as the querying node is simply going to select a single host, rather than query n_t nodes. The result of this could be that a node requests more nodes than exists in a particular group.



(a) Duplication (b) Non aggregation

Figure 2.7: Aggregation and non-aggregation of group membership

Figure 2.7(a) shows the problem faced as node n_d receives membership of Group g from nodes n_b and node n_c. The issue of duplication can occur for two reasons, node n_d is unaware of membership data beyond one hop and that membership data is aggregated. Nodes are typically unaware of membership beyond the previous hop because should a node have to transfer the full history of membership on each hop, the size of data transferred would become prohibitive. Similarly, aggregation of groups is done for this reason as the number of groups increase without some means of data compression the transfer of membership lists would grow exponentially with the number of groups and result in an unscalable overlay.

In the case of Figure 2.7(a), node n_d will not be able to discern that node B and C are sending it the same nodes from g. Therefore it will count g's membership as four and could then report that to other nodes, whereby the same process would occur, resulting in membership of g ballooning. Ultimately the lack of integrity in group membership figures would manifest itself when a node tries to send queries to more nodes than exists in a group.

If no aggregation takes place, node n_d would know the membership figures of Group g being reported by n_b and node n_c were of two different nodes as shown on Figure 2.7(b). n_d then has an accurate view of g, with four nodes, not the same two that has been counted twice. This is possible only if there is no aggregation of group membership data but that each node on the overlay has more than just a view of the previous hop. Both potential solutions result in a significant data transfer on the overlay and would not scale as the number of nodes and groups increase.

The problem of having accurate group membership is further complicated by the notion of churn (Section 2.3). Although nodes could poll group membership figures, this would result in an increase in messages being sent and represent a waste of resources every time there was no change in group membership. A balance of nodes announcing their arrival in a group such as through server pushing and global announcements would be required in order to keep the number of messages nodes have to forward to a reasonable level in large Anycast groups.

The issue of mitigating double counting in manycasting overlays is investigated in Chapter 5, where a data structure is proposed that ensures that accurate group membership is conveyed without the need for complete overlay knowledge.

Anycasting provides a routing mechanism that takes into account particular performance characteristics that make it particularly useful in QoS enabled P2P overlays. While deploying Anycast at the network layer has largely been unsuccessful, at the application layer it is possible to look at ways of aggregating data such that storage and transfer of routing tables does not affect the scalability of the overlay.

Work on creating locality aware anycasting in presented in Chapters 3, 4 and 5. Through work presented on those chapters, it is shown that Anycast and N-casting is a viable, efficient and useful routing scheme used in both structured and unstructured overlays.

Chapter 3

Distributed Overlay Anycast Table

The Distributed Overlay Anycast Table (DOAT) is an overlay proposed to meet the challenges presented by low latency content distribution.

The DOAT is a structured overlay which implements application-layer Anycast. It facilitates the discovery of the closest host in a given group and the closest member in a given location in the delay space. DOAT is designed to optimise accuracy and query time, two characteristics that are vital for real-time applications such as Peerlive, a near-live P2P video distribution overlay. DOAT is fully decentralised, with distribution of computation, storage and bandwidth among most of the nodes on the overlay.

DOAT aims to return the closest member of a particular Anycast group and can be used in applications such as content distribution networks (CDN), near-live video streaming, virtual worlds and peer-to-peer service discovery. In recent years effort has been placed on this area with published systems [92], [93], [140] on video streaming already deployed. It is clear that the advantages in P2P networks previously discussed can be leveraged to create cost effective, scalable CDNs. Calculations by Xiang [92] and Liu [141] shows traditional client-server CDN based architecture does not perform well in the task of large scale multimedia distribution. It is also argued that the edge servers used by CDNs cannot always cope with the phenomenon known as flash crowds.

Flash crowds are a sudden influx of users requesting the piece of information or data. This is typically found with news websites when world events occur such as the 11 September 2001 terrorist attacks or the football World Cup Final. Managing flash crowds is a lesson in scalability and in the client-server paradigm over-provisioning is typically used to overcome this problem, however inefficient this may be. However P2P on its own will not suffice when

requiring strict QoS. Critical support systems such as a scheme in which nodes would be encouraged to provide their bandwidth for the system are also required.

In the case of the video streaming use-case, each DOAT group could be a single stream. The streams could be a television channel such as *BBC 1*, a commercial movie or a home video. The first node on a particular stream will typically be the content's publisher. This may or may not be a high-bandwidth node however it is there as the initial offering of that stream to the network. As other nodes latch onto the stream, the job of the producer is typically one of being closest to the production of the stream, it is after all the node which is producing it. However, it may not be the node which all other nodes connect to.

DOAT's role in such a system is to provide a list of nodes present in a particular stream which are geographically closest to the querying node, and in traditional Anycast terms, each stream is an Anycast group. It manages this through a system of *Local Trackers* (LTs) that are aware of nodes subscribing to a particular stream within a local area. Using an abstraction, described in detail later, the DOAT system assigns nodes to a part of the overlay that corresponds to their delay. Each stream has its own LTs which keeps track of node IP addresses and their co-ordinates. LTs are nodes which are given the extra work of providing a list of nodes which are geographically close to the querying node.

The LTs are the first point of contact for nodes joining the system and therefore the performance of LTs in providing a fast lookup service for new DOAT nodes is vital in lower the start-up time. For this reason the assigning of LTs is important, however due to the size of local areas varying, the number of nodes being serviced by a single LT can be controlled. While LTs are the linchpin of the DOAT system each node is vital in the initial discovery of an LT. The following sections will describe the DOAT system.

The choice of node is based on its stability which is key in the DOAT system. DOAT nodes have to maintain Bloom filters to retain group membership. Bloom filter size concerns are mitigated by the selection of stable overlay nodes to become DOAT nodes with a similar set of characteristics used for the choosing of new LTs. As presented in earlier chapters, previous work has shown that nodes selected on the basis of past uptime provides a reliable indication of future uptime.

Nodes that join a system that uses DOAT will need to be served with the IP address of the closest LT of the stream they are wishing to watch. The quicker this query is, the lower the start-up delay is.

It is important to realise what DOAT is not. DOAT provides a method of node discovery and message routing, it plays no part in the final selection of nodes by clients. Selection of nodes depends on other factors within the underlying system such as their available upload bandwidth, network delay, stability, trust and past performance. Some of these characteristics are entirely separate modules within the underlying system and are out of the scope of this research.

3.1 Design considerations for DOAT

Aside from meeting the overall objectives outlined in the introductory section, the goal of DOAT is to efficiently marry a node discovery service harnessing the underlying topology of the Internet, in many ways take the advantages of Anycast and present them in a scalable manner. As described in Section 2.4.1, network layer Anycast adoption has been hampered by the cost placed on edge routers, so with DOAT that cost is removed from routers and placed on the hosts. The overlay network is simply used for node discovery not routing. Application level Anycast allows for other metrics to be considered with making routing choices, metrics which cannot be measured at the router level.

DOAT can, with the aid of ISP powered Non-Consuming Peers (NCPs), increase the amount of intra-AS traffic. Even without the use of NCPs, due to the local aware nature of DOAT we would expect to see a reduction of inter AS traffic from the nodes that are provided by an LT. Aside from the use of NCPs the overlay construction is vital in creating a topologically aware overlay and for that DOAT utilises space-filling curves in order to translate multi-dimensional coordinates into easily manageable figures.

Experiments on DOAT were conducted through a discrete event simulator programmed in C. The simulator sequentially adds nodes to the DOAT overlay following a uniform random distribution in the Euclidian space as presented in Figure 4.3. The input file is a list of nodes' coordinates and latencies from every other node and is read sequentially to create the DOAT overlay.

As the simulator introduced a new node to the DOAT overlay it went through the joining pro-

tocol as would be the case in a real-world environment. While the newly joining node's routing tables are populated, the routing tables of nodes already present in the overlay are modified with the new information.

During the creation of the overlay, the simulator records the number of messages being sent. Once the overlay has been created randomly generated queries for group members are generated. For the purposes of simulation there is one Anycast group that members can join, with the percentage of node members in that group being varied in simulator runs.

Along with recording the number of messages sent, the total message propagation delay is recorded, the quality of the results returned, known as the accuracy. Metrics such as accuracy and end-to-end query delay is possible due to the simulator knowing the shortest theoretical path between two nodes on the DOAT overlay from the input file that has the coordinates of all nodes on the DOAT overlay.

The simulator's lifecycle starts at the overlay creation with a predetermined number of nodes in a single group as bound as an input to the simulator at 1,000. The nodes send messages to other overlay nodes as defined by the DOAT protocol, after which the simulator terminates.

3.2 Space-filling curves

DOAT makes use of space-filling curves to bring a notion of locality to the overlay. The performance of the space-filling curve is critical to the performance of the DOAT. The space-filling curve allows for a two-dimensional coordinate to be transformed into a single dimension allowing nodes to be placed on a specific point on the DOAT overlay.

The translation from network coordinate to DOAT value is simply the transformation of the curve. Initially DOAT was simulated using the well known Hilbert Curve [142] and then moving onto the H-Curve [143]. The problem with Hilbert Curves is that they do not wrap around, that is to say *0.9* is not close to *0.1* on the curve.

In some cases space-filling curves may introduce inaccuracy. In Xu et al. [96] the system tries to map the three-dimensional landmarking space into a single dimensional value using the Hilbert Curve. The authors found that by reducing the number of components in the landmarking vector got a more accurate value of distance.

For DOAT the H-Curve was chosen because it has good locality preserving properties. The distortion between multi-dimensional locality values such as the ones generated by Vivaldi and the corresponding H-Curve value is small compared to other space-filling curves.

3.3 Overlay space

DOAT manages nodes within an abstracted overlay space which maps nodes to their delay space. The aim is to make nodes query and transfer chunks with nodes which are close in the delay space.

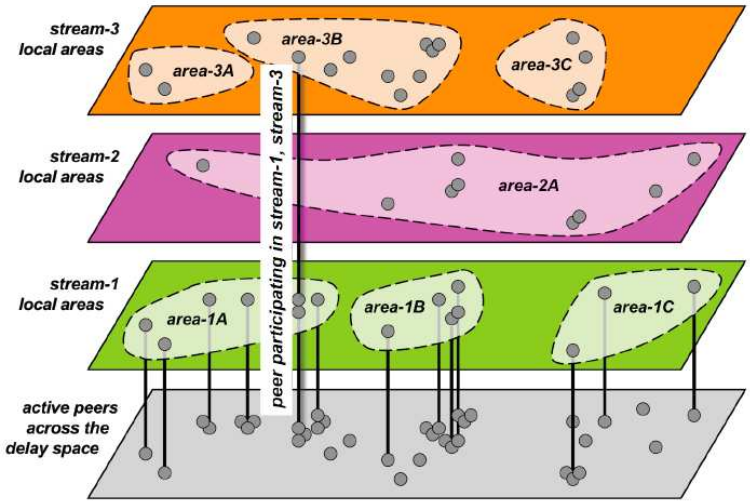


Figure 3.1: DOAT overlay

Figure 3.1 represents the DOAT overlay for three streams. In Figure 3.1, the delay space is the mapping of nodes' locations relative to their latencies from each other, which is used as an estimation of the geographical distance between each node. The underlying delay space is mapped onto each of the stream's local areas. As the number of nodes involved in a stream grows the number of local areas will grow. This allows load to be balanced even if a single local area covers a distance on the delay space which is small, for example, 100ms.

The practical motivation for this is within regions certain streams may be more popular than others. This may be particularly true for regional television channels and national sporting events. In these cases nodes would be clustered with a low standard deviation in delay however having one local area would mean the LT being overloaded. This is important because LTs are standard DOAT nodes it is vital that they are not viewed as servers with over provisioned connectivity and hardware resources. Situations may arise when nodes join the overlay and are only

provided information of nodes that are belonging to their LT. This would lead to disconnected regions and suboptimal connections.

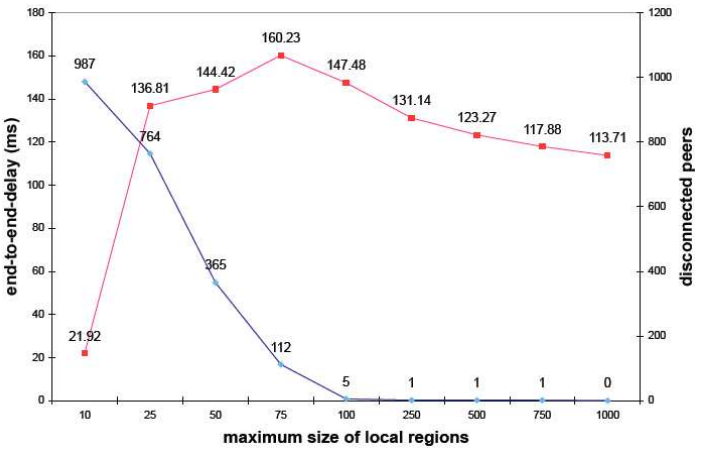


Figure 3.2: DOAT local region size

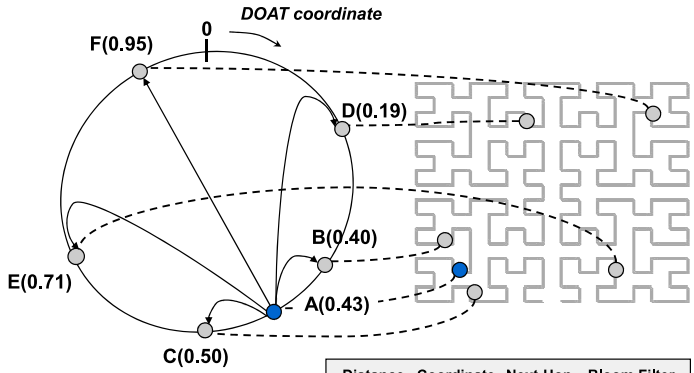
Figure 3.2 displays the relationship between the size of local regions, in nodes and the effect it has on end-to-end delay and the number of disconnected nodes. As the size of local regions increase the number of disconnected nodes quickly falls. Aside from the initial jump in end-to-end delay between *10* and *25*, size increase causes a steady decrease in delay. The reason for the increase in end-to-end delay when the size of local regions is increased from 10 is due to an increase in neighbours that are geographically further away as in the case of this simulation 'local neighbours' - those physically close to the node - had been exhausted.

The conclusions that can be drawn from Figure 3.2 is suffocating a local area with too few nodes will lead to the available capacity of LTs being exhausted. The upshot of this is growing numbers of nodes which join after the limit has been reached being shunted into local areas which are disconnected. When local areas are 100, there is acceptable end-to-end delay and a very low number of disconnected nodes. This result also provides motivation for the long-range nodes which are kept in the routing table of each DOAT node which provide inter-region links.

3.4 Protocol

The process of joining the DOAT is primarily a process by which a node's network coordinates are translated into a single dimension. The motivation behind this is that nodes which have values that are close on a space-filling curve is also close in the delay space. The job of the space-filling curve is to transform the latitude-longitude coordinates (the place of a node on a World map) to a single value. In reality the node can transform a synthetic co-ordinate which

may not have any relation to its latitude or longitude, into a single-dimensional number, therefore the need for geographically accurate co-ordinates is vital for the efficient functioning of DOAT.



Distance	Coordinate	Next-Hop	Bloom Filter
0.00	0.43	A	BF.local
0.03	0.40	B	BF.B
0.07	0.50	C	BF.C
0.24	0.19	D	BF.D
0.28	0.71	E	BF.E
0.48	0.95	F	BF.F

Figure 3.3: The DOAT address space

For DOAT the use of space-filling curves is to facilitate a lightweight method of performing landmarking on nodes. Nodes joining the DOAT first determine their position using a distributed coordinate system such as Vivaldi [59]. This gives a multi-dimensional delay space value. Using a space-filling curve will transform this figure into a single-dimensional unit which can be used for routing and query forwarding.

The property of this single dimension coordinate is distances between two coordinates *d* and *d'*, in the single-dimension space *a*, has a similar distance between the corresponding two nodes in the multi-dimensional space a', given a small constant. Formally, $a(d, d') < ka'(d, d')$ for some small value of k. It should be noted that the transformation is not accurate in the opposite direction, that is to say two nodes which are close on the multi-dimension overlay may not be close on the single-dimension DOAT overlay. This one time calculation requires low processing and memory resources (the function consists of less than 40 lines of code) after which no further information is required or stored.

After the DOAT node is provided with its H-curve value, the ring overlay, similar to that of Chord [4] is used to position nodes within the network. It should be noted that the overlay distance between a node at *0.9* and one at *0.0* is 0.1 and that all distances are measured clockwise.

A node starts building its routing table by contacting a node on the overlay it knows about. As a predetermined system design decision, a DOAT node will know of its neighbouring nodes in clockwise and anti-clockwise positions on the ring. The node is given the information of another node during the bootstrap process by an external service, which could be another distributed overlay or a centralised service. The node will attempt to contact nodes at logarithmically farther distances on the ring, as shown in Figure 3.3.

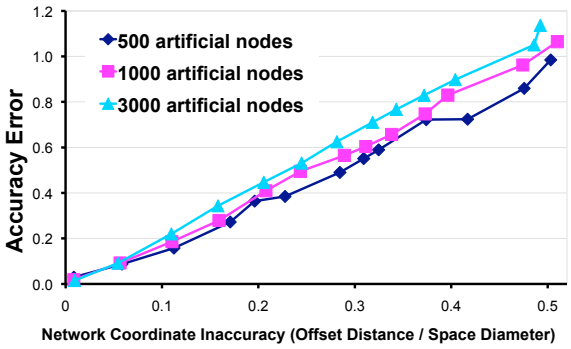


Figure 3.4: The effect of coordinate accuracy

DOAT nodes may have to rejoin the DOAT if they experience network coordinate drift [144] or because of changes in the underlying network performance. In the case of coordinate drift, the rejoin procedure is undertaken to get a more accurate reading of where the node is and increase the accuracy of the closest group member selection. The effect of inaccurate coordinates is shown on Figure 3.4 which shows a near linear relationship between the DOAT returning the closest node and the offset coordinate value.

Each DOAT node maintains a routing table displayed in Figure 3.3 which contains one entry for the local registry and one entry for each of its neighbours. Each entry includes the DOAT coordinate and the distance of the node from itself. The group identifiers are aggregated using Bloom filters in order to keep maintenance costs at a minimum. A trade-off between the size of the Bloom filter and the probability of false positives has to be made.

A node is aware of its neighbours by following the routes built to discover Anycast group members with the difference being that the destination is an explicit DOAT coordinate not a group identifier. The routing table of each node contains entries of these nodes with their H-curve value, distance from itself and a Bloom filter [145] of the streams that node is a member of.

The number of entries in a DOAT node routing table is a trade-off between routing speed and maintenance costs as seen in other structured overlay systems [103]. As with networks such as OneHop [146] routing knowledge can be used to decrease the number of hops a query needs to travel at significant cost to maintenance and scalability, with the number of entries being a system parameter.

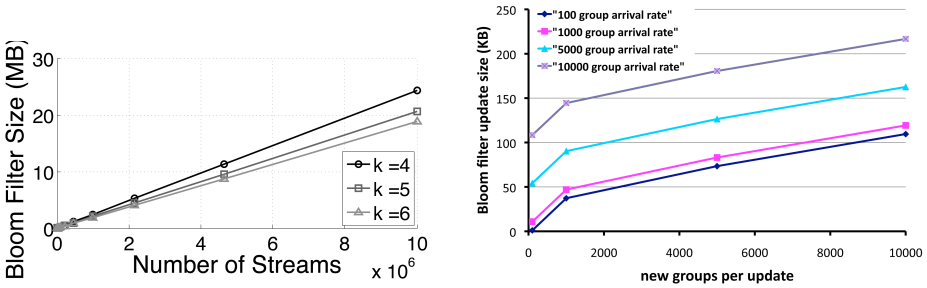
Knowing *'far away'* nodes is important as it provides a routing shortcut to those nodes rather than having neighbours repeatedly contacting their neighbours to forward queries. The principles behind this is borrowed from the Chord DHT routing system as described earlier and simulations carried out show that while it does increase the memory usage at a node by a small amount, it significantly reduces the hop-count to those distant nodes.

In a global scale system nodes may not be distributed uniformly across the delay space, that is to say the same number of nodes which are present in Central Europe or North America may not be the same as in central Africa or Northern Russia. However, taking a real world video streaming scenario, it could be possible that a family in Australia wants to watch the Ashes cricket taking place in Lords, London where the distance in geographical and delay space is large. While the latency of nodes within a small radius of itself will provide low latency and possibly high throughput, they would not be able to receive the stream if long distance links were not present.

Aside from preventing the creation of a rich club of geographically close nodes, distant nodes also provide a measure of resilience should a region of the underlying network be disrupted. By creating a topologically aware overlay we increase the exposure to such failures. Therefore by keeping knowledge of nodes that we know are on the *'other side of the world'*, a measure of resilience can be restored should region dependant churn occur.

The Bloom filter provides an efficient and accurate storage mechanism to store and verify set membership. A Bloom filter provides a mechanism for knowing whether an object is part of a set by applying a hashing function and checking whether any of the bits in the '*filter*' has been set to 1. If any one of the bits in the filter has been set to 1 then in the context of DOAT, the node is a member of the Anycast group. A documented problem [147], [148] with Bloom filters is the return of false positives however in the case of DOAT, a false positive will not yield a complete failure but rather a suboptimal LT node being returned. The trade-off between

Bloom filter size and efficiency is important as it is impractical for nodes to be transferring large amounts of data to simply maintain the foundations of content distribution system, let alone consume and sustain it.



(a) Bloom filter size (b) Bloom filter update size

Figure 3.5: Bloom filter and update size

Using work published by Broder [148], DOAT uses Bloom filters with four hash functions and a false positive probability of 0.001. The size of the filter would be between 10 and 20MB. This is illustrated in Figure 3.5(a), with the number of streams expected to be less than eight. The trade-off is an important one not only because of data transfer, the number of streams each DOAT node can be aware of is important in allowing for smaller local areas.

The Bloom filter aggregation requires the integration of full filters each time not just ones which include new LTs. As LT churn should be longer than DOAT routing update intervals, the differences in consecutively calculated Bloom filters are minimal.

Route announcements from n to one of its neighbours n‘ contains the groups present in the local registry of n and the other groups that n can reach through other nodes in the DOAT overlay. In essence the DOAT node is announcing all the Anycast groups which have member hosts in its local area to nodes which are further away. In addition to the Bloom filter, the routing update contains the immediately closest neighbour of the node in both directions. This allows a fallback should the current neighbours disgracefully disconnect from the overlay.

In order to lower maintenance costs, routing updates do not occur synchronously with updates. Instead, a minimum interval is set whereby updates are sent to a neighbour. To lower the amount of data being sent between nodes, the whole Bloom filter does not need to be sent between nodes. Instead only the difference is sent and compression occurs by sending only the

positions of the bits that need to be changed rather than the whole Bloom filter bit vector.

$$U \approx \log_2(m)(\beta(k, m, n + \rho_+) - \beta(k, m, n - \rho_-)) \tag{3.1}$$

$$\beta(k, m, n) = m \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right) \tag{3.2}$$

The size of the update is represented by equation 3.1, where ρ_+ is the number of groups added and ρ_- is number of groups decreased. The second function *β(k,m,n)* gives the average number of ones in a Bloom filter with *n* stored elements. The filter update size displayed in Figure 3.5(b) shows the Bloom filter updates scaling as the number of new groups increase. From 5000 to 10,000 new groups, the update size increases only around 50KB, less than 30%.

