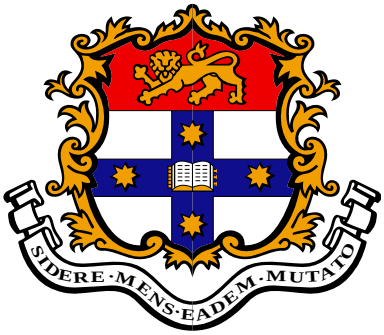
BALANCING IMPLICIT GROUP MESSAGING OVER PEER-TO-PEER NETWORKS



A thesis submitted in fulfilment of the requirements for the
degree of Doctor of Philosophy in the School of Information Technologies at
The University of Sydney

Daniel Cutting

June 2007

To mum and dad, for always being there and instilling in me
a passion for all things literary, artistic, musical and technical.

To Simon for keeping me on the planet and always making me laugh.

And to Loan for her endless patience, inspiration and belief in me.

Abstract

The ever-increasing role of Internet-based communication and the great quantity and variety of new niche content compels new paradigms for many-to-many messaging. This thesis introduces *implicit group messaging* (IGM) which proposes that messages should be delivered to "implicit groups" of consumers based upon their characteristics or interests. Such groups are undefined until the moment of publication and may be very large, posing significant delivery and load distribution problems.

This thesis argues that the loading problems of IGM can be effectively addressed with a structured peer-to-peer (P2P) design that promotes low and uniform loading across peers (*fairness*). IGM is formally specified and a framework for analysing implementations is defined which considers loading and fairness over peer, network and performance metrics. An effectively load-balanced P2P design called SPICE is presented, based around a novel substrate called ICE which features hierarchical tesseral addressing and efficient multicast routing. Through detailed simulation and analysis with both extreme and realistic data sources and network topologies, SPICE is shown to exhibit extremely low and fair peer and network loading both for individual publications, and over many publications. The load distribution techniques employed by SPICE are highly capable of dealing with the variable load distribution challenges presented by IGM.

Acknowledgements

I've been lucky enough to have three supervisors over the course of my degree. I clearly recall the day my principal supervisor, Aaron Quigley, convinced me to quit my job and become an academic. Although I sometimes wish I hadn't listened, his unwavering enthusiasm and creativity have never failed to inspire and motivate, and his support has been tireless and various over the entire duration of my graduate studies. My second supervisor, Björn Landfeldt, took me on as a student midway through my degree and has always made time for me. His ability to get me thinking about my work at different scales has been the perfect antidote to my typically stratospheric viewpoint. The Nymity work I did with my original associate supervisor, John Zic, was a lot of fun and I learnt a lot about the research process from him. I'd also like to thank my honours supervisor from way back, Gernot Heiser. It was he who really got me fascinated in systems, and his supervision has influenced all of my technical work since then.

The University of Sydney has been a stimulating research environment. I'd like to thank every member of the school for their help and support over the years, particularly Chen Wang, Remo Di Giovanni, Witold Janus, Fabian Brites, Chris Collins, Sharon Chambers, Josephine Spongberg, Michael McCabe and Candice Loxley. Working alongside Bob Kummerfeld and Judy Kay has been a great privilege and widened my perspective enormously, and I must thank James Curran for providing access to some quick simulation machines. I'm also very grateful to the powers that be for the UPA/APA scholarship that allowed me to pursue a Ph.D.

The scholarly and financial support of the Smart Internet Technology Cooperative Research Centre has given me the opportunity to collaborate and form friendships with many other students around Australia. I must put in a special word for Lisette Cochineas, whose outstanding commitment to the students is really appreciated by all of us.

Getting data for my research was tricky, but it was made so much easier with the help of some great people. Paul Ortyl's deeply appreciated assistance in obtaining the CCSB data was crucial for the case study in this

dissertation, as were the excellent physical network topologies published by Michael Liljenstam, Jason Liu and David M. Nicol. Maurice Coyle and Peter Briggs from University College Dublin spent far too much of their time helping me obtain and process both the I-SPY data, and the Excite data kindly made available by Bernard Jansen and Evelyn Balfe. Thanks also to Matthew Hurst for his terrific rendition of the blogosphere.

The simulations in this dissertation were partly executed on the big iron at the Australian Centre for Advanced Computing and Communications (AC3), and my experiments were made much easier generally with a tool called ROMNeT designed by David Symonds. The superb LaTeX template for this dissertation was originally created by Harish Bhanderi.

I'm indebted to Norman and Marshia Cipriano for setting me up with a brilliant study desk at which I spent many long nights toiling on my code and writings. It made things just a bit easier.

I can't begin to express the amount of fun I've had working with the denizens of G61b, especially those who travelled the same path with me and shared many amusing moments: Adam Hudson for his awesome encouragement and camaraderie over the years; Derek Corbett for his lively collaboration and capitalist agenda; Mark Assad for always being able to completely baffle me (both intentionally and otherwise); Mick Avery for his rapier-like wit and 1337 skillz; and David Carmichael's beard for always allowing David to follow it around. The G61b latecomers, hangers-on and gatecrashers have also become good friends and respected colleagues: David Symonds, William Niu, Andrew Lum, Trent Apted, Daren Ler, David Storey, Anthony Dang, Khaled Almiani, Tara McIntosh and Charles Prabhakar.

I'm also extremely grateful to the very many people who proofread and improved this dissertation: the G61b gang, Gavin Sinclair, Ross Shannon, Mum, Dad, Loan, Simon, Bruce Waskett, Stephen O'Young, Rob Rowland, Linda Sun, Martyn Huckerby and Nick Watts.

Well, I think I hear the orchestra striking up, so I'll just leave it at that.

Publications

Some of the material and concepts in this thesis have appeared in published journal, conference and workshop papers.

Journals

- Daniel Cutting, Aaron Quigley and Björn Landfeldt, "Special Interest Messaging: A Comparison of IGM Approaches," *The Computer Journal*, 2007, doi:10.1093/comjnl/bxm076.
- Daniel Cutting, Aaron Quigley and Björn Landfeldt, "SPICE: Scalable P2P implicit group messaging," *Computer Communications*, 2007, doi:10.1016/j.comcom.2007.08.026.

Conferences

- Daniel Cutting, Björn Landfeldt and Aaron Quigley, "Implicit group messaging over peer-to-peer networks," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P2006), Cambridge, United Kingdom*, A. Montresor, A. Wierzbicki, and N. Shahmehri, Eds. IEEE Computer Society, September 2006, pp. 125–132.
- Daniel Cutting, Derek J. Corbett and Aaron Quigley, "Context-based messaging for ad hoc networks," in *Third International Conference on Pervasive Computing (Pervasive 2005), Munich, Germany, Adjunct Proceedings*, A. Ferscha, R. Mayrhofer, T. Strang, C. Linnhoff-Popien, A. Dey, A. Butz, and A. Schmidt, Eds. Oesterreichische Computer Gesellschaft, May 2005, pp. 33–36, ISBN: 3-85403-191-2.
- Daniel Cutting, "Middleware in pervasive computing environments," in *Fifth Association of Pacific Rim Universities (APRU) Doctoral Students Conference, Sydney, Australia*, APRU, August 2004.

Workshops

- Daniel Cutting, Björn Landfeldt and Aaron Quigley, "The long, interesting tail of Indie TV," in *International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB) at the Fourth International Conference on Pervasive Computing (Pervasive 2006)*, Dublin, Ireland (invited paper), May 2006.
- Daniel Cutting and Aaron Quigley, "Mapping context to situations in a pervasive computing environment," in *First International Workshop on Software Aspects of Context (IWSAC'04), ACS/IEEE International Conference on Pervasive Services (ICPS'04), Beirut, Lebanon*, July 2004.

Parameters, notation and metrics

Table I: Implementation parameters.

SPICE (and ICE)	Symbol	Typical range	Default
Storage limit	s	4–1024	32
Frequency limit (casts/s)	f	0.0125–0.05	0.0125
Surface dimensionality	d	2–4	3
Branch factor (^{c})	ϕ	0–π	$\frac{\pi}{3}$
Number of landmarks	l	$2^{jd}, j \in \mathbb{N}, j \neq 0$	$j = 2$
BROKER			
Number of brokers	b	$0.01n$–$0.2n$	$0.01n$ (1%)
Routing error			10%
CENTRALISED		No separate parameters	
Simulation			
Tags per peer	k	5–50	18
Tags per cast		1–10	2
Casts per second			1.0
Number of casts	c		1024
Number of peers	n		4096
Peer registration skew		0.0–2.0	1.0
Cast skew		0.0–2.0	1.0

Table II: Notation.

	IGM modelling languages
$\mathcal{P}$	Set of all participants.
p_r	The registration of participant p.
c_t	The target expression of cast c.
$\mathcal{P}_t$	Implicit group selected by t.
$\lvert\mathcal{P}_t\rvert$	Size of group selected by t.
$\mathcal{T}$	All target expressions in a modelling language.
$\mathcal{R}$	All registrations in a modelling language.
$t \triangleright r$	Target expression t selects registration r.
$C(q)$	The set of all casts published by participant q.
R	The set of all registered participants.
	Tag-based modelling language
$T(t)$	Target set of expression t.
$R(r)$	Registration set of registration r.

Table III: Summary of IGM metrics.

	Efficiency and unfairness	
METRIC_M	Efficiency	Highest loading of a metric.
METRIC_G	Unfairness	Gini coefficient of a metric, ranging from 0.0 (representing complete fairness) to 1.0 (complete unfairness).
	Peer facet	
TOUT	Total out	Low TOUT_M indicates no peer forwards many cast messages over time. Low TOUT_G indicates peers have a similar total outgoing load.
TIN	Total in	Low TIN_M indicates no peer has to handle many casts over time. Low TIN_G indicates most peers have a similar total incoming load.
POUT	Per cast out	Low POUT_M indicates no peer has a high outgoing load during a single cast.
PIN	Per cast in	Low PIN_M indicates no peer has a high incoming load during a single cast.
STOR	Stored bytes	Low STOR_M indicates no peer stores a lot of state. Low STOR_G indicates most peers store a similar amount of state.
	Network facet	
TINK	Total link load	Low TINK_M indicates no link is heavily loaded over all casts. Low TINK_G indicates links have a similar total load.
PINK	Per cast link load	Low PINK_M indicates no link is heavily loaded during a single cast.
	Cast facet	
RTH	Ratio of Total Hops	Cost of a cast in terms of the ratio of total overlay hops compared to an ideal CENTRALISED implementation.
RAH RMH	Ratio of Avg./ Max. Hops	Ratio of average and maximum overlay hops to each consumer compared to the CENTRALISED implementation.
Latency		The actual time taken to deliver a cast from a publisher to a consumer.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The Internet is rapidly becoming a dominant medium of social interaction; more than *one billion* people are already online [1]. Although initially supporting only limited text-based email and news groups, the introduction of the World Wide Web has heralded greater interaction on a global scale by enabling the interconnection of documents and multimedia. The intention of the web has always been "read/write" [2] and recently, countless content-oriented tools such as Blogger [3] and Flickr [4] have significantly lowered the technical difficulty of producing and publishing content. Combined with increasingly ubiquitous broadband connectivity (some two-thirds of all users in 2006[1] [5]), these tools have driven the proliferation of specialised weblogs (*blogs*), films, technical documents and podcasts [6, 7, 8].

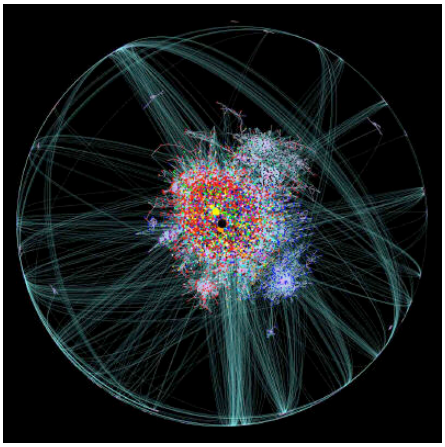


Figure 1.1: The blogosphere is a vast web of online journals by amateurs and professionals alike. Image courtesy of Matthew Hurst. Reprinted with permission.

The Internet is now a mass medium facilitating tremendous social interaction. As of early 2007, for instance, the web site Technorati [9] was tracking more than 80 million interconnected blogs in the "blogosphere" (Figure 1.1). "Open source journalism" is taking root [10] and the concurrent evolution of connected mobile devices such as camera phones enables people to publish and receive photographs and other content "on-the-go" as participants in mobile virtual communities [11].

It is increasingly difficult to focus on what is personally interesting and relevant [12]. There are many possibilities for conveying content from publishers to consumers. Public web pages are typically found via search engines or aggregation sites (*portals*)

[1] Based on forecasts from 2004 data.

such as Slashdot [13]. Many sites are also supplemented by RSS or Atom syndication mechanisms that permit consumers to subscribe to updates when they discover a publisher that interests them. Some more sophisticated aggregation services also exist whereby consumers can receive aggregated, customised updates by combining and filtering RSS feeds from multiple sources [14]. Even more generally, content-based publish/subscribe messaging allows consumers to subscribe to articles matching specific criteria and have only those that match delivered as they are published, regardless of their original source [15].

Search engines excel when consumers are able to specifically describe what they are looking for. Portals, by contrast, are suitable for finding non-specific content around a general theme; there are numerous technology-oriented portals such as Slashdot, for example. They serendipitously benefit consumers by exposing them to articles from a variety of sources that may be of interest, even if they would not think to actually search for them. However, since they need to appeal to a somewhat broad readership, it is uncommon to find portals that *precisely* cater to individuals' interests.

In general, these approaches place the onus on consumers to hunt for information, sifting through search results, or progressively narrowing their browsing through predetermined categories. This thesis terms these forms of delivery *consumer-selected*, since it is the consumer that selects the content they receive. For example, web site URLs may be explicitly entered into a browser (*selection-named*), or content may be implicitly described by keying terms into a search engine or publish/subscribe subscription (*selection-described*).

Complementary to consumer-selected delivery is *publisher-selected*, which relies on the publishers of articles selecting the consumers. Prime examples are email and instant messaging where recipients are specified by the publisher with names or addresses. An important feature of this approach is that consumers do not initiate delivery: information arrives unannounced. Unlike consumer-selected approaches, most publisher-selected techniques require publishers to explicitly name consumers (i.e., they are selection-named).

Table 1.1 categorises various messaging approaches, both consumer- and publisher-selected, and selection-named and -described. Each has drawbacks, depending on the application domain. For example: email requires the publisher to know the names of all interested consumers, which is unsuitable for large audiences; blogs require consumers to find and subscribe to those they suspect will produce relevant articles; and search engines require consumers to describe sought content accurately. The difficulty confronted is that of *findability* [16]—the ease with which articles can be found—and the difficulty can only mount as the number of publishers, consumers and articles increase.

Table 1.1: Classification of techniques for connecting publishers and consumers.

	Publisher-selected	Consumer-selected
Selection-named	Email, instant messaging, unicast	Web sites, portals, directories, blogs, multicast, mailing lists, chat rooms (IRC), channel- and topic-based publish/subscribe, RSS, Atom
Selection-described	Implicit group messaging, anycast, broadcast, interest management	Subject- and content-based publish/-subscribe, search engines

1.2 Implicit group messaging

This thesis proposes that because publishers create content with a specific audience in mind, **publishers should specify the consumer demographic for each article of content**. Underpinning this concept is the notion of an *implicit group*: a set of consumers that have some inherent features in common. An implicit group is defined by the characteristics of its members, rather than their explicit names.

Implicit group messaging (IGM) delivers content from publishers to specified implicit groups of consumers. It can be classified as publisher-selected and selection-described (Table 1.1). IGM can be most easily conceptualised by considering the removal of the subscription language of publish/subscribe from the consumers to the publishers. Instead of consumers selecting the type of messages they want to receive, the publishers select the type of audience they wish to reach. As a straightforward example, Figure 1.2 conceptually illustrates publishers sending messages to only those consumers that have certain combinations of features. The Selector component finds selected consumers and the Notifier component delivers messages to them. Note that in actual implementations these components may be combined, distributed or implemented in any number of ways.

Implicit group messaging enables implicit groups of consumers to receive targeted, relevant information from a multitude of publishers in a timely fashion, without searching or subscribing. The approach combines the serendipity of portals with the precision of search engines and the unannounced nature of email. **Many-to-many IGM over the Internet is the motivating context of this thesis.**

1.3 The loading problem

When considered in the context of content publication over the Internet, IGM must support thousands to millions of participants with very frequent publications to both small and large implicit groups. IGM must be scalable in terms of the number of

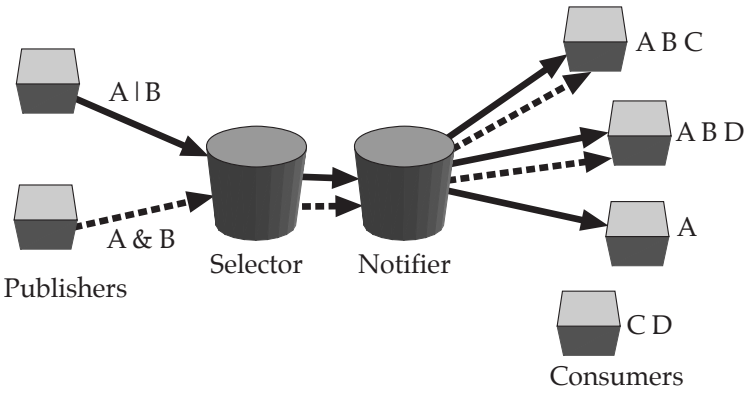


Figure 1.2: A conceptual model of implicit group messaging.

participants, the sizes of implicit groups and the number of publications. Potentially ephemeral implicit groups may be messaged frequently, rarely or not at all, with membership varying as new participants join, leave or change.

The power of the IGM concept poses inherent design problems. Implicit groups are unbounded in size, undefined until the moment of publication, and may cross-cut the population of consumers along any number of dimensions. Implementations of the Selector and Notifier components may therefore be subjected to extremely variable loading characteristics. For example, a centralised implementation of the Selector will receive many messages (high incoming load), while the Notifier component must forward many messages (extreme outgoing load). Underlying network links will also be subjected to carrying many duplicates of messages. In distributed IGM implementations, non-uniform group selections by publishers and skewed distributions of consumer features may cause points of heavy loading, either during individual publications, or manifest only after many.

This thesis addresses the IGM loading problem caused by many participants and highly skewed publication and feature distributions.

1.4 Hypothesis

Social interaction over the Internet is inherently decentralised, egalitarian and many-to-many. A solution to the loading problem should embrace these same principles for reasons both technical and social. Although success stories such as Google [17] show that the client/server approach can scale to millions of users, it comes at great administrative and financial cost. By contrast, distributed peer-to-peer (P2P) systems offer the promise of smooth scalability from small to large numbers of participants without dedicated infrastructure [18, 19, 20]. P2P systems can also avoid centralised bottlenecks and points of failure, and eliminate inherent bias or censorship of content, whether intentional [21] or not [22]. Of importance to some applications is that

distributed systems may also be less vulnerable to legal or technical attacks [23]. However, it is important in such systems to ensure the load placed on peers is "fair", precisely because no single peer claims responsibility for the whole system.

In the context of this thesis, *fairness* is defined as the degree to which all participants contribute equally to the operation of the IGM system. A system is *efficient* when the maximum contribution of any peer is low relative to the total work that must be done. **This thesis posits that a distributed structured P2P network supplemented by adaptive load distribution techniques affords efficient and fair IGM, irrespective of scale or skew.**

Distributed structured P2P networks generally organise peers and connections as abstract, regular topologies supporting various routing guarantees, in contrast to the more random topologies found in unstructured P2P networks. Some examples of structured P2P networks are Chord [18], Pastry [24], Tapestry [25] and CAN [19]. The SPICE implementation presented in Chapters 7 and 8 specifically addresses the IGM loading problem, and is built on a distributed structured P2P network called ICE. Extensive evaluation shows that the ICE/SPICE combination fairly and minimally loads peers both on a per publication basis and over a period of time. SPICE performs well even under extremely skewed feature distributions across peers, and frequent publications to very large implicit groups.

1.5 Contributions

This thesis advances the state of the art in load distribution in P2P networks, Internet messaging generally, and analysis of load distribution in distributed systems, by way of the following enumerated contributions.

Chapter 2

1. A classification of various Internet messaging paradigms along the two axes presented in Table 1.1: publisher-/consumer-directed; and selection-named/-described.

2. A detailed review of load distribution techniques in peer-to-peer networking.

Chapter 3

3. A formal specification of IGM based on linear temporal logic [26], including an IGM interface, and "safety" and "liveness" conditions. Preliminary work on IGM-like schemes appear as conference [27, 28] and workshop publications [29].

4. A tag-based modelling language implemented according to the formal specification, suitable for an array of Internet messaging applications.

5. A characterisation of the heavy loading patterns incurred by IGM implementations, especially those caused by large, frequently selected implicit groups. Early exploration of these loading problems forms a conference publication [30].

Chapter 4

6. A comprehensive analysis framework for evaluating IGM implementations capable of determining participant and network load distribution fairness and efficiency, both per publication and over many publications. Fairness is based on the Gini coefficient measure of inequality.

Chapters 5–8

7. Three IGM implementations: a baseline implementation called CENTRALISED (Chapter 5); a decentralised implementation called BROKER, optimised for minimal network load and latency (Chapter 6); and a P2P design called SPICE (Chapters 7 and 8).

8. A common battery of simulation-driven experiments performed for each implementation testing peer, publication and data skew scalability. Analysis of the progressively decentralised implementations demonstrates the extent to which a fully distributed system can fairly and efficient balance the loads generated by IGM. CENTRALISED is both inefficient and unfair under all conditions. BROKER exhibits low latency, and reduces loading but remains generally unfair. SPICE vastly reduces the maximum loading on any single participant, while greatly improving fairness across all.

Chapter 7

9. A novel distributed structured P2P substrate called ICE which uses hierarchical tesseral addressing and "amortised routing" to provide efficient and lightweight multicast messaging for higher-level systems. ICE is used as the basis of the SPICE IGM implementation; its unique features enable SPICE's effective load distribution techniques.

Chapter 8

10. "Registry distribution" and "registry replication" techniques in SPICE that are shown to promote very fair and efficient dissemination of publications to implicit groups of any frequency or scale. Investigation of these techniques appear in a journal publication [31].

Chapter 9

11. A case study of how IGM could be effectively used in a real scenario, specifically exploring its use as the basis of a communication tool for implicit groups of researchers working in specialised domains. A similar application of IGM forms an earlier workshop publication [32].

12. A corroboration of the implementations' evaluations via comparative simulations driven by a real data source and actual network topologies with accurate broadband connectivity traits. This analysis also appears in a journal publication [33].

1.6 How to read this dissertation

The remainder of this dissertation is organised as follows. Chapter 2 categorises background and related material. Chapter 3 formally specifies implicit group messaging and characterises its heavy loading patterns. Chapter 4 presents an IGM analysis framework and describes the design of the simulation environment. Chapters 5–8 offer three IGM implementations—CENTRALISED, BROKER, and SPICE—combined with a comprehensive evaluation of their load distribution characteristics. Chapter 9 explores the use of IGM as a communication tool for researchers working in implicit special interest groups, presenting comparative simulations of the implementations driven by realistic data and actual network topologies. Finally, Chapter 10 summarises this thesis and proposes future work in the area of implicit group messaging. Appendix A provides an overview of the pseudocode language used to present the algorithms in this dissertation.

This dissertation can be read in its entirety or according to the reader's specific interests. Table 1.2 suggests relevant chapters for various implicit groups of readers.

Table 1.2: How to read this dissertation.

Interests	Chapters
Messaging or IGM	2, 3
IGM and implementation	3, 4, 5, 6, 7, 8, 10
P2P or Distributed Hash Tables	2, 7
Messaging and applications	2, 3, 9, 10

Chapter 2

Background

This chapter begins by describing some of the existing messaging paradigms in use over the Internet, according to how they connect publishers and consumers (Contribution 1). Peer-to-peer (P2P) networking is also discussed with an emphasis on structured overlay networks, which are central to this thesis' approach to load distribution. The chapter then explores the load distribution techniques used by other systems, particularly with respect to selection-described messaging over P2P networks (Contribution 2). The chapter closes with background material necessary for the BROKER and SPICE IGM implementations presented in Chapters 6 and 7.

2.1 Selection-named messaging

Composing and sending email is one of the most familiar ways of publishing messages to people. Instant messaging is also becoming increasingly popular. These technologies typically rely on publishers knowing the names or addresses of consumers. Although they can be used to reach large groups, this is not their intended usage; better solutions exist for many-to-many communication, including RSS syndication and chat rooms. Generally, these approaches require consumers to actively seek participation.

When enduring groups of consumers wish to repeatedly receive messages from publishers, they may explicitly join specific groups. In these cases, prior construction of messaging infrastructure can be used to achieved efficient distribution. For example, IP multicast [34] allows hosts to explicitly join a group and receive all messages sent to the group address by constructing an efficient spanning tree between all members. This tree can be source-rooted in the case of a single producer, or rooted at a rendezvous point when there are multiple producers. An advantage of multicasting at the IP level in the network stack is that redundant packets can be avoided, optimising the usage of physical network links. Similar application-level approaches have also been explored [35, 36, 37], designed around hubs that route messages between participants. Although unable to offer network utilisation as optimal as IP multicast, they

have the advantage of not requiring additional network support. A number of similar multicast schemes constructed over peer-to-peer networks have also been proposed, such as SCRIBE [38], CAN multicast [39] and Bayeux [40].

The technical designs of these kinds of techniques are generally inappropriate for selection-described messaging paradigms because they only need to deliver messages to participants known in advance. In particular, these approaches are not directly applicable to implicit group messaging as they would require the creation and maintenance of separate multicast groups for each implicit group, of which there are a potentially unlimited number.

2.2 Selection-described messaging

Publish/subscribe messaging [41] allows consumers to create "standing queries" that push matching content to them as it becomes available. This approach is consumer-selected because it is the consumers that select the set of messages they receive, and selection-described because they use a subscription language to group messages. Many publish/subscribe systems have been proposed, including SIENA [42], Elvin [43], REBECA [44], Gryphon [45] and INS/Twine [46].

SIENA [42] supports content-based publish/subscribe messaging over a network of routers which may be structured hierarchically, as an acyclic graph or as a cyclic graph. Each router maintains a routing table that contains covering subscriptions for each of its neighbours. SIENA has three main algorithms: *receiver advertisements*, which insert subscriptions into the system; *sender requests*, which allow routers to refresh their routing tables by asking consumers for their subscriptions; and *event publications* that prune a broadcast tree based on the content of events and the subscriptions along each edge from the current router. Additional enhancements such as address simplification allow multiple subscriptions to be rewritten as simpler constraints. This technique can be applied to IGM, and is one of the implementations presented in this thesis. Chapter 6 presents a broker-based design based on an acyclic graph that uses routing tables similar to SIENA.

In publisher-selected-described messaging, publishers select the consumers that receive their messages by describing them. The PEACH project [47] uses the concept of "implicit organization" in order to communicate between various components of an interactive museum guide and Chambel et al [48] describes the use of implicit groups to aid navigation of and retrieval of data from large distributed "hyperbases". However, these approaches are limited to their specific application and are not directly applicable to large distribution networks.

Khambatti [49] describes the discovery and utilisation of interest-based communities in unstructured peer-to-peer networks for directing distributed searches. The

approach connects peers with common interests, but does not offer the guarantee required by IGM that all group members receive messages. Instead, interest-based communities are used to prune the search space for queries by directing them to areas of the network where they are more likely to be resolved.

Interest management in war game simulations and virtual environments [50, 51] refers to the distribution of update messages from entities in the environment to all other entities that need to know about them. Entities create an expression of interest in events within a certain virtual distance, in a similar manner to content-based publish/subscribe. An additional feature of interest management is that extrinsic attributes of the publisher of a message (such as their logical position or name) can be matched in addition to the intrinsic attributes of the message.

SelectCast [52] allows a sender to multicast a message to peers matching an SQL-like query. The expression language supports aggregation functions (e.g., MAX or MIN) for specifying implicit groups of recipients but relies on hierarchically structured domain "superpeers" responsible for maintaining aggregation information and forwarding messages to deeper domains. SelectCast also supports publish/subsribe through the use of Bloom Filters (see Section 2.6.1). This approach is conceptually similar to the broker-based design of Chapter 6 which is also based on a distribution tree.

2.3 Implicit groups

This section identifies the concepts of implicit groups and communities, especially with respect to their increasing roles in the Internet.

A community is an enduring group of people defined by common characteristics. Historically, a community has described residents of a particular geographic area, or a social group within a larger society. More recently, Communities of Practice (CoP) [53] have described groups of people participating in shared social or business settings. CoP has been extended to several specific types of communities including Communities of Interest (people who share a common interest or goal) [54] and Communities of Circumstance (people with illnesses or minority groups, for example).

These notions have also progressed into the online domain. First used by Rheingold in relation to the WELL web site in the early nineties, the term "virtual community" [55] described physically distant members bound intellectually, socially and technically. The term now refers more broadly to practically any online social activity such as frequent visitors to web sites, chat rooms, and online games [56].

Communities have been used to improve the functionality of P2P systems, by allowing users to discover groups in which they may like to participate, or as a way of limiting searches to peers likely to resolve them [49, 57, 58]. However, such ap-

proaches are usually used to improve existing services such as search queries, and do not seek to deliver messages to every member of a community.

Implicit groups are wider in scope than traditional communities. Although they can describe communities of people with shared interests or circumstances, this thesis defines them broadly as subsets of populations of objects selected by the requirements for membership. They may be enduring or ephemeral, generic or highly specific, social or technical. Although this thesis is motivated primarily by social scenarios, implicit groups may specify groups not considered to be communities in the traditional sense, such as a set of computers with specific functionality and available resources.

2.4 Peer-to-peer networking

This section describes peer-to-peer (P2P) networking. It is described before examining selection-described messaging in Section 2.2 because it is common approach to such forms of messaging.

P2P is a style of networking characterised by its focus on many participants contributing to the operation or maintenance of the system as a whole. Although most well-known for file sharing applications, P2P is applicable to many domains such as resilient data backup [59, 60], distributed file systems [61], workflow management [62], distributed search [63], web caching [64], email [65], video streaming [66], and publish/subscribe messaging [44, 45, 67, 68, 69, 70]. In a loose sense, P2P has existed since the beginning of the Internet, and by many definitions includes heterogeneous directory-based approaches such as the original Napster. In this discussion, however, P2P refers to systems where all peers have homogeneous roles. Of this type of P2P there are two main approaches: *unstructured* and *structured*.

2.4.1 Unstructured P2P

In unstructured P2P, peers connect to one another randomly or according to certain heuristics, in order to produce small-world networks [71, 72] or groupings clustered by interest [49]. The approach is exemplified by early file sharing networks such as Gnutella.

Applications built on unstructured networks must be designed to work with the limited assumptions that can be made. Since each peer only connects to its direct neighbours and there are no central directories available for global system communication, it can be difficult to implement functionality that requires guarantees. For example, searching for objects that are replicated throughout a network is an appropriate application for unstructured P2P as in these cases it is sufficient to find just one or a few objects matching the search criteria. Queries are typically flooded from the source to a certain depth, although some advanced techniques for limiting or directing

these floods have been explored [73, 74, 75].

Applications that are required to find unique instances of objects, or every instance of an object are less suitable for such networks. For instance, IGM is designed to deliver a message to all members of an implicit group. It is not sufficient to find only those nearby to the publisher. In order to remain relatively efficient at large scales (i.e., without needing to flood messages to all peers), a network could be seeded with hints that improve the likelihood of finding selected peers. Such techniques include gossiping [76], Rumor Routing [77], Directed Diffusion [78], interest-based communities [49] and routing indices [79]. Even so, it is difficult to guarantee that significant fractions of groups will receive messages, and finding all members of very small, scattered groups is infeasible.

2.4.2 Structured P2P

Structured P2P networks take a very different approach to connecting neighbours. Peers are organised according to an abstract, regular topology supporting various routing guarantees which allow certain design assumptions absent from unstructured P2P. A peer is usually assigned a unique address or range of addresses. Routing rules are defined such that a message can be deterministically forwarded from peer to peer towards the peer that is nearest to a destination address.

A panoply of structured P2P designs exist, based on various routing schemes such as Plaxton routing (Pastry [24], Tapestry [25]), a one-dimensional ring (Chord [18]) or a d-dimensional Cartesian surface (CAN [19]). Many of these have $O(\log n)$ storage and routing costs (e.g., Chord, Pastry and Tapestry).

Collectively, these are often referred to as Distributed Hash Tables (DHTs) or Distributed Object Location and Routing (DOLR) designs. A DHT conceptually operates like a classical hash table, storing and retrieving objects over the network via fixed length keys. Peers are typically responsible for a portion of the address space and store and serve the objects that are hashed to it. DOLRs are very similar but support routing of arbitrary messages to objects or nodes in the overlay, rather than just storage and lookups.

CAN is of particular relevance to this thesis as the structured overlay introduced in Chapter 7, ICE, bears some similarities. CAN addressing and routing occurs on a d-dimensional plane, which is divided among peers. Peers' regions are specified using Cartesian coordinates, and data is logically stored at points on the surface by routing it to the peer that owns the region containing it. The point-to-point routing available in CAN incurs higher routing costs than many structured networks, $O(dn^{\frac{1}{d}})$, but benefits from just $O(d)$ storage complexity.

DHTs and DOLRs are often used as a substrate for more advanced applications, such as storage or messaging systems. For instance, the multicast designs mentioned

in Section 2.1 are based on structured overlays: to create a multicast group, an identifier is typically hashed to the addressing system of the underlying DHT and a creation message is routed to the nearest node. This node then acts as the root of the multicast group. When a node wishes to join the group, it too hashes the group identifier and routes a join message to the group root, thereby including itself in further group messages. Some systems use the group root to store a list of all members, while others take an approach similar to more traditional IP multicast and create a tree from the root to all members along the route of nodes in the DHT from the source to the root (e.g., SCRIBE [38]).

It is theoretically appealing to order or arrange peers randomly in the address space of a structured overlay, as uniform randomness simplifies design and permits assumptions of system behaviour and costs at the overlay level. However, such randomness does not consider physical network locality. Because neighbouring peers may be physically separated by intercontinental distances, such overlays may exhibit appalling performance. Therefore, it is beneficial for the overlay surface to map well to the physical underlying network, although this must be tempered by the desirable properties of random placement. Many systems offer functionality to aid creation of overlays that map well to physical networks. These are generally based upon measuring the latency of connections to other peers, or to a set of well-known "landmarks" around the network and include the ping-based Vivaldi [80], Madhyastha et al's traceroute-based approach [81] and CAN's landmark binning scheme [82].

2.5 Load balancing

When messages are delivered from publishers to consumers, some degree of load is placed on the message delivery infrastructure. Some of this load is in the form of state that must be retained by the infrastructure. Other load includes instantaneous or ongoing stress on various components, such as network links forwarding packets, peers receiving incoming packets, and servers transmitting multiple copies of messages.

Loading of Internet hosts has long been recognised as a significant problem. Popular web sites typically address the problem by deploying multiple servers that act as one. However, the load on web sites generally grows linearly with the number of users because the server need only send information back to a single host for each access. There has been much effort to distribute load in P2P networks, and DHTs in particular, as described below.

2.5.1 Storage/registration distribution

Storage load is caused by many objects hashing to the same location in a structured overlay, and hence the same peer. Ordinarily, fair storage distribution is trivially

achieved with a consistent hashing function which uniformly distributes keys over an identifier range. If the identifier range is also uniformly partitioned across peers, then the load is fairly distributed.

However, some systems require support for range queries. These are usually implemented by replacing consistent hashing with locality-preserving functions which can lead to imbalance when many data items have similar values or ranges. This is often handled by partitioning the identifier range non-uniformly to relieve load [83, 84], or by mapping the skewed data items to a uniform range [85]. For instance, a CAN-like surface may be uniformly partitioned by default, but a load-balanced version could instead split a region such that half of the data objects currently stored there remain in each region. Such an approach would result in a balanced structure, but since the surface would not be uniformly distributed, additional techniques for distributing routing load may be required. Ongoing adaptation could also be difficult since overlays are partitioned according to the loading profile of particular points in time.

Mercury [86] is a protocol for supporting multi-attribute range queries and supports load balancing of attributes that are popularly registered. Mercury peers participate in *attribute hubs* which are logical rings of nodes pertaining to a single attribute. Each peer must link to every hub, so the number of hubs (and thus attributes) cannot be particularly great. Peers register data items in the hub for each attribute they contain. They are stored around the hub using a locality-preserving function such that range queries can be resolved. Conjunctive range queries are routed to a single hub (for one attribute from the query) which then resolves the whole query. Popular attributes are handled by moving lightly loaded hub nodes to regions which are heavily loaded such that data items can be partitioned.

Mirinae [85] is a content-based publish/subscribe system that uses a hypercube overlay routing network to group similar subscriptions. Subscriptions are mapped to a corner of the hypercube based upon an application schema, and events are mapped to partial identifiers according to the same schema and sent along the edges to corners that cover it. This scheme works by clustering subscriptions in the topology of the hypercube that are semantically close to one another.

Such storage distribution techniques may be required for IGM systems built on structured overlays because many peers will likely register the same tags, leading to imbalance. IGM systems also have an additional caveat. Unlike search engines, for example, which resolve queries and return results to a single peer, IGM systems must forward publications to many consumers. If an IGM system has a storage loading problem, it will also have an outgoing load problem, since the peer storing the data will also need to notify many consumers. Storage distribution is thus of particular importance in IGM systems.

2.5.2 Access/query distribution

The other major type of loading is access or query loading. In this instance, a peer must frequently respond to queries for objects (i.e., it has a high incoming load). Query loading may also be a problem in IGM systems because publications may not be expected to follow uniform distributions. It is likely, therefore, that some points in the system will need to handle a large number of publications.

HotRoD [87] is a ring-based DHT supporting range queries that addresses the access loading problem by managing multiple virtual rings corresponding to increased levels of replication. When a range of peers becomes "hot" they are replicated and rotated to another region of a virtual ring. Queries can be routed to either the original or any of the replicas to be resolved.

Wang et al [88] addresses the access loading problem for DHTs by replicating data to the neighbours of the peers that are ordinarily required to forward the query. A similar approach is used by Beehive [89], LAR [90], P-RLS [91] and Tapestry [25]. This approach exploits the routing algorithms of DHTs which must typically route through these neighbouring peers to reach the original. If a replica is encountered first, the original does not need to be contacted.

2.5.3 Comprehensive load distribution

Some systems aim to solve both storage and access loading problems. The SPICE implementation in Chapters 7 and 8 distributes storage of registrations to neighbouring peers in a structured overlay ("registry distribution"), and addresses the access loading problem by replicating registrations entirely ("registry replication"). The following systems employ similar strategies.

Meghdoot [68] is a content-based publish/subscribe system based on a structured overlay network similar to CAN. It provides support for arbitrary-range subscriptions, but requires the pre-definition of a schema defining the names, types and limits of possible fields. A $2k$-dimensional surface akin to CAN is formed and maintained by peers (where k is the number of fields in the schema). Consequently, for a schema of any appreciable complexity, the cost of surface maintenance is relatively high. Subscriptions are installed on the surface by mapping them to a $2k$-dimensional hyperplane, bounded by the ranges specified in the subscription. Events are mapped to a single point on the surface, then broadcast to all neighbours within the region near the point that contains all possible matching subscriptions. Mechanisms for load balancing of subscription storage and event processing load are presented. Newly arriving peers split the regions of overloaded peers and take half of their subscriptions. When a peer is processing too many events, a new peer replicates the content and takes joint ownership of the region. When messages are routed through that region, the precise

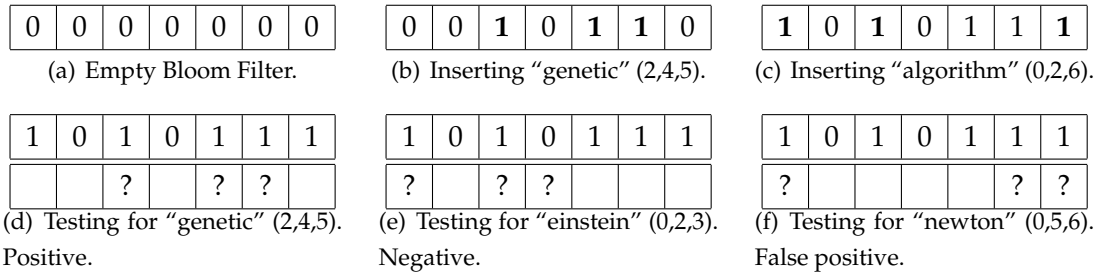
peer contacted is chosen at random. Meghdoot claims good and fair performance for scaling up to 10 000 peers, even with heavily skewed subscription and event sets. However, the need for a schema to describe the possible events and the particular requirement that fields are named and ordered in the addressing scheme, means that this approach cannot be applied to an unordered tag-based scheme such as the one adopted in this thesis (see Section 3.3).

Gao's DHT-based Content Discovery System (CDS) [92] is technically similar to SPICE. The main functional difference between the two is that SPICE delivers messages to members of implicit groups while CDS is essentially a distributed search engine that returns a matching set for a query. Although CDS addresses *content discovery* (as distinct from content delivery), it does provide a novel load distribution technique called the Load Balancing Matrix (LBM) which bears conceptual resemblance to the registry distribution and replication techniques of SPICE. Similar to SPICE, LBM manages skewed storage and query loads by increasing the number of peers responsible for storing and responding to queries. However, the approach used is technically very different: LBM is based on replicating and partitioning data at disparate parts of the surface using a hash function, while SPICE locally clusters distributed registrations to promote quick resolution, and widely spreads replicas to reduce initial routing costs.

The LBM is an adaptive 2-dimensional matrix that stores registrations of content, where columns partition registrations and rows replicate the partitions. Queries are resolved by routing to one member of each column (in any row) and collating results at the querying peer. As in SPICE, each registration contains enough information to resolve conjunctive queries without needing to contact other matrices.

However, each partitioned peer must be individually contacted to resolve a query, resulting in potentially many distant routes over the DHT. Furthermore, LBM uses a hash function that does not colocate partitioned peers, so amortised routing cannot be used to reduce message loading. A benefit of SPICE's registry distribution is that only a single long-distance route is needed, followed by a set of short-range routes to find nearby distributed registrations. Combined with overlay mapping, such a scheme reduces overall system load.

The LBM requires that replica nodes store the same registrations, which does not permit the possibility of peers with different capabilities handling more or fewer than other peers. In SPICE, "storage" and "frequency" limits may be individually set according to a peer's capabilities. The number of partitions in an LBM matrix is limited to a pre-determined value, unlike SPICE which scales from a single peer to the entire network. Additional state must also be held by the "head node" which maintains the current size of the matrix so that all registrations can be found. SPICE needs no such state beyond what is stored by each peer for its own routing purposes. Likewise with replication, LBM requires peers to find a replica by trial and error or by asking the



(a) Empty Bloom Filter. (b) Inserting "genetic" (2,4,5). (c) Inserting "algorithm" (0,2,6).

(d) Testing for "genetic" (2,4,5). Positive. (e) Testing for "einstein" (0,2,3). Negative. (f) Testing for "newton" (0,5,6). False positive.

Figure 2.1: Example of Bloom Filter usage.

head node. SPICE detects replicas as a natural consequence of routing towards the original.

2.6 Implementation details

This section describes material needed to implement the BROKER and ICE/SPICE IGM implementations in Chapters 6 and 7. BROKER uses Bloom Filters (Section 2.6.1), while ICE employs hierarchical tesseral addressing (Section 2.6.2) and divisive clustering (Section 2.6.3).

2.6.1 Bloom Filters

Bloom Filters [93] are compact representations of sets of objects, which support membership tests with an adjustable error rate of false positives. Stored as bit strings, Bloom Filters can incorporate new objects by hashing them with h different functions, each returning a position in the bit string to be set. An object can be tested for membership by hashing it with the h functions and ensuring all returned bit positions in the filter have been previously set. Figure 2.1 demonstrates the usage.

It is preferable to make Bloom Filters as short as possible, while retaining a certain rate of false positives. The optimal numbers of hash functions and bits needed for a Bloom Filter are dependent on the desired error rate and the expected number of elements it needs to store. By increasing the permissible error rate, fewer bits are needed to store the set, effectively trading precision for storage.

Let m be the number of bits, k the expected number of elements the Bloom Filter will need to store, h the number of hash functions and ρ the false positive probability. Broder and Mitzenmacher [94] shows that for a given false positive rate ρ the minimal value of h that maximises the information encoded in the Bloom Filter can be approximated by:

$$h \approx \frac{\log \rho}{\log \frac{1}{2}} \tag{2.1}$$

Table 2.1: Derived Bloom Filter values for h and m given ρ and k.

False positive rate (ρ)	Hash functions (h)	Elements (k)	Bits (m)
0.1	4	10	48
		100	480
		1 000 000	4 792 479
0.01	7	10	96
		100	959
		1 000 000	9 584 958
0.001	10	10	144
		100	1438
		1 000 000	14 377 436

The number of bits needed, m, can be similarly determined [94]:

$$m \approx \frac{k \log \rho}{\log 0.6185} \tag{2.2}$$

For example, if the false positive rate is 10% ($\rho = 0.1$), the optimal number of hash functions is approximately four. To store ten elements requires approximately 48 address bits. Table 2.1 enumerates derived values for h and m given ρ and k.

2.6.2 Hierarchical tesseral addressing

This section briefly describes hierarchical tesseral addressing, an addressing scheme employed by the ICE P2P substrate used by the SPICE IGM implementation in Chapter 7. The general idea is to efficiently decompose a plane by subdividing it into a regular tessellated pattern, so that each element can completely contain another such division. Squares and triangles have this property. Figure 7.3 in Chapter 7 illustrates a hierarchical tesseral address space based on squares; such a structure is often referred to as a quadtree.

Hierarchical tesseral addressing has been applied to many domains including geographic data storage and querying, computer graphics and robotics [95]. It has not been used to decompose the address space of a structured P2P network, although some systems have stored such structures over existing structured networks [96]. The properties of hierarchical tesseral addressing are crucial to the load distribution features presented in this thesis in Chapters 7 and 8.

eCAN [97] seeks to improve the routing efficiency of CAN [19] by building "expressways" between regions of the CAN surface. For this, progressively larger container regions are linked and can be used to bypass many of the small hops that would be required in CAN. The concept of hierarchical regions is similar to the tesseral ad-

dressing used by ICE but the purposes are quite different. eCAN peers actually record long-distance links to peers in other regions, maintaining a distributed tree-like structure that permits $O(\log n)$ point-to-point routing similar to Chord [18]. In ICE, tesseral addressing is a convention used to describe areas of the surface for the purposes of clustering destinations and multicasting, but all routing is local.

2.6.3 Divisive clustering

Divisive clustering is a hierarchical clustering technique that transforms a single set of objects into a number of clustered sets. It works by first finding the maximum distance between two elements in a set and creating a cluster for each. Each remaining element is then assigned to the cluster to which it is closest. This continues until each element is in its own cluster, or the maximum distance between every element is below a threshold.

For example, this set of five numbers is clustered according to their difference, resulting in first level clusters of two and three elements, and so on until each is individually clustered.

$$
\begin{array}{cccccc}
 & & & 13568 & & \\
 & & 13 & & 568 & \\
 & 1 & 3 & & 56 & 8 \\
 & 1 & 3 & & 5 & 6 \quad 8
\end{array}
$$

Divisive clustering is used in the ICE P2P substrate to cluster the parts of the network that remain to be visited by the routing algorithm. Each cluster is then visited independently. By setting a suitable threshold value, this approach permits the routing algorithm to reduce the number of messages that must be routed over the surface. See Section 7.2.2 for a full description of this algorithm.

2.7 Summary

This chapter has classified and discussed existing Internet messaging techniques according to whether they are publisher- or consumer-selected, and selection-named or -described (Contribution 1). The chapter has also reviewed peer-to-peer networking and the load distribution techniques employed by systems contending with loading problems similar to that imposed by IGM (Contribution 2).

Chapter 3

Implicit group messaging

This chapter begins by introducing and formally specifying the new messaging paradigm of implicit group messaging (IGM) in Sections 3.1 and 3.2 (Contribution 3). Section 3.3 defines a specific "modelling language" to be used in this thesis, based on the specification of IGM (Contribution 4). Inherent challenges associated with concrete implementations are then characterised in Section 3.4, particularly load balancing in the face of potentially large and spontaneously defined groups (Contribution 5).

Implicit group messaging is a many-to-many messaging paradigm for delivering messages from publishers to groups of consumers. Publishers select groups of consumers by their attributes rather than by name or address. Any participant is capable of publishing messages to any implicit groups at any time and publishers do not need to be members of the implicit groups to which they publish. The actual membership of the same implicit group may vary from message to message as participants join and leave the system or as their attributes change over time. Publishing to an implicit group is termed *casting*. Cast messages may themselves be referred to as *casts*.

Each participant in an IGM system has certain associated attributes or aspects of their identity. These are codified with a descriptive *modelling language* which can be tailored specifically for the application domain. An implicit group is defined as a subset of all participants that are described by a *target expression* over the same language. An expression conceptually segments participants into those that are described by the expression and those that are not. An expression is said to *select* those participants it describes.

Figure 3.1 shows a simple example of three different implicit groups, where each participant has associated sporting and musical interests. Implicit groups may be selected by combining interests, e.g., "football" (thick blue dashed line), "samba" (thin green dashed line), and "football & samba" (solid black line). Groups are not predefined but determined on an ad hoc basis. No participant, for example, explicitly joins or subscribes to the group "football & samba", but such a group can still be specified.

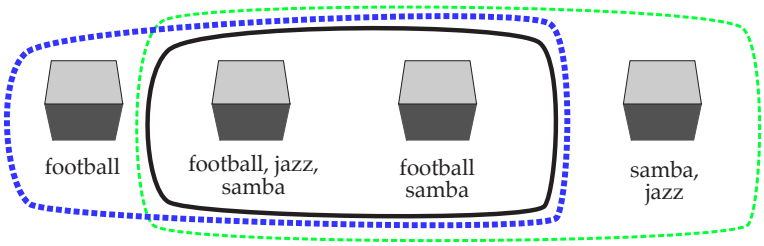


Figure 3.1: Three implicit groups.

Figure 3.2 illustrates a more complex example involving thousands of participants scattered over a wide area network. Implicit groups potentially cut across all network and geographic boundaries and may be extremely large or small (or even empty). Since implicit groups are not actually specified until publication, there is no trivial way to group participants in advance for the purposes of optimising network traffic for every potential group, in the way that IP multicast constructs minimal distribution trees, for example.

Implicit group messaging can be used as a fundamental messaging component in many domains, both social and technical. Collaborative group applications may use IGM to direct problems towards people that are likely to be able to solve them, or to facilitate chat sessions between people with similar interests. Such applications may be appropriately combined with the concept of communities of interest (see discussion in Section 2.3) but IGM is also applicable to machine–machine interaction and could be used to send commands to network nodes with particular configurations, or act as a resource discovery service.

IGM can be used to actually deliver content, or act purely as a notification service. This thesis is motivated by social interaction over the Internet where this choice depends upon the type of content and the specific IGM implementation. For example, it may be inefficient to route large content such as video over P2P networks, when traditional content delivery networks such as Akamai [98, 99, 100] are more suitable. In such instances, IGM could be used to notify consumers to begin retrieving new content from alternative sources.