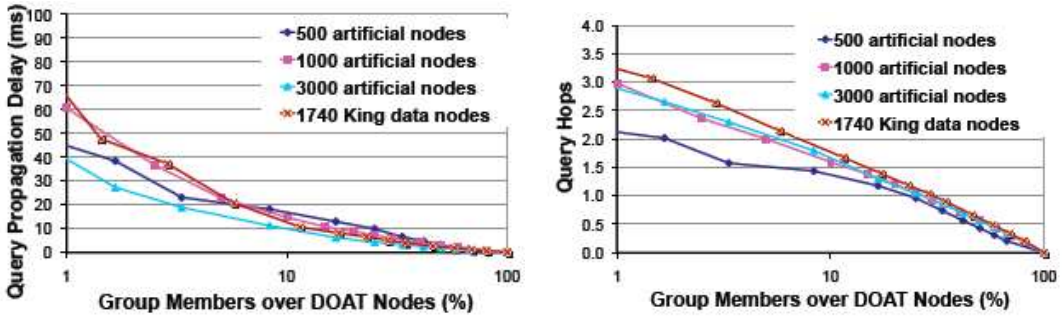
Basic routing follows that of the Chord overlay [4] as described in Section 2.2.3 with no modifications. Chord was chosen primarily due to its overlay structure, a ring and the state each node stores especially the knowledge of distant nodes.

3.5 Querying the DOAT

An overlay node queries a DOAT LT to find a group member. The node queries its local DOAT node with the stream identifier. It does this by sending a *REGISTER* message to the closest DOAT node it knows including its synthetic co-ordinate. The message is routed to the neighbour closest to the peer's co-ordinate. When the DOAT node receives messages, it sends a *REGISTER_ACK* message back to the original peer.

The DOAT node iterates over its routing table and forwards the query to the neighbour in the first routing entry where the Bloom filter yields a positive result for the stream identifier. Once the query reaches a DOAT node which returns a match on the stream identifier in its first entry, it has found an LT which is registered to the stream identifier. The query is finally passed onto the LT which then returns a list of nodes within the querying node's local area.

The results presented in Figure 3.6 plots query delay and query hops against the number of group members as a percentage of total DOAT nodes in the system. The artificial nodes are ones that are generated with coordinates from a random distribution, while the King dataset [149] is of nodes that have distances that are from real-world measurements. This was chosen because it shows how DOAT performs when the number of group members, not simply the



(a) Query delay

(b) Number of hops per query

Figure 3.6: DOAT query results

number of total overlay members, is varied as a node's query can only be served by group members and therefore the total overlay size is not neccessarily indicative of the DOAT membership.

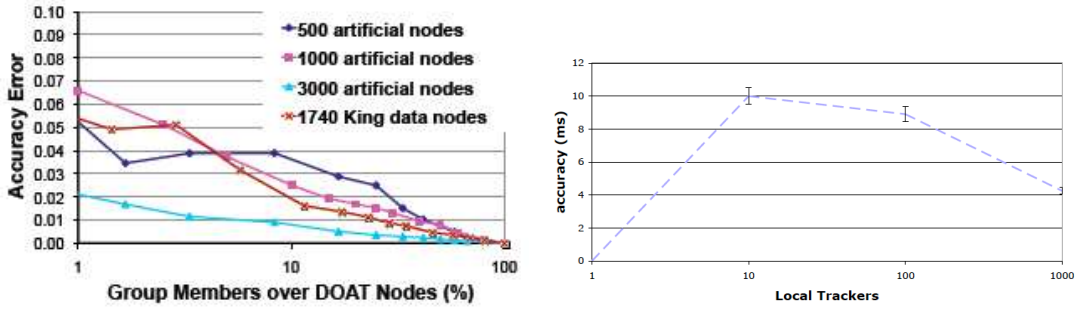
The average delay to discover nodes is shown in Figure 3.6(a) - DOAT query delay and hop count shows that average query delay to discover the closest member is below the average delay between nodes even for the smallest groups. The growth of member group subscribers leads to more nodes with smaller delays and each hop is *'shorter'* with less delay between them.

One of the key performance metrics for DOAT is the time it takes for this process to occur and was tested in simulations. Not only is overall delay time important but the difference in latency between the LT returned and the optimal LT, which is classed as the accuracy. DOAT strives to provide the optimal LT, that is to say, the closest LT to the querying node.

The delay decrease when having 10 LTs in a system of 1000 nodes is considerable however the same decrease is not observed when the ratio between nodes and LTs drops to 1:10. This is a favourable property to have as it shows that in the initial stages of a stream, where the publisher may be the only LT and the overlay has yet to hit the threshold for creating new LTs.

Figure 3.6(a) shows the scope of improvement that LTs bring in decreasing query time. The idea of having a single LT within a system is one that is rooted in the client-server paradigm and it shows that the DOAT system scales well with increased numbers of LTs given differing ratios of nodes to LTs.

As the number of nodes in a group increases the accuracy error decreases to the point at which it



(a) Accuracy as group membership increases

(b) Accuracy against number of local trackers

Figure 3.7: DOAT accuracy

is negligible. The errors introduced by converting a two-dimensional coordinate into a number by a space-filling curve becomes increasingly visible at this stage.

The accuracy of the LTs being returned increases well with the number of LTs. When there is only a single LT the accuracy is 100% simply because there are no others to choose from. However the decrease shows that even when there are many LTs to choose from, the choices are efficient. In all reported cases the LTs provided have a minimal effect on latency and the trend recorded is encouraging.

Most favourable of all results shown in Figure 3.7(b) is the accuracy increasing when there are more LTs present to a point. This shows that even though the number of choices that are available increases the nodes are on average being presented with an LT which has lower latency.

3.6 Conclusion

In results presented here, DOAT has shown that by using a space filling curve it has been possible to introduce the notion of locality awareness to a structured overlay. By using an efficient data structure, such as a Bloom filter, it was also shown that one of the major problems with Anycast, its inability to aggregate groups, has been mitigated.

Not only has DOAT shown good performance in overlays populated by ‘artificial’ nodes, but performance remained good when the real-world King dataset was used. Significant improvements are noted in query propagation and the increase in routing table size was very encouraging.

Results show that DOAT can provide scalable and accurate resource discovery using innovative features such as a space-filling curve and Bloom filters. The simulations presented offer a glimpse into the DOAT system and were presented in papers [150] and [151]. They also show its viability in a overlays that have stringent QoS requirements, such as video streaming. However it is important to note some caveats that come with these presented results.

One of the main limitations is the size of DOAT that was simulated. Due to restrictions on hardware at the time simulations of over 10,000 DOAT nodes were impossible. Currently network churn has not been implemented in the simulator. In order for any P2P overlay to be successful it is vital that simulations be carried out in an unstable overlay. Currently all nodes join at a single epoch with nodes not leaving the system during the simulation's lifetime. Although this gives us an accurate view of performance, it does so only under stable conditions which will not be present in a real-world network such as the Internet.

Future work to extend DOAT is outlined in Section 6.1.1, where the preliminary work on a hierarchical DOAT is presented. Further work on bringing locality awareness and efficient use of routing tables is presented in Chapters 4 and 5.

Chapter 4

Gossip Overlay Anycast Table

Gossip Overlay Anycast Table (GOAT) is an overlay by which a node is returned a set of other nodes in the overlay without the need for prior knowledge of the overlay. The proposed system aims to do away with the need for 'tracker' nodes to mitigate the effects of churn on the overlay's performance as outlined in Section 2.3.

Overlays such as DOAT, proposed in Chapter 3 utilise a deterministic method of building a node's routing table, the use of probabilistic methods allow for greater fairness as all nodes on the overlay would get an equal chance of being present in a node's routing table. The problem of routing bias could be particularly detrimental to a system in the presence of churn whether due to network or load conditions. To that end GOAT proposes a probabilistic system of populating routing tables.

The ultimate aim of GOAT is to supply a list of nodes on the overlay which are part of an Anycast group. To achieve that there are four macro steps, these are

- Neighbour acquisition
- Group membership
- Route propagation
- Querying

Neighbour acquisition focuses on building up a picture of the current overlay. Ideally there should be no prior knowledge, assumed or derived in order to maintain robustness. However there has to be a starting heuristic. For instance in BitTorrent systems, the starting heuristic is the list of trackers provided in the torrent file. As GOAT is presented as a decentralised system with the tracker not being a particular node for all the subscribers on the overlay, the joining

node needs to discover other nodes. This is further described later in this chapter, particularly in Section 4.1.

By not assuming a particular topology such as DOAT's ring (Chapter 3), the ability to build up a topology map on demand brings forward a new set of challenges. The time taken to build the topology map has to be considered, the number of messages that the process creates may distort the view of the overlay, how to create a fair and representative sample of the overlay ensuring an accurate picture and the mechanism to keep that map up-to-date with current overlay subscription.

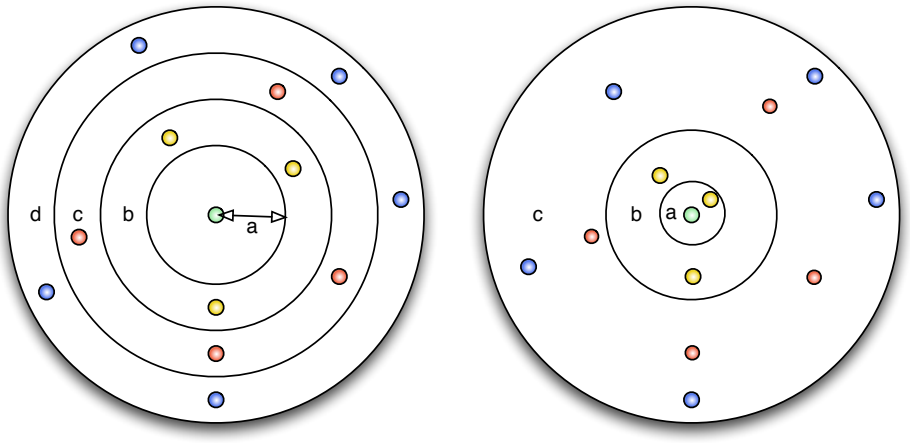
4.1 Overlay design

To gain knowledge of nodes currently on the overlay, a number of methods can be employed. One has to consider a number of issues with regards to neighbour acquisition such as the time taken and the number of messages, a factor discussed in the area of random walks in section 2.1.4. Both characteristics are vital in determining the efficiency of a particular algorithm, with the number of messages being particularly prevalent in an unstructured overlay where the number of messages to discover nodes or to maintain routing table correctness.

As Ripneau et al. [44] presented, nodes in the Gnutella network, a decentralised network, 2001 transferred 330TB/month just to maintain the overlay. Ripneau et al. had concerns as to whether Gnutella would be able to "*reach larger deployment*". The cost of maintaining the overlay needs to be considered in both neighbour acquisition, as it is envisaged in a P2P system that node membership will be changing. The first implementation had the notion of expanding rings by Lv et al. [11], which can place a distance range on nodes found.

In GOAT expanding rings have been implemented in two ways. The unit of distance a in Figure 4.1(a) is seconds, whereby nodes on the overlay are separated by their latency, used as a crude measurement of distance from the source node. The motivation behind this would be to be aware of nodes that are within a certain latency range, allowing for relatively predictable performance from those nodes.

The expanding ring heuristic is implemented with a fixed gap to the first ring, a, with subsequent rings having a range r which is dependant on the difference between the ring a and the furthest node f on the overlay. That distance is then divided by a pre-determined number, a system parameter p, which would set the number of rings in the system. Formally, the linear



(a) Uniform expanding rings (b) Logarithmic expanding rings

Figure 4.1: Uniform and logarithmic expanding rings

expanding ring heuristic is represented in Equation 4.1.

$$r = \left(\frac{f - a}{p} \right) \tag{4.1}$$

The problem posed by an expanding ring system with uniform distance between rings is the number of rings required to reach far away nodes. Given each ring has a minimum number of nodes will be selected, the routing table will not only have nodes from previous rings but also when it comes to routing the information, the incremental jumps, or shortcuts, will be smaller.

It should be noted that while in simulation the rise in routing table size could be mitigated by having knowledge of "*universe size*", that is not a realistic option for a real world system. By introducing logarithmic ring distances, not only are farther nodes reached in fewer hops but the amount of memory in each node's routing table is reduced, when taken as a factor of overlay coverage.

While the use of logarithmic expanding rings does not solve the problem of the number of rings r_n required to cover a universe the size of which of which is unknown, it does mitigate the issue of cost associated to overlay coverage.

$$r_n = 2\left(r_{n-1} - r_{n-2}\right) \tag{4.2}$$

Equation 4.2 formally describes the method for calculating the distance of a ring, logarithmically. The previous ring distance between, '$r_{n\text{-}1}$' and '$r_{n\text{-}2}$' is doubled. This is represented, not

to scale, in Figure 4.1(b).

Logarithmic expanding rings Figure 4.1(b) are able to reach the furthest nodes in the overlay in fewer hops than uniform expanding rings displayed in Figure 4.1(a). Rings which encompass greater distances also raise the possibility of a larger set of nodes from which *n* is chosen. This is an important factor in the efficiency of any expanding ring design as it influences the size of routing tables and routing efficiency.

While the effect of more rings resulting in more neighbours has an obvious effect on routing table sizes, a greater number of nodes between source and sink could result in a higher routing efficiency as messages are travelling shorter distances before being '*guided*' towards the final node. With large distances between nodes, a message could well be sent far off course before reaching the next node and being directed towards its destination.

Figures 4.1(a) and 4.1(b) show a series of nodes aligned at the six o'clock position. This was done to highlight a possible problem with a basic expanding ring algorithm. The problem could allow for '*tunnel vision*' when it comes to node neighbours. In a one dimensional universe this is not an issue, however in a more realistic two dimensional universe, especially in low membership overlays, it is a distinct possibility.

4.2 Protocol

Node *n* joining GOAT is presented with the address of another GOAT node. The service that provides this initial GOAT node information could be another distributed overlay such as DOAT or a DNS system. Once *n* has the address of another node in the GOAT overlay, it receives information on the nodes it knows of.

Once *n* receives the list of nodes it randomly picks n_x nodes to become neighbours, where *x* is a system parameter. Node *n* goes through the list of n_x nodes to find the furthest away node and divides this by the number of rings it will create, as shown in Equation 4.1. After this *n* sequentially processes the n_x nodes to calculate which ring they are put into and once a ring is full - the membership of each ring is a system parameter - the node then ignores all further nodes that fall in that ring in the subset of *x* nodes it has been sent.

For logarithmic expanding rings, when node *n* receives the set of n_x nodes, it proceeds to find

the furthest node in that list but then uses Equation 4.2 in order to calculate the range of each ring. Node *n* then proceeds to bin the nodes in each ring, following the same protocol as used with uniform expanding rings.

Nodes that have been placed into rings by *n* receive a *TRANSFER* message whereby they transfer their routing tables along with group membership data, to node *n* and those n_x nodes send a *JOINUPDATE* message to all the nodes in its routing table that are farther away from *n* than them with routing updates from node n_t.

In later experiments the protocol was tweaked to introduce a bias when placing nodes in rings based on distance. Whereas there would be a set number of nodes in each ring, a preset bias is introduced so that when *n* receives the list of nodes, it picks n_t nodes based on a function of its distance and the bias.

4.3 Experiments

GOAT was simulated in a discrete event simulator programmed in C. The simulator takes an input file that has node coordinates based on a dataset that has been embedded in a two-dimensional coordinate space. The simulator then creates the GOAT overlay based on the protocol proposed in Section 4.2.

The simulator sequentially adds nodes to the GOAT overlay, running through the node joining protocol each time in order to build the overlay. During the creation of the overlay, the simulator records the number of messages being sent. Once the overlay has been created randomly generated queries for group members are generated and the metrics mentioned in Section 4.3.1 is recorded. For the purposes of simulation there is one Anycast group that members can join, with the percentage of node members in that group being varied in simulator runs as it is the number of group members not the overlay membership that has an effect on the performance of GOAT.

Along with recording the number of messages sent, the total message propagation delay is recorded, the amount of the overlay a node has coverage of with the entries in its routing table and how much of the overlay space is replicated - meaning how inefficient is the neighbour acquisition algorithm. The routing table analysis occurs during a phase between the overlay creation and the query generation.

Metrics such as query routing efficiency are possible due to the simulator knowing the shortest theoretical path between two nodes on the GOAT overlay from the input file. The simulator stops once every node has randomly tried to select another node that is in the group.

In experiments the first innermost ring was set to be 10ms. The first ring is termed the node's local area and in Figure 4.1(a) is labelled as *a*. Nodes within area *a* are reachable within a single hop. The number of nodes a single node has knowledge of within each ring is set as a maximum of four. If there is fewer than four nodes, then all the nodes are selected.

4.3.1 Performance metrics

To evaluate the performance of GOAT, the following metrics are proposed

- Failed queries
- Accuracy error
- Query delay
- Query routing efficiency delay
- Routing update messages exchanged

Failed queries are defined as those that are sent to group members that do not exist. The node may have existed at some point in the overlay's lifetime however due to stale routing tables, the node's departure has not been reflected. This metric is essentially a worst case scenario whereby nodes have already propagated the query, meaning an increase in queries only having to drop the query after either a predetermined maximum query propagation time or number of hops. The result is a delay for the querying node and increases the load on the overlay load without resulting in a productive output. Naturally, the lower number of failed queries there is, the higher the system's performance.

Accuracy error *b* is the normalised difference in milliseconds between the returned result *r* and the closest possible result *c* over the diameter of the universe d_{univ}. This is to evaluate the node that is returned, with a lower figure denoting a smaller difference between the optimal result and the returned result. Formally,

$$b = \left(\frac{r - c}{d_{univ}} \right) \tag{4.3}$$

Query delay and its subset, query routing efficiency delay, measure the delay over the query path and the difference between the the query path and the absolute distance between the nodes, respectively. Also recorded is the query delay normalised of the '*world size*'. This is to remove the overall size of the world from affecting the result, formally the normalised query delay l_{norm} in a world size of d_{univ} is $l_{norm} = \frac{l}{d_{univ}}$.

Whereas query delay produces an absolute answer, it doesn't provide a comparative performance measurement. The query routing efficiency delay is a comparative measure against the optimal result. '*Routing efficiency*' is the term given to when a query doesn't move towards the destination host.

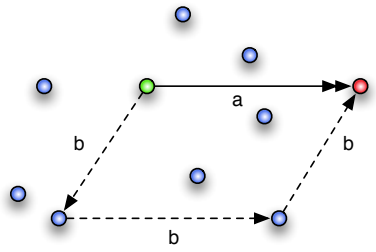


Figure 4.2: Query routing efficiency

The query paths in Figure 4.2 show an extreme example of query routing inefficiency. Path A represents the optimal result, with the distance *d* represents the absolute distance between the query originating node and the responder. Path B, the summation of *b* is clearly an inefficient path for the query. The query routing efficiency delay would be the difference, in milliseconds between path A and path B.

The absolute query efficiency delay does not provide a respective measure of performance, rather it is useful to record the ratio of query inefficiency to ascertain the severity of inefficient routing. This is the total query routing efficiency delay delay over the known optimal query delay, which is calculated from the dataset. In effect it shows what percentage of the query path consisted of inefficient routing. Formally,

$$\text{ratio of query efficiency} = \frac{\text{recorded delay}}{\text{known optimal query delay}} \tag{4.4}$$

Equation 4.4 describes how the metric of routing efficiency ratio is calculated. This metric provides some perspective with respect to query size, overlay membership and the efficiency of the system as group membership increases.

Given that the number of messages in an unstructured overlay has been its Achilles' Heel, it is vital that any such overlay should keep a tight rein on the number of messages being transferred. In the case of GOAT, the terms messages and route updates are used interchangeably. While an increase in overlay membership will inevitably lead to a rise in the number of update messages exchanged however the goal here is to mitigate the rate of increase. This in turn would make GOAT viable for large scale deployment and in environments where churn exists.

Routing updates are the table updates sent to nodes in order to maintain the overlay. A large number of routing updates not only mean that nodes have to deal with the actual query handling from nodes on the overlay but exchange routing updates with nodes present in its routing table. A large routing table also means increased memory and bandwidth consumption in storing and transferring the table an issue that was discussed in section 2.2.8. A trade-off has to be made as having large routing tables would mean an increase in routing update messages but lower the number of hops and therefore total messages on the overlay.

4.3.2 Node distribution

Nodes can be inserted following a heuristic to create a particular overlay. The distribution of the nodes can be used to model the location of nodes in an overlay which encompasses the real world. The most common and simplistic model is uniform node distribution.

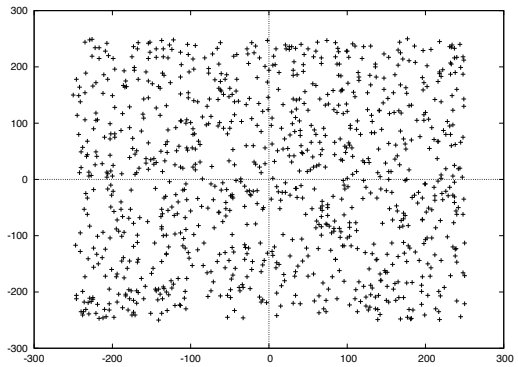
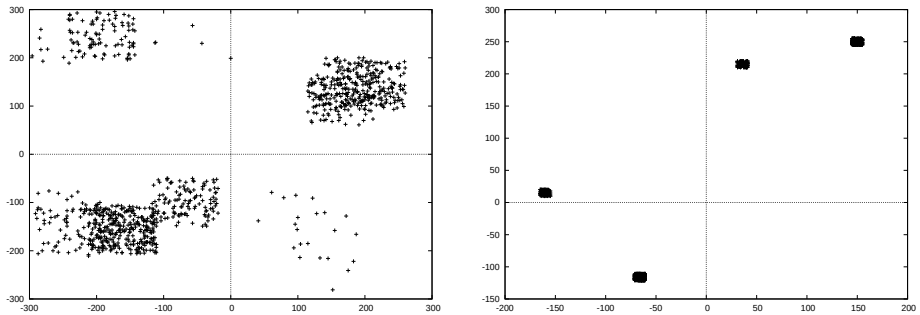


Figure 4.3: Uniform node distribution

Nodes spread out uniformly throughout the universe such as in Figure 4.3 is a rather simplistic and best case scenario. In the real world, nodes would be bound by geographical boundaries such as oceans. Similarly, areas of dense population, such as cities will affect the distribution.



(a) Low node clustering (b) High node clustering

Figure 4.4: Low and high clustering distributions

The clustering distribution models presented in Figures 4.4(a) and 4.4(b) offer a more realistic scenario for performance in the real world. In general a bias towards using clustered node distributions is favoured for the reason that it provides a more realistic set of node locations, as shown by Peerwise, but also a tougher test for nodes to obtain data from other nodes which are '*further away*'. Clustering could be due to underlying network design, such as nodes on a particular AS or geographical reasons such as the Atlantic ocean. The topologies generated used strategies and methodologies in Clegg [152]. The two topologies presented are referenced by Clegg as "*loosely and tightly clustered node distributions*". Once nodes are generated, they are given random identifiers, which, according to Clegg is vital to accurate topology generation.

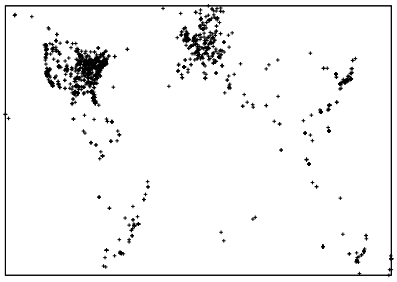


Figure 4.5: Peerwise node distribution

These node distributions are not based on real Internet measurements. It could be argued that in the case of '*high node clustering*' in Figure 4.4(b) is unrealistic based on the extreme nature of clustering, however it does provide a close to worse case scenario to test the performance of any overlay. However, it is also vital that systems are tested in realistic scenarios and for that reason, the Peerwise data set was used. That data set was borne out of the authors' desire to exploit errors in the network coordinate data sets, such as King [149]. The authors argue that

by using edges with embedding errors, it is possible to discover "*helpful peers*".

The Peerwise data set [153] is a full mesh of latency measurements between 1715 nodes with AS path information. The project works on the notion that triangle inequality errors can exist thanks to peering. It argues that coordinate systems such as Vivaldi [59] predict latency based on the assumption that $AB + BC \geq AC$ also known as the Cauchy-Schwarz inequality [154]. While coordinate systems try and preserve triangle equality, it results in unrealistic latency predictions for inter-node connections.

The latency between n_b and n_c does not respect the triangle equality previously mentioned. Though Figure 4.6 is an extreme example of what is termed triangle inequality violation (TIV) by Lumezanu et al., they argue that due to peering agreements it could be possible that two nodes have lower latency than their distance implies. It could also be said that proxies, caching and other 'middle boxes' could result in increased latency but the fact is as Lumezanu et al. states, Internet routing is not based on latency and for that reason TIVs are relatively common. Nevertheless, the sentiment remains, estimating latencies to maintain triangle equality can result in unrealistic results. When referring to the Peerwise data set in experiments, it is indicating use of the 1715 node data set presented on the project's website.

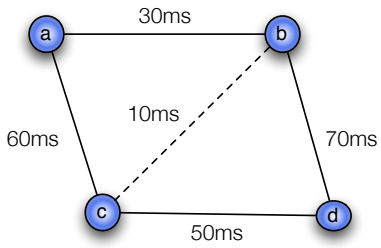


Figure 4.6: Triangle equality being broken

Although Lumezanu et al. say the existence of TIVs is not necessarily a negative, they state that the presence of TIVs mean that the three points which make up a 'bad triangle' cannot be embedded into coordinate systems without introducing errors. They state that the "*the more TIVs there are, the more imprecise the embedding [is]*", however the authors say some TIV errors can be exploited, especially as absolute embedding errors can be used to indicate the presence of TIVs. In a follow-up paper, Lumezanu et al. [155] propose a system that can detect TIVs to create an overlay that creates edges which are of "*mutual advantage between pairs of*

hosts".

What Lumezanu et al. presented was that sometimes the 'long sides' in a three node triangle can benefit from having a detour. For example in a three node triangle ABC where AC is the long side, the short sides, AB and AC could be part of shorter parts. Experiments conducted by Lumezanu et al. in [153] found that a pair of nodes that have a negative estimation error due to TIV has a higher chance of needing a shorter path, whereas nodes with a positive estimation error are more likely to be part of a shorter path for another node.

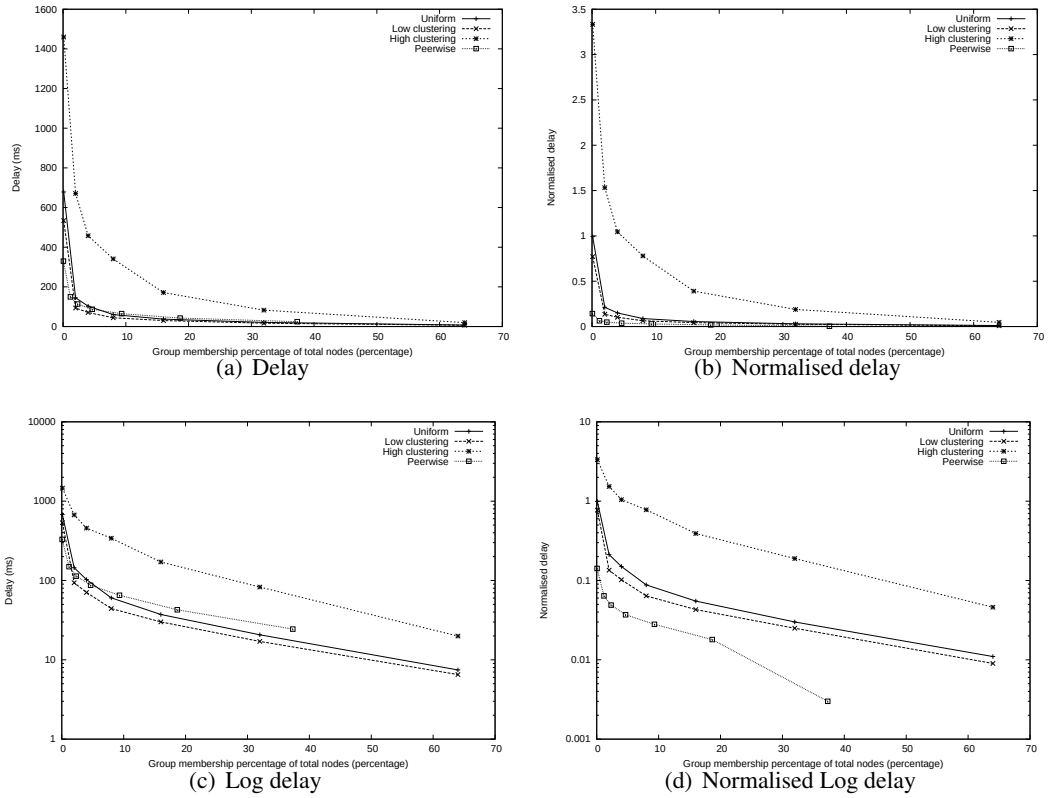


Figure 4.7: Absolute and normalised delay utilising uniform expanding rings

Figure 4.7 shows the query delay within GOAT when utilising uniform expanding rings as presented in Equation 4.1. In Figure 4.7(b), the delay has been normalised over the distance of the '*world*' to produce d_{norm} which has units of ms/m.

Calculating the uniform delay allows us to compare various node distributions where the world size is different. For instance, a distance of 60ms where the maximum edge-to-edge distance is 200ms is more significant than the same distance is present where edge-to-edge distance is 800ms.

As we are testing the viability of Anycast, a system which is designed to return either one or a subset of nodes from a particular group, it is more accurate to display results as the group membership varies as a percentage over total overlay membership. This is useful as it takes into account that while overlay membership can grow, it is group membership that actually has an influence on performance as users may want to access content which is being carried in both popular and unpopular groups. As node membership increases, it is intuitive to assume performance in metrics such as average delay and accuracy error will improve, however the real test of GOAT is its performance with low group membership.

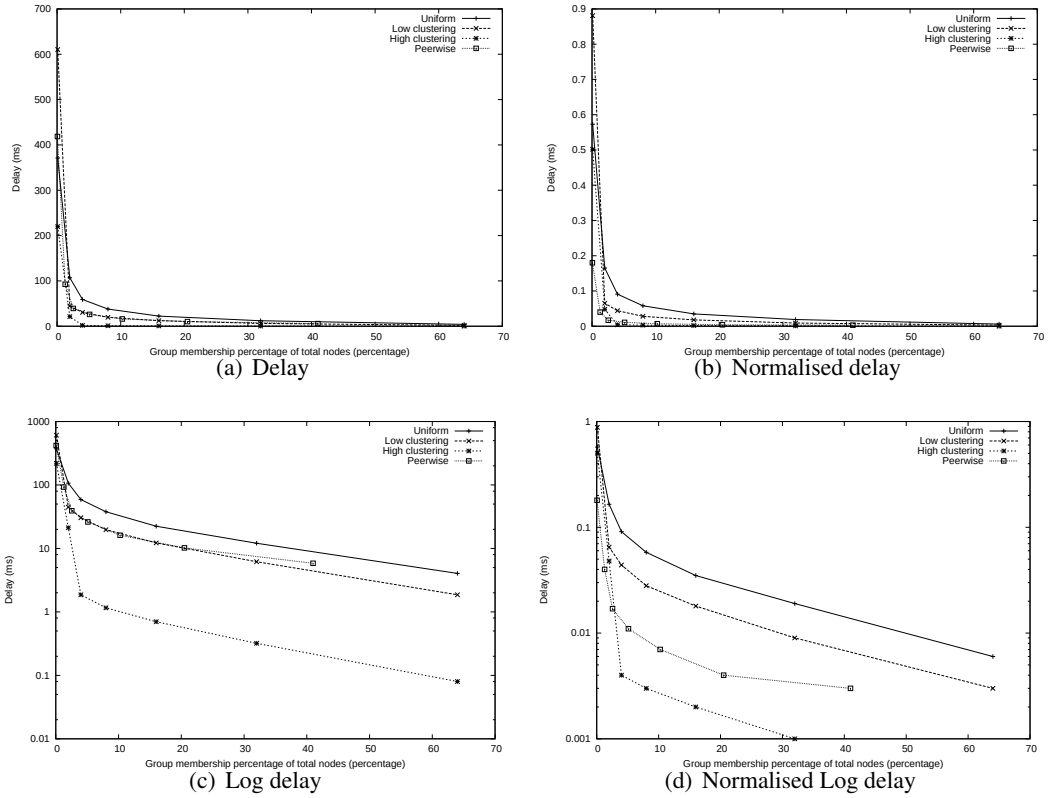
In experiments all three node distributions follow the same delay pattern, with the average query delay dropping as the percentage of group members increase. It is clear that the highly clustered distribution results in a higher delay, an expected result given the lower probability of finding group members within the cluster. As Figure 4.4(b) shows, the delay cost is high when having to '*jump*' from one cluster to another due to the distances between clusters.

The results show improvements in delay occur with a relatively modest increase in group membership. In the case of uniform and low clustering node distributions, delay drops significantly when group membership reaches 10 percent. Further delay improvements are recorded, however it is encouraging to record such improvements with relatively low group membership figures. Although significant improvements in delay are recorded with the highly clustered distribution, to attain the sub 100ms delay figure that was reached within 5 percent group membership with low clustering and uniform distributions, group membership has to surpass 30 percent.

It is particularly encouraging to witness the performance of GOAT when using the Peerwise node distribution. The observed delay follows very closely to that of the low clustering distribution. On visual inspection, the low clustering distribution (Figure 4.4(a)) is a closer '*match*' to the Peerwise distribution (Figure 4.5). Due to the maximum distance being significantly lower in the Peerwise distribution (438 units), normalised delay performance is favourable.

The Peerwise data series ends prematurely due to the smaller number of nodes in the distribution (1715), the percentage of GOAT nodes over the total membership would be lower at the same figure as the other uniform and clustered distributions. For instance 640 GOAT nodes in a 1000

node distribution would be 64 percent while the same number of GOAT nodes in the 1715 node Peerwise distribution is 41 percent.



(a) Delay (b) Normalised delay

(c) Log delay (d) Normalised Log delay

Figure 4.8: Absolute and normalised delay utilising logarithmic expanding rings

With logarithmic expanding rings as presented in Figure 4.1(b) and calculated through the formula presented in Formula 4.1, the overall performance looks very similar to that of linear expanding rings. However, while the performance follows the same trend, in absolute performance terms there is a significant increase. The most visible illustration of this is the absolute delay presented in Figure 4.8(a). When compared to the equivalent results presented in Figure 4.7(a), it is evident that for all the distributions within 5 percent group membership, the delay had dropped to below 100ms.

The most evident change in performance between uniform and expanding rings came when the high clustering node distribution was used. With the use of uniform rings, performance is worse than other distributions as shown in Figure 4.7(c) and 4.7(d). In those results the uniform expanding ring performs significantly worse than the best case node distribution. An explanation for this can found in the difference in the number of hops the query takes when using uniform and logarithmic expanding rings.

In both Figure 4.7(c) and 4.8(d) logarithmic expanding rings perform well in low clustering distributions beyond a five percent group membership. Not accounting for overlay size, the delay with both uniform and logarithmic rings are similar in low clustering and Peerwise distributions. This would suggests that an expanding ring system treats the low clustering distribution of nodes that was generated similarly to the real world representation provided by the Peerwise distribution.

Ring type	Uniform	Low clustering	High clustering	Peerwise
Uniform	14	6	6	13
Logarithmic	11	10	3	11

Table 4.1: Average hop count for 0.1% node membership over total nodes

Table 4.1 displays 0.1 percent group membership over total overlay membership, logarithmic rings performs as expected, reducing the number of hops for query propagation over uniform rings in all node distributions apart from low clustering. It should be noted that as group membership increases the hop count when using logarithmic rings matches that of uniform rings. When these results are compared against those recorded using a uniform node distribution they are very similar.

	Uniform		Low clustering	
Group membership %	Uniform	Logarithmic	Uniform	Logarithmic
0.1	14	11	6	10
2	5	5	4	4
4	3	3	3	3
8	2	2	2	2
16	2	2	2	2
32	1	1	1	1
64	1	1	1	1

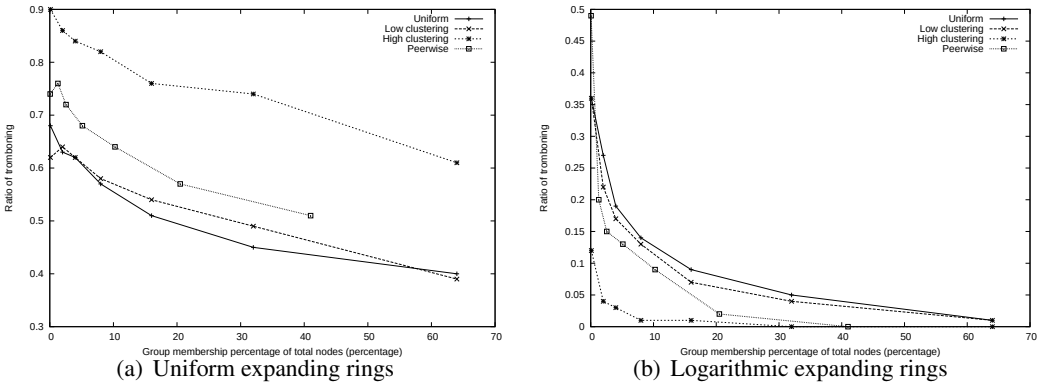
Table 4.2: Hop count on uniform and low clustering distributions

The similarity in performance between uniform and logarithmic expanding rings is shown in Table 4.2. Aside from the result with 0.1 percent group membership with uniform expanding rings in a low clustered distribution, the hop count is similar as the group membership increases. It is important to remember that 0.1 percent membership represents a single node in the group,

therefore as the clustering increases, the performance of logarithmic rings is likely to be similar to uniform expanding rings as the first in both heuristics is the same distance away.

Query delay shown in Figure 4.7 and Figure 4.8 low clustering and the Peerwise node distributions lead to similar results. It also brings up the link between delay and hop count, further motivating the need for fewer query hops (it should also be noted that lower the number of hops a query takes lowers the chance for a query to be '*stalled*' or re-routed in its path due to a node being offline).

The most graphic illustration of the performance benefits of logarithmic expanding rings comes when a high clustering distribution is used. Due to the extreme separation of clusters shown in Figure 4.4, it is likely that if a group member is in another cluster, the distance from one cluster to another is large. It is likely that with uniform rings many would be empty or with closer clusters have few nodes in them. Should the latter scenario arise, then it is possible that a query would trombone, increasing delay and this is exactly what was observed, with the ratio of routing efficiency showing a large decrease when logarithmic expanding rings were used, while the number of neighbours a node has increased.



(a) Uniform expanding rings (b) Logarithmic expanding rings

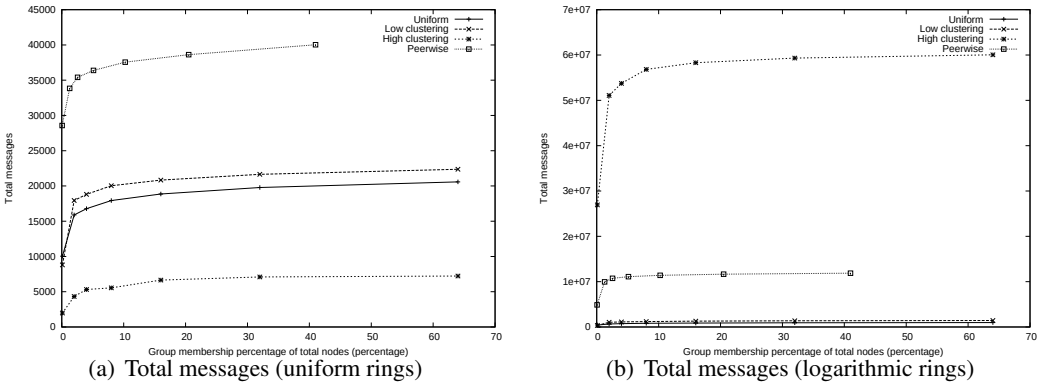
Figure 4.9: Ratio of routing efficiency with uniform and logarithmic expanding rings

When considering uniform and expanding rings using the ratio of routing efficiency as described in Equation 4.4, it is clear logarithmic rings result in significantly higher routing efficiency. To understand how significant the performance difference in between uniform and logarithmic rings one has to consider that in the best case with uniform rings, with a low clustering distribution and 64 percent group membership, the ratio is 0.39. The only observed routing efficiency ratio that is worse than 0.39 with logarithmic rings is with the Peerwise node distribution with

0.06 percent group membership.

The result is due to the distance increase between rings in the logarithmic system requiring fewer intermediary nodes resulting in fewer nodes diverting queries with improved routing efficiency. Fewer nodes result in a more direct route being taken and this is especially true as node distributions become increasingly clustered. In Figure 4.9(b), the highly clustered distribution not only provided the best initial result with low group membership, the routing efficiency ratio hit 0.0 beyond 32 percent group membership. In separate tests it was observed that uniform distributions would also record a routing efficiency ratio of 0.0, though group membership would need to reach 75 percent.

When considering the number of messages being sent, it is important to differentiate between the total number of messages, the sum of all messages sent by every node on the overlay, and the number of route updates. A message is one that is sent by a node that does not necessarily cause the recipient to update its routing table, while the number of route updates are messages that cause a node to update its routing table. The difference between the two represents overhead, messages that are sent but result in no changes to the routing table of nodes that receive them. The importance of route updates as a performance metric explained in Section 4.3.1.



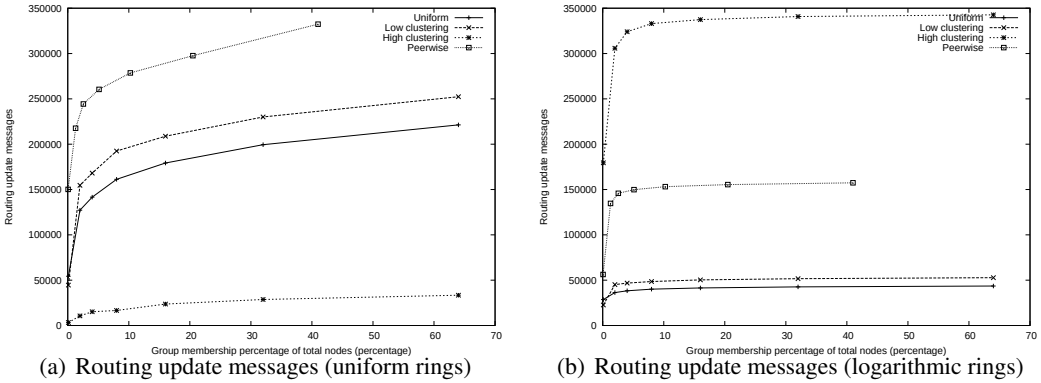
(a) Total messages (uniform rings) (b) Total messages (logarithmic rings)

Figure 4.10: Total messages with uniform and logarithmic expanding rings

The performance of uniform expanding rings with increasing clustering although looks favourable must be put into context by considering other metrics such as the average number of neighbouring nodes - those that are present in a node's routing table - was nine. Compare this to 24 for the low clustering and 44 for Peerwise node distributions and it is clear that uniform expanding rings, offer lower messages due to knowledge of fewer nodes. Consequently, impact on other performance metrics such as average delay, tromboning ratio and the number

of failed queries, which for 0.1 percent group membership resulted in 154 failed queries and for two percent group membership, three failed queries. In every other test, both with uniform and logarithmic expanding rings in the four node distributions, there were no failed queries apart from the aforementioned two instances. So while lowering the overall number of queries within the overlay, there is a decrease in performance as node clustering increases with uniform expanding rings.

With logarithmic expanding rings the total number of messages has increased by two orders of magnitude, highlighting the cost of having a greater number of neighbours. The number of neighbours increases significantly with node clustering with 45 neighbours from in a uniform node distribution to 358 neighbours with high clustering. With over 35 percent of the total overlay is being kept in a node's routing table results in the query hop count reducing to zero, however as shown in Figure 4.11(b) this causes an increase in the number of routing updates that are sent.



(a) Routing update messages (uniform rings) (b) Routing update messages (logarithmic rings)

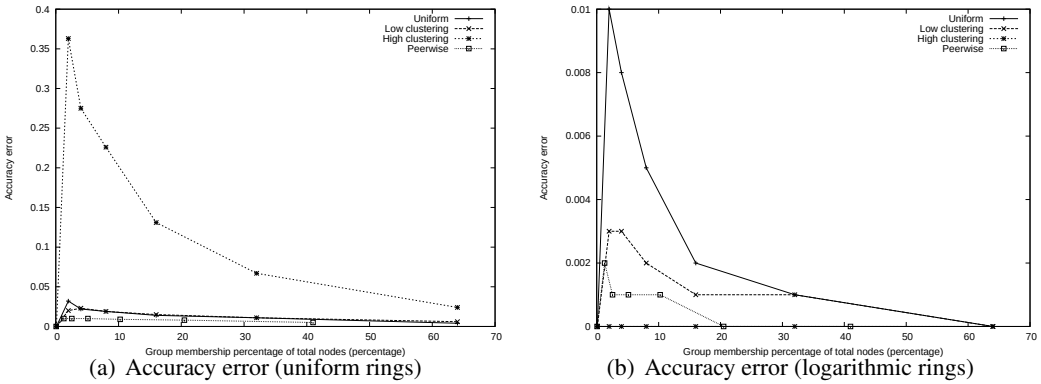
Figure 4.11: Routing updates with uniform and logarithmic expanding rings

The number of routing updates has a strong correlation with the total messages observed in Figure 4.10. With both uniform and logarithmic expanding rings, the order of messages and updates being sent irrespective of node distribution is the same. With uniform rings, the increasing order of messages is high clustering, uniform, low clustering and Peerwise, while using logarithmic expanding rings in increasing order of messages is uniform, low clustering, Peerwise and high clustering distributions.

This paints a picture of uniform expanding rings being favourable in high clustering environments, however as explained previously considering a single metric such as number of messages it is not advisable if one wants to glean an accurate picture of performance. With the

number of route updates exchanged directly linked to the number of neighbours a node has and that figure itself is related to other performance metrics such as hop count, delay and routing efficiency. The performance of logarithmic expanding rings is nearly opposite is in accordance with a system that has nodes spaced far apart and if queries are not to fail, which could be argued is a far more catastrophic failure.