Any concrete implementation of an implicit group messaging system should have the following basic properties:

1. All selected consumers eventually receive messages.
2. Messages are only delivered once to each consumer.
3. Only messages that are published are delivered to consumers.
4. Non-selected consumers do not receive messages.

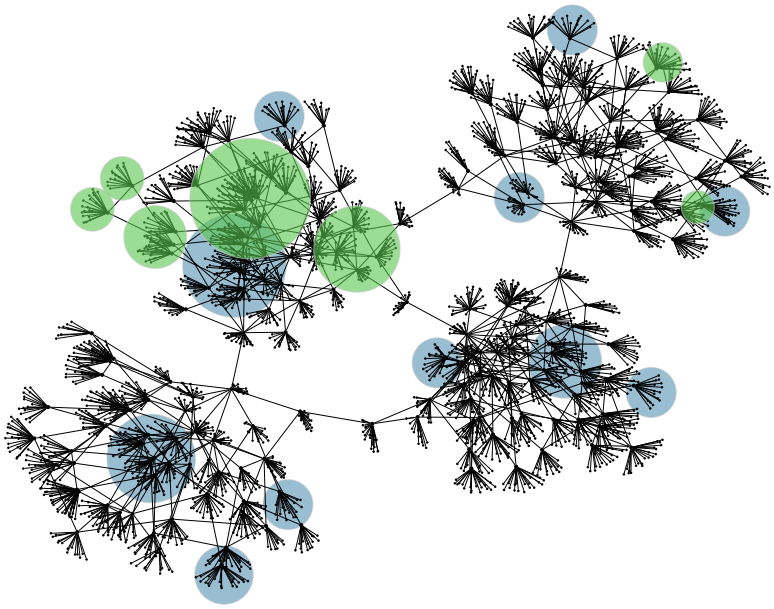


Figure 3.2: An example of a large IGM network. “football” is a widely spread interest (blue); the “samba” genre of music is more regional (green). The intersection of the two is small and sparse.

These properties define IGM in this thesis, but additional properties may also be desirable for some applications, such as the guarantee that messages are delivered to consumers in the same order they were published or that they must arrive within a certain amount of time. Such extensions to the IGM model are not addressed in this thesis but are considered in Chapter 10.

An implementation should also generally seek to minimise overall network and participant load, and deliver messages as quickly as possible, although the relative importance of these properties is application-specific. The focus of this thesis is an IGM implementation that conforms to the IGM properties while minimising and fairly distributing load on participants.

When designing a model such as IGM without a directly applicable analogue, it is important to construct a theoretical foundation describing its salient features. The following sections formalise IGM by defining implicit groups, specifying the selection semantics, defining an interface for participants, and formally restating the properties of IGM.

3.1 Modelling implicit groups

The notation described in this section is summarised in Table II. Implicit group messaging does not prescribe any particular modelling language to describe consumers and implicit groups; it may take many forms and can be chosen to suit a particular application domain. As such, the following generic definitions are independent of

language, although Section 3.3 defines a concrete modelling language to be used for the remainder of the dissertation.

Let $\mathcal{P}$ be the set of all participants in the system. Each participant $p \in \mathcal{P}$ has a *registration* p_r which describes it according to a modelling language. In fact, a participant may have many registrations, but without loss of generality this thesis assumes exactly one registration per participant.

Any language used to model implicit groups must define a surjective function $\triangleright$ (read "*selects*") which maps a target expression and registration to $true$ or $false$. $\mathcal{T}$ and $\mathcal{R}$ are the set of all target expressions and registrations for a language respectively.

$$\triangleright : \mathcal{T} \times \mathcal{R} \rightarrow \{true, false\} \tag{3.1}$$

The notation $t \triangleright r$ is read as "target expression t selects registration r". $\mathcal{P}_t$ is then defined as the implicit group (a subset of $\mathcal{P}$) selected by target expression t.

$$\mathcal{P}_t = \{p \in \mathcal{P} | t \triangleright p_r\} \tag{3.2}$$

3.2 Messaging implicit groups

This section formalises the properties of IGM enumerated above, based upon an interface of IGM operations, and specifies conditions that define which orderings of operations are legal. Any *event-based* messaging paradigm that supports distribution of messages from publishers to consumers can be specified similarly, including publish/subscribe and IP multicast. Mühl's specification of publish/subscribe message [44], founded on "traces" and "linear temporal logic", is thus adopted for IGM. In the interests of clarity and brevity, the specification presented here lacks some of the rigour required of a complete specification. The reader is referred to Mühl for a more rigourous treatment of trace-based specifications.

A minimal IGM model in which participants may only register and cast messages is sufficient for demonstrating the load distribution characteristics of the IGM implementations presented in later chapters. Table 3.1 lists the operations necessary for this model. Consumers may register with the IGM system and publishers may cast messages. Implicit group members are individually notified by the system after a cast.

A subtle semantic aspect of IGM is that specific membership of an implicit group is dependent on the actual time at which it is reified. For example, peers that join a system after a publisher casts a message but before it is delivered should conceivably be members of the implicit group. Implementations that invoke a distributed design (such as SPICE) are even more susceptible to an ambiguous group definition because the decision to notify consumers may be made independently by many peers.

Table 3.1: The IGM interface.

Method	Description
$reg(p)$	Consumer p registers p_r.
$cast(p, c)$	Publisher p casts c to group selected by target expression c_t.
$notify(p, c)$	Consumer p notified of cast c.

This dilemma also arises in publish/subscribe systems (and, in fact, any event-based system). It may be addressed by adjusting the definition of an implicit group to include only those consumers that have completed a registration at the time they are notified of a cast. Note that in distributed systems this definition permits the possibility that there is no single "instant" at which the consumers notified concurrently form an implicit group according to the ideal definition in Section 3.1. Such a utilitarian definition clarifies system implementation and, in practice, the semantic effect on publishers and consumers is minimal.

An IGM system can be thought of as a state machine, where invocations of the operations in Table 3.1 move it between states. Different combinations of operations will move the system to different states, and will often be dependent upon the existing state. A *trace* is an infinite sequence of states interleaved with operations.

$$trace = state_1, operation_1, state_2, operation_2, state_3, \ldots \tag{3.3}$$

Any ordering of operations and states can form a trace, but only a small subset of these are legal for a system observing the IGM properties. For example, it is legal for the system to notify p of cast c (with target expression c_t) if and only if p has registered a selected registration such that $c_t \triangleright p_r$. Note that the states themselves can be omitted from the trace since it is the relative orderings of operations that are of interest.

$$trace = reg(p), cast(q, c), notify(p, c), \ldots \tag{3.4}$$

This instance satisfies all informal properties of an IGM system. It would, however, constitute an illegal ordering if $c_t \not\triangleright p_r$ since p would be notified of a cast for which it had not previously registered a selected registration, violating Property 4.

The complete set of legal traces can be defined using *linear temporal logic* (LTL). This is a modal logic that allows reasoning about systems along a future timeline. A set of operators allow statements to be made about which conditions can or must occur at future points in the trace. Three temporal operators are needed to specify the IGM model.

- $\Box X$ means X is true in all future states.
- $\Diamond X$ means X is true in at least one future state.

- $\circ X$ means X is true in the next state.

Using these operators, it is possible to define predicates that hold for some traces. For example, the predicate $cast(p, c) \Rightarrow \circ notify(q, c)$ holds for all traces where q is notified of c immediately after c is cast by p.

$$trace = cast(p, c), notify(q, c), \ldots \tag{3.5}$$

This predicate may be weakened to allow notification at some point in the future rather than immediately: $cast(p, c) \Rightarrow \Diamond notify(q, c)$.

$$trace = cast(p, c), \ldots, notify(q, c), \ldots \tag{3.6}$$

LTL is often used in conjunction with *safety* and *liveness* conditions. For IGM, these can be used to describe traces where unwanted operation orderings do not occur, and where desirable orderings eventually occur, respectively. A *correct* system satisfies both safety and liveness conditions. Such approaches have been used to specify and verify the correctness of other event-based messaging systems such as REBECA [44] and Gryphon [45].

Liveness A liveness condition is used to express IGM Property 1. It states that if a consumer registers, it will eventually (and subsequently) be notified of all casts from any publisher that select its registration.

$$\Box\,[reg(p) \Rightarrow \Diamond\Box[(cast(q, c) \wedge c_t \triangleright p_r) \Rightarrow \Diamond notify(p, c)]] \tag{3.7}$$

Safety A safety condition is used to express Properties 2, 3 and 4. It states: when a consumer is notified of a cast, it will not be notified again; the message was previously cast by some publisher; and its registration is selected by the cast's target expression. Additionally, the condition requires that the consumer has completed a registration in order to agree with the adjusted definition of an implicit group. Let $C(q)$ be the set of all messages cast by q so far, and R be the set of all peers that have registered so far.

$$\Box\,[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c) \wedge \exists q.c \in C(q) \wedge c_t \triangleright p_r \wedge p \in R] \tag{3.8}$$

These two conditions are used to show that the implementations presented in Chapters 5–8 conform to the IGM specification.

3.3 Tag-based modelling language

In order to construct the IGM implementations in later chapters, a concrete modelling language is required. It must be clear so as not to obscure IGM itself, yet sufficiently

rich for a broad range of Internet-based social applications.

Structured languages such as controlled vocabularies, taxonomies and ontologies have been used extensively in many domains including library catalogues, context modelling for pervasive computing [101, 102], and the "semantic web" [103]. Structured languages are particularly good at arranging specialised domain-specific information. They are also able to reduce confusion by dictating a common language for participants, and often allow automated reasoning by the system [104]. Such *top-down* schemes also have some drawbacks: creating them may require much effort for complex systems; they may be culturally biased or language dependent [105]; they may not be agile enough to move with social trends; and they may not be structured as expected by the final participants [106, 107]. Despite their benefits, they are not applicable to all domains and may be most suited to constrained or special purpose applications.

A simple language commonly used for Internet-based "social software" allows participants to label content with arbitrary descriptive *tags* [9, 108]. Tags are simply keywords in a flat namespace. There is no hierarchy or intrinsic meaning to any tag; each is assigned meaning by the person who uses it. Indeed, tags are often loaded with personal connotations and their inconsistent usage may introduce semantic ambiguity to a system. However, common usage does tend to emerge even among small populations of participants [109]. An advantage of such a *bottom-up* vocabulary, or *folksonomy* as it is sometimes called [110], is that it can fluidly adapt over time to better represent changing application domains [111].

Due to its existing popularity in the motivating domain, this thesis uses a concrete modelling language based around this idea of simple descriptive *tagging*. Instead of tagging content, however, as is frequently the case with Internet-based content, participants tag *themselves*. The advantages of a bottom-up approach over a structured language are particularly important in the loosely federated, transitory environments envisioned for many IGM applications. Such a language is suitably expressive for many applications, including content production and the special interest research tool explored in Chapter 9.

Consumers describe themselves using tags (non-empty strings). Target expressions used by publishers are strings that combine tags using conjunctive (`&`) and disjunctive (`|`) operators, and parentheses to modify evaluation order, according to the following grammar:

$$tag := [a - z]+ \tag{3.9}$$

$$expression := factor\ (\ \text{“|”}\ factor \mid \text{“\&”}\ factor)* \tag{3.10}$$

$$factor := tag \mid \text{“(”}\ expression\ \text{“)”} \tag{3.11}$$

The selection function $\triangleright$ operates as expected under Boolean logic. For example, the target expression `(samba|jazz)&football` selects exactly those consumers that have registered `football` and at least one of `samba` or `jazz`.

More formally, let a registration r describing a participant's attributes be represented by a *registration set* $R(r)$ containing a tag for each attribute. E.g., a registration for a peer with tags `samba` and `football` has a registration set $\{\texttt{samba}, \texttt{football}\}$. Let a target expression t be represented by a *target set* $T(t)$ which is a set of *target elements*, each of which represents the conjunctive terms forming a disjunctive term of the expression. E.g., the expression `(samba|jazz)&football` expands to two disjunctive terms, `(samba&football)|(jazz&football)`, each containing two conjunctive terms. This expression is therefore represented by a target set $\{\{\texttt{samba}, \texttt{football}\}, \{\texttt{jazz}, \texttt{football}\}\}$. The selection function $\triangleright$ can then be defined as $true$ if and only if any non-empty subset of the registration set is an element of the target set:

$$t \triangleright r \Leftrightarrow \exists K \in T(t); K \subseteq R(r), K \neq \emptyset \tag{3.12}$$

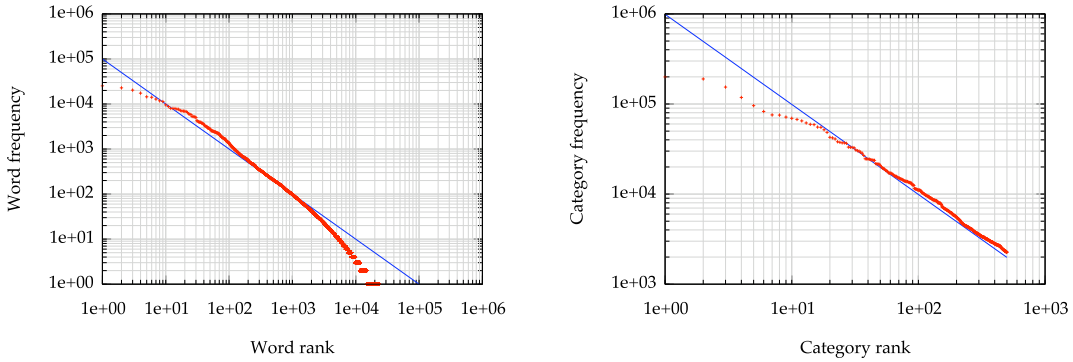
The definition of an implicit group (Theorem 3.2) can thus be restated for the tag-based modelling language using the selection function of Theorem 3.12 to yield:

$$\mathcal{P}_t = \{p \in \mathcal{P} | \exists K \in T(t); K \subseteq R(p_r), K \neq \emptyset\} \tag{3.13}$$

That is, an implicit group is the set of participants that have registered all of the tags in at least one element of a target set.

This definition is now contrasted with a simplified version of publish/subscribe messaging to emphasise the difference between publisher- and consumer-selected messaging approaches. In publish/subscribe, consumers subscribe to messages matching an expression. When messages are published, those that match are forwarded. The essential difference between publish/subscribe and IGM is the way in which the consumers are selected for each message. In IGM, publishers select the implicit group of consumers for an individual message; in publish/subscribe, consumers select the implicit group of messages they receive over time.

To illustrate this point using the definitions above, let a publish/subscribe message be published with metadata tags m (with a set representation analogous to a registration set, i.e., $R'(m)$). Let $\mathcal{P}'$ and $\mathcal{P}'_m$ be the set of all participants and the subset that receive the message, respectively. As before, it is assumed that each participant p' has exactly one subscription p'_s specified using a language similar to the tag-based target expression language defined above. $T'(s)$ represents a target set created by parsing a subscription in a manner akin to an IGM target expression.



(a) Frequencies of words in the complete works of Shakespeare is approximately Zipfian.

(b) Frequencies of top 500 categories assigned to Wikipedia articles.

Figure 3.3: Examples of real phenomena with Zipfian distributions.

Let $\triangleright'$ be a function that operates similarly to $\triangleright$ but defined such that $s \triangleright' m$ if and only if subscription s is matched by metadata tags m. Then it can be shown that the set of publish/subscribe consumers for a message has a similar, but converse, definition to that of IGM (Theorem 3.15).

$$\mathcal{P}'_m = \{p' \in \mathcal{P}' | p'_s \triangleright' m\} \tag{3.14}$$

$$= \{p' \in \mathcal{P}' | \exists K' \in T'(p'_s); K' \supseteq R'(m), K' \neq \emptyset\} \tag{3.15}$$

3.4 Characterisation of implicit group messaging

This section describes the types of peer registrations and cast distributions expected of IGM systems and discusses how these distributions may affect IGM implementations.

There are two types of data in a tag-based IGM system: the tags peers use to describe themselves; and the tags used in casts' target expressions. These registrations and casts, like many complex systems, will likely follow Zipf's law [112], which states that the frequency of a word in a language, $F(w)$, is inversely proportional to its rank, $R(w)$:

$$F(w) \propto \frac{1}{R(w)} \tag{3.16}$$

The law has been observed not only in large natural language corpora, but also in many Internet phenomena such as the number of visitors to web sites and the number of connections between autonomous computer networks like ISPs and universities [113]. Figure 3.3 shows two examples of actual Zipfian distributions on log-log scales: the frequency of all words appearing in the complete works of Shakespeare, and the top 500 categories assigned to articles on Wikipedia (on 17 June 2007) [114]. Zipf's law

Table 3.2: Various classes of implicit groups.

	Frequent cast	Seldom cast
Common tags	large, frequent	large, seldom
Rare tags	small, frequent	small, seldom

has also been observed in the frequency distributions of tags used on recent social web sites such as the bookmarking site, del.icio.us [109, 115, 116].

Zipf's law can be made more general by introducing an exponent s to the denominator, in order to describe its *skew*. An exponent of 0.0 describes a uniform frequency distribution (all elements are equally frequent), while exponents greater than 1.0 describe extremely skewed distributions where a few elements are vastly more popular than all others.

$$F(w) \propto \frac{1}{R(w)^s} \tag{3.17}$$

Although the precise exponents should be expected to vary across application domains, this thesis assumes that IGM peer registrations and casts will follow Zipfian distributions. Exponents between 0.0 (uniform) and 2.0 (extremely skewed) are considered during the evaluation in Chapters 5–8. The assumption that IGM registrations and casts are Zipfian is validated in Chapter 9 where real data are used.

Depending on the distribution of the tags registered by peers, variously sized implicit groups can result, ranging from empty to significant fractions of all peers. Similarly, the frequency with which the same groups are selected will vary with the skew of the casts. For illustrative purposes, implicit groups can be described by four classes, caused by the distributions of peer registrations and casts (Table 3.2), although the actual distribution of group sizes and selection frequencies is not discrete.

When peer registrations are highly skewed, many peers register the same tags. Thus, target expressions composed of common tags typically define larger implicit groups than those using rare tags, and tend to cause greater load on an IGM system during the cast. Implicit groups may be potentially any size (i.e., they are $O(n)$, where n is the number of participants in the system). If certain peers or servers in the system are responsible for handling individual casts, they need to notify many consumers. Hence it is desirable to divide the responsibility for large implicit groups over many peers.

When the cast distribution is highly skewed, the same implicit groups are selected frequently. If particular peers are responsible for these casts, they will need to handle a high frequency of requests. Again, distributing this load over many peers is preferable. If the frequently selected groups are also large, an IGM system is placed under

extreme pressure because many messages must be repeatedly delivered to the same large set of consumers. A fair and efficient IGM system must be capable of delivering casts to all of these classes of implicit groups without overloading peers either on a per cast basis or over many casts.

Section 4.3.2 describes the method used to create the Zipfian distributions of peer registrations and cast target expressions used in the evaluation of the IGM implementations.

3.5 Summary

This chapter has formally specified implicit groups and implicit group messaging using trace-based specifications and safety and liveness conditions rooted in linear temporal logic (Contribution 3). A concrete tag-based modelling language has also been defined, for use in the remainder of the dissertation (Contribution 4), and the types of expected implicit groups and attendant loading problems have also been described (Contribution 5).

Chapter 4 defines an analysis framework for analysing IGM implementations against these loading concerns, and presents the experimental design used for the evaluation in subsequent chapters.

Chapter 4

Analysing implicit group messaging

This chapter presents a framework for analysing the load distribution and performance properties of IGM implementations (Contribution 6), and describes the experimental design of the evaluation of concrete IGM implementations in Chapters 5–9.

4.1 Analysis framework

The analysis framework is specifically designed to measure **casting** in IGM implementations, as this is the main concern of the thesis. In particular, the framework does not address other aspects of implementations such as maintenance traffic or joining costs. Such costs are analysed theoretically in the design of each implementation.

Every participant in an IGM system is considered a peer. This includes client peers, servers and brokers in the CENTRALISED and BROKER implementations, and peers in the SPICE implementation.

By joining an IGM system, peers permit the use of four resources: computation, storage, incoming network traffic and outgoing network traffic. With ever-increasing computational power available to desktop computers, computational load is assumed to be unproblematic. However, storage requirements and incoming and outgoing load on peers should be considered. Network connectivity in particular is often charged according to the amount of data transmitted and received, and it is generally good practice to minimise the impact on other applications sharing network resources.

It is also important to consider a design's overall effect on the physical network. Client/server implementations would tend to load links near the server, for example, while structured P2P networks can have a detrimental effect if the overlay structure is incongruous to the physical structure (see Section 2.4.2).

Finally, the efficiency with which casts are published may be an important consideration for some IGM applications.

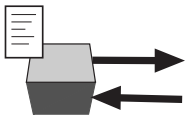
Accordingly, analysis is broadly broken into three facets reflecting these various concerns: *peer*, *network* and *cast*. The relative importance of these facets will vary

for different domains: some applications may require very low latency casts at the expense of network load, for example. The orthogonal distinction between these facets permits design trade-offs to be better understood and considered prior to design and deployment.

As outlined in Section 3.4, load in an IGM system can be considered on a per cast basis, and over many casts. Distinguishing between the two during analysis permits detection of peers that must handle flash load and those that are generally unfairly loaded over time. Each facet is considered against both of these types of load.

Peer facet

The peer facet analyses the cost to participate in the system from a peer's perspective. Consumers join the system in order to receive messages that match their attributes. They expect to receive matching messages exactly once, and ideally do not expect to have to handle or forward messages that do not match them. Table 4.1 shows the primary concerns of an IGM peer.



Storage load	Outgoing load
	Incoming load

Table 4.1: Peer facet.

Network facet

The network facet (Table 4.2) measures the impact of an implementation on the underlying physical network. The network can be abstractly described as a graph, the nodes and edges of which are routers/hosts and physical network links. This thesis is concerned only with fixed physical networks and mobility in particular is not considered.

Peers participating in an IGM system reside at the fringe of the network and communicate exclusively with unicast messaging in order to form overlays and higher level messaging constructs. Links in the network are considered to be directed, and should preferably carry the minimum number of packets needed to deliver messages, without overloading any in particular. The load on routers is not specifically measured, as it is safely assumed to be related to link load.

Link load

Table 4.2: Network facet.

Cast facet

The cast facet examines the overall cost and efficiency of publishing individual messages to implicit groups. Overall cost is measured in terms of the total overlay hops needed to reach all consumers, and the number of hops and time taken to reach each consumer (Table 4.3). Although the cast facet is not directly related to load distribution, it does provide a summary of the performance overhead incurred by an implementation.

Total hops	Hops to each consumer
	Latency to each consumer

Table 4.3: Cast facet.

4.1.1 Load matrix

This thesis is concerned with the efficient and fair distribution of load in IGM implementations on both a per cast basis and over many casts. Central to this concern is the determination of load distribution over peers and network links. If it can be shown that all peers have similar loading for a metric over all casts, an implementation is considered fair with respect to that metric. If no individual peer has a high load for any single cast or over all casts, an implementation is considered efficient for that metric. If these conditions can be shown for all metrics in the analysis framework, an implementation is considered both fair and efficient generally.

Table 4.4 shows a sample matrix of casts and peers for a single run of an arbitrary IGM implementation. In this example, each cell value is the number of messages a peer forwards on behalf of others for a particular cast. For instance, Peer 3 transmits 15 messages in the delivery of Cast 1. The matrix can be summarised to describe the loading for a particular run. To do this, the maximum of each row and the sum of each column are calculated, which represent the highest outgoing load of a single peer per cast (POUT), and the total outgoing load for each peer (TOUT), respectively. The maximum of each of these vectors is then taken to represent the absolute maximum loading across all peers and casts, and is denoted with a subscript M: i.e., in this example POUT_M is 35 and TOUT_M is 48. This indicates that a single peer was required to forward 35 messages during a single cast, and there was a single peer that forwarded a total of 48 messages over all casts.

The matrix approach used for the peer facet can be applied analogously to network links to produce the per cast link load (PINK) and total link load (TINK) metrics. A low PINK_M, for example, means no network link carried an excessive number of packets for any single cast. Note that because the loading values derived from these matrices are absolute numbers of messages, comparisons between implementations

Table 4.4: Sample outgoing load matrix of peers/casts for a run. POUT_M = 35, TOUT_M = 48 and TOUT_G = 0.3125.

	Peer 1	Peer 2	Peer 3	POUT
Cast 1	12	2	15	15
Cast 2	1	0	9	9
Cast 3	35	1	21	35
TOUT	48	3	45	

are only meaningful if the experimental design is identical, including the distributions of registrations and casts. These metrics are summarised in Table III.

The total load matrix vectors are also summarised for *fairness*: the similarity of loading across all peers and network links over many casts. One potential approach for calculating fairness is the standard deviation. Although such a statistic is useful when variables are expected to be normal around a mean, it is less useful for describing an entirely unknown distribution. There is no reason to suppose an IGM implementation will normally load peers or network links. In fact, the characterisation of IGM in Section 3.4 suggests there may be a large loading imbalance.

A cumulative distribution frequency (CDF) plot of load precisely describes a distribution but is not suitable for comparing implementations because it does not summarise the data; a single reduced value is preferable.

Hence the Gini coefficient (G) may be used. The Gini coefficient is often found in the field of economics as a measure of *inequality* of the distribution of income in a society. When every member has the same level of income, the distribution is completely fair. When one member has a positive income and the rest have none, the distribution is completely unfair. The analogy and applicability to loading on peers is clear.