It is important to consider the increase in route updates as group membership increases. In this case, both expanding ring policies show favourable results, displaying $log(n_t)$ increase of route update with group membership increase in all node distributions. Logarithmic expanding rings, the growth in routing updates is smaller as group membership increases, behaviour that shows GOAT scalability. Similarly the number of routing updates does not increase significantly between uniform and logarithmic expanding rings, as was the case with total messages in Figure 4.10. This shows that while the total number of messages in the overlay was significantly higher in Figure 4.10(b), the cost of maintaining the overlay did not significantly increase.



(a) Accuracy error (uniform rings) (b) Accuracy error (logarithmic rings)

Figure 4.12: Accuracy error with uniform and logarithmic expanding rings

As defined in Equation 4.3, accuracy error is a measure of the system's quality. The ideal system would return the '*best*' node, and in the case of GOAT, is the closest node as defined by delay, which would be represent an accuracy error of 0.0. This is the case when there is a single node group member in the group, as quite clearly the system cannot return any other node, which explains why in Figures 4.12(a) and 4.12(b) there is an initial spike following the 0.0 accuracy error when there is just a single node group member.

For uniform expanding rings, the performance when uniform, low clustering and Peerwise distributions are used is similar, though the system clearly performs better with the Peerwise. The higher accuracy error when using the high clustering distribution is not only due to the

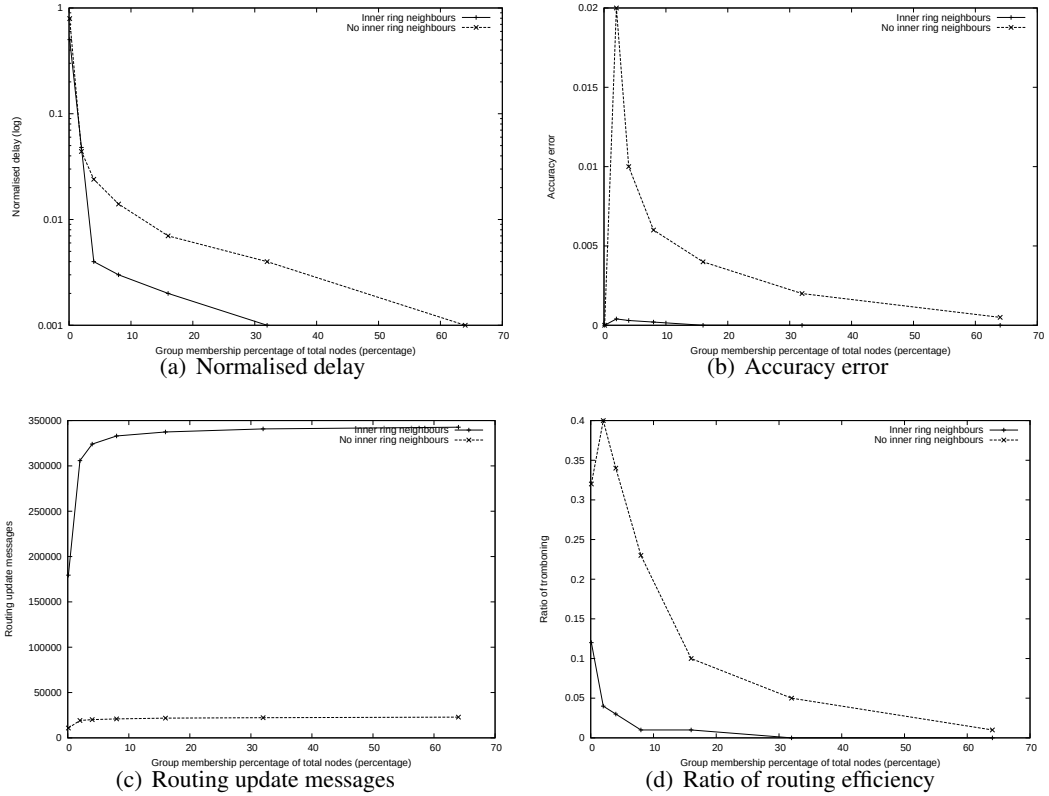
failed queries as mentioned in the analysis of route update results, but the low number of neighbours each node has, meaning that there is a far higher likelihood of a sub-optimal result being returned.

Accuracy error is significantly reduced when logarithmic expanding rings are used, by at least an order of magnitude in almost every test case. After the initial spike, regardless of which node distribution is used, at a number of points of group membership the accuracy error reaches 0.0, meaning GOAT returns the best result. For uniform and low clustering we witness this at 64 percent membership, though with higher clustering distributions such as Peerwise the 0.0 accuracy error figure is hit at just 16 percent group membership. The performance of logarithmic rings increases with clustering and on the high clustering distribution in all cases the accuracy error was 0.0.

In an attempt to reduce the cost of logarithmic expanding rings, GOAT's protocol was altered to remove the first ring. This effectively meant that the first ring, which in the experiments that were conducted was placed at a '*distance*' of 10ms away from the node was removed. The notion behind this move was to see what effect removing the closest neighbours would have on delay, messages and routing updates. The tests were conducted on an overlay that had high clustering node distribution as this was found in previous experiments to bring the highest cost when using logarithmic expanding rings.

By ignoring nodes in the first ring, the results presented in Figure 4.13 show that the cost of having logarithmic expanding rings is curtailed. Route updates are significantly reduced as shown in Figure 4.13(c) as the number of neighbours goes down from 358 to 25 when discounting the first ring. The downside to this is found in other performance metrics. The normalised delay, Figure 4.13(a), shows promising performance when group membership is low but as it increases, there is a widening gap.

The accuracy error (Figure 4.13), though significantly lower than when using uniform expanding rings, is higher than when accounting for nodes in the inner ring. Similarly, the ratio of routing efficiency was higher, suggesting that a significant proportion of queries were diverted '*off course*' by not having a node in the inner ring. This also suggests, at least with highly clustered node distributions, the first node the query reaches has a significant bearing on the routing efficiency.



(a) Normalised delay (b) Accuracy error

(c) Routing update messages (d) Ratio of routing efficiency

Figure 4.13: Performance with and without inner ring neighbours in high clustering node distributions

Further investigation can be carried out in varying the distance of the first ring. In experiments presented here, the first ring's boundaries from the node up to a radius of 10ms. If this was made shorter it would likely increase the number of neighbours, causing an increase in messages, however decrease the performance hit as witnessed in Figure 4.13. The results presented in Figure 4.13 show the cost/performance trade offs that need to be made in order to provide a balanced performance, with an inflection point somewhere between 1ms and 10ms.

The figures for accuracy error when high clustering goes some way to justify the high number of neighbours and the total number of messages which were considerably higher than in other distributions. The results presented on Figure 4.12(b) display the motivation for having a trade-off between total number of messages on the overlay and the accuracy. While it is desirable to have the '*perfect*' result every time, the high number of messages poses questions of node resources in the overlay.

Another protocol alteration to lower the cost of the GOAT overlay was to introduce a bias that favour closer node selection. The motivation behind this was to improve performance in

clustered overlays where the majority of group members would be close by rather than far away.

The bias changes alters the probability of a node being selected. The three bias variants used altered the probability with distance by formula $p = d^{0.3}, d^{0.5}, d^{0.8}$ meaning that nodes that are closer to the orginator node have a higher probability of being selected as the distribution used in the simulator to select nodes is purposefully skewed to favour closer nodes. The probability distribution function resulting from the bias are displayed in Figure 4.14.

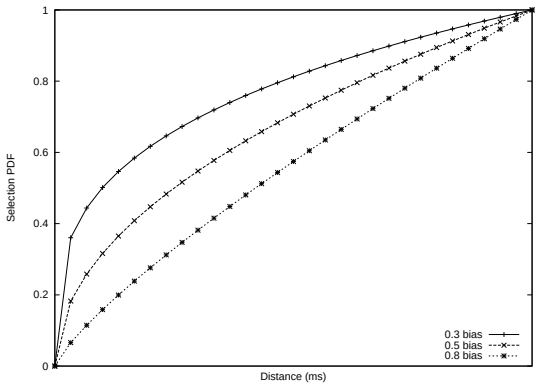


Figure 4.14: GOAT node selection bias

The probability distribution shown in Figure 4.14 highlights the degree to which selection probability can be biased in GOAT. A '1.0 bias' is one that does not place any preference on the distance of a node from the originator, therefore the '0.3 bias' skews the selection probability towards nodes that are closer to the originator.

The metrics measured were; the actual number of neighbours a node had after establishing bi-directional connections, the coverage over all peers in the GOAT overlay and the replication, the average number of one-hop neighbours per covered node.

Experiments were carried out using the uniform distribution (Figure 4.3), low clustering (Figure 4.4(a)) and high clustering (Figure 4.4(b)) as described previously. In the case of uniform node distributions, the number of nodes used were 1000, 2000 and 5000. For the low and high clustering distributions, the number of nodes was either 1000 or 2000 nodes as these are deemed to be representative group sizes allowing for the number of GOAT nodes to be varied.

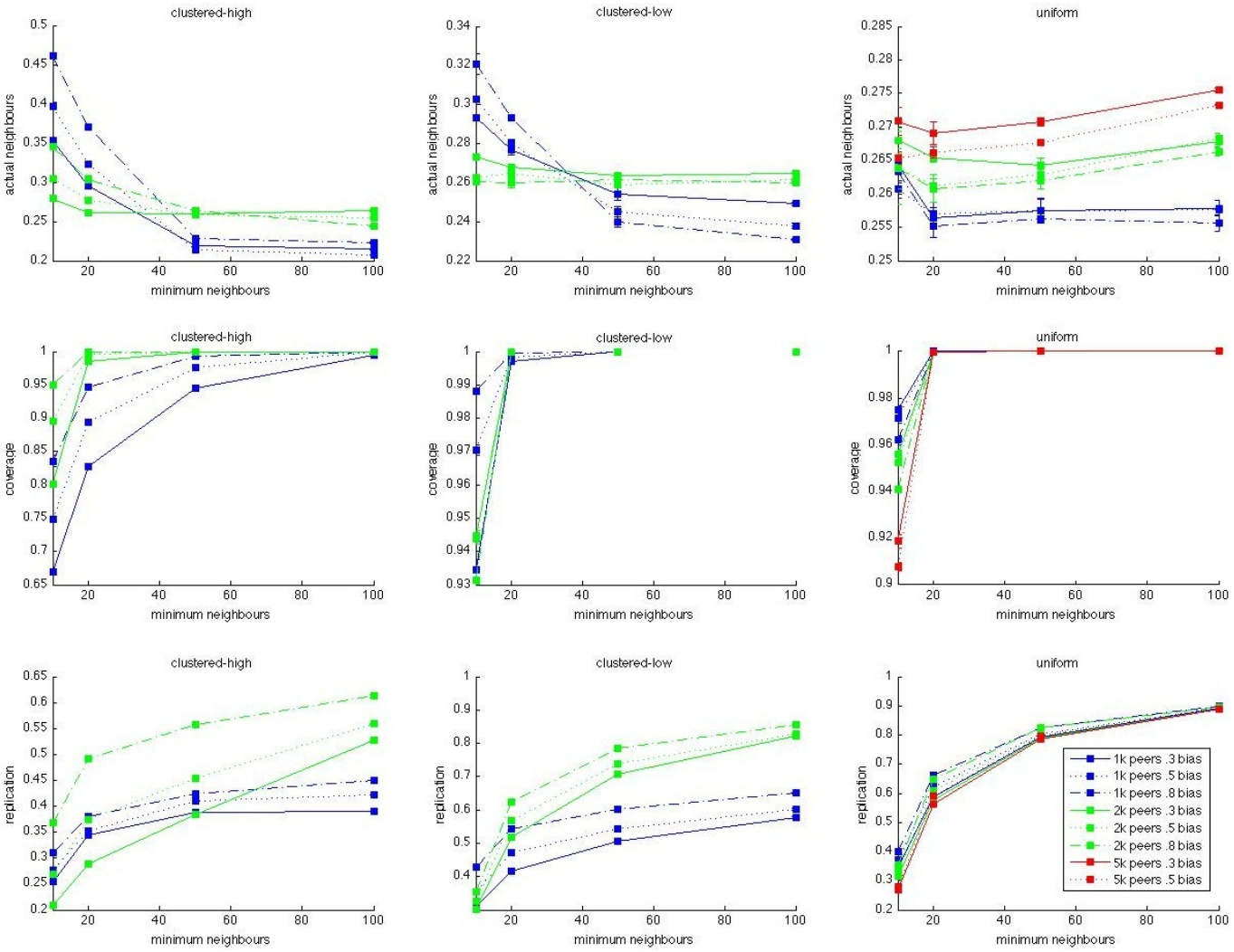


Figure 4.15: The relationship between node knowledge and overlay coverage

The first row of graphs show the actual neighbour percentage decrease significantly as the minimum neighbours increases. The second row shows coverage as the number of minimum neighbours increases, with uniform and low clustering distributions showing good results regardless of bias selection. Perceptible difference is seen below 20 minimum neighbours, where fewer nodes result in higher coverage. In the highly clustered node distribution it is clear that with 2000 nodes complete or near complete coverage is achieved at 20 minimum neighbours. However with 1000 nodes, especially with 0.3 bias, there is never complete coverage of the overlay. It should also be noted that as with 2000 nodes, the best performance is recorded when the bias is 0.8.

The coverage results was expected due to the bias placed towards far away nodes is unlikely

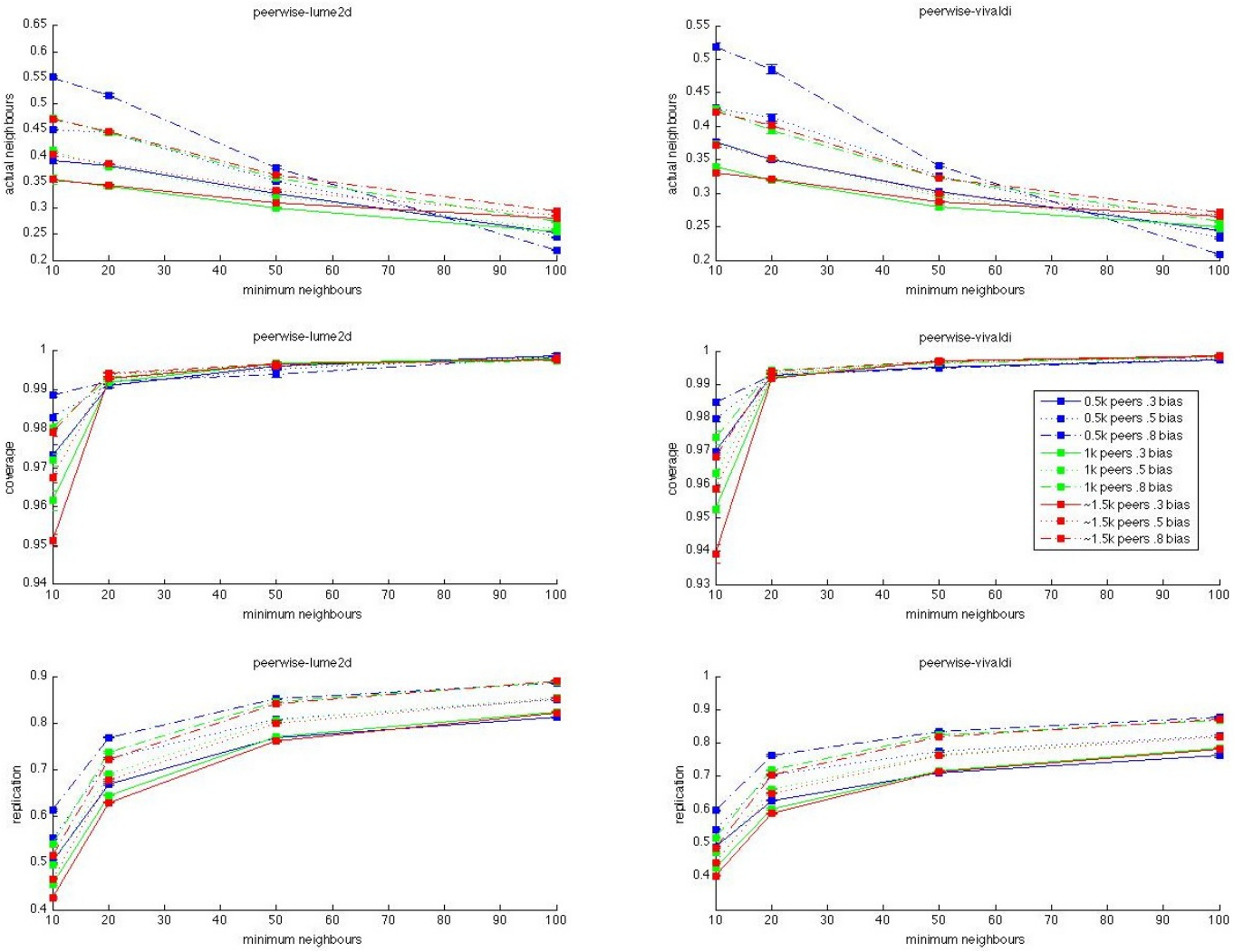


Figure 4.16: Node knowledge and overlay coverage using real-world distributions

to produce good results which has vast areas where there are no nodes. Introducing a bias to select distant nodes would produce worse results as clustering increases, which was confirmed by these results. Coverage increases with more peers due to the increase in sampling in highly clustered distributions, results in new clusters which makes the overlay behave more towards one with a uniform node distribution.

Replication, the percentage of entries in a node's routing table that is the same as other nodes on the overlay was generally high, especially with a uniform node distribution as seen in Figure 4.15. While replication was lower when clustered node distributions were used due to the number of neighbours required to achieve coverage.

Experiments were re-run using the Peerwise dataset with lume2D and Vivaldi embeddings to see how the changes performed in realistic node distributions. The general trends for neighbours, coverage and replication were similar to when uniform, low and high clustering node distributions were used. Perhaps the biggest change was in overlay coverage where the number of nodes and the bias had little effect on overlay coverage. In all cases with both lume2d and Vivaldi embeddings, over 99 percent of the overlay was covered with knowledge of 20 nodes.

4.4 Conclusion

The results presented in Section 4.3 justify the use of logarithmic expanding rings for neighbour acquisition. The choice is one of balancing the performance trade offs by employing logarithmic expanding rings while being wary of the cost to nodes. From a performance perspective, logarithmic expanding rings have shown to provide higher accuracy (Figure 4.12), a decrease in delay (Figure 4.7 and Figure 4.8) and routing efficiency (Figure 4.9).

Logarithmic expanding rings perform well in these metrics against uniform expanding rings, but it is clear that there is a cost associated to this performance gain. The main cost associated to logarithmic expanding rings is the rise in total number of messages that was observed over uniform expanding rings. It should be noted that the number of route updates sent when using logarithmic rings did not significantly increase, except in the case of high clustering.

It is in high clustering node distributions that the performance of uniform expanding rings falters and the advantage of using logarithmic expanding rings is greatest. The average number of neighbours that each node had in such an overlay increase from just nine with uniform rings to 358 with logarithmic rings. The upshot of this is the number of routing updates that are sent is increased and while having over 35 percent of the overlay membership in a node routing table is expensive, it ensures that unlike uniform rings, there are no query failures. However, it is not acceptable to have such a high percentage of total overlay membership in a routing table is acceptable and further work is needed in order to strike a balance between the unsatisfactory outcome of uniform rings while curtailing the expensive but high performance of logarithmic rings.

In some cases uniform expanding rings do provide perfectly adequate performance and in the case of average query hop count, the different between uniform and logarithmic expanding rings is very little, as presented in Table 4.2. It is possible to say from the experiments that

the advantages of using logarithmic expanding rings are far more pronounced as clustering increases in the node distribution. This may change in larger overlays in which the size results in the fewer rings of a logarithmic ring algorithm reaching nodes with fewer hops.

The performance metrics listed in Section 4.3.1 have been evaluated with the note that for the number of failed queries, the only experiments where GOAT produced failed queries was for uniform expanding ring, the figures of which were listed in Section 4.3. Further work should be carried out to consider lowering the number of messages that result in no updates to routing tables, thus lowering the cost of maintaining the overlay.

These experiments have shown that both heuristics for neighbour acquisition have their advantages and disadvantages in GOAT. It has also shown that the performance of such a system is largely dependant on the node distribution of the overlay. Certain design decisions have to be made which are undoubtedly trade-offs between performance and cost. This cost is not only to the nodes maintaining the overlay but for the queries themselves.

Optimising for a single performance metric is relatively easy as shown by the use of logarithmic expanding rings, which perform extremely well, especially in highly clustered node distributions, nevertheless place a great cost upon the nodes on the overlay. It is the balance that is considerably harder to achieve.

The results presented here not only to shows the viability of a gossip-based Anycast system but that challenges are present in producing such a system. Some of the challenges are addressed in the N-casting overlay.

Chapter 5

N-casting Overlay

This chapter proposes N-casting, a new messaging paradigm that extends Anycast to sending to the n_x closest group members. One can see N-casting as a hybrid of Anycast and Multicast, where every group member receives the message.

With the N-casting overlay it was deemed necessary to investigate the issues of overlay coverage, a reduction in message/storage overhead and avoiding double counting. Tackling these challenges are non-trivial and offer the opportunity to create a system that is efficient in the use of network resources and overlay coverage. In other words, the biggest challenge was to trade-off accuracy on the knowledge of where the n_x closest members are and amount of messages/state needed.

An N-casting query is defined as a query that can be routed to one or more group members. Unlike Anycast, where the query is routed to the node which is determined to be the best 'fit' for the selection criteria, in a N-casting system the sender can specify $n_x \geq 1$ nodes that the query will be routed to.

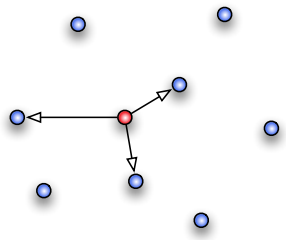


Figure 5.1: N-casting - multiple nodes receiving a node's query

The n_x nodes that best meet the selection criteria will service the query. This definition of

manycast is similar to those outlined in [156] and [157] in the context of optical burst-switched networks. The area of manycasting is investigated in Section 2.4.3 and N-casting is presented in Figure 5.1.

Motivation for an N-casting overlay is generated from the need to "contact" multiple group members that can provide relevant data to be used in aggregating bandwidth or backup. In the case of N-casting, the aim is to provide the n_x geographically closest group members, a result that is useful for a wide range of applications such as content distribution networks, content-centric networks, distributed virtual worlds and distributed storage networks among others.

By creating a hierarchical decomposition of the Internet based on latency, the N-casting overlay can provide latency awareness that is typically associated with IP-level Anycast. However as routing tables and addressing schemes are handled on the application layer, the costs associated with IP-level Anycast (Section 2.4.1) are mitigated.

Figure 5.2 shows how tiers in the overlay could represent different geographical areas depending on the number of nodes. Figure 5.2 should be viewed as one tier of an hierarchical overlay, visualising the number of nodes originating from the UK or Germany being greater than the number of nodes originating from areas such as Russia or parts of Africa.

Issues surrounding the advantages and disadvantages of location aware distributed overlays are discussed in Section 2.2.5 and in previous work such as DOAT (Chapter 3) and GOAT (Chapter 4) while work presented by Ciullo [35] evaluates the advantages of location aware overlays in P2P video streaming overlays to have a positive effect on performance. In this case it is important to note that N-casting will be location aware and that the hierarchical clustering will provide a varying resolution of overlay membership to each node. Each tier will provide a particular resolution and to that end, Figure 5.2 is a representation of a tier that provides a country resolution in the overlay. The tier below Figure 5.2 could, for example, provide a city-wide resolution.

Although each tier represents a '*compression*' of node distance information, it is also used to represent cluster membership. If the lowest tier of the hierarchy is a single node then the layer above is an aggregation of two or more nodes. The distance, or rather the resolution of that higher tier, could be determined by the distance of two single nodes on the lowest hierarchy, the aggregation decision is made by the linkage, which is discussed in Section 5.3.1.

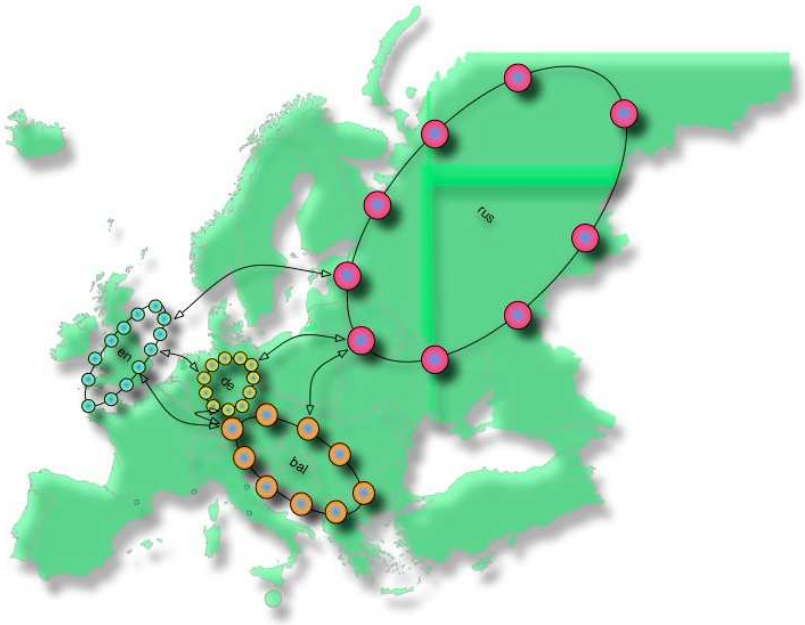


Figure 5.2: Regional overlays

The ability to contact multiple group members would require multiple Anycast queries, while the N-casting overlay reduces this down to a single query. This ability, made possible through the use of a '*floating-point*' data structure, will considerably reduce the number of messages, especially for popular groups.

The N-casting overlay brings together a number of important features such as hierarchical decomposition of the Internet, an efficient scalable data structure that provides a varying and predictable resolution of group membership and a protocol that allows queries to be sent to multiple group members.
N-casting brings location aware message passing by using the following techniques

- Cluster nodes on the overlay based on geographic location
- Aggregate clusters to form hierarchies
- Route and message propagation between clusters

Clustering nodes on an overlay to create groups of nodes that are geographically close to each other is the first step to allowing group-wide communication. Nodes could be clustered based on many characteristics however as the motivation is to provide geographically close clusters,

node delay will be used as the basis for clustering. Other metrics could include available bandwidth, uptime, or an ability to answer *q* queries in time *t*. The choice of metric to cluster on will be discussed later this in this chapter.

As the N-casting overlay's clustering algorithm is the first aspect of joining the overlay and to evaluate this the dataset is of utmost importance, a description of the dataset used for simulation is presented prior to the explanation of clustering and protocol.

5.1 Overlay protocol

The process of providing the overlay address of the first node given to joining nodes is provided by an external service such as a DNS or a tracker. The bootstrapping of N-casting nodes is outside of the scope of this work, however it could possibly be provided by a DHT such as DOAT or a load-balanced pool of DNS servers.

When nodes join the N-casting overlay, a node has to determine its position in the clustering hierarchy. In order to do this it either performs measurements to landmark nodes or by using IP geolocation. Once the node has determined its position in the hierarchy it then exchanges routing information with other nodes present in the N-casting overlay.

The node's local cluster is a node's lowest tier and the node maintains connections with every node in that cluster. For foreign clusters, the N-casting overlay follows a protocol that is similar to that of Kademlia [7] and other overlays, to know of one node in every other cluster. Formally for cluster *c*, $c_a^v = c(n_i, v)$ where node n_i belongs to tier *v*. Then for each cluster in tier *v* in [1,max_{ni} (*v*)] node n_i will establish a connection with each cluster c_b^v such that $c_b^v \subset c(n_i, v-1)$ and $c_b^v \neq c(n_i, v)$.

Once the node has been given the location of another node on the overlay, it sends a CONNECT_INIT message in the format CONNECT_INIT($c_a^{max_{ni}(t)}(n_i)$) where n_i is the node that has been located. This message gets propagated to node n_j in the $c_a^{max_{ni}(v)}(n_i)$ cluster and since every node needs to know at least one node in a high level area, this process will always terminate.

Node n_j will then send INFO messages to node n_i with information on its local and remote clusters, including local node and group membership information in a list of NodeID:GroupID

tuples. For simplicity NodeIDs are the IP address and port number of a node but could be a hash. The INFO message also contains information on remote nodes in the form of NodeID:ClusterID tuples.

Node n_i then sends a CONNECT($c_a^{max_{n_i}(v)}(n_i)$) to every local and remote node it has discovered. The NodeID:ClusterID tuples sent by node n_j are identical to the ClusterIDs node n_i will connect to.

Every node n_k that receives a CONNECT message from node n_i determines at which tier in the hierarchy the two nodes will be connected at. This is done by taking the longest prefix match of $c_1^{max_{n_i}(v)}(n_i)$ and $c_2^{max_{n_k}(v)}(n_k)$ to be c_3^p. Then, if $p = max_{n_i}(v)$, the two nodes belong to the same leaf cluster and node n_k establishes a connection with node n_i sending a list of NodeID:GroupID tuples. If $p \neq max_{n_i}(v)$, node n_k is in a remote cluster $c^{p+1_d}(n_k)$ to node n_i. In this case, node n_k will try and find another node in $c^{p+1_d}(n_k)$ to become a neighbour for node n_i. This is accomplished by sending a message to a randomly selected node in cluster $c^{p+1_d}(n_k)$, incrementing the hop counter. The process repeats for a fixed number of hops and the last responding node will establish a routing connection with node n_i.

As a further performance enhancement, in some deployment scenarios, such as nodes being on the same sub-net or DSLAM, CONNECT and INFO messages can be multicasted.

Distance	ClusterID	NodeID	FloatingPointStructure
10ms	$c^2_{2.2}$	n(2, .2.2)	FloatingPointStructure(2, .2.2)
20ms	$c^2_{2.3}$	n(2, .2.3)	FloatingPointStructure(2, .2.3)
25ms	$c^1_{.1}$	n(1, .1)	FloatingPointStructure(2, .1)
30ms	$c^1_{.3}$	n(1, .3)	FloatingPointStructure(2, .3)
60ms	$c^1_{.4}$	n(1, .4)	FloatingPointStructure(2, .4)
65ms	$c^1_{.6}$	n(1, .6)	FloatingPointStructure(2, .6)
65ms	$c^1_{.7}$	n(1, .7)	FloatingPointStructure(2, .7)
90ms	$c^1_{.5}$	n(1, .5) = n_j	FloatingPointStructure(2, .5)

Table 5.1: N-casting overlay node routing table

Each N-casting overlay node maintains two routing tables. The first routing table contains NodeID:GroupID tuples of nodes in the local cluster, with the other routing table containing

all the remote clusters the node is connected to. The second routing table is used to forward queries for groups that are not in the node's local cluster.

Each entry in the node's remote cluster routing table, displayed in Table 5.1 contains information on the identity of the node in the remote cluster, the distance to the cluster's centroid that node resides in and the set of groups and the number of members in those groups available at that cluster.

When a node becomes a group member, it creates a new entry in its local cluster table and sends a NodeID:GroupID message to all the nodes in its local cluster. For every update in its local cluster or in the remote clusters for which it is a hop for other nodes, the node sends a route announcement that aggregates the information across all the clusters it represents. For example in Table 5.1, node n_j will send an announce for cluster $c^1_{.5}$ to node n_i including the data structure that has information from its local cluster $c^1_{.5.1}$ and the data structure it received from the node in cluster $c^1_{.5.2}$.

To forward queries, a node receiving a query for n_{near} group members given a particular GroupID *g* will first check how many members of *g* it can find in its local cluster. If the number of group members in the local cluster is less than n_{near} the node will iterate through its routing tables checking against the data structure to find nodes in g. When an entry for *g* is found the cardinality estimator adds to the running count of g members. The iteration stops once this running count is equal to or greater than n_{near}.

As a node's routing table is sorted in ascending order of distance to clusters, the next entry is the closest cluster that has announced members for this group. The process is repeated at each hop until a node with group members in its local cluster is found. This routing policy results in each hop being shorter in distance than the previous one.

5.2 Datasets

In order to simulate N-casting and as a follow-on from the simulations conducted on GOAT (Section 4.3), datasets collected from Internet data solely used in experiments conducted on N-casting. In GOAT, three generated datasets were used along with the Peerwise dataset, as detailed in section 4.3.2, and using datasets derived from Internet measurements allow for a more accurate, real-world scenario in which N-casting can be evaluated. In order to do this, a

number of datasets have been considered and are outlined here.

Datasets such as iPlane [158] and Peerwise provide raw IP address data, however for N-casting lat-long coordinates are required and this is provided by IP geolocation services. In DOAT (chapter 3), geolocation was provided by using a space filling curve, however for finer grain geo-location, in N-casting IP geolocation will be used. There are a number of IP geolocation companies that provide 'IP intelligence' used by firms such as Google and Facebook to serve targeted adverts based on the location of the user's IP address. MaxMind [1], Digital Element [2] and Neustar IP Geolocation [3] provide latitude and longitude figures for IP addresses as well as other information, such as in the case of Neustar, bandwidth estimation and latency.

Although IP geolocation provides latitude and longitude, it may not provide the precise location of the node. The resolution of IP geolocation is not has high as coordinates provided by the global positioning system, something alluded to by IP geolocation firm Neustar when it states accurately locate with a high degree of certainty the location of a node within a "20-30 mile radius" though in some cases the coordinates could point to the DSLAM of the broadband connection, which should in most cases be nearer the node [4]. This places a limitation on the granularity of leaf clusters in a hierarchical clustering algorithm as it would be to be at least the maximum resolution of the IP geolocation service.

To show the coverage of iPlane [158] and Peerwise datasets, the nodes' lat-long coordinates were plotted in Figure 5.3 by taking the raw IP addresses and using the aforementioned Neustar IP geolocation service. The Peerwise distribution (Figure 5.3(b)) is significantly 'lighter' compared to that of the iPlane distribution due to the number of data points. The Peerwise dataset has 1,715 nodes, while the iPlane dataset has 187,314 nodes. Even with the difference in unique data points, the Peerwise distribution shows that it has reasonable coverage of North America, Western Europe and Japan.

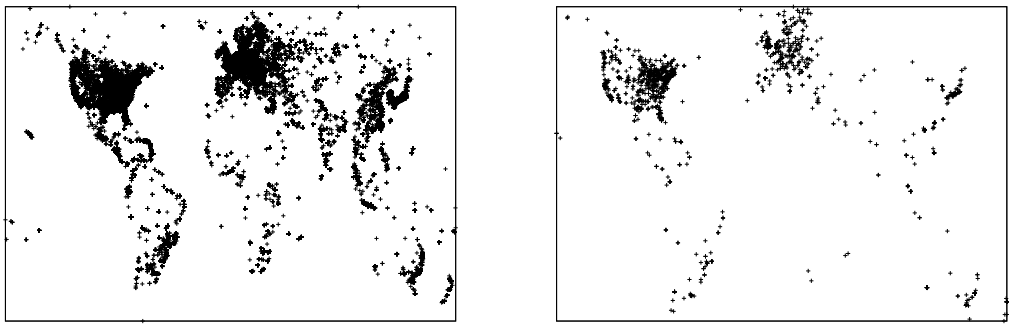
Figure 5.3(a) displays node locations in the iPlane dataset and clearly shows the outline of a world map, with a concentration of results in North America, Europe, Japan, coastal cities in

[1] MaxMind - http://www.maxmind.com

[2] Digital Element - http://www.digitalelement.com

[3] Quova - http://www.quova.com

[4] IP geolocation specialist says an IP doesn't identify a person - The INQUIRER - http://www.theinquirer.net/inquirer/news/2081054/ip-geolocation-specialist-ip-address-doesnt-identify-person



(a) iPlane node distribution (b) Peerwise node distribution

Figure 5.3: Node distributions

China and the east coast of Australia. Other countries, such as India, New Zealand and Hawaii can also made out from the plot. From Figure 5.3(a) it is possible to conclude that the iPlane dataset provides a global coverage that can be used to provide realistic hierarchical clustering.

Agarwal et al. [33] propose the use of median round-trip time (RTT) values to show correlation between latency and distance. In the evaluation of datasets presented provided by iPlane, Peerwise and DNS measurements, the median RTT value was plotted against distance. It should be noted that Agarwal et al. use MaxMind's IP geolocation, the geolocation data in our experiments is provided by Neustar IP Geolocation.

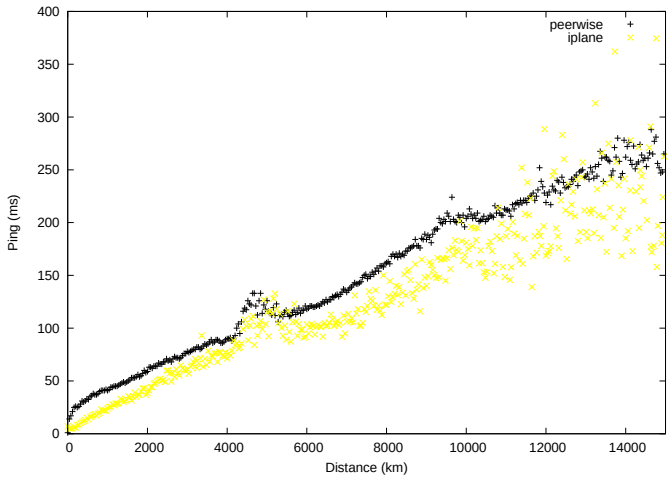


Figure 5.4: The correlation between distance and latency in iPlane and Peerwise datasets

The iPlane results represented in Figure 5.4 is the geolocation of a node within iPlane's own clusters. iPlane aggregates trace routes into clusters, which it says includes nodes that are "*similar from a routing and performance perspective*". Madhyastha et al. [158] state clusters could contain nodes within the same point of presence and those that are in the same geographic location of the AS. It is a reference to the fact that some ASes can cover vast geographical areas and it is unrealistic to assign a particular performance characteristic such as latency as an AS-wide generalisation.

Figure 5.4 shows the the median latency every 40km. There is a clear similarity between Peerwise and iPlane datasets, and that presented by Agarwal et al. Of particular note is the 'bump' near 5000km present on Peerwise, iPlane and Agarwal's dataset. No formal explanation for this bump has been provided and while it is possibly a geolocation error, given that it is present on all three datasets, one of which uses MaxMind while the other two, presented in Figure 5.4, uses Neustar for IP geolocation it is fair to assume it is a property of the network itself.

The iPlane data is not used directly as an input to the simulator, rather as a starting point for measurements in Landa [159] which employs TurboKing methodology [160] to determine latencies between measurement points. The measurements conducted by Landa is based on DNS servers found by resolving IP addresses from iPlane, DNS server lists, forward DNS replies from known hostnames, BitTorrent block lists and BitTorrent *ANNOUNCE* messages and randomly generated routable IP addresses. From those datasets around 350,000 DNS servers were found with around 54,000 non-forwarding DNS servers used as measurement points. The measurement itself followed TurboKing.

The IP addresses of the non-forwarding DNS servers was geolocated using the Neustar service. The breakdown of DNS servers' location, on a regional basis is presented in Table 5.2.
Both North America and Europe (East and West) have a similar number of nodes in the dataset at 21,276 and 19,751 respectively. China and the Asia Pacific has fewer than expected DNS servers given the population in particular of China. This could be down to a number of very large ISPs in China.

RTT measurements on the DNS servers were carried out taking 10 samples spaced 10 seconds apart. Following from the methodology used to correlate RTT with distance in Figure 5.4, the median of the 10 values was used in order to filter out any RTT measurements affected by

Subcontinental Zone	Servers
Africa	519
Central Asia and the Middle East	1,490
Asia Pacific and China	7,730
Indian Subcontinent	449
North America (North)	21,276
North America (South) and the Caribbean	526
Oceania	1,116
South America (West)	270
South America (East)	1,333
Eastern Europe	6,798
Western Europe	12,953

Table 5.2: Geographic distribution of measurement servers

variables such as spikes in network utilisation.

To simulate the N-casting overlay, the DNS server dataset is used as an input into the clustering algorithm, which groups the nodes based on the latency measurements recorded through experiments conducted by Landa. The next stage is to perform agglomerative hierarchical clustering on this data to present it as the input into the N-casting overlay simulator.

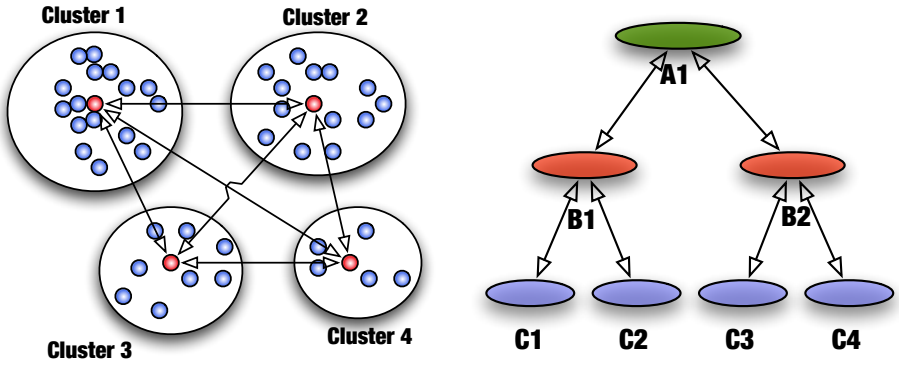
5.3 Clustering

By decomposing the overlay in a hierarchical manner, nodes in the overlay can have knowledge of global overlay membership without the need for excessive neighbour connections. Complete overlay knowledge is expensive, both in terms of storage at the node level and maintenance demands placed on the overlay as observed in Buford [161] and Monnerat [83]. Partial overlay knowledge, employed by overlays such as Chord, Pastry, Kad, Tapesty and CAN, is scalable however N-casting, through hierarchical decomposition provides the advantages of complete overlay membership knowledge with the scalability of partial membership knowledge through the use of compression.

The clustering of overlay nodes can produce a flat overlay where all clusters are placed on the same tier, or in a hierarchy, with clusters aggregating those clusters on lower tiers. Clustering usually refers to flat clustering where objects, or in this case nodes, would be grouped depend-

ing on the clustering coefficient and in hierarchical clustering, the tiers are created from the linkage method used.

Figure 5.5 shows a simple representation of what a hierarchical clustering overlay may look like.



(a) Flat clustering (b) Hierarchical clustering

Figure 5.5: Flat and hierarchical clustering

The representation of a flat clustered overlay in Figure 5.5(a) is shown with inter-cluster gateway nodes highlighted. The issue of having deterministic gateway nodes such as those presented in Artigas [100] introduces concerns over resilience and scalability.

Having a single deterministic gateway node produces a single point of failure for those nodes in the same cluster. Should that node become overloaded or disconnected from the overlay, the cluster it represented on the overlay would become an island, at least until a new gateway node was selected and that information was disseminated to other gateway nodes. These issues are detailed further in Section 2.3.

5.3.1 Linkage

Linkage is the term given to the heuristic used to aggregate clusters to create new tiers in the hierarchy. The linkage used determines the characteristics of clusters and has a significant bearing on the performance of N-casting. To that end there are three primary types of linkage that needs to be considered.

- Single-link clustering
- Complete-link clustering

- Average link clustering

Single link clustering aggregates clusters based on closest distance of nodes in clusters [162], resulting in aggregating clusters that are the closest distance to each other. Conversely, in complete-link clustering the clusters are merged based on the distance between the two furthest nodes in two clusters. Average-link clustering, sometimes called group average clustering, is seen as a 'half-way' compromise between single and complete-link clustering, merging clusters after evaluating every data point before aggregation.

The need for average-link clustering comes from the significantly differing behaviour of single and complete link clustering. The difference between the two is that single-link clustering the outliers in each cluster is not taken into account until the very highest tier of aggregation, meaning outliers can significantly alter the '*quality*' of clustering, whereas complete-link acknowledges outliers sooner, aggregates outliers at a far lower level, that may lead to smaller differences between hierarchies at higher tiers.
Even in a small data sample, the linkage methods result in differences in aggregation, as can be seen by the vertical lines on the dendogram. In such a graph is it important to look at the vertical lines as a measure of how big the step is between each tier, typically the y-axis in a dendogram is a similarity value, though in the case of Figure **??** it would be ping times in milliseconds.

Figure **??** shows single link clustering can lead to outliers being included at the very top level. For example the right most node is only aggregated at the second highest tier, meaning that taking a top down view, at the second tier there would be two clusters with a very large latency gap to the next tier. There are large gaps in latency for complete-link clustering (Figure **??**) and average link clustering (Figure **??**), though by visual inspection it is clear to see there is an intermediary tier that provides less of a jump.

The decision over which linkage type method to use boils down to how to handle outliers. In the case of an overlay clustered by nodes' distance, an outlier could be one of two things; a node that has a high ping time due to being geographically far away, and one that has a high ping time due to link congestion. Either way, ignoring outliers is simply not an option, at the very least, far away nodes are important to the resilience of an overlay but more immediate is the need to accommodate such nodes in order to facilitate access to services regardless of location. However, incorporating outliers at a very stage, such as in the case of complete-link clustering in Figure **??** could lead to 'wide' clusters - clusters that cover large geographic areas.

Ideally such a trait would be preferred at the higher tiers, providing a country or continental view of the overlay.

The ability to aggregate multiple lower-level clusters into a single cluster may add hops to the message path, however it allows nodes to have knowledge of a greater breadth of the overlay with fewer route updates and smaller routing tables. As overlay membership grows, instead of nodes keeping track of more clusters, the hierarchical clustering algorithm can either create more tiers to abstract the great numbers of clusters or clear one-to-many mappings between one node in a cluster pointing to many nodes in many clusters in lower tiers. This type of clustering is formally known as agglomerative hierarchical clustering.

5.4 Clustering in the N-casting overlay

In the N-casting overlay, cluster creation has three stages; the creation of higher tiers that contain stable nodes, the creation of leaf clusters using constrained clustering to create a balanced cluster size and the merging of leaf clusters into a multi-tier hierarchy that bounds the number of connections each N-casting overlay node needs to maintain.

The high-tier clusters in the N-casting overlay consists of nodes that represent continents and countries that respectively make up the first and second tiers from a top-down view of the overlay. The number of clusters in these two tiers are fixed, meaning there is no overhead associated with the reconfiguration of the hierarchy between nodes that are in different countries.

The low-level leaf clusters in the N-casting overlay are defined by using a constrained clustering algorithm [163] to balance the number of nodes assigned to each leaf cluster. This is important as nodes within a leaf cluster maintain full, unaggregated, information on every member.