The Gini coefficient is based on Lorenz curves, which are normalised representations of cumulative distribution frequency plots. G is defined as the area between the diagonal *line of equality* and the Lorenz curve, as a fraction of the entire area below the line of equality. It can thus take a value from 0.0 (fairness, when the curve matches the line of equality) to 1.0 (unfairness). Equation 4.1 defines the Gini coefficient for a population with n values x_i and mean $\bar{x}$, sorted in ascending order.

$$G = \frac{\sum_{i=1}^{n}(2i - n - 1)x_i}{n^2 - \bar{x}} \tag{4.1}$$

Figure 4.1 is an example of a Lorenz curve (unrelated to Table 4.4), with an associated Gini coefficient of 0.72 representing quite high unfairness. Because a higher value for G actually represents *unfairness*, this title is used on plots in the evaluation in subsequent chapters, but both terms may be used to describe systems.

The Gini coefficient is applied to the totalled vectors from the load matrix. For

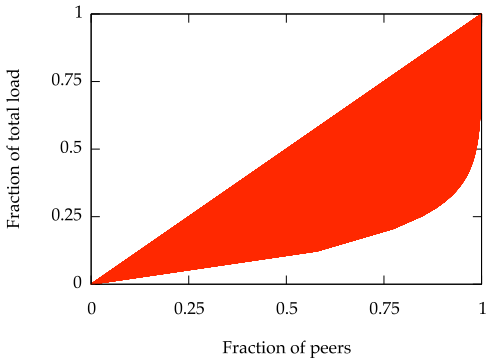


Figure 4.1: A Lorenz curve under a line of equality. The Gini coefficient is the red region as a fraction of the whole area under the diagonal (0.72).

example, G is calculated for the TOUT row in Table 4.4 using Equation 4.1, yielding a value of 0.3125. This low value indicates the distribution of total outgoing messages after all casts is quite fair over all peers (although because only three peers are involved this measurement is not particularly meaningful). The Gini coefficient is applied to all such totalled rows from the load matrix, i.e., TIN, TOUT and TINK (see Section 4.2). The metric is denoted with a subscript G. For example, an implementation exhibiting low TINK_G imposes similar total link load across network links after all casts. It is only useful to apply the Gini coefficient to metrics for total loading. It is not desirable to "fairly" load all peers in a system for *each* cast. It is sufficient instead to insist that no individual peers are heavily loaded for any single cast.

The Gini coefficient is used to measure inequality in the HotRoD P2P network [87]. The authors report that within the context of peer loading in a DHT, a fair distribution has a Gini coefficient in the range 0.5–0.65, while an unfair load distribution is in the range 0.85–0.99.

Note that fairness (METRIC_G) and efficiency (METRIC_M) are independent. A high degree of fairness does not mean the system is lightly loaded; it could be that every peer is equally heavily loaded. However, if a system exhibits high fairness and low maximum loading, the system can be said to be fair and efficient.

4.2 Metrics

The previous section introduced the three facets by which an IGM implementation can be analysed. Each facet comprises specific metrics which are detailed in this section. Peer and network facet metrics are defined in terms of the load matrix of Section 4.1.1. Cast facet metrics are defined separately. Table II summarises the notation used for analysing implementations, and Table III summarises all metrics.

Table 4.5: Sample storage load vector of peers for a run. $STOR_M$ = 4800 and $STOR_G \approx$ 0.14.

	Peer 1	Peer 2	Peer 3
STOR (bytes)	2600	4800	3100

4.2.1 Peer facet metrics

The peer facet relates to the load on individual peers as they participate in an IGM system. These metrics are based on the summary vectors from the load matrix.

Outgoing load (POUT/TOUT)

When a publisher casts a message, it expects to send just one outgoing message. Depending on the delivery model employed some peers will sometimes need to forward messages on behalf of others. Each peer tracks the number of cast messages it forwards on behalf of others for each cast. When used to record these messages, the aggregate column of the load matrix yields the per cast outgoing load (POUT), and the aggregate row yields the total outgoing load (TOUT).

Incoming load (PIN/TIN)

By participating in an IGM system, peers expect to receive exactly one message for each cast that selects their registration. Depending on the delivery model employed some peers will sometimes need to handle additional incoming messages. When used to record these messages, the aggregate column of the load matrix yields the per cast incoming load (PIN), and the aggregate row yields the total incoming load (TIN).

Storage load (STOR)

This metric measures the storage load of peers (in bytes) and includes all data a peer must store in support of the system, or on behalf of others. STOR is independent of the number of casts, so efficiency and fairness metrics are calculated using a storage load vector (Table 4.5).

4.2.2 Network facet metrics

The metrics comprising the network facet measure the link loads imposed by individual casts and over all casts. Each is calculated in terms of physical packets between routers and/or hosts. Links are considered to be directional so there are two links between each pair of routers and the number of packets they carry is measured independently.

Table 4.6: Sample matrix of load of all physical links and casts in a single run.

	Link 1	Link 2	Link 3	PINK
Cast 1	12	2	15	15
Cast 2	1	0	9	9
Cast 3	35	1	21	35
TINK	48	3	45	

Link load (PINK/TINK)

The load matrix is adapted for link load (Table 4.6). When used to record link packets, the aggregate column of the link load matrix yields the per cast link load (PINK), and the aggregate row yields the total link load (TINK).

4.2.3 Cast facet metrics

These metrics measure performance of an IGM implementation, as distinct from the load distribution and effect on the physical network. While not directly related to load distribution, performance is important when determining which implementations to use for particular domains. The primary cast facet metrics are measured as ratios to the baseline CENTRALISED implementation (Chapter 5).

Ratio of total hops (RTH)

In the CENTRALISED implementation, a publishing peer sends a single message to the server, which replicates the message to all matching peers in the system. This results in a minimal total number of overlay hops for each cast c. $|\mathcal{P}_{c_t}|$ is the size of the implicit group selected by c_t.

$$hops(c) = 1 + |\mathcal{P}_{c_t}| \tag{4.2}$$

The total message overhead of an IGM implementation is found by calculating the ratio of total overlay hops needed for a cast as compared to this minimal number. The RTH for cast c is shown in Equation 4.3, where $out[p][c]$ is the number of overlay messages forwarded by peer p for cast c. Note that the CENTRALISED implementation will always have an RTH of 1, by definition.

$$RTH(c) = \frac{\sum_{p=1}^{n} out[p][c]}{1 + |\mathcal{P}_{c_t}|} \tag{4.3}$$

Ratio of average/maximum hops (RAH/RMH)

The Ratios of Average and Maximum Hops measure the overlay hops taken for an implementation to deliver a cast to the average and final group member, as compared to

the CENTRALISED implementation. In Equations 4.4 and 4.5, $hops[c][p]$ is the number of hops taken to deliver cast c from the source to consumer p. In the CENTRALISED implementation, it always takes two hops to deliver a cast from a publisher to a consumer (via the server).

$$RAH(c) = \frac{avg(hops[c][p])}{2} \tag{4.4}$$

$$RMH(c) = \frac{max(hops[c][p])}{2} \tag{4.5}$$

Latency

Latency is measured in seconds from the time a message is published to the time it is received by each consumer. The overall latency of an implementation determines the types of possible applications. Latency measurements are presented in Chapter 9, where realistic network topologies and peer links are modelled.

4.2.4 Summary

This section has presented an analysis framework for IGM systems that permits an understanding of load distribution and performance from three perspectives: peer; network; and cast. The metrics used in the peer and network facets are based upon a load matrix and can determine whether any individual peers are overloaded on a per cast basis or over many casts. Using the Gini coefficient, the framework can also show how fairly the overall distribution of load is in a system. The remainder of this chapter describes the experimental design used in the evaluations of the IGM implementations presented in Chapters 5–8.

4.3 Experimental design

This thesis is concerned with the load distribution of casting in IGM models, and in particular the hypothesis that the structured P2P SPICE implementation fairly and efficiently addresses the inherent loading problems of IGM characterised in Section 3.4.

The argument proceeds by evaluating a set of casts under each implementation against the peer, network and cast facets. The implementations are OMNeT++ / INET [117] discrete-event simulations run on machine clusters coordinated with a bespoke OMNeT++ management tool called ROMNeT [118]. The properties of each implementation are determined via three primary experiments which test load distribution with respect to:

1. number of peers;

2. number of casts;

3. Zipfian skew of registrations and casts.

Each of these experiments tests an aspect of how well load is distributed by an implementation. Specific implementations also require additional experiments, detailed in each chapter.

By default, each simulation arranges 4096 peers around a generated network topology and simulates 1024 casts between them. Peer registration and cast skew (collectively "data skew") are Zipfian with exponent 1.0. Results are the average of five runs with different peer registrations, casts and random seeds; results with more runs do not show an appreciable difference. Errors shown on graphs in the following chapters are 95% confidence intervals. Table I shows the default values for all parameters in the simulations.

4.3.1 Router-level network topologies

Networks may be modelled at many levels, depending on the type of simulation that is needed. Large "semi-managed" networks like the global Internet are particularly difficult to represent as they are not controlled or designed by a single authority but by many *autonomous systems* or ASes (such as ISPs, companies, etc.). Each of these usually has a planned topology connecting *points of presence* (POPs), which are essentially client access points to the network. At a lower level still are the *routers* which are the physical machines that connect to one another via physical network links such as fibre optic cable.

The earliest approach to generating network topologies was by Waxman [119] and was based on Erdös-Rényi random graphs [120]. They are formed by placing nodes at random in a bounded 2D area and creating an edge between all pairs according to a probability function weighted by their Euclidean distance. Subsequent efforts focused on mimicking the apparent hierarchical structure of the Internet. These structural generators, such as Tiers [121] and GT-ITM [122], are based on design heuristics that network designers may employ when constructing a network (such as connecting high-level nodes via a minimal spanning tree to minimise the cost of cabling). More recently, there has been strong interest in applying graph theory to topology generation. For instance, the Internet has been shown to follow a power-law distribution at the AS level [123], a fact exploited by modelling tools such as BRITE [124], INET [125] and GLP [126].

Concurrently, there have been projects designed to measure the real topology of various parts of the Internet via several techniques such as BGP and WHOIS traces. The Rocketfuel project [127] has accurately mapped several large ISPs in high detail at

the router level. The DIMES project [128] aims to discover the Internet topology properties at the AS, POP and router levels. Unlike other approaches, DIMES is designed to measure networks from multiple vantages which may be combined to approximate the actual topology better.

There is some dispute as to which network topology generators create more realistic topologies. Claims that the Internet is best described by power laws at all levels suggest generators built on this notion are the most realistic [123, 129]. Some research, however, suggests that although the AS level may follow power-law distributions, router level topologies do not [130, 131]. Recent measurements of real ISP router level topologies also support this [127]. Thus, tools like GLP are most appropriate for AS level topologies and structural tools such as GT-ITM generate more realistic router level topologies [132, 133].

The network facet of the analysis framework measures packets over physical links between routers. Therefore, based on this review of network topology generation, the topologies used in the individual implementation evaluations are created with GT-ITM. The parameters proposed by Heckmann et al [132] are used to generate national ISP-style router level topologies, based on the U.S. AT&T national network. These generated topologies have approximately 150 routers interconnected with 180 network links. Links are reliable (meaning no packets are corrupted, lost or delivered out of order) and in order to measure the number of packets transmitted by implementations without reaching contention limits, the links are also assumed to have infinite bandwidth. Cross-traffic is not introduced but link latency is modelled, as this is very important for analysis of implementations under the cast facet.

Actual wide area and intercontinental network topologies are used in Chapter 9 based upon measurements compiled from projects such as Rocketfuel. A more detailed description of those networks is provided in that chapter.

4.3.2 Peer and cast data

There are two important data sources required to simulate an IGM system: peer registrations and cast target expressions. The simulator (described in detail in Section 4.3.3) accepts a list of peer registrations (sets of tags that are registered by peers as they join) and a list of casts that are to be delivered over the course of the experiments.

As discussed in Section 3.4, it is important to be able to experiment with increasingly skewed peer registrations and casts in order to assess how load distribution is affected. Synthetic data sources can be made to approximate a Zipfian distribution with a specific skew using Equation 4.6 which describes the fraction of the time the i^{th}

Table 4.7: Probability of most common tag being selected by a peer.

Exponent	Tags per peer				
	1	6	12	18	24
0.00	0.000	0.000	0.000	0.000	0.001
0.50	0.003	0.016	0.031	0.046	0.061
1.00	0.090	0.432	0.678	0.817	0.896
1.50	0.384	0.946	0.997	1.000	1.000
2.00	0.608	0.996	1.000	1.000	1.000

element of an m-element Zipfian distribution with exponent s appears.

$$frac(i, s, m) = \frac{1}{i^s H_{m,s}} \tag{4.6}$$

$H_{m,s}$ is the m^{th} generalised harmonic number:

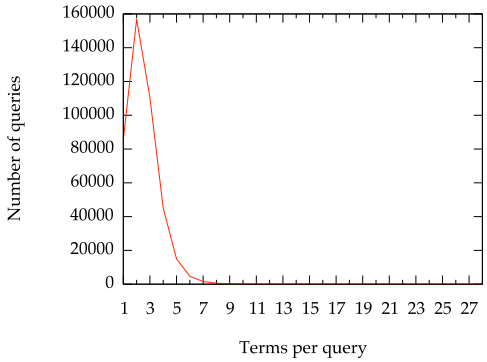
$$H_{m,s} = \sum_{k=1}^{m} \frac{1}{k^s} \tag{4.7}$$

It is straightforward to generate a distribution using this formula from which random elements can be drawn with appropriate probability. In these simulations, cast target expressions and peer registrations are created by selecting a number of distinct tags at random from a distribution of $\approx$ 37 000 tags, so there is a 9% probability of selecting the most common tag from a Zipfian distribution with exponent 1.0. If peers select k tags each, the probability of one of these being the i^{th} most common is:

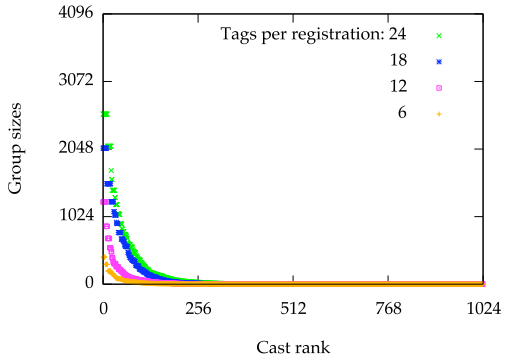
$$P(i) = 1 - \left(1 - \frac{1}{i^s H_{m,s}}\right)^k \tag{4.8}$$

Table 4.7 shows Zipfian exponents with the corresponding probability of an individual peer registering the most common tag, for different numbers of tags per peer. This gives an indication of the sizes of implicit groups that may result. For instance, if 4096 peers register 18 tags each from a Zipfian distribution with skew 1.0, a cast to the implicit group using just the most common tag will need to be delivered to approximately 3346 peers.

The number of conjunctive tags per cast chosen for the following experiments is based on an analysis of search queries from a major Internet search engine, Excite [134], originally provided by Spink et al [135, 136], who have performed several studies of such data. The search logs are from an unknown period of time in 1999 and reveal that of 455 743 queries, two terms per query is by far the most frequent occurrence (Figure 4.2(a)). This value is used in subsequent experiments. However, Chapter 9 uses a real data source as the basis of casts which has a variable number of tags per target expression.

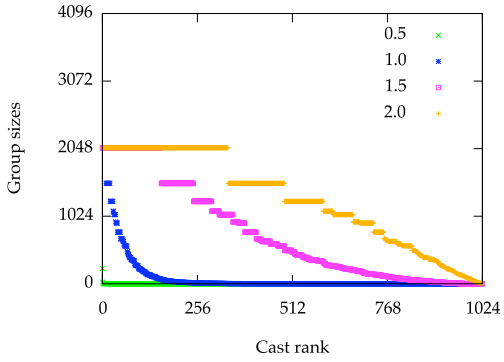


(a) Frequency of terms per query in the Excite search engine query logs.

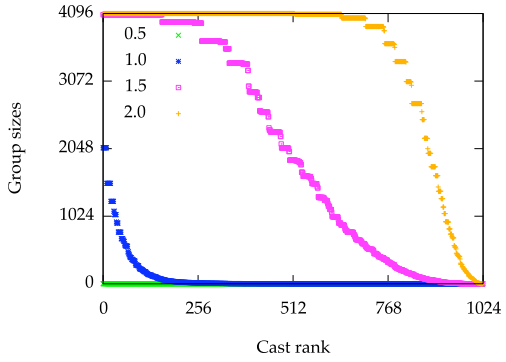


(b) Sizes of implicit groups formed (from population of 4096) as peers register more tags each.

Figure 4.2: Varying tags per peer registration and cast target expression.



(a) Peer registration skew fixed at 1.0, cast skew varied from 0.5 to 2.0.



(b) Peer registration and cast skew varied simultaneously from 0.5 to 2.0.

Figure 4.3: Sizes of implicit groups (from population of 4096 peers) for each of 1024 casts as Zipfian skew varies.

Figure 4.2(b) shows the effect on implicit group size when peers register different numbers of tags (maintaining two conjunctive tags per cast). Group size and frequency do not increase greatly beyond 18 tags per peer. This is the value chosen for the evaluation in later chapters as it results in a good variety of small and large groups suitable for evaluation, and is a reasonable number of tags for a real application.

Figure 4.3 illustrates empirically the sizes of implicit groups that result from varying the skew of both peer registrations and casts. Each peer registers 18 tags and casts are composed of 2 conjunctive tags. Figure 4.3(a) fixes the peer registration skew to 1 and varies the cast skew, while Figure 4.3(b) binds the two and varies them simultaneously. Generally, a high skew for peer registrations results in larger implicit groups and increasing the cast skew increases the number of casts selecting these large groups.

In the evaluation in Chapters 5–8, corresponding peer registrations and cast target expressions are generated from the same tag distribution so that the common tags appearing in casts are also common among the peer registrations. This ensures that as the skew increases, more peers are selected by more casts.

Multiple distinct groups of similar size

It is necessary for the experiments in Chapter 9 to cast messages to many different groups of similar size in order to investigate the effect of group size on performance and load distribution.

A set of tags is created, each of which is assigned to a random set of peers. The number of peers assigned each tag is constant, but the actual peers are random. Thus, the probability of any single peer having any tag is known. Depending upon the number of terms desired in each cast, the fraction of peers to receive each tag can be calculated. Let g be the probability of any given peer having a particular tag. Then the probability of a peer having any k tags is g^k. Thus, to ensure groups of a given size for any combination of k tags, each tag must be distributed over j of n peers, where:

$$j = n \sqrt[k]{g} \tag{4.9}$$

The total number of distinct tags needed, a, is dependent on the number of casts, c, and the number of conjunctive tags in each cast, according to the following inequality.

$$\binom{a}{k} \geq c \tag{4.10}$$

Thus, to generate c casts, all k-combinations of a distinct tags are generated. To generate the peer registrations, the a tags are each randomly assigned to g of n peers. Additional unique attributes are added to peers such that all have approximately the same number. These extra attributes are never part of a cast and thus do not affect the group sizes.

4.3.3 Simulator

The simulator is designed to run each implementation through several phases in the order detailed below. In a deployed system, these phases would not exist, but they are needed in these simulations to support the analysis of the casts using the analysis framework. The framework is designed specifically to analyse load distribution during casts, so the evaluation in Chapters 5–8 is conducted over the cast phase (see below).

Landmark phase

Landmarks are well-known peers in a network. During this phase, each peer contacts each landmark and records the round trip time (RTT) in a table. The table is used for different purposes depending on the implementation. In the BROKER implementation, landmarks take the role of brokers to which client peers connect. In SPICE, landmarks help map the structure of the P2P overlay to the physical network topology for efficiency reasons. The implementation chapters describe these processes in detail.

Note that the landmark phase is not a necessary part of SPICE or IGM in general. Implementations could be constructed not to rely on well-known landmarks. For instance, client peers could use service discovery to find servers or brokers, and SPICE peers could use a decentralised system such as Vivaldi [80] to improve the overlay/network mapping.

Join phase

In this phase, every peer joins the IGM system. For the CENTRALISED and BROKER implementations, peers need only connect to a server or broker. Brokers then form a connected overlay using the landmark tables as a guide. In SPICE, each peer joins the distributed structured P2P network. Chapters 6 and 7 describe these processes fully.

Registration phase

At this point, each consumer registers its tags with the system, according to the data provided by the process described in Section 4.3.2. For the CENTRALISED and BROKER implementations, clients register with the server or their local broker. For SPICE, peers need to register their tags at various points in the structured P2P network (see Chapter 7).

Replication phase

This phase is specific to the SPICE implementation and is used to perform replication of data structures as part of the load distribution process (see Chapters 7 and 8 for a full description). The BROKER and CENTRALISED implementations do not include this phase.

Cast phase

During the cast phase, all of the casts are initiated and delivered serially at a rate of one per second, without the interference of other traffic. Casts are issued in this way to isolate clearly the costs of individual casts, which are needed for the various facets of the analysis framework. The casts are provided to the simulator as a list generated by the process described in Section 4.3.2. This phase is the most important, as it tests

the load balancing properties of the implementations. All evaluation results are based on this phase of simulation.

4.4 Summary

This chapter has presented a framework for analysing implementations of IGM systems according to three facets—peer, network and cast—and detailed the specific metrics used (Contribution 6). Fairness across peers is measured with the Gini coefficient, which is commonly used in other fields to quantify inequality. The experimental design used for the evaluation of IGM implementations in the following chapters has also been described, including the phases of the simulation process. Physical network topologies are generated by GT-ITM, and the data sources are synthetically created according to Zipfian distributions of variable skew.

Chapter 5

CENTRALISED

The following chapters describe in detail the design and evaluation of three implementations of implicit group messaging with the tag-based modelling language described in Chapter 3 (Contributions 7 and 8). These are called the CENTRALISED, BROKER and SPICE implementations. Each is treated individually in the following chapters and accompanied by a theoretical and simulation-driven analysis of its properties based on the IGM analysis framework of Chapter 4.

The client/server CENTRALISED implementation is conceptually the simplest, and well-suited to theoretical analysis. Chapter 6 follows with the BROKER implementation which seeks to address the shortcomings of the CENTRALISED implementation by decentralising the single server into a broker backbone spanning the physical network. Chapters 7 and 8 are devoted to the intricacies of SPICE. This implementation extends the theme of decentralisation to its logical extreme in the form of a distributed structured peer-to-peer overlay designed to distribute load fairly over all peers and network links irrespective of the frequency of casts or sizes of implicit groups.

5.1 Design

IGM is readily implemented as a client/server CENTRALISED overlay as it directly corresponds to the conceptual IGM model outlined in Chapter 3 (reiterated in Figure 5.1). This implementation serves to demonstrate the inherent loading problems of IGM, and act as a baseline argument for more advanced implementations.

The CENTRALISED implementation consists of a single server peer (or equivalently, a server cluster) with the remainder of peers acting as clients communicating only with the server. When a consumer joins the system, it sends its registration to the server where it is stored. A publisher casts a message by sending it to the server which then calculates all selected group members from the registrations and unicasts it to each.

The server thus has three responsibilities: storing registrations from consumers; selecting implicit group members based on cast target expressions; and acting as the

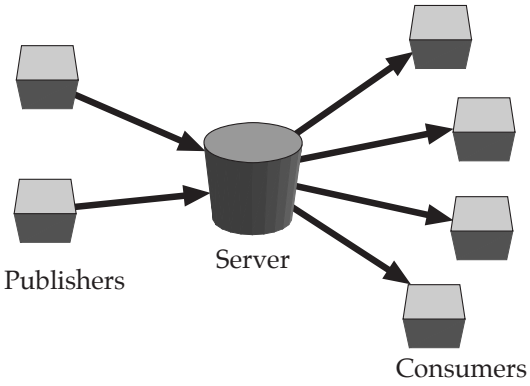
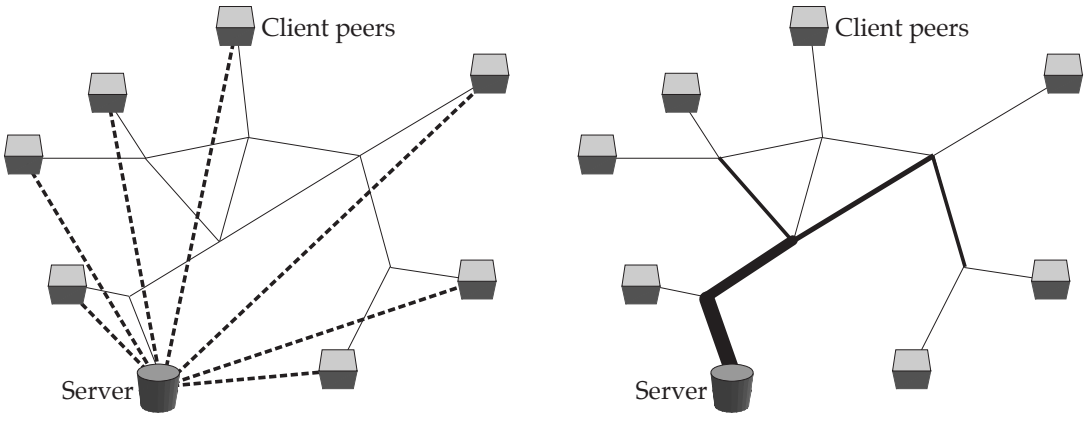


Figure 5.1: The CENTRALISED implementation maps directly to the conceptual model of IGM. All registrations and casts are routed through a central server.



(a) The overlay on a physical network.

(b) Links near the server are heavily loaded.

Figure 5.2: The CENTRALISED implementation.

nexus for all messages between publishers and consumers. Client peers do no work beyond publishing and receiving casts.

Figure 5.2(a) demonstrates how the overlay design maps to an underlying physical network. All communication is in the form of unicast messages between client peers and the server. Consequently, links near the server carry many incoming messages. In the case of casts to large groups, links radiating from the server carry myriad redundant copies of the same cast (illustrated in Figure 5.2(b) with emphasised links).