Smaller countries that have a nominal number of nodes can be placed in a single leaf cluster and not split into further tiers. For countries where there is a larger number of nodes (USA, Japan, South Korea, Great Britain), the number of leaf clusters k is decided by dividing the total number of nodes in that country by the maximum size of a leaf cluster in nodes, which is set prior to the simulation. The constrained clustering algorithm is then applied to generate k clusters with a minimum number of nodes that is less than $\frac{n_{total}}{k}$ so that the algorithm can trade off the balance between cluster sizes and cluster similarity.

The trade off between cluster size and cluster similarity is an important one as large clusters may reduce inter-cluster delays, but nodes will be farther from the centroid of the cluster. However having small clusters will lead to large latency gaps between tiers in the hierarchy parts of the overlay so a balance needs to be met in order to have clusters of comparable similarity when nodes announce aggregated group membership data.

Outliers, in the form of nodes that are geographically far away from others such as those on an island, must be handled with care. If outliers are incorporated in an existing cluster then they could skew the intra-cluster distance, creating a very large cluster. However if outlier nodes are placed in their own cluster there could be a very large number of clusters and tiers, which will result in every N-casting overlay node requiring more entries in their routing tables and thus increasing protocol overhead. For this reason, outliers are included within other clusters but are given a small weighting to offset their distance from the centroid.

Mid-tier clusters are the clusters that are between the two top tiers representing continents and countries and the lowest level leaf clusters. The mid-tier clusters are formed using a hierarchical agglomerative clustering algorithm by using the centroids of leaf clusters. The hierarchical clustering in the N-casting overlay uses average linkage to create the dendogram.

To determine the number of tiers and the number of clusters per tier it is possible to calculate the number of connections an N-casting overlay node would have given a set a choices. Equation **??** has been taken from [164]. In a balanced tree where each cluster at tier *v* that has c_v number of clusters at tier $v+1$ the total number of connections is given by $connections = v_{total}(x-1)$ where v_{total} is the total number of tiers. For *k* leaf clusters generated using the previously described method, the tree will contain v_{total} tiers as calculated by;

$$\sum_{i=1}^{v} c_v^i = K \implies \frac{c_v - c_v^{v+1}}{1-c_v} = K \implies c_v^v = 1+k(1-\frac{1}{c_v}) \implies v = \frac{log\left(1+k\left(1-\frac{1}{c_v}\right)\right)}{log(c_v)} \tag{5.1}$$

c_v is calculated such that *connections* is bounded and *k* is minimised, which allows the control of the number of connections a N-casting overlay node needs to maintain with mid-tier clusters. In a bid to create to balanced tiers, some areas the overlay will have more tiers than others due to node density, which could mean that the actual number of connections is higher than the theoretical maximum.

Using this methodology on the dataset, the probability distribution function of the actual number of mid-tier connections in different continents was calculated with *connections* equal to 20. On average through the entire population the average number of mid-tier connections is 26 per node.

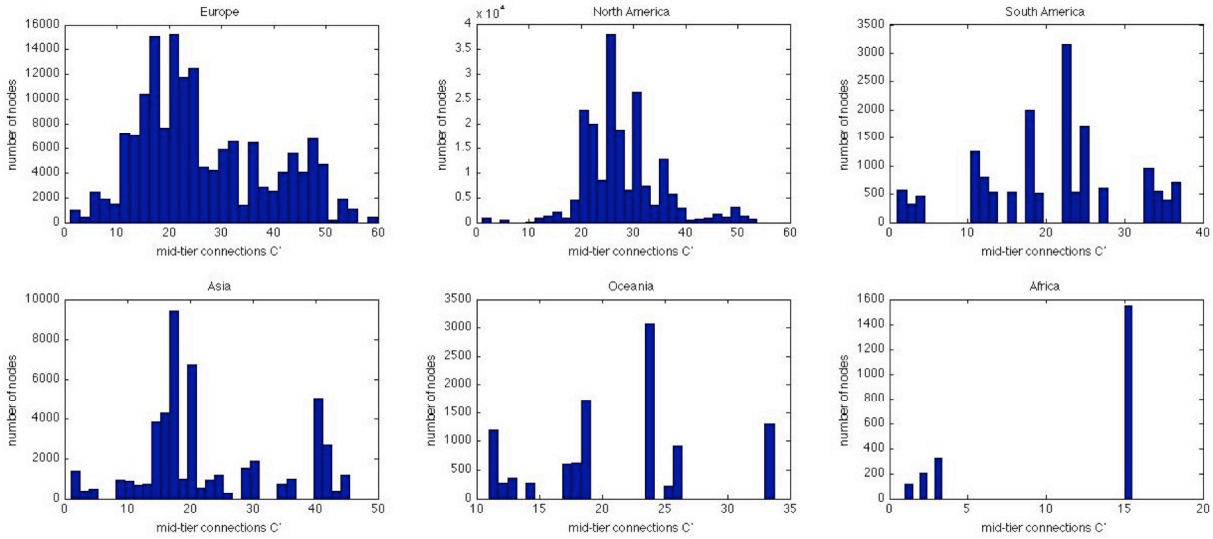


Figure 5.6: Probability distribution function of mid-tier calculations

Using equation 5.1 to calculate c_v, the algorithm starts at the top of the dendogram and creates a tier every c_v mergings, and traverses the tree for each created cluster. As an example using the dataset previously described (Section 5.2)), the full hierarchy for Australia using sample values of k = 94 and *connections* = 20 creates t = 2 middle tiers with a maximum of c_v = 13 clusters and leaf clusters spread at tier 3, 4 and 5.

5.5 Floating-point data structure

One of the key innovations in N-casting is the varying resolution a node has of group membership, with the idea being the node doesn't need the same level of accuracy as group membership gets larger. In order to implement this, a '*floating-point*' data structure is proposed.

To provide a variable resolution of group membership, the N-casting overlay employs a floating-point style mantissa and exponent data structure, where the mantissa provides a high resolution at low group membership, where the knowledge of a single node is important for reliable N-casting. For larger groups, through the use of incrementing the exponent, a predictable variable resolution that decreases as group membership increases.

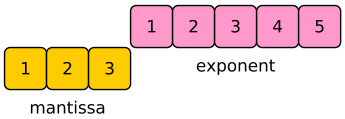


Figure 5.7: Floating-point data structure for group membership

By representing group membership using the simple data structure presented in Figure 5.7, it greatly simplifies the graceful degradation of group membership with distance. The use of a mantissa-exponent technique allows a predictable resolution degradation as group membership increases and in three lines of code, the node can calculate whether the group membership update needs to be pushed to other nodes.

Increasing the size of the mantissa will increase the resolution, however if the overall mantissa+exponent size is kept static then the highest number that can be represented decreases. Therefore the decision to use a mantissa of three and an exponent of five allows for the accounting of over two billion nodes in a single group. While there is seven or less group members, the data structure provides absolute accuracy, with only exponent changes being transmitted from then on.

As with all floating point numbers the exponent *exp* determines the scale of the number for a given base. Formally the highest possible representation ρ is determined by the base b as $b^{2^{exp}} \leq \rho < b^{2^{exp+1}}$ while the mantissa provides the precision between the range of ρ. Formally by splitting the range of ρ into equal $2^{\text{size of}m)}$ parts it is possible to determine what sub-range the number belongs to as presented in Equation 5.2.

$$b^{2^e} + m\frac{b^{2^{exp+1}} - b^{2^{exp}}}{2^{\text{size of}(m)}} + \leq \rho < b^{2^{exp}} + (m+1)\frac{b^{2^{exp+1}} - b^{2^{exp}}}{2^{\text{size of}(m)}} \tag{5.2}$$

Using Equation 5.2 with *m* = 3 bits and *exp* = 5 bits and *b* = 2, it is possible to represent 2^{2^5} or 4,294,967,296 group members. While that figure does not account for every single person in the world to be part of a single group, given the diversity of what groups represent (video streams, websites, virtual world slice), a single byte per group is deemed to be a large enough to accommodate popular groups yet maintain high resolution for less popular groups.

N-casting overlay nodes will transfer compressed pairs of GroupID:FloatingPointFormat($g_{members}$),

where group information would be aggregated in lower tiers to provide a 'compressed' view for higher tiers. Formally the estimation for the number of members in a group is $g_{members}$ = $\sum_{i \epsilon c_{aggregated}} n_i$ where c is the set of clusters to be aggregated, with n_i being the only FloatingPointFormat of numbers available.

As the clusters are disjoint, double counting is eliminated. That is to say, a node cannot appear in two clusters.

5.6 Simulation

For experiments, a discrete event simulator was programmed in Java. The simulator takes an input file that is generated by an external clustering program to generate an overlay based on the protocol previously mentioned. Once the simulator has created the overlay, it enters a loop to simulate an epoch in the life of the overlay. An epoch is defined as one minute of time and is a parameter set at runtime. The simulation terminates once the simulator completes the final epoch, meaning simulation occurs for a predetermined time in minutes.

As the simulator is designed to mimic an overlay deployed on the public Internet, in each epoch there are nodes joining and leaving N-casting groups. For the purposes of simulation, the overlay membership is static and is a system parameter with the routing tables and message passing in one group is simulated, while the other clusters and groups are calculated but no message passing takes place.

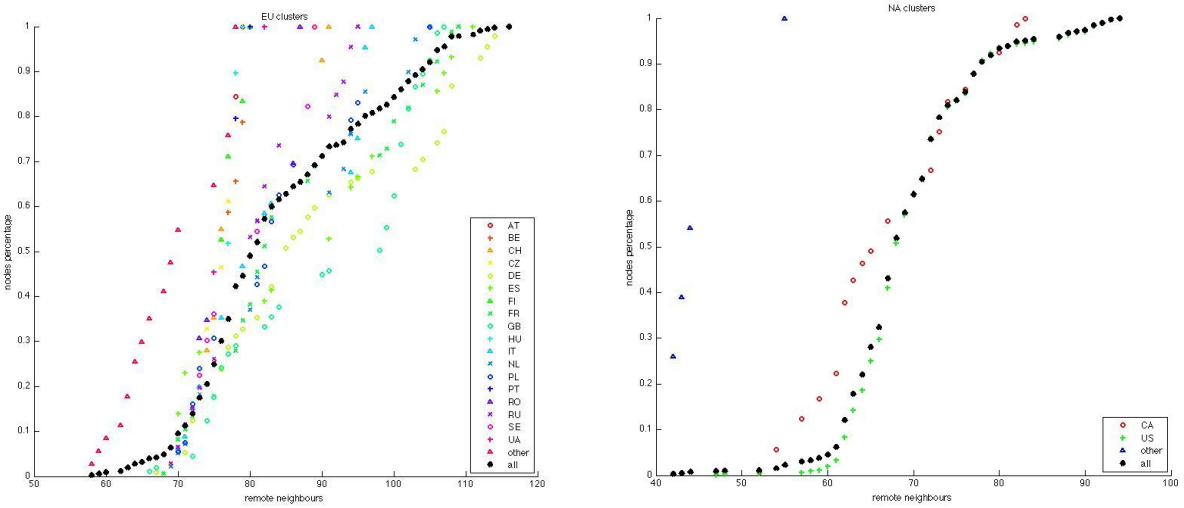
To simulate realistic group join and departure rates, probability distributions were used. Group join rates per epoch were calculated based on a Poisson distribution with a λ that is related to the size of the group. For the simulated group, the nodes are generated and join the group, passing messages to other group members to generate its routing table. A Poisson distribution as used by Terpstra [165], Rhea [17] with other distributed networks exhibiting similar node arrival patterns as recorded by Krishnamurthy [166] in Chord.

Node session lengths follow a Weibull distribution, a behaviour that was observed by Stutzbach [73]. Nodes that leave the group are still connected to the overlay and can in the future re-join the group. Nodes that have been a member of the group in previous epochs and left have the same probability as nodes that have never joined the group when it comes to new node joins.

The simulator uses epochs to denote time and based on previous research, an average session length for an N-casting overlay node was set at 21 minutes, the average session length measured by Stutzbach [73], meaning a single epoch is equivalent to 21 minutes of 'real world' time. In order to simulate run-times, the notion of an epoch window was introduced, which represents a division of the 21 minutes. An epoch window of 1 minute is equal to 21 minutes, while a 2 minute epoch window represents 11.5 minutes. Formally, run time $= \frac{21}{\text{epoch window}}$.

5.7 Results

Following the work presented in clustering in Section 5.3 and its importance to the performance of the N-casting overlay, a breakdown of how the number of clusters in a continent is shown for Europe (Figure 5.8(a)) and North America (Figure 5.8(b)).



(a) Europe

(b) North America

Figures 5.8(a) and 5.8(b) show that both in Europe and North America respectively the number of neighbours needed by a node to achieve coverage of a continent rises sharply after a slow start. The reason why both graphs do not start at zero is because the node needs to know neighbours in the five other continents, therefore the baseline, in the case of Europe is 58 while for the North America experiment it is 42.

For the continent of Europe (Figure 5.8(a)) there is a steep increase in coverage, rising from 10 percent to 60 percent by adding 12 neighbours. From this point the increase in coverage is steady with the increase in neighbours, however in countries where there were fewer nodes in

the dataset, such as Hungry, Portugal and Switzerland, complete coverage is attained sooner. Countries such as France, Germany and Great Britain, where the dataset is considerably richer, requires more neighbours to achieve complete coverage.

The reason for this disparity in the number of clusters increases with the amount of nodes as was shown on a continent scale in Figure 5.6 and Table 5.2, meaning there will be more messages being sent to achieve complete coverage of these particular continents.

In North America, for the purposes of readability, the graph has broken down the continent into the United States and Canada with Caribbean Islands and Mexico being classified as 'other'. As the dataset has more North American nodes than any other country it should provide an accurate view of how many neighbours a node needs to have coverage of the country.

Initially the relationship between neighbour knowledge and coverage doesn't scale. This could be explained by the majority of the country's population being located at opposite coasts, meaning the clusters that represent the geographical middle of the United States do not provide much overall coverage of the country. Further credence is given to this theory as the coverage rises steeply going from 4 percent to over 90 percent with 20 neighbours being added.

In order to investigate the number of messages sent to a group, the 'popularity' of a group was varied. The popularity is defined as the number of nodes of a particular cluster that are group members. A real world example of this would be N-casting overlay nodes that has data on a particular television channel or players in a particular area of a virtual world.

The cluster and node chosen as a measurement point was done so at random. The results aim to show the number of messages being received by a single node as a sign of how scalable the N-casting overlay is as group membership increases.

The tail in each run, visible in all runs in Figure 5.8 is due to the start-up phase of the simulator, where nodes are added sequentially and routing tables are populated. The effect of this start up phase is an increased number of messages being recorded before reaching a steady state. The effect of this on the number of messages recorded above the steady state is more pronounced as group popularity increases.

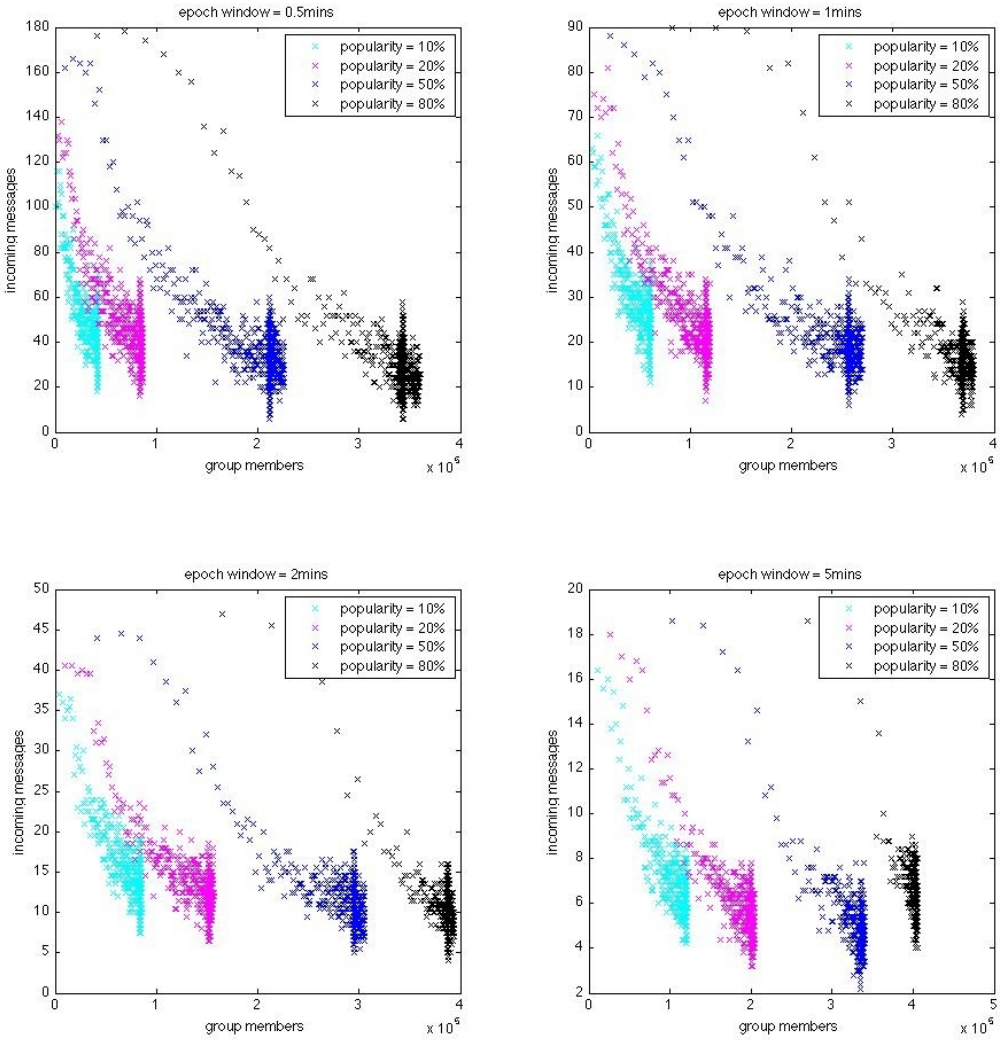


Figure 5.8: N-casting Overlay messages as popularity changes

Figure 5.8 shows the number of incoming messages for a single node from all its neighbours in both local and remote clusters. As the epoch window length is increased, the number of messages decreases regardless of the popularity. The abrupt stops on each run is due to hitting the artificial limit imposed by the popularity constraint, meaning for a given popularity there will not be more than a subset of the cluster membership as group members.

Regardless of group popularity, the number of messages decreases as group membership nears its artificially imposed limit. The variance in the number of messages at the popularity limit is an artifact of the epoch window size as smaller snapshots do not show the system nearing a steady state. As the epoch window is increased, it can clearly be seen that at the variance in messages at the membership limit is lower.

For a given epoch window, the number of messages settle close to the same level at the bounded group membership. This strongly shows that as group sizes increase the protocol scales well, not requiring nodes to handle an ever growing number of messages from nodes in remote clusters.

5.8 Conclusion

The N-casting overlay presents a number of innovative features, including a hierarchical decomposition of the Internet, an efficient data structure and the ability to send messages to multiple members of a group. The overlay brings together an efficient node discovery protocol, which promotes resilience and scalability.

Experiments were conducted using is rich dataset collected using the TurboKing method for clustering. Through the use of hierarchical clustering, experiments have shown that coverage of large, highly populated continents can be achieved with a reasonable number of neighbours. The ability to abstract locality into clustering results in both locality and the ability to lower the number of messages sent to maintain group membership knowledge.

The results presented show that as group popularity increases, the number of messages being sent does not significantly increase, meaning the N-casting overlay is scalable. The use of the floating point data structure ensures compression of membership data with the protocol limiting the number of updates being sent, which as experiments have shown helps the N-casting overlay's scalability.

Through the use of simulations that model realistic group churn, it has been shown that the N-casting overlay can provide a scalable and dependable N-casting service.

Chapter 6

Conclusion and future work

The work presented here highlights the advantages of locality-aware in overlays in both structured and unstructured overlays. All three contributions focus on the area of Anycast, where there exists great scope for contributions that look at efficient service or content discovery in P2P networks.

P2P overlays bring considerable challenges when it comes to protocol design due to the lack of well provisioned, high availability servers. While P2P overlays can provide resilience due to their distributed nature and geographical disparity, the aim is to harness the advantages of P2P overlays while using underlying characteristics of the network in order to provide close to optimal path message routing.

In Section 1.1, the goal is described as creating a resource efficient overlay, where resource was defined as message passing and latency. Three overlays have been presented that tackle the issue of message passing and latency, while keeping a close eye on overlay maintenance costs and with the use of innovative data structures, scalability was also achieved.

DOAT, presented in Chapter 3 is a structured overlay where the notion of locality was introduced to an overlay through the use of space filling curves. The experiments conducted showed the space filling curve allowed messages to be routed efficiently to nodes that were physically close (Figure 3.7). The use of Bloom filters to store routing tables and group membership data provided an efficient way of storing large amounts of data, with accuracy able to be varied as a system parameter.

Structured overlays, while providing deterministic and predictable performance, can be susceptible to high overlay churn. Although work has been done to mitigate the effects of churn

on the performance degradation in structured networks, due to the design of the overlay and consequently routing policies being predetermined prior to overlay construction, flexible probabilistic unstructured overlays that could maintain performance in environments that exhibit churn was proposed.

Unstructured overlays promote a probabilistic method of creating an overlay and while resilience to membership churn can be improved, there could be performance degradation in terms of message routing if locality is not taken into consideration. To introduce locality in an unstructured overlay, GOAT presented in Chapter 4 is an unstructured overlay that makes use of expanding rings in node discovery to bring the notion of locality to the overlay.

GOAT showed unstructured, locality aware overlays can be viable through the use of techniques presented in DOAT, such as Bloom filters for group membership storage and the use of landmarking and expanding rings. While GOAT showed the viability of unstructured overlays being used to route messages efficiently, the N-casting overlay presented in Chapter 5 showed how an unstructured overlay can cope in churn environments while introducing further innovations, including the ability to send requests to $n+1$ nodes in a group.

DOAT and GOAT both presented overlays that would return the requesting node with a single member of a group, if such a member existed. However there are times when a node may want to know of more than one member in a group, for example to aggregate multiple nodes' upstream bandwidth in order to receive a file. This has been defined as N-casting, where $n > 1$, though N-casting can also work for cases where $n = 1$. Facilitating N-casting requires a finer grain knowledge of group membership, not just whether a group has a member or not.

Further contributions include a move away from Bloom filters for group membership information, with the introduction of an simple and elegant floating-point representation that allows for membership knowledge to have varying resolutions depending on group membership. While total group membership with absolute accuracy is the best possible scenario, the practicalities of this makes it unrealistic, and therefore the solution proposed here is simple, elegant and allows for nodes to maintain knowledge of a very large number of groups without significant negative impact on its resources or that of the overlay.

With N-casting, a number of important features were introduced to unstructured overlays for

location awareness and scalability. The N-casting overlay is created following a hierarchical decomposition of the Internet, where nodes are clustered based on their distances between each other. This forgoes the need for external systems such as Vivaldi, creating a self-contained location aware overlay. The N-casting overlay brought hierarchical clustering, an elegant datastructure to minimise overlay maintenance costs and a protocol that was shown to scale with group membership.

DOAT, GOAT and N-casting have a number of real world use cases that can exploit each overlay's characteristics. DOAT is particularly well suited where nodes have long session lengths, such as servers or nodes placed on an ISP's core network to provide service discovery and content delivery networks. GOAT, through its probabilistic routing is more suited to unstable overlays where nodes have shorter session lengths such as P2P overlays to distribute content.

N-casting's ability to decompose the Internet and provide a variable resolution overlay means it is particularly well suited to virtual worlds, as described in Appendix A. N-casting can also be used in high churn environments to provide localised content delivery.

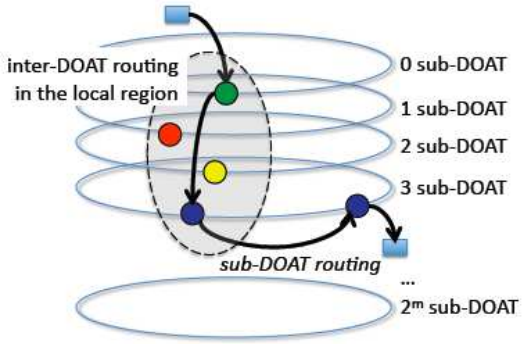
Work presented in DOAT, GOAT and N-casting show that it is possible to introduce locality in scalable P2P overlays that can maintain group knowledge membership. Through the use of space-filling curves, expanding rings and hierarchical clustering, it has been shown that overlay nodes can be made to respect geographical locality in order to efficiently route messages. By using innovative data structures such as Bloom filters and a floating-point representation for group membership, scalability was achieved.

6.1 Future work

6.1.1 Hierarchical DOAT - D^{2}OAT

Hierarchical structured overlays have been described as a way of distributing expense and sandboxing of failures. Intending to leverage the advantages displayed by this design paradigm DOAT (Chapter 3) can be modified to work on multi-tiered overlays in order to lower the demands on a particular node allowing for a greater number of nodes within the network to become DOAT nodes.

Figure 6.1(b) displays an early design of what a D^{2}OAT may look like along with a node's routing table. At this stage a very basic protocol is presented as a starting point for future work



(a) D^{2}OAT overlay

sub-doat (m)	DOAT Distance	DOAT Coordinate	NodeID
0	0.01	0.35	id044
0	0.04	0.40	id42
0	0.20	0.16	id45
1	0.01	0.37	id2
1	0.03	0.39	id222
1	0.05	0.31	id80
⋮	⋮	⋮	⋮
2^m	⋮	⋮	⋮

(b) D^{2}OAT routing table

Figure 6.1: The D^{2}OAT overlay and a node's inter DOAT routing table

on D^{2}OAT.

Each node maintains a subset of Bloom filters and each group identifier is split into m+n bits. A sub-level DOAT is created for each value of m maintaining the routes for the group identifiers with the hash value of m. The formation of the sub-DOAT overlay is the same as a non-hierarchical DOAT however nodes maintain knowledge of other nodes in order sub-DOATs with each node knowing of at least one node in another level indexed by m.

The node which sends a query has to first find its closest D^{2}OAT node in any sub-DOAT. This node, based on the first m bits of the group identifier in the query finds the corresponding sub-DOAT from the contents of its inter-DOAT routing table. It then forwards the query to the corresponding sub-DOAT and routing follows the usual DOAT procedure.

Work on determining an efficient protocol for nodes to join and maintain the overlay needs to be done, while considering how churn can affect the performance of a hierarchical deterministic overlay. Hierarchical overlays has been shown to be a viable proposition in distributed Anycast with N-casting in Chapter 5, and D^{2}OAT needs to be compared to its probabilistic rival.

6.1.2 Replica placement

The placement of replicas was discussed in Section 2.3.1 as an important tool in mitigating the negative performance effects of churn. The N-casting overlay has been simulated with realistic churn models however it does not implement replication, something that could increase its performance. By incorporating replication in N-casting, it could allow for group members being reachable in fewer queries and perhaps more importantly improved accuracy during periods of

group membership change.

Node placement in the DOAT, GOAT and N-casting overlays is based solely on geographical proximity, however in the case of replication other placement strategies could be investigated. Replication strategies based on overlay characteristics such as demand and group membership churn with the notion on dynamic positioning of nodes in possible problem areas.

With group member nodes being positioned in different parts of the overlay depending on characteristics other than geographical locality, mechanisms for the efficient transfer of nodes to, in the case of N-casting, different clusters needs to be considered. Replica nodes for a particular group may be in one cluster, whereas they may be normal group members in other groups present in a different cluster entirely.

Work on efficient replica placement needs to demonstrate whether employing polices will affect the operation and scalability of the overlay. The amount of replication and when it needs to occur should also be investigated. Questions over the popularity threshold before group members from other clusters become replicas must be answered, but perhaps the most challenging work will come in determining at what time does replication occur if it is dependant on churn.

Questions need to be answered over what monitors group membership churn and for how long. Once again a balance will need to be struck in order to preserve overlay scalability while enforcing the advantages of replication. Then there is the question of whether the number of replicas vary as a function of group membership churn, and if this relationship is a linear one.

Replication is an important mechanism that can mitigate the ill-effects of churn. However implementing an efficient way in an overlay such as N-casting is a considerable challenge and one that should be investigated.

6.1.3 Virtual worlds

Distributed Anycast overlays such as DOAT, GOAT and N-casting could be deployed on a number of use-cases. One such is near-live video streaming as discussed in previous chapters, however another challenging scenario is in the area of virtual worlds.

An initial literature review of virtual worlds is presented in Section A.1, and highlights the challenges of deploying a distributed overlay in this area. However an N-casting capable over-

lay is well suited to virtual worlds in which there are regions of interest. Using a hierarchical decomposition of the Internet, similar to that presented in Section 5.3, a virtual world's areas of interest can be represented.

As explained in Section A.1.4, avatars in virtual worlds do not require global overlay membership view at a linear location, meaning objects or avatars in the virtual world that are farther away from the node do not require the same resolution as those close to it. The N-casting overlay presented in Chapter 5 already provides a variable resolution of global overlay membership with the use of hierarchical clustering and an elegant '*floating point*' data structure.

In virtual worlds, an overlay would be required to provide accurate results, in that the users will expect to have low latency with those nodes it interacts with. Aside from the latency challenges, the overlay must deal with consistency as explained in Section A.1.3, meaning the reliable dissemination of information in a particular group to other groups in the overlay. Not only does the node's view have to be accurate in order to achieve integrity in the view presented to the user but it has to maintain state, meaning actions that occurred previously may have a bearing on the current view presented to nodes. Such overlay maintenance requirements require investigation in order to maintain scalability as the overlay's membership grows and as time passes.

Bibliography

[1] G. Bates and V. Miller. The effects of the thermal environment on health and productivity. In *8th International Occupational Hygiene Association Conference, Pilanesburg*, 2005.

[2] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. In *Proceedings of Peer-to-Peer Computing 2003*, 2002.

[3] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, San Diego, California, United States, 2001.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001*, page 12, San Diego, California, 27/8/2001.

[5] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, number 3-540-42800-3 in Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[6] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[7] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 258:263, 2002.

[8] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. *Peer-to-Peer Systems II*, pages 160–169, 2003.

[9] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 73–84. ACM, 2005.

[10] N. Chang and M. Liu. Revisiting the ttl-based controlled flooding search: optimality and randomization. *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 85–99, 2004.

[11] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, 2002.

[12] N. B. Chang and M. Liu. Controlled flooding search in a large network. *IEEE/ACM Transactions on Networking*, 15(2):436–449, 2007.

[13] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, 2003.

[14] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[15] D. Kato and T. Kamiya. Evaluating dht implementations in complex environments by network emulator. 2007.

[16] Various. Gnutella - http://www.gnutella.com. *N/A*, NaN, NaN.

[17] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. *Proceedings of the USENIX Annual Technical Conference*, pages 127–140, 2004.

[18] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. *Proc. IPTPS*, pages 87–99, 2004.

[19] Y. Liu, X. Liu, L. Xiao, LM Ni, and X. Zhang. Location-aware topology matching in p2p systems. In *Proceedings of IEEE INFOCOM*, volume 4, 2004.

[20] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, 2002.

[21] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a dht for low latency and high throughput. *Proceedings of NSDI*, 4:85–98, 2004.

[22] H. Zhang, A. Goel, and R. Govindan. Incrementally improving lookup latency in distributed hash table systems. *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 114–125, 2003.

[23] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242. ACM, 2002.

[24] A. Ganesh, A.M. Kermarrec, and L. Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. *Networked Group Communication*, pages 44–55, 2001.

[25] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. *Proceedings of Multimedia Computing and Networking*, page 152, 2002.

[26] I. Ullah, G. Doyen, G. Bonnet, and D. Gaiti. A survey and synthesis of user behavior measurements in p2p streaming systems. *Communications Surveys & Tutorials, IEEE*, 14(3):734–749.

[27] P. Dhungel, X. Hei, D. Wu, and K.W. Ross. A measurement study of attacks on bittorrent seeds. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

[28] S. Agarwal, J.P. Singh, A. Mavlankar, P. Baccichet, and B. Girod. Performance and quality-of-service analysis of a live p2p video multicast session on the internet. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pages 11–19. IEEE, 2008.

[29] Salvatore Spoto, Rossano Gaeta, Marco Grangetto, and Matteo Sereno. Analysis of pplive through active and passive measurements. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–7. IEEE, 2009.

[30] N. Hopper, E.Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[31] M. Steiner and E. Biersack. Where is my peer? evaluation of the vivaldi network coordinate system in azureus. *NETWORKING 2009*, pages 145–156, 2009.

[32] Y. Sun and S. Marcos. High frequency trading and server locations. *ISSN 1931-0285 CD ISSN 1941-9589 ONLINE*, page 592, 2012.

[33] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive p2p systems. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, number 978-1-60558-594-9 in SIGCOMM '09, pages 315–326, New York, NY, USA, 2009. ACM.

[34] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *Peer-to-Peer Systems*, pages 85–93, 2002.

[35] D. Ciullo, M.A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: a measurement study. *IEEE Transactions on Multimedia*, 12(1):54–63, 2010.

[36] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[37] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172. ACM, 2001.

[38] M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, and M. Van Steen. Gossip-Based Peer Sampling. *ACM Transactions on Computer Systems*, 10:1275517–1275520, 2007.

[39] R. Baraglia, P. Dazzi, B. Guidi, and L. Ricci. Godel: Delaunay overlays in p2p networks via gossip.

[40] G. Dán, N. Carlsson, and I. Chatzidrossos. Efficient and highly available peer discovery: A case for independent trackers and gossiping. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 290–299. IEEE, 2011.

[41] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[42] S. Voulgaris and M. Van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer. In *In Proceedings of the 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management,(DSOM 2003), number 2867 in Lecture Notes in Computer Science*, 2003.

[43] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *IEEE Computer Society; 1999*, volume 24, pages 102–111, 2004.

[44] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P '01)*, 2001.

[45] P. Eugster, P. Felber, F. Le Fessant, et al. The art of programming gossip-based systems. *Operating Systems Review*, 2007.

[46] M. Jelasity, R. Guerraoui, A. M. Kermarrec, and M. van Steen. Gossip-based unstructured overlay networks: An experimental evaluation. Technical report, Technical Report 2003-15, University of Bologna, Department of Computer Science, Dec. 2003, http://ftp.cs.unibo.it/pub/UBLCS/2003/2003-15.pdhttp://ftp.cs.unibo.it/pub/UBLCS/2003/2003-15.pdff.

[47] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. *Symposium on Foundations of Computer Science*, 41:565–574, 2000.

[48] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 1, 2004.

[49] S. Datta and H. Kargupta. Uniform data sampling from a peer-to-peer network. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, page 50. IEEE Computer Society, 2007.

[50] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking (TON)*, 17(2):377–390, 2009.

[51] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087, 1953.

[52] Asad Awan, Ronaldo A. Ferreira, Suresh Jagannathan, and Ananth Grama. Distributed uniform sampling in unstructured peer-to-peer networks. *Hawaii International Conference on System Sciences*, 9:223c, 2006.

[53] N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, page 246. ACM, 2007.

[54] L. Lovász. Random walks on graphs: A survey. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.176.6453*, 1993.

[55] K. Horowitz and D. Malkhi. Estimating network size from local information. *Information Processing Letters*, 88(5):243, 2003.

[56] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. *IEEE/ACM Transactions on Networking (TON)*, 16(2):267–280, 2008.

[57] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical Review E*, 64(4):46135, 2001.

[58] N. Ning, D. Wang, Y. Ma, J. Hu, J. Sun, C. Gao, and W. Zheng. Genius: Peer-to-peer location-aware gossip using network coordinates. *Computational Science–ICCS 2005*, pages 133–292, 2005.

[59] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.

[60] J. Zhao and J. Lu. Solving overlay mismatching of unstructured p2p networks using physical locality information. In *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 75–76. IEEE, 2006.

[61] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang. Location-aware topology matching in p2p systems. In *INFOCOM 2004*, volume 4, pages 2220–2230. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 2004.

[62] B. Wong, A. Slivkins, and E.G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 85–96. ACM, 2005.

[63] M. Jelasity, W. Kowalczyk, and M. Van Steen. Newscast computing. Technical report, 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.5411.

[64] A. M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 248–258, 2003.

[65] P.T. Eugster, R. Guerraoui, SB Handurukande, P. Kouznetsov, and A.M. Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems (TOCS)*, 21(4):341–374, 2003.

[66] M.J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. In *Proceedings of 14th International Symposium on DIStributed Computing (DISC 2000)*, pages 253–267, 2000.

[67] L.W. Beineke and F. Harary. The connectivity function of a graph. *Mathematika*, 14(02):pages 197–202, 1967.

[68] M.J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, pages 364–379. Springer-Verlag London, UK, 1999.

[69] M. Castro, M. Costa, and A. Rowstron. Should we build Gnutella on a structured Overlay. 2003.

[70] L. Massoulie, E. Le Merrer, A.M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Annual ACM Symposium on Principles of Distributed Computing: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*. Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA,, 2006.

[71] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. *Proceedings of the eleventh international conference on Information and knowledge management*, pages 300–307, 2002.

[72] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[73] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. *Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 189–202, 2006.

[74] F. E. Bustamante and Y. Qiao. Friendships that last: peer lifespan and its role in p2p protocols. *Proceedings of the 8th international workshop on Web caching and distribution*, 2004.

[75] Carlos M. Gutierrez and Patrick Gallagher. Secure hash standard. 2008. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.3809.

[76] R. Rivest. Rfc1321: The md5 message-digest algorithm. *Internet RFCs*, 1992. http://www.ietf.org/rfc/rfc1321.txt.

[77] I. García, S. Lefebvre, S. Hornus, and A. Lasram. Coherent parallel hashing. *ACM Transactions on Graphics (TOG)*, 30(6):161, 2011.

[78] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, 1997.

[79] C. Gross, D. Stingl, B. Richerzhagen, A. Hemel, R. Steinmetz, and D. Hausheer. Geodemlia: A robust peer-to-peer overlay supporting location-based search. In *Peer-to-Peer Computing (P2P), 2012 IEEE International Conference on*. IEEE, 2012.

[80] Y.J. Joung, C.T. Fang, and L.W. Yang. Keyword search in dht-based peer-to-peer networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, 2005. (ICDCS 2005)*, pages 339–348, 2005.

[81] D. Carra, M. Steiner, and P. Michiardi. Adaptive load balancing in kad. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 92–101. IEEE, 2011.

[82] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proc. First Symposium on Networked Systems Design and Implementation*, 2004.

[83] L.R. Monnerat and C.L. Amorim. D1HT A Distributed One Hop Hash Table. In *Proceedings of the 20th IEEE International Parallel Distributed Processing Symposium (IPDPS)*, 2006.

[84] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology table of contents*, pages 293–304. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1994.

[85] X. Wang, D. Feng, X. Lai, and H. Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. *Short talk presented at CRYPTO 2004*, 4, http://web.mit.edu/fustflum/documents/crypto.pdf.

[86] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA,, 1997.

[87] Moni Naor and Udi Wieder. Know thy neighbor’s neighbor: better routing for skip-graphs and small worlds. In *Proceedings of the Third international conference on Peer-to-Peer Systems*, IPTPS'04, pages 269–277, Berlin, Heidelberg, 2004. Springer-Verlag.

[88] C.C. Wang and K. Harfoush. Shortest-path routing in randomized dht-based peer-to-peer systems. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 52(18):3307–3317, 2008.

[89] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. *Future directions in distributed computing*, pages 103–107, 2003.

[90] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical report, Microsoft Research, 2003. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.4319.

[91] Z. Xu and Z. Zhang. Building low-maintenance expressways for p2p systems. *Hewlett-Packard Labs, Palo Alto, CA, Tech. Rep. HPL-2002-41*, 2002. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.3751.

[92] Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, and Y. Q. Zhang. Peer-to-peer based multimedia distribution service. *IEEE Transactions on Multimedia*, 6(2):343–355, 2004.

[93] X. Zhang, J. Liu, B. Li, and T. S. P. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE INFOCOM*, volume 3, pages 13–17, 2005.

[94] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. *Future directions in distributed computing*, pages 103–107, 2003.

[95] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of INFOCOM 2002*, volume 3, 2002.

[96] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *23rd International Conference On Distributed Computing Systems*, pages 500–508, 2003.

[97] Z. Xu, R. Min, and Y. Hu. Hieras: A dht based hierarchical p2p routing algorithm. In *Proceedings of the International Conference on Parallel Processing*, pages 187–196, 2003.

[98] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in g major: designing dhts with hierarchical structure. In *24th International Conference on Distributed Computing Systems, 2004*, pages 263–272, 2004.

[99] L. Garcés-Erice, E. Biersack, P. Felber, K. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. *Euro-Par 2003 Parallel Processing*, pages 1230–1239, 2003.

[100] M. S. Artigas, P. G. Lopez, J. P. Ahullo, and A. F. G. Skarmeta. Cyclone: a novel design schema for hierarchical dhts. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pages 49–56, 2005.

[101] S. Zoels, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical dht design. In *Proceedings of the 6th International Conference on Peer-to-Peer Computing, Cambridge, UK*, 2006.

[102] M. S. Artigas, P. G. Lopez, and A. F. Skarmeta. A comparative study of hierarchical dht systems. In *32nd IEEE Conference on Local Computer Networks. LCN 2007*, pages 325–333, 2007.

[103] J. Li, J. Stribling, R. Morris, M. F. Kaashock, and T. A. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. *Proceedings of IEEE INFOCOM*, 1, 2005.

[104] K.P. Gummadi, H.V. Madhyastha, S.D. Gribble, H.M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, pages 13–13. USENIX Association, 2004.

[105] R.E. Brown and J.G. Koomey. Electricity use in california: past trends and present usage patterns. *Energy Policy*, 31(9):849–864, 2003. http://enduse.lbl.gov/info/LBNL-47992.pdf.

[106] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. pages 256–267, 2003. http://www.springerlink.com/content/ehcfgw36n3j1ypr6/.

[107] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. Watching television over an ip network. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 71–84. ACM, 2008.

[108] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[109] P.B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–158. ACM New York, NY, USA, 2006.

[110] W. Terpstra, J. Kangasharju, C. Leng, and A. Buchmann. Bubblestorm: Resilient, probabilistic and exhaustive p2p search. In *Proceedings of ACM SIGCOMM*, Kyoto, Japan, 2007.

[111] S. Tewari and L. Kleinrock. Analysis of search and replication in unstructured peer-to-peer networks. *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 404–405, 2005.

[112] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[113] T. Hardie. Rfc3258: Distributing authoritative name servers via shared unicast addresses. *Internet RFCs*, 2002. http://tools.ietf.org/html/rfc3258.

[114] Dina Katabi and John Wroclawski. A framework for scalable global ip-anycast (gia). In *Proceedings of the SIGCOMM 2000*, SIGCOMM '00, pages 3–15, Stockholm, Sweden, 2000. ACM.

[115] T. Stevens, T. Wauters, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester. Analysis of an anycast based overlay system for scalable service discovery and execution. *Computer Networks*, 54(1):97–111, 2010.

[116] S. Yu, W. Zhou, and M. Chowdhury. Quality-of-service routing for web-based multimedia servers. In *Proceedings of the 7th Joint Conference on Information Sciences: September 26-30, 2003, Research Triangle Park, North Carolina, USA*, pages 1349–1353. Association for Intelligent Machinery, 2011.

[117] F. Weiden and P. Frost. Anycast as a load balancing feature. In *Proceedings of the 24th international conference on Large installation system administration*, pages 1–6. USENIX Association, 2010.

[118] L. Andel, J. Kuthan, and D. Sisalem. Distributed media server architecture for sip using ip anycast. In *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, page 5. ACM, 2009.

[119] H.A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe. A practical architecture for an anycast cdn. *ACM Transactions on the Web (TWEB)*, 5(4):17, 2011.

[120] X. Qin, L. Taoshen, and G. Zhihui. A qos anycast routing algorithm based on genetic algorithm and particle swarm optimization. In *Genetic and Evolutionary Computing, 2009. WGEC'09. 3rd International Conference on*, pages 125–128. IEEE, 2009.

[121] Y. Zhang, Y. Zhang, and Z. Yu. Anycast qos routing based on clone strategies. *Jisuanji Gongcheng yu Yingyong(Computer Engineering and Applications)*, 47(35), 2011.

[122] L. Fan and L. Taoshen. Implementation and performance analyses of anycast qos routing algorithm based on genetic algorithm in ns2. In *Information and Computing Science, 2009. ICIC'09. Second International Conference on*, volume 3, pages 368–371. IEEE, 2009.

[123] M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. *Group Communications and Charges. Technology and Business Models*, pages 47–57, 2003.

[124] Z. M. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response timeof a replicated service. *Proceedings of IEEE INFOCOM'98*, 2, 1998.

[125] C. Partridge, T. Mendez, and W. Milliken. Rfc 1546: host anycasting service. 1993.

[126] I. Avramopoulos and M. Suchara. Protecting the dns from routing attacks: Two alternative anycast implementations. *Security & Privacy, IEEE*, 7(5):14–20, 2009.

[127] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, and V. Shah. Application-layer anycasting. *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 3, 1997.

[128] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 8(4):455, 2000.

[129] H. Ballani and P. Francis. Towards a global ip anycast service. *ACM SIGCOMM Computer Communication Review*, 35(4):301–312, 2005.

[130] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[131] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz. Analyzing stability in wide-area network performance. *Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 2–12, 1997.

[132] J. Gwertzman and M. Seltzer. The case for geographical push-caching. *Proceedings of the 1995 Workshop on Hot Operating Systems*, pages 51–55, 1995.

[133] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–86, 2002.

[134] Casey Carter, Seung Yi, Prashant Ratanchandani, and Robin Kravets. Manycast: exploring the space between anycast and multicast in ad hoc networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 273–285, New York, NY, USA, 2003. ACM.

[135] C. Zhu and H. Liu. Research on k-anycast routing schemes for ipv6. In *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pages 330–333. IEEE, 2010.

[136] CR Dow, PJ Lin, SC Chen, and SF Hwang. An efficient anycast scheme for discovering k services in cluster-based mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 11(9):1287–1301, 2010.

[137] Y. Zhang, X. Jiang, and Z. Lin. K-anycast dtn routing in predictable environment. In *Electrical and Control Engineering (ICECE), 2010 International Conference on*, pages 5296–5299. IEEE, 2010.

[138] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):100, 2002.

[139] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding close friends on the internet. In *Ninth International Conference on Network Protocols*, pages 301–309. IEEE, 2001.