The modelling language used for the implementations in this thesis is based on tags (Section 3.3). Peers register by sending a registration message to the server. The server stores the registrations in a table in order to map cast target expressions to the selected implicit group members. In this implementation, the table is simply a list scanned by the server. In practice, this may be efficiently implemented with an inverted index (a technique commonly employed by search engines [137]), where each tag is mapped to the addresses of peers that have expressed it. By finding the intersections and unions of these sets based on the target expression, the server can determine the set of peers that are selected by the target expression.

Listing 1 shows simplified algorithms for the CENTRALISED implementation. The listings in this dissertation are presented in object-oriented pseudocode resembling a high-level language called Ruby [138, 139]. Appendix A offers a short guide to reading the language.

Table 5.1 defines the important fields in the objects used by the algorithms (and the *selects?* method of the target expression object which implements the $\triangleright$ function). The `reg` and `cast` methods of the IGM interface from Chapter 3 are implemented by client peers. `handle_X` methods are invoked on the server when it receives a message of type `X`. The `send` method is assumed to transmit a message between two peers (over unicast). The `notify` method (from the IGM interface) is identical to `send` but is used to distinguish a cast's final hop to a consumer.

5.2 Conformance

This section shows that the design satisfies the liveness and safety conditions of the IGM specification (Section 3.2). These conditions discretise time into an interleaved series of states and operations called a trace. Note that because this is a distributed system, the ordering of operations invoked by separate components (peers) must be serialised for analysis. The act of sending a message between peers (with the `send` and `notify` methods) is thus assumed to be an operation that is reliably handled by the recipient prior to some future state, without guaranteeing that it is necessarily the

Table 5.1: Object structure for CENTRALISED algorithms.

(a) A peer object.

Peer	registration

(b) A registration object.

Registration	peer	tags

(c) A cast object.

Cast	target	payload

(d) A target expression object.

Target	*selects?(registration)*

Listing 1 CENTRALISED algorithms.

```
1  # Client peer p registers.
2  def reg(p)
3     # p sends the server its registration.
4     p.send(server,p.registration)
5  end
6
7  # Client peer p casts c.
8  def cast(p,c)
9     # p sends c to the server.
10    p.send(server,c)
11 end
12
13 # Server handles registration from peer p.
14 def handle_reg(server,r)
15    # Server stores the registration.
16    server.registry.push(r)
17 end
18
19 # Server handles cast c.
20 def handle_cast(server,c)
21    # Iterate over all consumers and notify those selected.
22    # Since peers have exactly one registration, no peer can
23    # be notified more than once.
24    server.registry.each do |r|
25       server.notify(r.peer,c) if c.target.selects?(r)
26    end
27 end
```

next state:

$$\Box[send(p, m) \Rightarrow \Diamond handle(p, m)] \tag{5.1}$$

5.2.1 Liveness

The liveness condition (Theorem 3.7) states that if p registers, then from some point in the future onwards, p will eventually be notified of each cast that selects its registration.

$$\Box\,[reg(p) \Rightarrow \Diamond\Box[(cast(q, c) \wedge c_t \triangleright p_r) \Rightarrow \Diamond notify(p, c)]] \tag{5.2}$$

If after state s_a a peer registers (Line 4 of Listing 1), then a registration is sent to the server and inserted into the registry (Line 16) by some future state, s_b (by Predicate 5.1). If after state s_c a publisher casts a message (Line 10), then it is sent to the server which scans the entire registry by state s_d. If the cast selects the registration previously inserted, then the server notifies the peer (Line 25), which receives the notification by state s_e.

Registrations are never removed from the registry and the entire registry is scanned each time a cast is received. Therefore the server cannot fail to notify selected consumers of new casts after a registration has been inserted into the registry.

Hence the CENTRALISED design satisfies the liveness condition.

5.2.2 Safety

The safety condition (Theorem 3.8) states that when a consumer is notified of a cast: it will not be notified again; the message was previously cast by some publisher; and its registration is selected by the cast's target expression. Additionally, the condition requires that the consumer has completed a registration in order to agree with the adjusted definition of an implicit group.

$$\Box\,[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c) \wedge \exists q.c \in C(q) \wedge c_t \triangleright p_r \wedge p \in R] \tag{5.3}$$

To demonstrate the design satisfies the safety condition, it is sufficient to show it satisfies each of the component conditions.

$\Box[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c)]$ This component says that a consumer must never be notified more than once for a particular cast. Communication between all peers is assumed to be reliable so any cast published will be received by the server exactly once (Line 10). By the definition of IGM for this thesis, each peer has exactly one registration. Therefore the server's registry may contain at most one entry for each

peer. Only the server notifies consumers of casts (Line 25) found by iterating over the registry once only. Therefore no peer can receive the same cast more than once.

$\Box[notify(p, c) \Rightarrow \exists q.c \in C(q)]$ This component requires that a message must have been cast before a consumer is notified of it. The only peer that notifies consumers is the server, and only then when handling casts it has received (Line 25). Cast messages are only initiated by publishers (Line 10). Therefore, consumer p can only be notified of cast c if some publisher q has cast it; i.e., if $c \in C(q)$.

$\Box[notify(p, c) \Rightarrow c_t \triangleright p_r]$ This component requires that only consumers with registrations selected by a cast's target expression are notified, which is expressly specified in Line 25. No peers besides the server notify consumers of casts.

$\Box[notify(p, c) \Rightarrow p \in R]$ This component requires that a consumer has registered before it is notified of any casts. The server only notifies peers if an associated registration is found in its registry (Line 25). The registry is only modified when the server receives a registration (Line 16). The server only receives a registration that has been sent by a peer (Line 4). Therefore the peer must have registered prior to notification of any cast, i.e., $p \in R$.

Because the CENTRALISED design satisfies these components individually, it satisfies the safety condition generally.

5.3 Analysis

This section analyses the message complexity for registration and casting in the CENTRALISED implementation in terms of the facets described in Chapter 3, using the nomenclature in Table II. The analysis and evaluation of this implementation are primarily to demonstrate the use of the metrics and the types of experiments to be performed on the more complex implementations.

5.3.1 Registration

A peer registers simply by sending a registration to the server (Line 4 in Listing 1) which stores it in a local table. Hence registration complexity is $O(1)$.

5.3.2 Peer facet

The CENTRALISED implementation is extremely unfair towards the highly loaded server. It stores all system state, handles all casts and forwards copies of all casts to group members. Because each cast is handled by the server, the maximum per cast

incoming load (PIN_M) over all peers is always 1. Similarly, the maximum total incoming load (TIN_M) is always the total number of casts published, $O(c)$, and TIN_G tends to 1.0 (total unfairness) with the number of peers. The maximum per cast outgoing load (POUT_M) is always the size of the implicit group selected by each cast, since the server must forward copies to each group member. Because groups may comprise the entire network, POUT_M is thus $O(n)$. The maximum total outgoing load (TOUT_M) is the sum of all group sizes messaged, $O(cn)$, and TOUT_G also tends to complete unfairness with the number of peers. Finally, client peers store no state but the server stores a registration for each client, so STOR_M is $O(n)$ and STOR_G tends to complete unfairness.

5.3.3 Network facet

The link from the network to the server carries each cast to be dispatched, and the corresponding link from the server to the network carries replicas of each outgoing cast to all consumers. The network metrics PINK_M and TINK_M are thus concomitant with metrics in the peer facet. PINK_M assumes the maximum value of POUT_M and PIN_M, $O(n)$, and TINK_M is similarly the maximum of TOUT_M and TIN_M, $O(cn)$. Network links radiating away from the server towards the clients are progressively less loaded, as illustrated in Figure 5.2(b), so TINK_G is slightly fairer than TOUT_G generally.

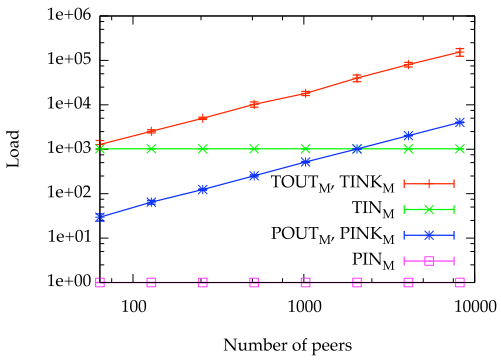
5.3.4 Cast facet

Casting in the CENTRALISED implementation involves almost the minimum number of peers possible for delivery: the publisher; the members of the selected implicit group; and the server. Thus the total number of hops for a cast is approximately the size of the implicit group, $O(n)$. The number of hops to each consumer from the publisher is $O(1)$ because each cast hops just once to the server and back to each consumer. Note that as the cast facet is defined in terms of the CENTRALISED implementation, RTH and RAH/RMH are always 1.

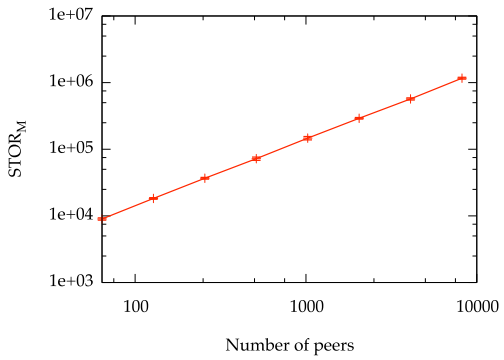
5.4 Evaluation

This section evaluates the CENTRALISED implementation through simulation. These experiments examine the load distribution properties of the implementation with respect to increasing numbers of peers and casts, and increasingly skewed peer registrations and cast target expressions. The default values used for all parameters are summarised in Table I.

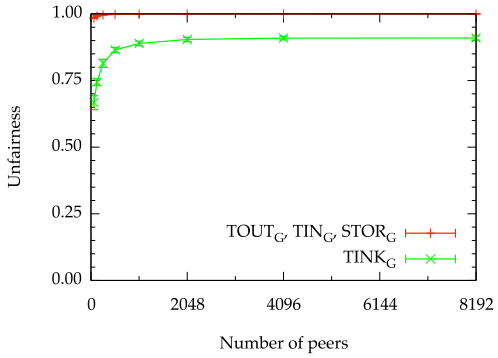
It serves no purpose to evaluate the cast facet here, as its metrics are defined in terms of the CENTRALISED implementation. Therefore, only the peer and network



(a) The server forwards casts to larger groups as the number of peers increases.



(b) The server stores a registration for every peer.



(c) Peer and network metrics show great unfairness because the server is the nexus of all casts.

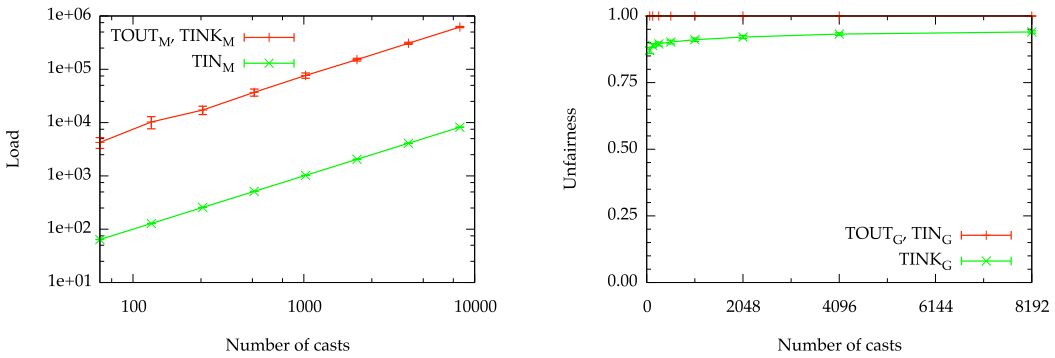
Figure 5.3: Peer scalability in the CENTRALISED implementation.

facets are evaluated in this section. The other implementations in subsequent chapters evaluate all facets: peer; network; and cast.

5.4.1 Peer scalability

The manner in which an implementation scales with the number of peers is of prime importance for a messaging system intended for many participants. Two effects are particularly important: the incoming and outgoing loads placed on individual peers; and the distribution of storage load over peers.

The sizes of implicit groups increase with the number of peers as a consequence of the peer registration distribution. For example, a target expression comprising a single common tag that is registered by 9% of all new peers will select increasingly larger groups with the number of peers. In these experiments, the data skew of 1.0, the 18 tags registered by each peer, and the two conjunctive tags per cast, produce some large implicit groups of approximately half of all peers (see Section 4.3.2). The server is required to forward casts to all of these group members, and Figure 5.3(a) shows that POUT_M rises at this rate. TOUT_M increases even more quickly because



(a) The server receives and forwards every cast over the life of the system.

(b) TINK grows more unfair since links near the server are consistently more loaded.

Figure 5.4: Cast scalability in the CENTRALISED implementation.

the server handles c=1024 casts in total. PINK$_M$ and TINK$_M$ grow equivalently due to their correspondence to POUT and PINK.

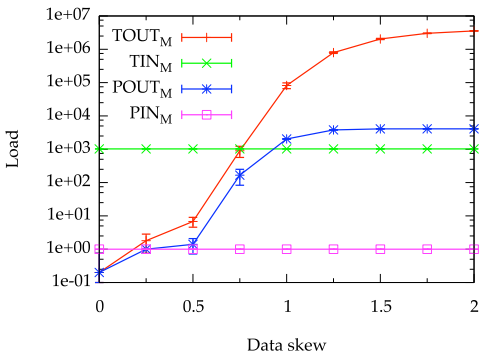
No peer stores any state except the server, which maintains a registration for every peer. Figure 5.3(b) shows that STOR$_M$ increases linearly with the number of peers.

Figure 5.3(c) confirms that in addition to loading the server heavily, the implementation is also very unfair. TIN$_G$ and TOUT$_G$ tend to total unfairness (1.0) because only the server sends and receives casts. Likewise, because only 1 of n peers stores any state, STOR$_G$ tends to 1.0. TINK$_G$ is a measure of total link load unfairness and is generally unfair since all casts traverse the same point in the physical network. It becomes more unfair with the number of peers because casts need to be forwarded to larger groups as more peers join the system. This is guaranteed to load links around the server heavily, but less likely to load distant links in the network. TINK$_G$ does not quite tend to 1.0, because although the load on more distant links in the network is low, it is not zero. The Gini coefficient used as the unfairness metric only reaches total unfairness when exactly one element is non-zero (as seen in TIN$_G$ and TOUT$_G$).

5.4.2 Cast scalability

The ability of an implementation to perform well over many casts is very important for a persistently operating IGM system. It is preferable not to load the same peers and network links consistently. By varying the number of casts, this experiment determines if any peers are consistently loaded. Clearly, storage load and per cast metrics such as PIN, POUT and PINK are not affected by the number of casts, so this section presents only metrics which measure totals over all casts, i.e., TIN, TOUT and TINK.

Figure 5.4(a) shows that as the server handles every cast, the maximum total incoming load (TIN$_M$) increases linearly. The server also delivers the cast to every



(a) Outgoing server load increases dramatically as large groups are frequently selected.

Figure 5.5: Data skew in the CENTRALISED implementation.

implicit group member, so the total outgoing load (TOUT_M) also increases linearly, though at a faster rate than TIN_M because the average group has many members. Figure 5.4(b) reveals that total link load (TINK_G) becomes increasingly unfair with the number of casts due to the consistent loading of links near the server.

5.4.3 Data skew

The default Zipfian exponent used in this evaluation is 1.0 but the precise data skew for an IGM system may vary with the application domain. This section measures the loading on peers as skew is varied from 0.0 to 2.0 for both peer registrations and cast target expressions simultaneously. As the skew increases, more peers register the same tags, and cast target expressions are more often composed of those tags. With skew 0.0, very few casts ever select a non-empty group. With very high skew, the same large groups will tend to be selected repeatedly. Generally, increased data skew leads to more frequent casts to larger implicit groups.

A high data skew is the most difficult for an IGM system, and particularly for this implementation. It results in very high outgoing load on the server and surrounding links. In contrast, incoming load is not affected by skew in this implementation. It is consistently high since the server handles every cast regardless of the tags used in the target expression. Figure 5.5(a) shows how POUT_M and TOUT_M increase rapidly once data skew reaches a level sufficient to ensure that some large groups are selected. Beyond skew 1.5, POUT_M plateaus at n=4096, the number of peers in the network. At this degree of skew, some casts select a group comprising all peers and the server must forward the cast to each. TOUT_M increases to more than 3.5 million ($\approx cn$) at skew 2.0, because almost every cast selects most of the peers in the network.

5.5 Summary

This chapter has presented a baseline IGM implementation based on a client/server overlay that may be used as a point of comparison for more advanced implementations (Contribution 7). The CENTRALISED implementation is simple in design and lends itself well to analysis. The evaluation under three types of loading (peer, cast and data skew) has illustrated the use of the peer and network facets (Contribution 8).

There are inherent strengths and weaknesses to the design. The low investment of resources by client peers is desirable and the centralised nature permits straightforward administration, which may be a requirement in some situations. Furthermore, the modelling language may be arbitrarily complex since the Selector component can be implemented in a monolithic fashion. However, since the server is the nexus of all messages, it requires an initial investment in infrastructure. It is also potentially vulnerable to failure or attack which can lead to periods of unavailability for all participants.

The design also exhibits unenviable loading characteristics. Incoming load to the server grows with the number of casts which may be caused by external events triggering a slew of casts, or as an indirect consequence of an increasing number of client peers. These problems are not restricted to IGM systems, of course. Many incapable web sites have succumbed to sudden increases in traffic resulting in unavailability[1], though such problems can often be ameliorated by over-provisioning the server.

More troublesome, however, is that unlike web servers which need only return information to a single client per request, the number of recipients for a single cast may be very large and grows with the number of participants.

Thus the design does not scale naturally with the number of peers or the number of casts, and is susceptible to casts selecting large fractions of the peers. Together, these issues necessitate highly capable network links and the provision of powerful servers capable of handling not only average incoming and outgoing load but also unexpected spikes.

The shortcomings of the CENTRALISED implementation are caused by the reliance on a single server peer for the distribution of every cast, which concentrates both peer and network load at a single point. The problems may be diffused by replacing the single server with a network of servers; this approach is the basis of content delivery networks such as Akamai [100], for example. The next chapter focuses on such a design for IGM, called BROKER.

[1] Commonly known as "the Slashdot effect" for the popular eponymous news site that regularly directs thousands of readers to the same site at the same time.

Chapter 6

BROKER

This chapter presents and evaluates the BROKER implementation, which seeks to address the shortcomings of CENTRALISED with an overlay network of servers.

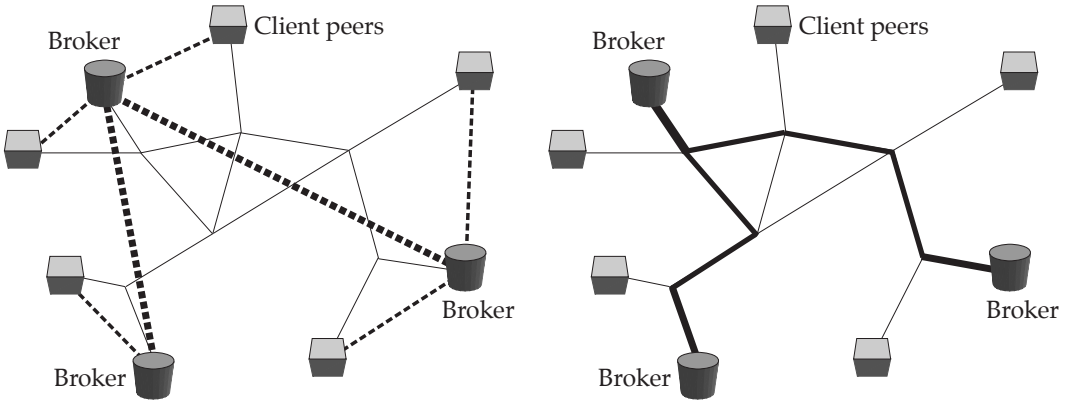
6.1 Design

The design of the CENTRALISED implementation concentrates peer and network load at a single point in the system, leading to general unfairness. The hypothesis behind the BROKER implementation is that this load can be diffused by decentralising the server into an overlay network of servers (or *brokers*), each of which services a share of the peers. BROKER distinguishes between client peers and brokers. Each client peer connects to and registers with a single broker which behaves towards its clients like a CENTRALISED server. Clients cast messages to their broker which propagates them to other brokers and onwards to other client peers.

A broker-based design is often employed by event-based distribution systems. Such publish/subscribe systems as SIENA [42], Gryphon [45] and REBECA [44] are based on the general concept of decentralised broker overlays. Brokers may be randomly, manually or automatically placed around the physical network. The overlay graph (also called the *backbone*) may be hierarchical, an unrooted tree or a more general cyclic graph.

In this implementation, b brokers are randomly placed and form a non-hierarchical, acyclic overlay. An acyclic overlay simplifies the implementation significantly yet still allows very efficient routing, and is one of the approaches used in SIENA [42]. This backbone overlay is constructed by finding the minimal spanning tree (MST), according to the round trip times (RTT) between each pair of brokers. Using the RTTs ensures that message delivery is quick, and more importantly that neighbouring brokers in the overlay are nearby in the physical network, reducing overall network traffic.

Because a minimum spanning tree is used to distribute single copies of casts between brokers, major network links between different parts of the network typically



(a) The overlay on a physical network.

(b) Links between brokers are heavily loaded over time.

Figure 6.1: The BROKER implementation.

need to carry just one copy of each cast. This design has several benefits: the outgoing load is spread over many brokers; the physical network is more efficiently utilised (although links between brokers must still carry many casts); and casts are delivered quickly to each consumer.

Figures 6.1(a) and 6.1(b) show a logical broker overlay mapped to a physical network. Figure 6.2 illustrates the acyclic overlay topology of a larger backbone. The backbone could be manually constructed by administrators or automated with distributed minimum spanning tree algorithms [140], but for the purposes of simulation it is calculated centrally.

The BROKER algorithms are shown in Listing 2 (with extended object structures in Table 6.1). To register, a client sends its registration to its broker which records it as if it were a server in the CENTRALISED implementation. The broker then floods the registration to the rest of the backbone so that casts published in distant parts of the network can be propagated back to selected consumers.

As a registration is flooded across the backbone, each broker records the tags it contains in *cast routing tables*. Each neighbour of a broker is associated with its own routing table so it is known which tags have been registered in specific subgraphs of the backbone (recall that the backbone is acyclic). Rather than flooding the entire backbone when a cast is published, this "backward link" allows the brokers to forward it to only those neighbours that may have client peers in their subgraph selected by its target expression.

In this scheme, brokers record the tags that a set of consumers have registered in a subgraph of the backbone, but the specific combinations of tags (i.e., the complete registrations) are not stored. Thus, it cannot be determined from a routing table whether

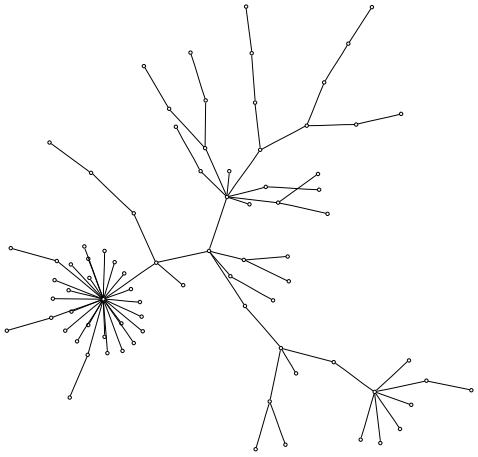


Figure 6.2: A backbone of 81 brokers, formed as part of an 8192-peer BROKER implementation.

any particular consumer in a subgraph has registered a specific combination of tags, only that for each tag there is at least one consumer that has registered it. As a cast is propagated along the backbone, the routing tables better approximate actual registrations of client peers because there are fewer consumers in the remaining subgraphs.

This approach is taken to prevent every broker potentially storing a copy of every registration. Brokers' routing tables are instead implemented as constant-sized Bloom Filters [93] into which the tags are inserted (see Section 2.6.1). The error rate of the Bloom Filters is called the *routing error*.

6.2 Conformance

This section shows how the BROKER implementation conforms to the liveness and safety conditions of the IGM specification (Chapter 3).

The BROKER design generalises the CENTRALISED implementation with an overlay of servers but client peers continue to communicate with exactly one broker, which stores their registrations and notifies them of new casts. It is sufficient, therefore, to demonstrate that the liveness and safety conditions are not violated by the introduction of the broker backbone.

6.2.1 Liveness

The liveness condition (Theorem 3.7) states that if p registers, then from some point in the future onwards, p will eventually be notified of each cast that selects its registration.

Presume initially that routing tables are not used to limit the flooding of casts over the broker backbone. Any cast sent to a broker from a client (Line 7 of Listing 2) can be flooded to all brokers using a breadth-first traversal (Line 33). Upon receipt of a cast

Table 6.1: Extended object structure for BROKER algorithms.

(a) A peer object.

Peer	registration	broker	*client?()*

(b) A broker object.

Broker	neighbours	cast_table

Listing 2 BROKER algorithms.

```
1  # Client peers.
2  def reg(p)
3     p.send(p.broker,p.registration)
4  end
5
6  def cast(p,c)
7     p.send(p.broker,c)
8  end
9
10 # Broker peers.
11 def handle_reg(broker,r)
12    if r.sender.client?  # Received directly from a client.
13       broker.registry.push(r)
14    else  # Received from another broker.
15       # Apply tags to the cast routing table associated with the
16       # neighbour from which this registration was received.
17       r.tags.each { |tag| broker.cast_table[r.sender].insert(tag) }
18    end
19    # Propagate registration to all neighbours.
20    broker.neighbours.each do |neighbour|
21       next if neighbour == r.sender # Don't backtrack to previous broker.
22       p.send(neighbour,r)
23    end
24 end
25
26 def handle_cast(broker,c)
27    # Notify selected client consumers of cast.
28    broker.registry.each do |r|
29       broker.notify(r.peer,c) if c.target.selects?(r)
30    end
31    # Selectively forward cast to other brokers.
32    broker.neighbours.each do |neighbour|
33       broker.send(neighbour,c) if neighbour != c.sender and
34          c.target.selects?(broker.cast_table[neighbour])
35    end
36 end
```

(from clients or other brokers), each broker notifies all selected consumers that have registered (Line 29) in the same way as the CENTRALISED implementation. Therefore the design satisfies liveness if every cast is flooded.