[140] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng. Anysee: Peer-to-peer live streaming. In *Proceedings of IEEE INFOCOM*, volume 6, pages 1–10, 2006.

[141] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao. Flash crowd in p2p live streaming systems: Fundamental characteristics and design implications. *Parallel and Distributed Systems, IEEE Transactions on*, 23(7):1227–1239, 2012.

[142] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, pages 124–141, 2001.

[143] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. *Discrete Applied Mathematics*, 117(1-3):211–237, 2002.

[144] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *Proceedings of NSDI, Cambridge, MA, April*, 2007.

[145] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[146] B. Leong and J. Li. Achieving one-hop dht lookup and strong stabilization by passing tokens. In *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, 2004.

[147] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking (TON)*, 10(5):604–612, 2002.

[148] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[149] P. Gummadi, Saroiu Stefan, and D. Gribble Steven. King: estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002.

[150] E. Mykoniati, R. Landa, S. Spirou, R. Clegg, L. Latif, D. Griffin, and M. Rio. Scalable peer-to-peer streaming for live entertainment content. *IEEE Communications*, 2008. http://eprints.ucl.ac.uk/14961/1/14961.pdf.

[151] E. Mykoniati, L. Latif, R. Landa, B. Yang, R. Clegg, D. Griffin, and M. Rio. Distributed overlay anycast tables using space filling curves. In *INFOCOM Workshops 2009, IEEE*, pages 1–6. IEEE, 2009.

[152] R. G. Clegg, D. Griffin, R. Landa, E. Mykoniati, and M. Rio. The performance of locality-aware topologies for peer-to-peer live streaming. In *UK Performance Engineering Workshop*, 2008.

[153] C. Lumezanu, D. Levin, and N. Spring. PeerWise discovery and negotiation of faster paths. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, 2007.

[154] J Michael Steele. *The Cauchy-Schwarz master class: An introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

[155] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in internet routing overlays. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 467–480. USENIX Association, 2009.

[156] R.R.C. Bikram and V.M. Vokkarane. Dynamic load-balanced manycasting over optical burst-switched (obs) networks. In *Conference on Optical Fiber Communication-incudes (OFC 2009)*, pages 1–3, 2009.

[157] B.G. Bathula, R.R.C. Bikram, V.M. Vokkarane, and S. Talabattula. Quality-of-transmission-aware manycasting over optical burst-switched networks. *IEEE/OSA Journal of Optical Communications and Networking*, 2(10):820–830, 2010.

[158] H.V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 367–380. USENIX Association, 2006.

[159] R. Landa, R.G. Clegg, J.T. Araujo, E. Mykoniati, D Griffin, and M Rio. The large-scale geography of internet round trip times. Submitted for review, 2012.

[160] D. Leonard and D. Loguinov. Turbo king: Framework for large-scale internet delay measurements. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 31–35. IEEE, 2008.

[161] J. Buford, A. Brown, and M. Kolberg. Analysis of an active maintenance algorithm for an o (1)-hop overlay. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pages 81–86. IEEE, 2007.

[162] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Number 0-13-022278-X. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[163] Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Number 1584889969, 9781584889960. Chapman & Hall/CRC, 1 edition, 2008.

[164] S. Basu, I. Davidson, and K. Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications*. Chapman & Hall/CRC, 2008.

[165] W. Terpstra, J. Kangasharju, C. Leng, and A. Buchmann. Bubblestorm: Resilient, probabilistic and exhaustive p2p search. In *ACM SIGCOMM*, Kyoto, Japan, 27-31 August 2007.

[166] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. A statistical theory of chord under churn. In *Proceedings of the 4th international conference on Peer-to-Peer Systems*, IPTPS'05, pages 93–103, Berlin, Heidelberg, 2005. Springer-Verlag.

[167] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 2003.

[168] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *The 11th IEEE International Conference on Networks (ICON2003)*, pages 137–141, 2003.

[169] R. Bartle. Interactive multi-user computer games. *MUSE Ltd*, 1990. http://www.mud.co.uk/richard/imucg.htm.

[170] T. Henderson. Latency and user behaviour on a multiplayer game server. In *Proceedings of Third International COST264 Workshop (NGC 2001)*, volume 3, page 1, London, United Kingdom, November 2001. Springer.

[171] B. S. Woodcock. An analysis of mmog subscription growth. 2008. http://www.mmogchart.com/analysis-and-conclusions/.

[172] M. Ye and L. Cheng. System-performance modeling for massively multiplayer online role-playing games. *IBM Systems Journal*, 45(1):45–58, 2006.

[173] T. A. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96)*, page 222. IEEE Computer Society Washington, DC, USA, 1996.

[174] Hu Shun-Yun and Liao Guan-Ming. Scalable peer-to-peer networked virtual environment. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, Portland, Oregon, USA, 2004. ACM.

[175] S.Y. Hu, J.F. Chen, and T.H. Chen. Von: a scalable peer-to-peer network for virtual environments. *IEEE Network: The Magazine of Global Internetworking*, 20(4):22–31, 2006.

[176] D. Hughes, K. Gilleade, and J. Walkerdine. Exploiting p2p in the creation of game worlds. In *Third International Game Design and Technology Workshop and Conference*, 2005.

[177] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, 2004.

[178] M. Mauve. How to keep a dead man from shooting. In *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204. Springer-Verlag London, UK, 2000.

[179] Angie Chandler and Joe Finney. On the effects of loose causal consistency in mobile multiplayer games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–11, New York, NY, USA, 2005. ACM.

[180] Y.W. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033, 2001.

[181] D. Frey, J. Royan, R. Piegay, A.M. Kermarrec, E. Anceaume, F. Le Fessant, et al. Solipsis: A decentralized architecture for virtual environments. In *1st International Workshop on Massively Multiuser Virtual Environments*, 2008.

[182] J. Keller and G. Simon. Toward a peer-to-peer shared virtual reality. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 695–700. IEEE, 2002.

Appendix A

Virtual worlds

Presented here is preliminary work on virtual worlds, which can be used as the motivation for future work. The literature review presented here provides a case for distributed overlays in the demanding application of virtual worlds, where QoS and data integrity is required.

A.1 Virtual worlds

The creation and maintenance of multi-user worlds represent a significant challenge to Peer-to-Peer overlay designers. A solution to this multi faceted problem brings together not only the need to find relevant nodes quickly but also maintain world *'state'* across many nodes. The use of these worlds in multiplayer gaming pushes the boundaries further as players or *'avatars'* require integrity in World events and expect close to client-server response times.

Work presented here concentrates on Worlds which are used in multiplayer gaming however these can easily be used for virtual reality and augmented reality solutions. Multiplayer gaming currently poses the toughest demands on the overlay, similar to, but greater than video streaming. Whereas video can be pre-buffered for several seconds with no discernible loss in experience in gaming however, tens of milliseconds usually affects the outcome of a battle/match/competition [167], with usage patterns seen to favour lower latency environments [168]. Compounding the challenge of reporting world events quickly is the accuracy and security of these updates all the while spreading the load across nodes on the overlay means it is easy to see that creating a high performance P2P virtual world is a non-trivial design problem.

A.1.1 Background to gaming

The notion of multiplayer games goes back over three decades to MUDs [169] which started in the late 1970s. These, unsurprisingly, low bandwidth games which used text to describe the worlds or Dungeons, provided the mainstay for online gaming until the early 1990s when

iD Software[1] released DOOM and 3DRealms [2], Duke Nukem. These were followed on by Quake, Half-Life, Unreal Tournament and derivatives. These games defined a new genre the first person shooter (FPS). It would be this genre with its fast paced action that would really push the boundaries of multiplayer gaming. Following this multiplayer versions of role-playing games called massively-multiplayer role playing games, MMOPRGs such as EverQuest [3], RuneScape[4] and World of Warcraft[5] were created. These were the modern day MUDs.

The two genres bring with it differing requirements from the overlay. FPS games typically take place in smaller worlds, commonly known as maps but have tighter tolerances for latency. Research [170] has shown that the *'frag'* (kill) rate is negatively proportional to the latency. With MMORPGs, the requirement for optimal latency is relaxed as games are more about strategy than particular actions however the game world is vast with more avatars present. In recent years games have made objects within the world mutable by avatars. Mutable objects increase the game state which has to be made available to avatars and may be associated with a particular avatar at a certain time (for instance a weapon which is picked up and used by an avatar may move its position in the overlay and may also have characteristics associated with it, such as the number of bullets left in its magazine).

The popularity of multi-player gaming [171] has meant the use of traditional client-server setup is becoming a non cost-effective solution to provisioning of services. Though the figures presented by Woodcock [171] are for massively multiplayer games such as World of Warcraft and Everquest it would be a reasonable assumption that other games based on the first person shooter and racing genres have also seem similar growth due to the number of titles which support multi-player interaction and increase in platforms offering simple connection to the Internet for gaming (Sony's PlayStation and Microsoft's Xbox).

A.1.2 Current technologies

Multiplayer gaming is generally provided through a standard client-server paradigm where users connect to servers which are either automatically load distributed or manually though server selection at the client side. The client-server paradigm, which has been challenged in other areas of network usage, remained popular in multiplayer gaming for a number of reasons.

[1] ID Software - http://www.idsoftware.com

[2] 3DRealms - http://www.3drealms.com

[3] EverQuest, Sony Entertainment - http://eqplayers.station.sony.com

[4] RuneScape, Jagex Entertainment - http://www.runescape.com

[5] World of Warcraft, Blizzard Entertainment - http://www.worldofwarcraft.com

- Synchronisation of world state
- Security in message handling
- Latency
- Resilience

The reliance on client-server technology in FPS games is helped by the fact that most servers accommodate less than 64 avatars and the intense demand for low latency, low packet loss performance. Not only does this lower the resource requirements for the server hosting games. It should be noted that while CPU usage is considerable in game servers, with the latest games able to fully utilise the latest processors from manufacturers this can be attributed to a couple of factors, both of which are outside this volume of work. The first is a lack of multi-processor support from the server *'binaries'* and the second is poor performance of binaries in different system architectures, for instance Linux over Microsoft Windows, though both issues are being improved upon.

In MMORPGs where worlds can consist of tens of thousands of avatars system and network resource becomes a significant factor in providing an adequate quality of experience. Unlike FPS games which are traditionally played on finite *'maps'*, MMORPGs take place in *'worlds'* which may increase in area as the game continues.

Due to the vast size of worlds, a process of creating zones was proposed and first used in the massively multiplayer online role playing game, MMORPG, Everquest [172]. Although this system wasn't P2P, it had a number of servers allowing demand on resources to be distributed among multiple servers. Players can chose which zone they join at connection however each zone has a limit on the number of players it can hold therefore still being limited by resource usage of the server which the zone resides on. Sony, the makers of Everquest introduced a concept of shards which are duplicated instances of the whole game world to allow unlimited number of players to compete in a single game world. Research by Ye [172] found most MMORPGs adopt the use of zones and shards in their architecture.

However the zone and shard architecture has issues, most notably the inability for players on different shards to interact with each other. The switching between zones requires a player's state to be written to external storage otherwise it will be lost. Furthermore, Ye found that the

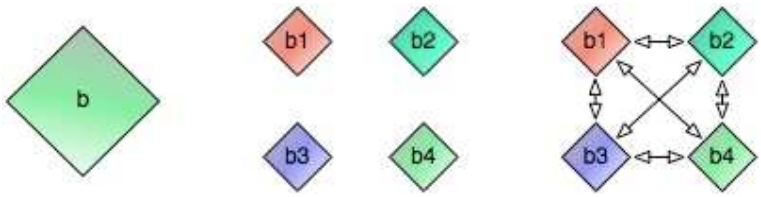


Figure A.1: The use of zones to duplicate game worlds

non-Gaussian distribution of players on each zone meant load on zone servers wasn't uniformly distributed resulting in inefficient use of resources. To overcome these limitations seamless architecture was developed. This is shown in Figure A.1 where the duplication of game world *b* initially led to zones which do not communicate with each other. The progression of this was to introduce a 'Seamless architecture' that allows communication between zones and shards.

The seamless architecture is similar to the zone and shard that preceded it however players can interact with objects which reside on other zones without any perceptible difference. Collaboration between different machines that run the different zones is required in order to maintain consistent information on players which occur on different zones. The boundaries of zones become dynamic allowing servers to balance their load. This full mesh architecture clearly has overheads as not all information will need to be replicated between all servers.

A modification a full mesh architecture is the adoption of hierarchical overlays first presented by Funkhouser [173] in which servers still communicate with each other but only share updates rather than complete game state between each other. This offers a more fine grain control of what update information gets sent to each other and is done so regardless of whether an avatar is in different zones or shards. The player can usually select which server from the pool *S1, S2, S3*, something which is done prior to the start of a multiplayer gaming session.

Games such as Warcraft or Starcraft define servers (called *'realms'* by geographic location, albeit in coarse terms such as *'Northern Europe'* or *'East Coast America'* with players typically playing on the one closest to them as it affords the most favourable network conditions. It should be noted that many separate games may take place on a single server with each game being a unique representation of the game world or different game worlds. The manual selection of servers can be automated through the use of Anycast (Section 2.4) in order to automatically route the user to whichever server is the closest or has the most resources. Strategies for assign-

ing users to servers may be more complex as players may be paired against others of a similar skill or particular set of rules.

This is still a client-server paradigm with the load spread over multiple servers; none of the network or resource load has been shifted to the clients. This provided the motivation for work being presented [171], [174], [175] which allows the creation of game-worlds to occur on P2P networks. Early research on topologies for multi-user virtual environments [176] highlighted possible problems for P2P architectures supporting such environments due to the number of update messages which need to be sent between zones. The author proposes the use of multicast in order to reduce the number of messages being sent from a particular node, with the router filtering which nodes receive the update.

The notion of using multicast to push state information is utilised by Knutsson et al. [177] with an overlay using multicast to inform other avatars of avatar location changes in the nearby region. It is also referenced as a possible solution in early work by Funkhouser [173]. Through the use of multicast, the limitation comes from the cost of multicast group membership, a factor that could be mitigated by having a greater number of regions and utilising more multicast trees.

A.1.3 Consistency

Once the virtual world has either been split or replicated on a number of servers the issue of providing a consistent view [178] to all participants becomes vital for overall experience and to avoid paradoxes such as a avatar who has been killed being able to attack another avatar. This problem increases in complexity when low bandwidth, high latency connections such as mobile devices are taken into consideration [179].

The consistency of virtual worlds are made more complex due to prediction mechanisms [180] built into clients, allowing for compensation of network latency. The notion of client-side prediction was borne out with the then high latency narrow-band dial-up connections that were used to play games such as *Quake* and *Half-Life* on the Internet. Users could expect to have a 300-500ms trip time on certain servers, so instead of waiting for the server to acknowledge and accept client updates, the client would assume that the updates will be accepted. Of course there will be times when the updates will be refused by the server and the client will notice an instant change in avatar position. The server's view of the World would over-rule that of the client.

A.1.4 Area of Interest

Avatars in the virtual environment do not require complete knowledge of the World; and in applications such as computer games would expressly forbid such knowledge to players. For this reason the notion of *"areas of interest"* were introduced.

The notion of areas of interest is to provide some form of mapping between the physical and virtual worlds. Topologically aware overlays may not represent an accurate area of interest. Solipsis [181] arranges nodes in a topologically aware fashion to this in order to reduce the distance messages have to travel on the overlay, however the neighbouring node on the overlay could be on the other side of the virtual world.

The area of interest of an avatar can be defined as the area of the virtual environment that can either directly or indirectly affect or be affected the avatar. Objects and avatars which can interact with the avatar are within that avatar's direct area of interest. Indirect areas of interest could be events which affect the avatar from the other side of the World. For instance teleportation points which allow other players to appear instantly within the avatar's direct area of interest.

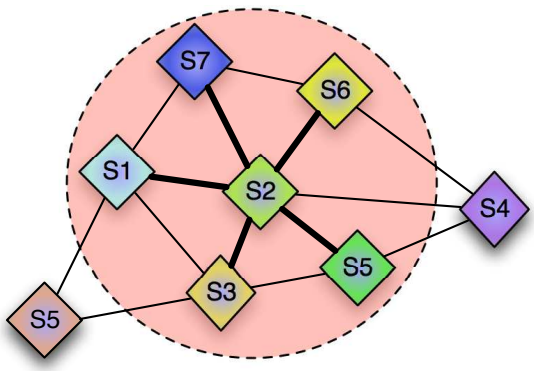


Figure A.2: Avatar *S2's area of interest*

The area of interest displayed in Figure A.2 shows other avatars within the node *S2*'s area of interest. These are nodes which *S2* is expected to have common interaction with. The size of the area of interest can be static or dynamic and items referenced in the areas of interest depend on the application. Applications which aim to provide the user with a greater level of immersion are likely to include more objects in the area of interest which require tracking. As the number of items in an area of interest grows the number of messages required to keep track of them increase. Early systems used static areas which were defined at the World's creation

but these were superseded by dynamically sized ones, lower requirements on particular nodes and increasing the versatility of the virtual environment.

Although decentralised networks offload much of the processing and bandwidth requirements from any particular node on the overlay, update messages must be filtered so that every peer doesn't receive every update. This type of selective flooding is discussed further in Section 2.1.2. The system may only allow update messages to those in the same zone as interactions with players who are in different zones may be fewer. It should be noted this hierarchy may not be similar to the topologically aware overlay as presented in Section 3.3 as players in Australia may not be the same distance away from a player in the United Kingdom in the game world as they are in the real world.

Even this strategy of sending updates to players in a particular zone fails to scale as all peers in the zone must be aware of all other peers in a zone. If peers join a zone infrequently, then join messages aren't the biggest problem but periodic *"polling"* updates of their state may be relatively small in size but the number of these messages grows $\Theta(NP)$ for N entities and P peers. As mentioned previously, the use of multicast can mitigate the message update problem and also moves the message filtering away from the peers and onto the network level, but bring forward the problems associated with multicast such as increased load on edge routers.

A hierarchical approach [176] allows a pool of servers to distribute the load of sending updates to peers. A peer will communicate with only one server and the server propagates the update to other servers in the pool. The server may be part of another zone. Filtering of updates could be done in order to only notify peers which interact with other peers regardless of zonal differences.

The hierarchical overlay has similarities with the seamless architecture presented in Figure A.3. This also has connotations of 'super peer' assisted P2P overlays, something which is a part of the DOAT system, in the form of NCPs (Section 3.1). The server pool, S1, S2, S3 in Figure A.3 communicate with each other, transferring location of world entities (player position, damage to scenery and other characteristics) to each other in a full mesh network. In turn players connect to the server which is closest to them, either automatically through some sort of software Anycast service or manually through player selection.

Systems that utilise hierarchies may not fragment the game world, instead replicate the world

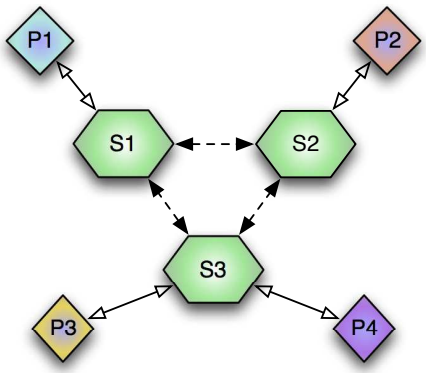


Figure A.3: Hierarchical overlays for game servers

on each server with incremental updates. The update policy could be a epidemic style burst of information which reaches a certain number of servers within a given radius, measured in latency or it could be routed messages to particular hosts, Keller et al. [182] labelled this as awareness area of radius *r*.

Nodes within an awareness radius of a particular node would receive updates of joins and departs with Keller's proposal is to allow nodes to know its local surroundings. Although flooding doesn't exhibit favourable scalability properties (Section 2.1.2) one could argue that the number of servers for a particular virtual world would be relatively small, in the order of 10s rather than 10s of thousands. The network in Figure A.2 shows the awareness radius of server S2. Updates are sent to those nodes within that radius, S1, S3, S4, S6 and S7. These nodes in turn forward updates to nodes within their awareness radius. In order to avoid synchronisation problems between nodes, updates occur synchronously among servers.

Simulations would have to be conducted in order to determine whether updates on servers several 'awareness radii' away from the source would have the update in time for its connected players. Should the virtual world be split up similarly to that of the real world, that is to say players in England play against other players from England on a particular server and the World is a virtual representation of England then it is unlikely that a player would interact with the Australian part of the virtual world and therefore a delay in updates due to many awareness radii hops may be acceptable. The size of each shard would be determined by the capacity of each server. Although servers could be supernodes placed on ISP backbones with high uptime and good connectivity characteristics there still lies a capacity bottleneck. The problem could be eased by the use of a system similar to the local tracker one proposed in DOAT (Section 3.4).

The idea of player nodes being selected to maintain part of the virtual world relies heavily on their connection time to the overlay. In certain types of games such as first person shooters the game world may get reset or changed completely in a short period of time, in the order of minutes rather than hours or days; as in common in massively multiplayer games. At these resets, known as map cycles, players may disconnect from the server for a number of reasons, such as not liking to play that particular world or not having a copy of that world on his system.

While incentive systems may promote players to keep connected to the overlay for longer the ability to delegate ownership of shards to players is one that would need to be investigated further.

A fluid delegation of shards could occur, so that the overlay node which crosses a shard boundary is responsible for the maintenance of that shard. The term 'shard' is a piece of the complete game world, rather than a replication of the complete game world. Other nodes present within a shard could be utilised as a server pool, with nodes utilising an Anycast service in order to communicate with the closest one. A lightweight protocol to transfer shard ownership with residual data would be required as players would obtain and release ownership in the order of seconds.

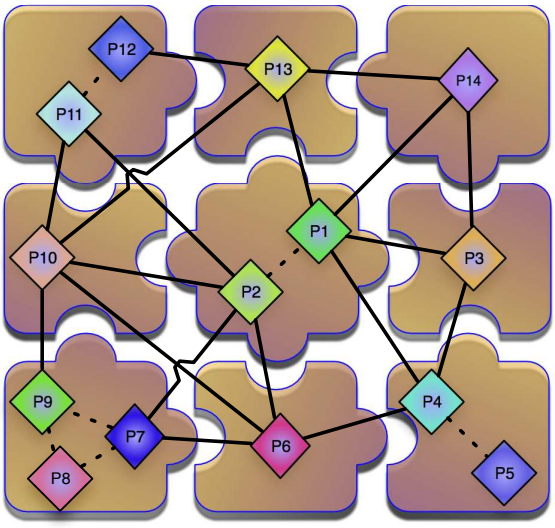


Figure A.4: Inter-shard player connection

The shards in Figure A.4 has players in them which take part in both intra (dotted line) and inter

shard (solid line) communication. The notion of a shard owner may be blurred somewhat when a large number players are in one shard. Each shard needs to have information on neighbouring shards, because an ability of events in other shards effecting gameplay.

Player nodes maintain knowledge of nodes through a DOAT-like system. Inter-shard communications may occur with different nodes within a particular shard, shown by P9 - P10 and P6 - P7. The selection of nodes can be to some usage metric as defined by the Anycast service. This behaviour could allow the transfer of game state transfer of a player to occur over a short period of time as the player will likely be the closest node in their new occupied shard to the players in its previous occupied shard.

This idea of ownership of the game world is similar to that of structured overlays where a node has ownership of part of the keyspace. The notion of the game world being represented as the keyspace of DHTs may allow the building on top of work that has been conducted on DHTs.