The cast routing tables are used to limit the casts that must be flooded. It must therefore be shown that no cast is prevented from reaching a broker with a selected client peer. Each broker maintains a cast table for each neighbour. The table can be thought of as a large "covering" registration containing many tags. A cast is forwarded to a neighbour only if it "selects" its cast table (Line 33), so every target expression that selects any of the client registrations beyond a neighbour must also select the neighbour's cast table.

More formally, let f be the cast table for a neighbour, and Q be the set of registrations of all clients beyond that neighbour in the broker backbone. It must be shown that:

$$\forall r \in Q, t \in \mathcal{T}, t \triangleright r \Rightarrow t \triangleright f \tag{6.1}$$

The cast table f is a *union* of all registrations $r \in Q$, meaning every tag in every registration of Q is also in f. When a client registers (Line 3) the registration is sent to its broker which stores and forwards it to its neighbours (Line 22). When a broker receives a registration, it inserts the tags into its cast table (Line 17) and continues propagating it (Line 22). Every registration in the subgraph of a broker's neighbour is therefore eventually received by the broker, and all tags are recorded in the appropriate cast table. Thus, f is the union of all registrations in Q.

$$\forall r \in Q, R(f) \supseteq R(r) \tag{6.2}$$

where $R(x)$ is the registration set of registration x (see Section 3.3).

By the definition of $\triangleright$ (Theorem 3.12 in Chapter 3) there must exist a target element K of $T(t)$ that is a subset of $R(r)$, for each $r \in Q$. But by Theorem 6.2, each such K must also be a subset of $R(f)$, so $t \triangleright f$ (by definition of $\triangleright$). Thus, any target expression selecting a registration beyond the neighbour also selects its cast table, and casts are forwarded to them.

Because of the acyclic structure of the backbone, this argument can be inductively applied to all subgraphs, so no cast is prevented from reaching a broker with a selected client. The BROKER design thus satisfies the liveness condition.

6.2.2 Safety

The design is shown to satisfy the IGM safety condition (Theorem 3.8) by treating each component of the condition individually.

$\Box[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c)]$ This component says that a consumer must never be notified more than once for a particular cast. Clients connect to and register with exactly one broker (Line 3). Internally, brokers behave like CENTRALISED servers when notifying consumers of casts, so the argument applied in that design also applies here. The broker backbone is acyclic and brokers never return casts to brokers from which they are received (Line 33). Hence each broker can only receive and process each cast at most once. Thus a consumer cannot be notified more than once of a particular cast.

$\Box[notify(p, c) \Rightarrow \exists q.c \in C(q)]$ This component requires that a message must have been cast before a consumer is notified of it. This argument works backwards from the consumers to the publisher. Only brokers notify consumers of casts (Line 29), and only when they receive them from a client (Line 7) or another broker (Line 33). If received from a client q, then that client must have published the cast, so $c \in C(q)$. If received from a broker, the cast must have originated from a client or other broker in the subgraph headed by that broker. The acyclic structure of the backbone ensures that there is a unique path from the broker to the cast's source because the same cast can never visit the same broker twice. Because the backbone is finite, this path must also be finite. Therefore, there exists a broker on the path of the cast that did not receive it from another broker. But the only other way for a broker to receive a cast is when a client publishes it (Line 7). Thus there exists a client q that published the cast; i.e., $c \in C(q)$.

$\Box[notify(p, c) \Rightarrow c_t \triangleright p_r]$ This component requires that only consumers with registrations selected by a cast's target expression are notified. Only brokers notify consumers of casts, and only then when their registrations are selected (Line 25).

$\Box[notify(p, c) \Rightarrow p \in R]$ This component requires that a consumer has registered before it is notified of any casts. Brokers act identically to CENTRALISED servers towards their client peers, so the same argument used in that design applies here. Upon receipt of a cast, brokers find all selected registrations for their clients and notify associated consumers (Line 29). Only registrations stored in the broker's registry can be found, and the registry is only updated when a registration is received from a client (Line 13). Registrations are only received from clients when a client has registered (Line 3); i.e., $p \in R$.

The BROKER implementation satisfies each component separately, and therefore satisfies the entire IGM safety condition.

6.3 Analysis

This section analyses the message complexity of registration and casting in the BROKER implementation. The analysis assumes that the number of client peers attached to each broker is equal. This may not always be the case in practice because clients join their closest broker. The distribution of publishers is assumed to be uniform (i.e., no client publishes more casts than any other). Peers are assumed to be unclustered in the sense that the location of a client in the network has no bearing on which tags it registers. In a real system, such an assumption may not hold: the geographic (and hence network) location of a peer will likely influence the tags registered.

6.3.1 Registration

Registration in the BROKER implementation essentially involves flooding a registration to all b brokers, so registration is $O(b)$. In practice, registration could be optimised by forwarding only those registrations that are not already covered by cast tables.

6.3.2 Peer facet

Generally, BROKER behaves similarly to CENTRALISED. Storage load is nil for client peers but each broker stores $O(\frac{n}{b})$ registrations. Brokers store an additional number of constant-sized Bloom Filters representing the routing tables. The number each stores is dependent on the number of neighbouring brokers, which is dependent on the physical network over which the broker backbone is constructed. If a degree-constrained minimal spanning tree were used, it would be possible to place an upper bound on this additional cost. In the implementation presented here, the backbone is not degree-constrained and it is possible for some brokers to be high-degree hubs with a consequently high storage load.

Due to the acyclic arrangement of broker peers, brokers receive no more than a single copy of each cast, and the cast routing tables may reduce this to nil for some brokers and casts. Thus the maximum per cast incoming load (PIN_M) is $O(1)$. The acyclic topology also produces bottleneck brokers that must propagate most of the casts published. The maximum total incoming load (TIN_M) is therefore $O(c)$.

Outgoing load is nil for client peers, as in the CENTRALISED implementation. Assuming a uniform distribution of client peers to brokers and an unclustered pattern of peer registrations, each broker has the same number of selected peers for each cast. Thus the maximum per cast outgoing load (POUT_M) is $O(\frac{n}{b})$. The maximum total outgoing load (TOUT_M) is similarly $O(\frac{cn}{b})$.

6.3.3 Network facet

Network links between brokers are consistently loaded, as they are required to carry copies of many casts. This loading can be reduced by limiting the number of brokers. However, this eventually reduces to the CENTRALISED implementation (with a single broker) and includes its attendant shortcomings. With a large number of brokers, inter-broker links are more loaded but each broker supports fewer, physically closer clients. This leads to a reduction of physical packets needed on links between clients and brokers. Figure 6.1(b) illustrates this point.

As in the CENTRALISED implementation, the network facet directly corresponds to the peer facet, with PINK_M assuming the maximum of PIN_M and POUT_M, and TINK_M the maximum of TIN_M and TOUT_M. This is because the most loaded physical network link is that connecting the most loaded broker to the network. Due to the spanning tree design, most physical links between brokers do not have to carry duplicate cast packets, so the total network load lessens as the ratio of brokers to clients increases.

Selecting the number of brokers is thus a design decision that trades broker–broker link load for broker–client link load. This parameter is investigated in Section 6.4.1.

6.3.4 Cast facet

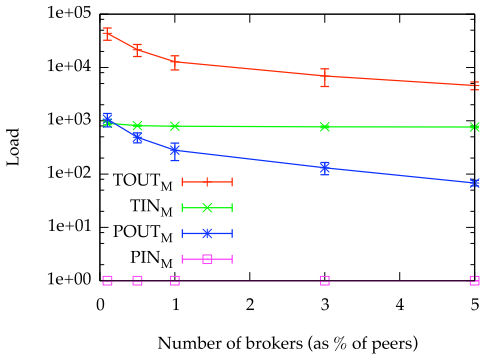
The number of overlay hops from publishers to consumers is dependent on the diameter of the broker backbone, which in turn is dependent on the physical network topology. In the worst case, the brokers may form a chain resulting in $O(b)$ RAH and RMH. In practice, the backbone is more likely to form a tree with $O(\log b)$ hops to each consumer from the publisher. The total number of hops needed to deliver a cast in the worst case includes every broker and every member of the selected implicit group.

6.4 Evaluation

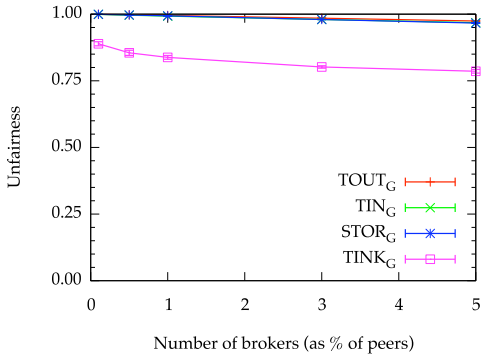
The BROKER implementation has two parameters—the number of brokers and the routing error—which are evaluated prior to the peer scalability, cast scalability and data skew experiments. In addition to the peer and network facets used in the evaluation of the CENTRALISED implementation, the cast facet is also employed to assess the comparative performance of the BROKER implementation. The default values used for all parameters are summarised in Table I.

6.4.1 Broker scalability

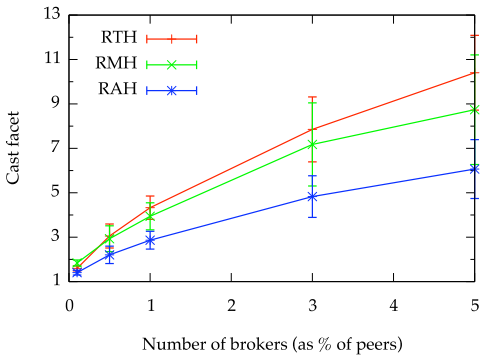
Increasing the number of broker peers (while keeping the number of client peers fixed) has three effects. Firstly, the outgoing cast load is spread over more peers. Secondly,



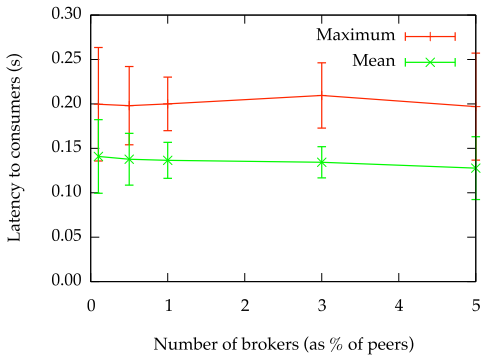
(a) POUT_M and TOUT_M are reduced by spreading cast notification over many brokers.



(b) All metrics become fairer to an extent with more brokers.



(c) Number of overlay hops to consumers increases as more brokers are added.



(d) The latency to consumers slightly decreases with additional brokers.

Figure 6.3: Broker scalability in the BROKER implementation

the network is utilised more efficiently by improving the congruence of the broker overlay to the physical network. Finally, because publishers and consumers are more likely to be physically near to brokers, casts can be more quickly routed over the backbone (which is a minimum spanning tree) from publishers to consumers, instead of routing via a potentially distant server.

Figure 6.3(a) shows that POUT_M and TOUT_M (and PINK_M and TINK_M) are indeed decreased by deploying additional brokers. Even deploying just 1% of peers as brokers significantly reduces maximum outgoing load per cast. Figure 6.3(b) shows that TINK becomes fairer as more brokers are deployed, due to the extended backbone restraining the number of duplicated cast packets traversing the same network links. Figures 6.3(c) and 6.3(d) present seemingly conflicting results. The number of overlay hops from publishers to consumers increases with the number of brokers because a larger backbone must be traversed. However, the average latency is reduced because these hops lie on a minimum spanning tree between publisher and consumer.

Based on these results, subsequent experiments rely on 1% of peers acting as brokers. This value is a good compromise between dramatically reducing POUT_M and TOUT_M, and maintaining low RTH.

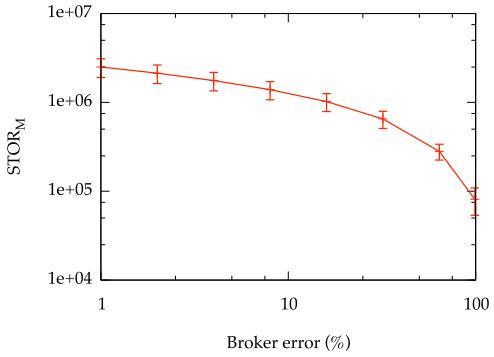
6.4.2 Routing error

Greater routing error means more Bloom Filter false positives are permissible, which results in smaller routing tables and reduced storage load (Figure 6.4(a)). However, the higher error rate potentially results in the unnecessary relay of casts between brokers in the overlay. TOUT_M, TINK_M, TIN_M and RTH increase with routing error (Figures 6.4(b) and 6.4(c)) because brokers are less able to filter messages from reaching parts of the backbone where they are unneeded.

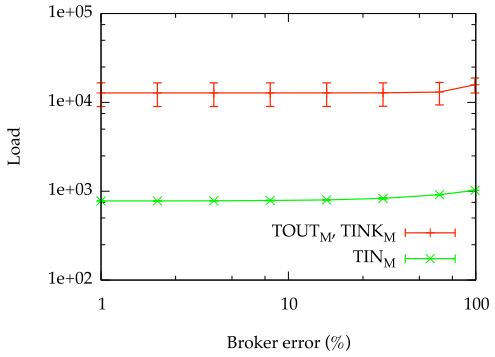
A routing error of 10% is used in subsequent experiments. This value is chosen to limit the disruption to TOUT_M, TINK_M, TIN_M and RTH as much as possible. A larger error does theoretically reduce storage costs but this is misleading since suitably large Bloom Filters must at any rate be deployed to accommodate unexpectedly many distinct tags. Moreover, as brokers are designed to act as routers and storage points in the network, they can be suitably provisioned for larger storage requirements.

6.4.3 Peer scalability

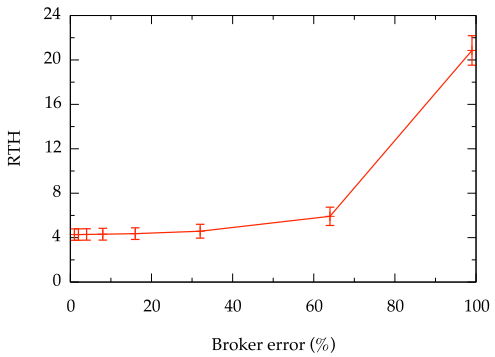
POUT_M and TOUT_M increase with the number of peers (Figure 6.5(a)), despite a proportionally equal fraction of brokers. There is considerable variation in the distribution of client peers to brokers due to the physical topology and the fact that clients join the broker that is closest. Hence, some remote brokers will have few clients whereas others may have several times the average. These more highly loaded brokers gener-



(a) STOR_M decreases significantly as more false positives are permitted in the brokers' routing tables.

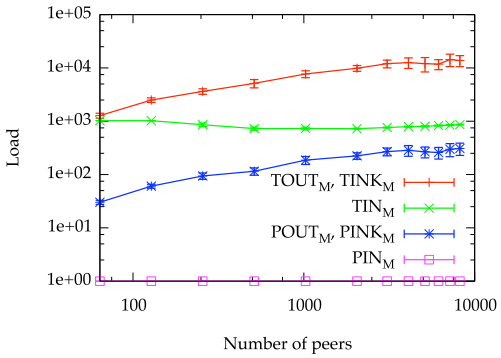


(b) TOUT, TINK and TIN increase slightly with routing error because fewer casts are prevented from flooding the backbone.

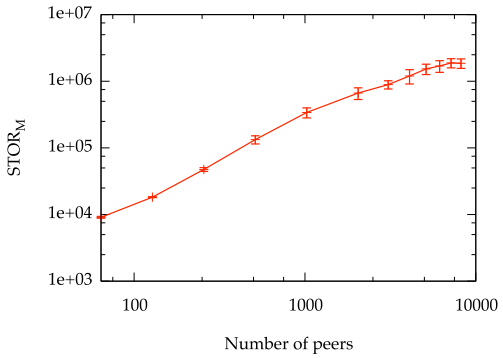


(c) RTH increases with routing error because the backbone is more often flooded unnecessarily.

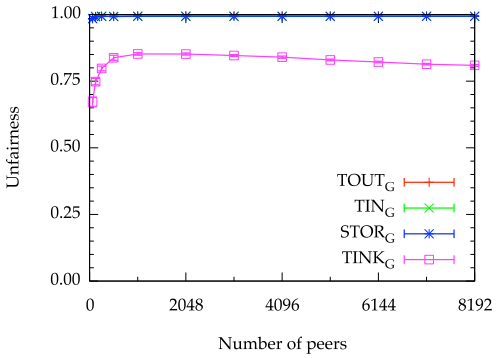
Figure 6.4: Routing error in the BROKER implementation.



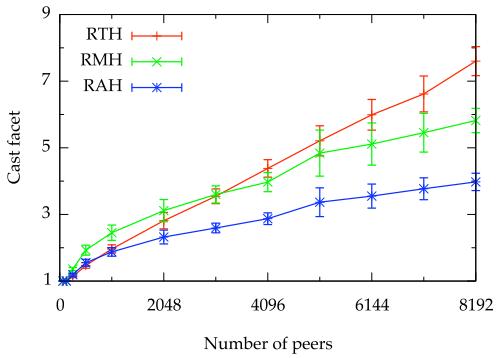
(a) Increased load is caused by unequal distribution of clients.



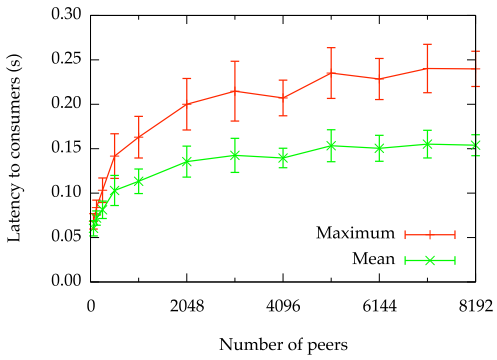
(b) More peers register distinct tags which must be incorporated into brokers' routing tables.



(c) More peers increases the number of brokers so fewer messages are replicated across links.



(d) The larger backbone formed in a larger network increases overlay hops to consumers.



(e) Latency plateaus because the growing backbone approximates the physical network.

Figure 6.5: Peer scalability in the BROKER implementation.

ally increase POUT_M and TOUT_M.

Although the decentralised BROKER implementation vastly inhibits the rate at which POUT_M and TOUT_M grow compared to CENTRALISED (Section 5.4.1), TIN_M is barely reduced. This is because some brokers, particularly those central to the backbone, must continue to process most casts.

STOR_M increases because an increased number of peers produce more distinct tags for registration, and larger routing tables between brokers are needed to maintain the same routing error rate (Figure 6.5(b)). Under the BROKER implementation, STOR_M actually exceeds that of the CENTRALISED implementation due to these routing tables.

Although the heavy loads are somewhat distributed, the overall unfairness of the implementation remains high. TINK_G becomes slightly fairer (Figure 6.5(c)) because the proportional increase in brokers means clients are nearer their selected broker. Hence broker–client link loads are more similar across all clients. The other metrics remain consistently unfair since there is always a small minority of loaded brokers compared to the total number of peers.

Figure 6.5(d) shows RTH increasing linearly with the number of peers, although RAH and RMH increase logarithmically. This is because the total size of the backbone is increasing, but the branching tree structure results in a logarithmic number of hops between any two brokers on average. Figure 6.5(e) shows cast latency growing very gradually. Beyond a certain size, the backbone closely approximates the bounded physical network topology, allowing efficient delivery of casts to consumers along a minimal latency path.

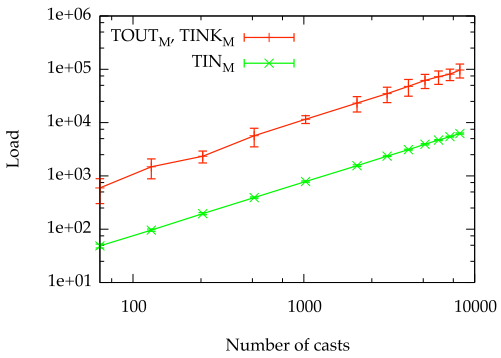
6.4.4 Cast scalability

Clearly, TOUT_M and TIN_M both increase linearly with the number of casts (Figure 6.6(a)), but it is important to note that TOUT_M increases at a much lower rate than the CENTRALISED implementation (Section 5.4.2). This is further evidence that decentralising the server to a backbone of brokers has a beneficial effect.

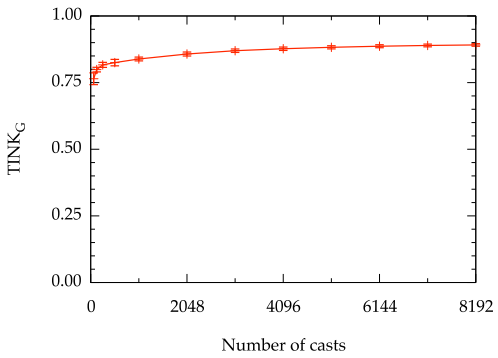
However, the implementation remains extremely unfair. The same broker peers handle casts while the majority of peers (clients) do not aid in cast delivery. Figure 6.6(b) shows that TINK_G grows more unfair as the number of casts increases, because the network links connecting brokers are more consistently loaded than any others.

6.4.5 Data skew

Figure 6.7(a) presents similar trends to data skew under the CENTRALISED implementation, though TOUT_M and POUT_M are an order of magnitude lower under high skew. This is essentially because the outgoing load is distributed over many brokers rather than a single server. TIN_M increases slightly with skewed data. With low skew,

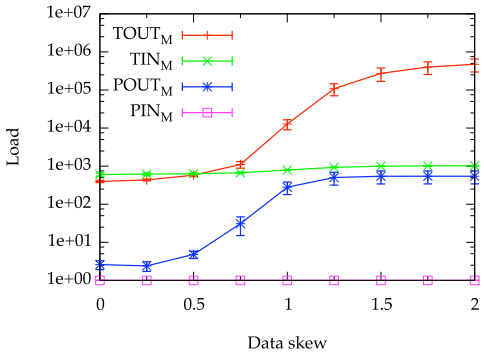


(a) Incoming and outgoing load on brokers increases with the number of casts.

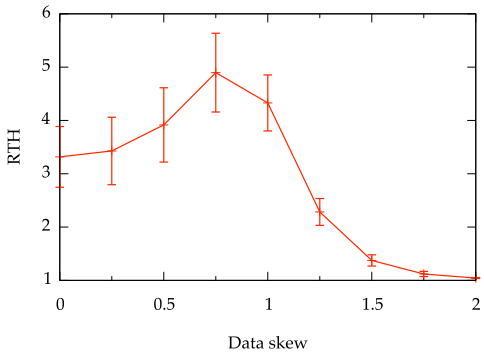


(b) Links between brokers are heavily loaded over many casts.

Figure 6.6: Cast scalability in the BROKER implementation.



(a) Outgoing load is a fraction of CENTRALISED, but still plateaus very high.



(b) RTH is highly dependent on the variety and sizes of implicit groups caused by data skew.

Figure 6.7: Data skew in the BROKER implementation.

groups are generally small or empty and casts do not need to be propagated across the whole broker backbone. Hence some brokers do not receive every cast, and TIN is generally lower. As skew increases, more casts need to be flooded across the backbone and the average TIN approaches c.

Figure 6.7(b) depicts the RTH rising and falling with increasing data skew. This effect is due to the variety and magnitude of implicit group sizes. With low skew almost all groups are empty and the higher RTH reflects the fact that the cast messages must nevertheless traverse part of the broker backbone. When skew is 1.0, many peers register some of the same tags. The frequent casts containing these tags are thus propagated across much of the backbone. However, only peers that have registered the whole combination of tags required by a target expression are selected—the backbone does not filter these—resulting in unnecessary messages and a higher RTH. As the sizes of implicit groups increase when skew is 2.0, the backbone efficiently reaches all

members with almost no overhead.

6.5 Summary

This chapter has presented and evaluated the BROKER IGM implementation (Contributions 7 and 8). The design clearly helps to reduce outgoing load compared to the CENTRALISED implementation. By decentralising the responsibility for forwarding casts to consumers over many peers, the maximum outgoing load on any single peer is reduced to a fraction. The implementation does not, however, address total incoming load; many brokers are required to process a majority of casts. This is due primarily to the acyclic backbone design. An overlay that permits cycles may be more able to distribute incoming load.

BROKER is not fair. Though fairer than the CENTRALISED implementation, the bipartite system of brokers and clients continues to divide the peers into those that perform work and those that do not. This distinction must be eliminated in order to produce true fairness, which is the focus of this thesis and the motivation of the SPICE design in the following chapter.

Chapter 7

Basic SPICE

This chapter introduces the P2P SPICE implementation, and its foundation, the ICE substrate (Contribution 9). The implementation extends the theme of decentralisation seen in the BROKER implementation in the form of a distributed structured peer-to-peer (P2P) overlay. This chapter presents the basic SPICE design, and Chapter 8 extends it with techniques designed to fairly distribute load over all peers and network links irrespective of the frequency of casts or sizes of implicit groups.

7.1 Design

The BROKER implementation aims to decentralise the points in the network that are relied upon for casting. The SPICE implementation[1] takes this to a logical extreme by delivering casts over an homogeneous network of peers that cooperatively serve consumers. A novel, structured P2P substrate called ICE (Section 7.2) provides a "surface" upon which the SPICE registration and casting algorithms are founded. Many "rendezvous points" on the surface are used to store tag registrations and marshal casts for delivery to consumers. Advanced distribution and replication techniques, described in Chapter 8, ensure the participating peers perform only limited work and store limited state information, independent of the number of peers, popularity of tags, or frequency of casts.

Many structured P2P designs exist; their common feature is an address space that permits deterministic routing, within which peers are arranged. DHT overlays conceptually operate like classical hash tables, storing and retrieving objects over the network via fixed length keys. DOLRs support routing of arbitrary messages to objects or nodes in the substrate. See Section 2.4.2 for further background regarding structured P2P networks.

A simple DHT-based IGM system is easily implemented in two parts. To register, each peer inserts a reference to itself at the hashed address of each of the tags in its reg-

[1] Named for the variety of groups addressable via IGM.

Table 7.1: Extended object structure for DHT/DOLR algorithms.

(a) A target expression object.

Target	or_terms	*selects?(registration)*

Listing 3 Naïve DHT-based IGM.

```
1  def reg(p)
2     # Store the peer's identity in the DHT for each tag.
3     p.registration.tags.each { |tag| dht[tag].store(p) }
4  end
5
6  def cast(p,c)
7     consumers = Set.new
8     # Find the union of the set of peers for each disjunctive term.
9     c.target.or_terms.each do |term|
10       # Lookup all peers with the last tag in the term.
11       peers_and = Set.new(dht[term.pop])
12       # Intersect with those that have every other tag in the term.
13       peers_and = term.inject(peers_and) { |sel,tag| sel & dht[tag] }
14       # Store the result with those from other disjunctive terms.
15       consumers |= peers_and
16    end
17    consumers.each { |q| p.notify(q,c) } # Notify each selected consumer.
18 end
```

istration, termed *rendezvous points* (RPs). To cast, a publisher retrieves across the DHT all peer references stored for each of the tags in the target expression. It then combines and intersects them according to the expression before notifying the resultant set of the message. Such an approach has much in common with distributed search indices for document collections which employ "vertical" or "keyword partitioning" [141, 142].

Listing 3 shows the simplicity of a pure DHT solution. The same basic object definitions used for the CENTRALISED and BROKER algorithms (Table 5.1) are used for these DHT and DOLR approaches. The `Target` object is extended with an `or_terms` field (Table 7.1), which is a collection of the disjunctive terms of the target expression.

A problem with this naïve *iterative* approach is that considerable intermediate data must be returned to the publisher before it can determine the selected consumers. A *recursive* alternative is to route the cast to the RPs in a chained sequence rather than returning results immediately to the publisher, as applied in the distributed search indices used in Panaché [143] and RDFPeers [144], for example. At each point, the intersection of results can be calculated and forwarded to the next RP, reducing the amount of data that needs to eventually return to the publisher.

However, the network traffic cost for lookups is non-negligible, whether iterative or recursive. The number of lookups can be reduced by each peer storing a complete

Listing 4 Improved DHT-based IGM.

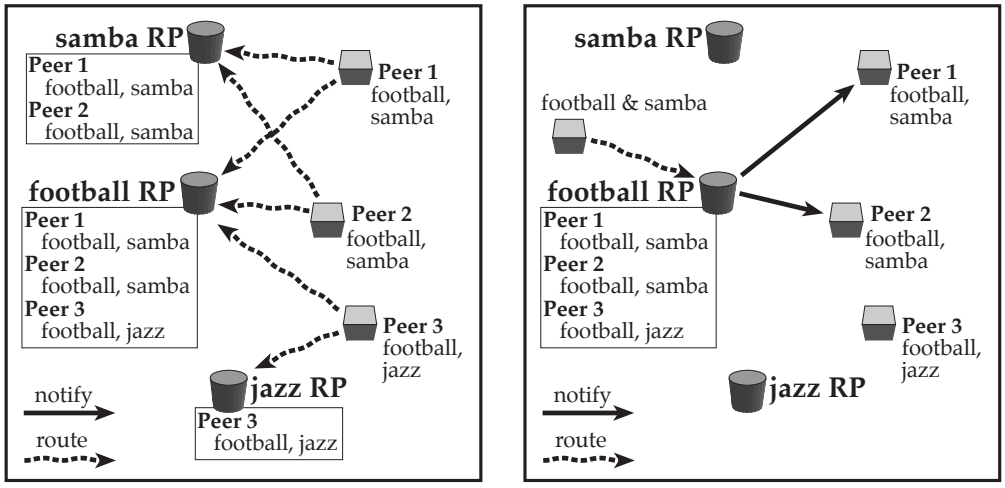
```
1  def reg(p)
2     # Store the peer registration in the DHT for each tag.
3     p.registration.tags.each { |tag| dht[tag].store(p.registration) }
4  end
5
6  def cast(p,c)
7     registrations = Set.new
8     # Find the union of registrations for each disjunctive term.
9     c.target.or_terms.each do |term|
10       # Any single tag in the conjunctive term can be looked up.
11       reg_and = Set.new(dht[term.first])
12       # Reject registrations that are not selected.
13       reg_and.reject! { |r| not term.selects?(r) }
14       registrations |= reg_and
15    end
16    registrations.each { |r| p.notify(r.peer,c) }
17 end
```

list of its tags at the RP for each of its tags. A lookup then returns all tags for a peer and allows the set of selected consumers to be calculated without having to lookup every tag individually. A somewhat similar strategy is employed in keyword fusion [145] and Keyword-Set Search System [146].

The RP for each tag must be informed when a participant's registration is altered, which may be a reasonably expensive operation in terms of network traffic. However, IGM registrations represent interests or inherent attributes of a participant which presumably will not change as quickly as casts are published. Since casts may occur frequently and to potentially very large groups, their efficiency is a priority. Thus SPICE is designed to optimise for casts, and adopts the technique of storing all of a peer's tags at the RP for each. Listing 4 presents this improved DHT approach.

Even with this improvement, results are still returned to the publisher before consumers are notified of a message. Since all the information necessary to find selected peers is stored at RPs, a simple optimisation is for the peer responsible for the RPs to forward casts directly to consumers instead, reducing the total amount of data transferred around the network. Note that as the selection logic is not centralised at the publisher, it is possible for duplicates to be delivered to consumers selected by multiple disjunctive terms of a cast. This issue is discussed and resolved in Section 7.3.2. Because other peers in the network must now participate in the actual delivery of casts, such an approach requires a DOLR (where peers execute specific algorithms upon receipt of messages) as opposed to a pure lookup-based DHT.

This is the fundamental SPICE design, illustrated in Figures 7.1(a) and 7.1(b). Listing 5 presents the pseudocode for the basic SPICE design. A new `route` method is added that routes a message over the DOLR to a hashed address. Method names pre-



(a) Registering. (b) Casting.

Figure 7.1: Schematic of basic SPICE on a DOLR.

fixed with `handle_X` are invoked when a message of type `X` is routed to any peer in the DOLR (recall that peers are homogenous and thus any is capable of handling a message). Two more object definitions are extended with additional fields (Table 7.2); these are explained in the listing as necessary, and detailed in Section 7.3.

Although conceptually straightforward, this approach does not work well when implicit groups are very large or frequently selected, as expected by the characterisation of IGM (Section 3.4). This is clearly demonstrated in Section 7.5 which submits the basic SPICE implementation to the IGM analysis framework. These problems are addressed in Chapter 8 which expounds SPICE's load distribution features. However, the following section first describes in detail the DOLR used by SPICE.

7.2 ICE: a tesseral P2P substrate

The basic SPICE algorithms require a structured DOLR substrate upon which to operate. This section introduces a novel substrate called ICE based around tesseral addressing and an efficient amortised multicast routing algorithm. ICE is a useful fundament for building P2P systems such as SPICE and has just two functions: to organise the peers into a structured overlay over the physical network; and to provide an efficient routing algorithm for delivering messages from source peers to multiple destination peers. Figure 7.2 illustrates the layered approach. IGM applications such as chat rooms and Special Interest Group blogs are built on SPICE, which uses the primitives of the ICE substrate.

The previous section briefly alluded to the ICE design. Peers are organised on the d-dimensional surface of a d-torus. The entire surface is claimed by peers; there are

Table 7.2: Extended object structure for SPICE algorithms.

(a) A registration object.

Registration	peer	tags	tag

(b) A cast object.

Cast	target	term	tag	payload

Listing 5 Algorithms for basic SPICE on a DOLR.

```
1  # Peer p registers.
2  def reg(p)
3     # Route the registration to the RP for each tag.
4     p.registration.tags.each do |tag|
5        # RP peers may store several registries (due to hash collisions).
6        # The tag field identifies the registry for this registration.
7        p.registration.tag = tag
8        p.route(tag.rp,p.registration)
9     end
10 end
11
12 # Peer p casts c.
13 def cast(p,c)
14    c.target.or_terms.each do |term|
15       # The term field indicates which disjunctive term the RP is
16       # being asked to resolve. This is used to prevent duplicate
17       # notifications (Section 7.3.2).
18       c.term = term
19       # Casts are handled by a single registry. Since some RPs hold
20       # several registries, the tag field identifies the one to use.
21       c.tag = term.first
22       p.route(c.tag.rp,c)
23    end
24 end
25
26 # Peer p handles registration r.
27 def handle_reg(p,r)
28    # Add it to the appropriate registry.
29    p.registry[r.tag].push(r)
30 end
31
32 # Peer p handles cast c.
33 def handle_cast(p,c)
34    registrations = p.registry[c.tag].find_all do |r|
35       c.target.selects?(r)
36    end
37    # minimum_term? prevents duplicate notifications (Section 7.3.2).
38    registrations.each { |r| p.notify(r.peer,c) if minimum_term?(r,c) }
39 end
```

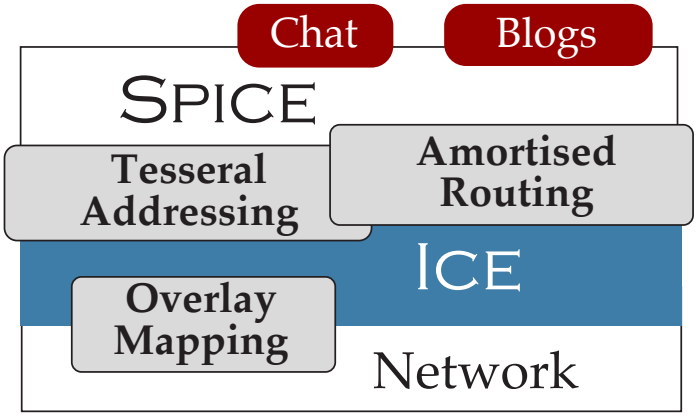


Figure 7.2: The layered approach to building IGM applications with SPICE.

no "holes". Each peer "owns" an exclusive region of the surface and communicates directly only with peers that own bordering regions (maintained in a *neighbourhood* table). Messages are routed across the surface by passing them from neighbour to neighbour in the direction of the destination. Figure 7.3 shows the 2-dimensional surface of a 2-torus. The projected surface is considered to adjoin continuously on all edges.

Peers become part of the surface by routing a join request to a chosen region on the surface from an arbitrary bootstrap peer. The peer that owns the chosen region handles the join request by halving its region, returning a join acknowledgement to the new peer, and informing its neighbours.

The only state information stored by an ICE peer is its neighbour table. The number of neighbours a peer has is dependent on the dimensionality of the surface. Because the surface is randomly partitioned between peers, each peer has approximately one bordering neighbour for each face of its region, or two neighbours for each axis of surface dimensionality. Thus, the storage complexity of ICE is related only to the dimensionality of the surface, i.e., $O(d)$. The number of peers does not affect the state stored by any individual peer.

The ICE surface is superficially similar to CAN [19]. In fact the basic SPICE algorithms outlined in Section 7.1 could be constructed on CAN without much difficulty. However, there is a crucial difference between CAN and ICE. Unlike CAN which uses a Cartesian coordinate space, ICE surfaces use hierarchical tesseral addressing. The distinction is motivated by the intended usage: CAN is designed to support point-to-point routing whereas ICE is inherently a multipoint routing substrate, especially designed to enable the load distribution features of SPICE introduced in Chapter 8.

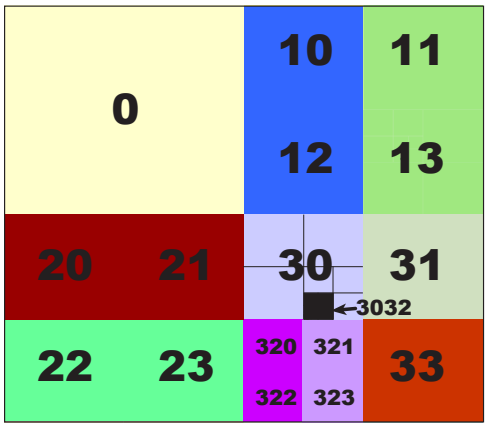


Figure 7.3: A projection of the surface of a 2-torus. Colours show ownership by different peers.

7.2.1 Tesseral addressing

Hierarchical tesseral addressing is a compact and elegant addressing scheme that describes regions of space instead of Cartesian points. It is particularly useful for arbitrarily decomposing a space to different granularities and is used to decompose the ICE surface. Further background information regarding tesseral addressing is provided in Section 2.6.2.

ICE regions are addressed by strings of d-bit digits (of base 2^d). These regions are termed *extents*, and each digit represents a progressive index into the hierarchically addressed surface. Extents may be large and coarse-grained or extremely small and fine-grained. The *depth* of an extent is the number of digits in its address. Deeper extents are necessarily smaller than shallow extents. The solitary extent of zero length is the *universal extent* (denoted $\mathcal{U}$) and represents the entire surface.

Figure 7.3 illustrates the addressing scheme, ordered left-to-right and top-to-bottom. For example, the address `0` specifies the top-left quadrant of a 2-dimensional surface, and address `3032` specifies a small extent towards the bottom-right corner. This mapping approach can be applied to surfaces of arbitrary dimensionality without loss of generality.

A set of extents is called a *tract*. Tracts can represent arbitrary regions of the surface by composing several extents, both contiguous and disparate. For example, the tract $\{0,2\}$ is the entire left half of a 2-dimensional surface. Every instance of a tract has a definite representation (i.e., a specific set of extents), although there are infinitely many such sets which can define the same tract. Tracts can be subtracted from one another but unlike ordinary set difference, the result is a new tract that describes the remaining region of the surface. For example, on a 2-dimensional surface, $\{0,2\} - \{20\} = \{0,21,22,23\}$.

Hierarchical tesseral addressing has some important features that make it suitable

for describing an overlay surface, and in particular for supporting the load distribution features of SPICE. Its hierarchical nature allows a deep address to naturally *scale* to increasingly large covering extents simply by omitting digits from its tail. Similarly, its self-similar properties means an address can easily be *translated* across the surface by replacing its head with digits from another extent.

It is important to note that although this addressing scheme is hierarchical, there is no need for a global data structure to be maintained across peers. In particular there is no "root" peer. It simply affords compact descriptions of large and small regions of the surface, and is a convenient way for peers to store surface information and communicate with one another. For instance, the region of the surface that each peer owns is represented as a tract. Likewise, peers' neighbourhood tables map extents adjacent to their tract to the peers that own them. The versatile hierarchical and self-similar properties of tesseral addressing are an integral part of the load distribution mechanisms of SPICE described in Chapter 8, and of the multicast routing algorithm now described in Section 7.2.2.

7.2.2 Amortised routing

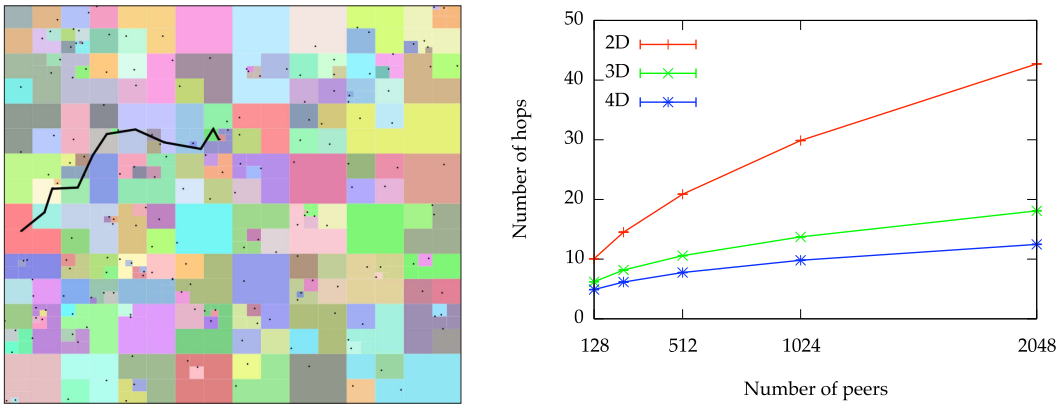
Besides tesseral addressing, the other major component of ICE is an efficient amortised multicast routing algorithm. Payloads are routed from a source peer to a *destination tract* by passing messages from neighbour to neighbour. All peers that own tracts intersecting the destination tract receive a copy of the message.

Point-to-point routing

ICE can emulate point-to-point routing by delivering to a very small destination tract that is covered by the tract of a single peer. Messages are routed geometrically across the surface between neighbouring peers. At each hop, peers forward the message to a neighbour that owns an extent nearer to the destination. Figure 7.4(a) shows an example of a message routed across a 2-dimensional surface.

For the purposes of finding a neighbour closer to the destination, a distance metric between extents is required. This is achieved by mapping extents from their tesseral addresses to bounding hypercubes in a Cartesian space. To do this, the surface is assumed to be wholly contained within a unit hypercube, within which extents are converted to sub-hypercubes. The distance between two extents is defined as the length of the minimum interval connecting their bounding hypercubes. For example, the distance between extents `3032` and `23` on a 2-dimensional surface is 0.125 (see Figure 7.3).

The message complexity of point-to-point routing depends on the dimensionality of the surface and the number of peers. An equally partitioned surface has $n^{\frac{1}{d}}$ distinct



(a) Routing across a 2D surface.

(b) Point-to-point routing is $O(dn^{\frac{1}{d}})$ overlay hops.

Figure 7.4: ICE point-to-point routing.

peers arranged along each edge of the surface. The longest route a message can take is from a corner of the surface to its centre (due to the surface wrapping on all sides). The surface is "quantised" by the peers so the message takes a Manhattan ("city block") path. Point-to-point routing over the ICE surface is thus $O(dn^{\frac{1}{d}})$ overlay hops. Figure 7.4(b) confirms this in simulation by routing messages between two random extents. As expected, higher dimensionality requires fewer hops to route between two parts of the surface.

In practice, actual routing performance can be improved by considering the physical link latency between neighbours when selecting the next hop, in addition to the distance from the destination. The "best" hop is chosen to be the one with the greatest distance progress to link latency ratio which tends to favour low latency links, when all else is equal. Listing 6 shows this fragment of the ICE routing algorithm, and Table 7.3 shows the object structure. The technique is one of the routing optimisations used in CAN [19], but other routing possibilities exist. For example, peers could attempt to minimise total load on neighbours over time, avoid known malicious peers, or monitor the reliability of neighbours in order to bias route selection. However, these options are not explored in this thesis; all peers optimise for the maximum distance to latency ratio.

Multipoint routing

The ICE routing algorithm is not restricted to point-to-point routing; it delivers copies of a message to all peers with tracts intersecting a destination tract. The destination tract may be a small contiguous region of the surface but this is not a requirement. A tract can specify *any* region of the surface whether contiguous or not. There are times when it is useful for a message to be delivered to disparate parts of a surface. For instance, a resilient storage layer built atop ICE could use the routing algorithm to

Table 7.3: Extended object structure for ICE algorithms.

(a) An ICE peer object.

ICE Peer	registration	tract	neighbourhood	*handled?(route)*

(b) A route object.

Route	destination_tract	branch_factor	payload

Listing 6 The "next hop" fragment of the ICE routing algorithm.

```
# Returns a reference to the neighbour that offers the most progress
# towards a tract considering both distance and latency.
def next_hop(p,tract)
   current_dist = p.tract.distance_to(tract)
   # Find the distance from each neighbour to the tract.
   hops = p.neighbourhood.distances_to(tract)
   # Calculate the progress ratios for each neighbour.
   progress = hops.map do |neighbour,next_dist|
      {
         'neighbour' => neighbour,
         'progress' => (current_dist-next_dist) /
            p.neighbourhood.latency_to(neighbour)
      }
   end
   # Return the neighbour that makes the maximum progress.
   max_progress = progress.max { |x,y| x['progress'] <=> y['progress'] }
   return max_progress['neighbour']
end
```

copy data to multiple regions of a surface.

The algorithm is given a destination tract and a payload. A peer routes the message by forwarding it to the neighbouring extent that is closest to any part of the destination tract. At each hop, the peer handling the route delivers the payload to the application layer if its tract intersects the destination tract. Its tract is then subtracted from the destination, reducing the total region left to visit. The remaining tract is then recursively visited in the same way.

When a tract is not contiguous, it is quicker to route separate copies of a message to each region instead of visiting each in turn. However, routing separate copies of a message from the source is inefficient if part of the route is the same for each copy. Because ICE is a structured overlay and the surface is a geometric construct, the routing algorithm clusters parts of the destination tract based on their direction away from the source in order to amortise the routing cost. If several lie in the same direction away from a source, it is sufficient for a single copy to be routed that way.

This technique is recursively applied. Not only is the destination tract clustered from the initial source, it is clustered again at each hop. This results in messages taking

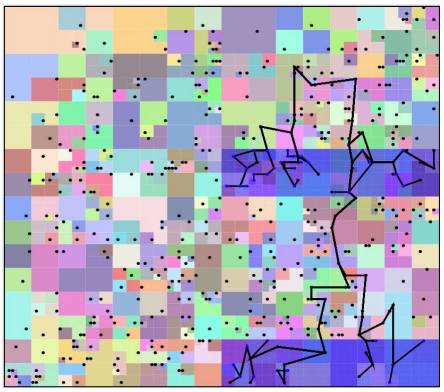


Figure 7.5: A message routed from a source at top-right to the shaded destination tract on a 2D surface.

a tree-like route from the source to all peers with tracts that intersect the destination tract. Initially the route has few branches but as the message approaches individual peers it branches as necessary to balance the total number of messages against direct individual routes. Figure 7.5 shows an example of amortised routing delivering a message to a highlighted destination tract (comprising two distinct regions).

A divisive hierarchical clustering algorithm (discussed in Section 2.6.3) is applied to the remaining destination tract at each hop resulting in a set of clustered sub-tracts. Each of these clusters is then individually routed a copy of the message. Thus, the message branches when there is more than one cluster but continues in a point-to-point fashion otherwise.

Recall that a tract is simply expressed as a set of extents; the clustering algorithm groups these extents according to the angular difference between their centres as measured from the point of view of the current peer's tract. Those with an angular difference larger than a threshold are placed in separate clusters. This threshold is called the *branch factor*, expressed in radians and taking a value in the range 0–π. Due to parallax, distant extents are more likely to be clustered together than nearby extents.

A branch factor of 0 means messages are never amortised; every extent in a destination tract is considered its own cluster and individually approached using point-to-point routing. A value of π means the entire destination tract is always treated as one cluster such that the message never branches but visits each extent in turn. The advantage of a high branch factor is that a minimal total number of messages are needed, which in turn reduces the incoming and outgoing load on peers and network links. The drawback is that messages take longer to reach their destinations on average, because more circuitous routes are taken.

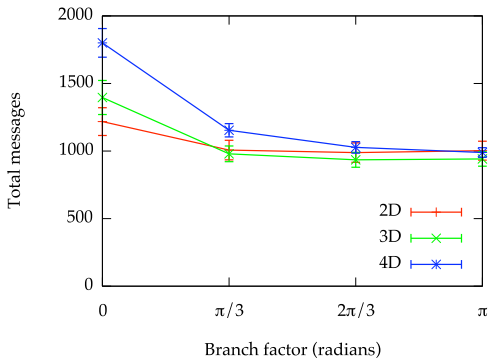
Listing 7 presents the complete ICE routing algorithm, building upon the "next hop" fragment implemented in Listing 6. Since ICE is designed as a general basis for P2P applications, it provides callback hooks that an application can use to

Listing 7 The ICE routing algorithm.

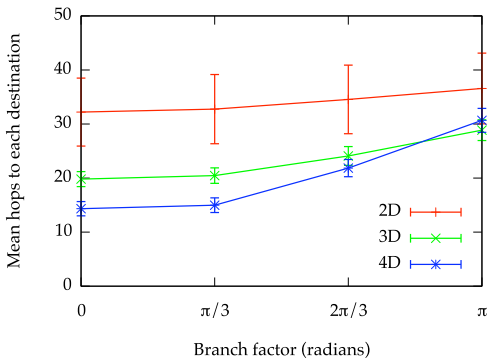
```
# Peer p routes 'route'.
def handle_route(p,route)
   # Deliver the payload to this peer if it's in the destination tract.
   if route.destination_tract.intersects?(p.tract) and
         not p.handled?(route)
      # A callback allows an application to alter what payload is
      # delivered to the peer, and what continues to be routed to the
      # remainder of the destination tract.
      route.payload = callback_deliver(p,route) do |deliver_payload|
         p.deliver(p,deliver_payload)
      end
   end
   # Subtract this peer's tract from the destination tract.
   remaining_tract = route.destination_tract - p.tract
   unless remaining_tract.empty?
      # Cluster the remaining destination tract.
      clusters = p.tract.cluster(remaining_tract,route.branch_factor)
      # A callback allows an application to alter what payload is
      # routed to each cluster.
      callback_branch(p,route,clusters) do |cluster_payload,cluster|
         route_dup = route.dup
         route_dup.destination_tract = cluster
         route_dup.payload = cluster_payload
         hop = next_hop(p,cluster)
         p.send(hop,route_dup)
      end
   end
   p.handled!(route)
end

# By default, don't alter the payload.
def callback_deliver(p,route)
   yield route.payload  # Yield to ICE to deliver the payload.
   return route.payload # Return the payload unaltered.
end

# By default, deliver the same payload to each cluster.
def callback_branch(p,route,clusters)
   clusters.each { |cluster| yield route.payload,cluster }
end
```



(a) Increased branch factor reduces the total number of messages needed.



(b) Increased branch factor increases the number of hops to individual peers.

Figure 7.6: Multipoint ICE routing.

modify the default multipoint routing behaviour. In particular, two callbacks may be implemented by an application: `callback_deliver` (Line 9 of Listing 7) and `callback_branch` (Line 20). These are called prior to the delivery of a payload to an application, and prior to forwarding a payload to a cluster of the destination tract, respectively. They may be used to alter the payloads that are delivered, or veto them altogether. SPICE uses these callbacks to implement registry distribution in Section 8.1.1 of Chapter 8, but the default implementations (Lines 32 and 38) behave as described above, delivering the same payload to every peer in a destination tract.

The effect of surface dimensionality and branch factor on multipoint routing are investigated via simulation. 1000 messages are routed from random sources to random destination tracts intersecting ≈ 500 peers of 1000 on an ICE surface. The total number of messages needed for each delivery and the average number of hops to each destination peer per message are measured.

Figure 7.6(a) shows that the branch factor has a very significant effect on the total number of messages. By increasing amortisation of packets (with a higher branch factor), the number drops to approximately 1000 irrespective of surface dimensionality, just twice the minimum number of messages needed to directly contact each destination peer from the source. Increasing the surface dimensionality generally increases the total number of messages because there are intuitively more directions in higher dimensions and destinations are less likely to be clustered. However, by increasing the branch factor beyond approximately $\frac{\pi}{3}$ the total number of messages can be reduced even in these higher dimensions because the algorithm is more permissive in what constitutes a "similar" direction.

Higher dimensionality also dramatically reduces the average number of hops to destinations (Figure 7.6(b)) because the number of hops between two parts of the sur-

face is reduced (as shown previously is Figure 7.4(b)). A lower branch factor also delivers messages in fewer hops since they do not follow longer generalised routes to many destinations. A branch factor of less than $\frac{\pi}{3}$ tends to produce the most direct routes to destinations.

Based on these results, a surface dimensionality of 3 and branch factor of $\frac{\pi}{3}$ offer the best compromise between low total messages and average hops to each destination. These values are therefore used in the subsequent SPICE load distribution experiments.

7.2.3 Overlay mapping

It is beneficial for the ICE surface consider the physical underlying network in order to reduce routing latencies and network loading. Any of the approaches discussed in Section 2.4.2 are applicable, but for the purposes of the load distribution simulations in this dissertation, it is sufficient to implement overlay mapping in a centralised fashion. Thus, ICE uses a centralised landmark binning scheme. The guiding principle is to attempt to place physically close peers near one another on the overlay, yet ensure the surface is uniformly partitioned among peers.

l well-known landmarks are randomly placed around the physical network and each is associated with an equally sized extent of the ICE surface. Each peer joining the system measures the round trip time (RTT) to each landmark. It then attempts to join at a random location within the extent associated with the closest landmark. If a landmark is the closest choice for more than $\frac{n}{l}$ peers, the next closest landmark is chosen by a newly joining peer. This ensures that each part of the overlay has a similar number of peers, and thus that the ICE surface is uniformly partitioned.

The mapping of landmarks to extents on the surface is achieved by finding the minimal spanning tree of the landmarks (based on RTTs between them). The order of landmarks resulting from a depth-first traversal of this tree is then mapped to a Morton linearisation [147] of the ICE surface. This strategy tends to associate physically nearby landmarks with extents that are nearby on the surface, and since peers within each extent are physically nearby, the overlay as a whole is better able to map to the physical network (see below).

A Morton linearisation permits only specific numbers of landmarks to be used, such that Equation 7.1 holds. For example, a 2-dimensional surface can be equally divided into four extents for $k = 1$ and 16 for $k = 2$. Similarly, a 3-dimensional surface can be divided into eight for $k = 1$ and 64 for $k = 2$.

$$l = 2^{kd}, k > 0 \tag{7.1}$$

Figure 7.7 shows the results of a simulation measuring the average number of physical packets needed to route 1000 messages between two random extents on a

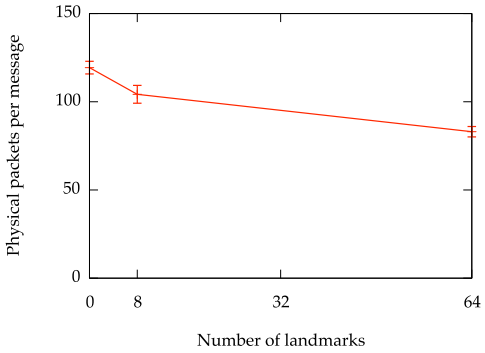


Figure 7.7: Additional landmarks increase overlay/network correspondence and reduce the physical cost of routing across the surface.

2000 peer, 3D ICE surface, against an increasing number of landmarks. When no landmarks are used, overlay routing results in many physical network packets. The overhead diminishes with the number of landmarks, due to an increased correspondence between the overlay and physical network topologies. 64 landmarks is the largest reasonable number that can be employed in networks of this size and dimensionality, because of Equation 7.1, although the trend indicates additional landmarks may improve congruence further. The SPICE simulations in subsequent sections use 64 landmarks.

7.2.4 Self-stabilisation

The unreliable nature of peers in a P2P system guarantees that some peers will unexpectedly fail or leave the system over time. When designing a structured overlay such as the ICE surface, it is important to consider how it will cope with structural damage when it is encountered and how it will repair itself despite peer flux. ICE does not inherently support data storage—it is implemented by higher layers if needed—so the only impact of a failed peer is routing over the surface. To ensure messages are not interrupted by *holes* in the ICE surface (unclaimed tracts), the routing algorithms must be capable of routing around failed peers. Additionally, peers should detect and repair holes. This section sketches how the ICE surface can be made "self-stabilising".

Self-stabilisation [148] is the process of converging a system to a stable state from an arbitrary starting state. This can be expressed as a set of invariants and rules that move the system state towards satisfying these invariants. In ICE, a stable surface is defined by three invariants. Firstly, the entire surface must be claimed by peers. Secondly, the tracts claimed by peers must not intersect. Thirdly, every extent adjacent to a peer's tract must be described by exactly one mapping in its neighbourhood table.

ICE is suited to the *correction-on-change* and *correction-on-use* approaches to self-stabilisation [149]. In this technique, problems are corrected as they are detected dur-

ing normal routing operations, rather than through the use of periodic beacons. This greatly limits the amount of maintenance traffic required and is especially applicable to ICE because the failure of a peer affects only $O(d)$ neighbours.

Correction-on-use dictates that each message routed across the surface must include maintenance information about intermediate source and destination peers. This allows peers to verify that their neighbourhood state agrees with the sender's. Deterministic agreements can ensure that any conflicts arising (such as two peers claiming intersecting extents) are resolved. Furthermore, the very act of receiving a route from a peer indicates the sender's existence and allows the recipient to ensure neighbourhood information is correct.

When routing messages, peers will occasionally detect failed peers. In these cases the message must be alternately routed. A message may be greedily routed around the periphery of a hole on an ICE surface by probing the "next best" neighbour, or more general techniques from sensor networking or geocasting may be applied, such as the FACE protocols [150]. In addition to routing around holes, the correction-on-change protocol repairs the hole by deterministically assigning it to a new peer. Detections of the hole by different peers routing messages result in a deterministic marshalling peer being informed. This peer can then claim the hole itself, or delegate it to a neighbour. Finally, update information can be delivered to neighbours so they can update their neighbourhood tables. If a marshalling peer has also failed, the technique may be recursively applied.

The focus of this thesis is load distribution in implicit group messaging systems. The practical matter of self-stabilisation is therefore not investigated in the evaluation of ICE, although it is clear from this sketch that implementing self-stabilising algorithms is not problematic. Note that a stabilised ICE surface does not negate the need for applications using ICE to employ their own data integrity algorithms. ICE's only guarantee is to route messages to all peers within a destination tract. Any application-specific stabilisation must build on this guarantee. The SPICE design tacitly assumes the ICE surface is self-stabilising and that routing is reliable, but Section 7.3.3 discusses how registration information is maintained when peers fail.

7.2.5 Summary

This section has described the ICE surface, a novel P2P structured overlay design based on hierarchical tesseral addressing and an efficient amortised multicast routing algorithm. ICE is a very lightweight basis for building more complex P2P systems. Its only functionality is to maintain a logical surface of peers, and allow routing of messages to all peers within arbitrary tracts of this surface. Though minimal, this functionality permits construction of complex systems such as the SPICE implementation of implicit group messaging, as seen in the following section.

7.3 SPICE on ICE

This section describes how the basic SPICE algorithms are constructed on top of ICE. Consumers register at the rendezvous points (RPs) for each tag on the ICE surface and publishers route casts to these RPs which notify all selected consumers. The fundamental approach is motivated in Section 7.1, and demonstrated in Figures 7.1(a) and 7.1(b). Listing 5 presents the algorithms. The following sections detail subtle design issues related to implementing SPICE on ICE. Note that for the remainder of the dissertation, RP can refer both to a rendezvous point on the surface and the actual peer storing registries.

7.3.1 Registration

A peer routes its registration over the ICE surface to the RPs for each of its tags where it is stored in registries. DHTs typically use consistent hashing to map objects to identifiers in the DHT address space [18, 24, 25, 82, 151]. The number of bits in the address space is often fixed at 128 (suitable for the MD5 hash function) or 160 (suitable for SHA-1).

Due to ICE's tesseral addressing, the rendezvous points for tags are in fact deep extents and are interchangeably referred to as *rendezvous extents*. The address of a rendezvous extent is found by hashing a tag, then taking d bits at a time for each digit.

However, each registry in SPICE should be "owned" by exactly one peer, so the depth of the rendezvous extent must be sufficient to ensure that only one peer's tract intersects it. The size of a peer's tract is dependent upon the number of peers, n, the dimensionality of the surface, d, and the way the surface is partitioned. If the surface is partitioned equally, the number of digits needed for a rendezvous extent to map to a single peer is:

$$digits = \left\lceil \frac{\log_2 n}{d} \right\rceil \tag{7.2}$$

For example, a 2-dimensional surface with four equally partitioned peers requires just one digit for a rendezvous extent to map a tag to a unique peer. Similarly, a 3-dimensional surface with 512 peers requires three digits. Since each digit is d bits, $O(\log_2 n)$ bits are required for rendezvous extent addresses. Thus only 20 bits are required to support a million peers. Since the surface partitioning will not be completely equal in practice, a safe margin of extra digits is required to ensure that a rendezvous extent maps to a single peer. However, a hash function such as SHA-1 is capable of easily supporting any realistic number of peers. Note that as addresses need to be expressed in terms of digits of a particular surface dimensionality, the tail of the hash is dropped as necessary to ensure the number of bits is divisible by d.

7.3.2 Casting

As outlined in Section 7.1, casting to implicit groups requires publishers to route messages to rendezvous points on the surface which then notify all selected consumers stored in their registries.

Listing 5 and Table 7.2 present the cast algorithms and supporting objects for the basic SPICE implementation. The cast's *tag* field contains the particular registry for which the cast is destined. Because an RP may be responsible for many tags, this allows the RP to determine for which registry it should be processing the cast. The *term* field is similar, containing the disjunctive term for which the message should be processed. This is needed to avoid notifying consumers repeatedly of casts when they are selected by more than one disjunctive term (explained below).

In the case of a target expression of just a single tag, the publisher routes the cast to the rendezvous extent of that tag. Upon receipt, the RP iterates through all registrations stored for the tag and notifies consumers.

A target expression containing conjunctive terms (e.g., `football&jazz`) is handled very similarly. The publisher selects any one of the tags and forwards the cast to the RP for only that tag. Upon receipt, the RP searches the registrations for that tag, but only notifies those peers that have registered all tags in the target expression. This is possible because the registration contains the entire registration of the peers. Note that the single tag case is actually a degenerate version of this case.

Casting to an implicit group using disjunctive terms is slightly more complex. Consider the target expression `samba|jazz`. This cast should be delivered to all consumers that have registered `samba`, or `jazz`, or both. Therefore, using a single RP will not suffice, because it will not necessarily have enough information to find all of the selected consumers. For instance, if the message were routed to the RP for `samba`, all consumers registering this tag would be found, as would all registering both `samba` and `jazz`. However, consumers that only registered `jazz` would not receive the cast.

To remedy this, the cast must be routed to one RP for every disjunctive term, each of which then notifies the consumers for each term in parallel. This introduces a subtle problem. Peers that have registered only `samba` or only `jazz` will receive the cast as expected, but peers that have registered both tags will receive a copy of the cast from each RP. Although duplicates could simply be discarded by consumers, it is undesirable to place extra load on RPs, consumers and the physical network.

The problem stems from the fact that no RP knows if another is also notifying a common consumer. A costly solution would be for the RPs to communicate in order to omit duplicates. However, this is unnecessary since the symmetry can be broken using a deterministic function without requiring any communication between RPs.

The RP can decide if it should forward the message to consumers matching more than one disjunctive term in the target expression by imposing a total ordering on the

Listing 8 Tests if a consumer should be notified of a cast based on the total ordering of terms, and whether the RP handling the cast is responsible for the minimum term.

```
1 def minimum_term?(reg,cast)
2    selected_by = cast.target.or_terms.find_all do |term|
3       term.selects? reg
4    end
5    # Total ordering of terms is lexicographic.
6    cast.term == selected_by.min { |x,y| x.sort.join <=> y.sort.join }
7 end
```

terms. If it is handling the "minimum" term that selects a consumer, it should handle the notification. Otherwise, it knows that another RP will do so. The disjunctive term can be ordered by sorting hashes of the terms, or by lexicographic sorting of their canonical forms, for example. Listing 8 shows one possible implementation that lexicographically sorts and concatenates each disjunctive term, then sorts the results.

In the example, a peer that has registered both `samba` and `jazz` tags will be stored in the registries for both of these tags. By only testing the summary against the target expression `samba|jazz`, both RPs will determine that this consumer should receive a copy of the cast. To determine which should notify it, each RP finds the minimum of the two disjunctive terms, e.g., `jazz` (because `jazz` $<$ `samba` lexicographically). The RP for `jazz` will therefore exclusively notify the consumer.

7.3.3 Self-stabilisation

Peers may join and leave SPICE at any time. When a new peer joins it claims part of the ICE surface which belongs to an existing peer. Any registries the existing peer is storing may need to be passed to the new peer if its part of the surface covers the rendezvous extent. Similarly, when a peer leaves, it can allocate its tract to a neighbour and transfer any registries it is storing. Sometimes, however, a peer fails or leaves without announcing its departure.

A self-stabilised ICE surface is capable of routing around failed peers, and repairing the surface to a stable state over time, but it does not ensure the integrity of data stored in layers on top of ICE, such as registrations stored at RPs in SPICE.

To address this issue, registrations may be periodically synchronised with one or two backup peers neighbouring the RP. When an RP fails, the peer claiming the hole in the surface can retrieve them from these backups as part of the repair process.

However, the routing algorithm provided by ICE simplifies this process. When registering, a peer can deliberately route its registration to a rendezvous extent several times larger than the tracts owned by any individual peers. Each peer that receives the registration stores a copy and is capable of taking over the role of the RP without the need for any explicit synchronisation or requests for backup copies.

As this thesis is focussed on the load distribution characteristics of IGM implementations, this technique is not evaluated as part of the simulations in the following sections. However, implementing self-stabilisation in the basic SPICE implementation is straightforward and efficient due to the amortised routing algorithm provided by the ICE substrate.

7.4 Conformance

This section shows how the basic SPICE implementation conforms to the liveness and safety conditions of the IGM specification (Chapter 3). Routing over the ICE P2P substrate is assumed to be reliable, and the storage of registrations in SPICE is assumed to be stable.

7.4.1 Liveness

The liveness condition (Theorem 3.7) states that if p registers, then from some point in the future onwards, p will eventually be notified of each cast that selects its registration.

$$\Box\,[reg(p) \Rightarrow \Diamond\Box[(cast(q,c) \wedge c_t \triangleright p_r) \Rightarrow \Diamond notify(p,c)]] \tag{7.3}$$

Every registration that is selected by a target expression has an associated registration set $R(r)$ that is a superset of at least one of the target elements, K, of the target set, $T(t)$ (by the definition of $\triangleright$). A peer's registration is routed to the RPs of each of its tags (Line 8 of Listing 5), whereupon it is added to the registry (Line 29). Therefore, all RPs of the tags of some $K \in T(t)$ must eventually contain registration r. Any of these RPs may therefore be chosen by the cast algorithm (Line 21).

The cast algorithm routes the cast to the RP of exactly one tag in each disjunctive term of a target expression (Line 14). Each RP then notifies all consumers that have a registration stored in the registry that is selected by the cast (Line 38), unless it is not handling the minimum term that selects the registration. The set of disjunctive terms in a target expression can be totally ordered (Listing 8) which means there is exactly one minimum term for every target expression. This condition must be true for one of the RPs to which the cast is routed because an RP for each term receives the cast (Line 14 of Listing 5).

Therefore the basic SPICE design satisfies the liveness condition.

7.4.2 Safety

The safety condition (Theorem 3.8) states that when a consumer is notified of a cast, it will not be notified again, the message was previously cast by some publisher, and

its registration is selected by the cast's target expression. Additionally, the condition requires that the consumer has completed a registration in order to agree with the adjusted definition of an implicit group.

As with the other IGM implementations, it suffices to show all components of the IGM safety condition separately to show it as a whole.

$\Box[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c)]$ This component says that a consumer must never be notified more than once for a particular cast. Consumers are only notified of a cast when an RP handles a cast routed to it over ICE (Line 38 of Listing 5). ICE delivers to peers at most one copy of each routed message (Line 5 of Listing 7), so casts are handled by each RP at most once. Therefore no single RP will ever notify any consumer of the same cast more than once.

Casts are routed to one RP for each disjunctive term in a target expression (Line 14 of Listing 5), so a consumer selected by more than one term could be notified by more than one RP. There exists a total lexicographic ordering of all disjunctive terms of a target expression under the tag-based modelling language (Listing 8). Because the ordering is total, there is only one minimum term in a target expression. Therefore at most one RP can notify consumers (Line 38 of Listing 5).

$\Box[notify(p, c) \Rightarrow \exists q.c \in C(q)]$ This component requires that a message must have been cast before a consumer is notified of it. This can be shown by working backwards from the consumer to the publisher. According to Listing 5, a consumer is only notified when a peer handles a cast that has been routed to it (Line 38). Furthermore, such messages are only routed to peers when a publisher invokes the cast operation (Line 22). Therefore, it follows that if a peer is notified of a cast, it must have previously been cast by a publisher.

$\Box[notify(p, c) \Rightarrow c_t \triangleright p_r]$ This component requires that only consumers with registrations selected by a cast's target expression are notified, which is expressly specified in Line 35. No peers besides RPs notify consumers of casts.

$\Box[notify(p, c) \Rightarrow p \in R]$ This component requires that a consumer has registered before it is notified of any casts. An RP only notifies peers if an associated registration is found in its registry (Line 38). The registry is only modified when the RP receives a registration (Line 29). The RP only receives a registration that has been sent by a peer (Line 8). Therefore the peer must have registered prior to notification of any cast, i.e., $p \in R$.

Because the basic SPICE design satisfies these components individually, it satisfies the safety condition generally.

7.5 Analysis

This section analyses the basic SPICE implementation without any load distribution techniques. Without these features, SPICE is in some ways conceptually similar to the CENTRALISED implementation. Casts are routed to an RP peer which, like the CENTRALISED server, notifies all selected consumers. Unlike CENTRALISED, more than a single RP may be employed for a single cast (if the target expression includes disjunctive terms), and different RPs are used for distinct casts.

The CENTRALISED implementation can be provisioned for heavy server loading. The basic SPICE implementation offers no way for RPs to be specially chosen or provisioned, which is a serious detriment. As such, this analysis is important for motivating and guiding the load distribution features of SPICE Unloaded in Chapter 8 which redress this problem.

This analysis assumes the ICE surface topology is equally partitioned between peers. This is a reasonable assumption, as supported by the discussion in Section 7.2.

7.5.1 Registration

Registration in the basic SPICE implementation involves a consumer routing a message to the RP for each of the tags in the registration. The analysis of point-to-point routing in ICE in Section 7.2 finds it to have message complexity $O(dn^{\frac{1}{d}})$ for n peers on a d-dimensional surface. Therefore, registration in basic SPICE is $O(kdn^{\frac{1}{d}})$, where k is the number of tags a peer registers. Note, however, that the ICE amortised routing algorithm reduces the cost in practice, since some RPs may be *en route* to others.

7.5.2 Peer facet

Every SPICE peer must store at least the state associated with the ICE substrate (Section 7.2). When a peer is acting as an RP for one or more tags, it also stores a number of registries containing peer registrations.

The RPs for tags are randomly chosen by mapping tags to rendezvous extents using rendezvous hashing (Section 7.3.1). As the ICE surface is uniformly divided between all peers, each acts as the RP for a similar number of tags. The number of distinct tags registered in the system determines the number of registries. When peer registration skew is 0.0, almost every one of the k tags registered by a peer is unique, so $O(kn)$ registries are required, meaning each RP has $O(k)$ registries. As skew increases, more peers register the same common tags and the number of total unique tags falls.

However, the popularity of tags greatly affects the number of registrations stored at a single RP. Table 4.7 in Section 4.3.2 shows that almost 82% of peers register the most common tag in a Zipfian distribution of skew 1.0, assuming each registers 18

tags. Moreover, a single peer may act as an RP for several tags, so the actual storage cost for a particular peer may be even higher. Generally, however, the storage load is more distributed than the CENTRALISED implementation leading to lower STOR_G.

In terms of incoming and outgoing load, the basic SPICE implementation behaves much like the CENTRALISED implementation. Because some RPs must store many registrations, a single RP or small group of RPs must notify every consumer in large implicit groups, resulting in $O(n)$ POUT_M. TIN_M is $O(c)$ and TOUT_M is $O(cn)$ in the case of a skewed cast distribution because the same RPs are involved for many casts. TIN_G and TOUT_G are slightly fairer than the CENTRALISED implementation, however, because not every cast is handled by the same RP.

7.5.3 Network facet

The basic SPICE implementation behaves much like the CENTRALISED implementation in the worst case, so its network facet can be similarly analysed. Because only one or a few RPs are responsible for notifying consumers for a single cast, PINK_M is $O(n)$. Furthermore, because a Zipfian cast skew means the same casts are published frequently, the same RPs are loaded over many casts, so TINK_M is $O(cn)$. TINK_G also shows unfairness similar to CENTRALISED because links radiating away from the loaded RPs are loaded more heavily than links near consumers.

7.5.4 Cast facet

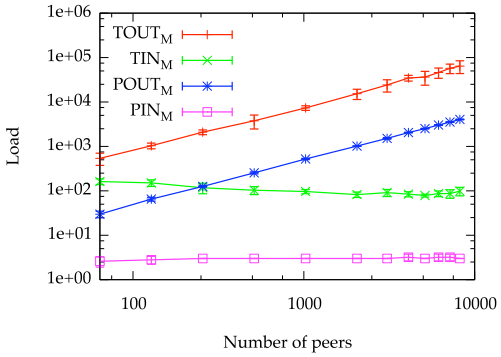
Adopting a structured P2P network approach over a client/server approach greatly affects the number of overlay hops needed to deliver a message both in total, and on average to each consumer. In a client/server approach, casts are sent to a server which then notifies all consumers. In SPICE, messages must first be routed across the ICE surface to the RPs for each of the j disjunctive terms of the target expression. The total number of hops to deliver a cast is therefore $O(jdn^{\frac{1}{d}} + n)$. Each consumer will be notified of a cast after it is routed to an RP, so RAH/RMH for SPICE is $O(dn^{\frac{1}{d}})$.

7.6 Evaluation

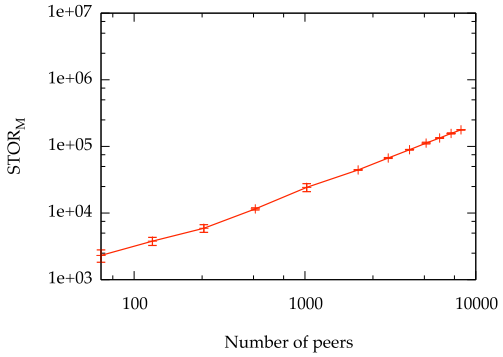
This section evaluates the basic SPICE implementation without load distribution features, to serve as a comparative baseline for their application in the following chapter. The default values used for all parameters are summarised in Table I.

7.6.1 Peer scalability

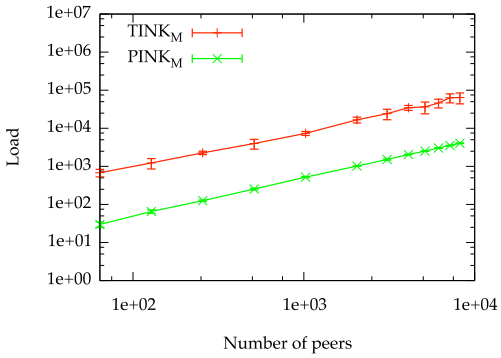
As the number of peers increases, the overall trends are very similar to CENTRALISED. Figure 7.8(a) shows that POUT_M exactly follows CENTRALISED because a single RP is



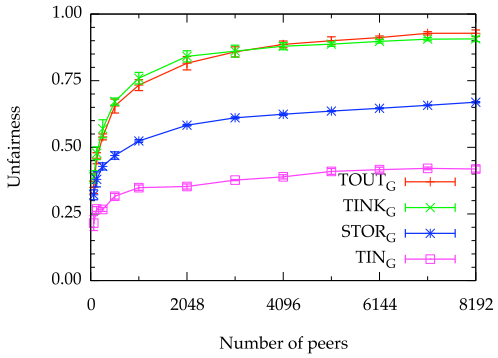
(a) TOUT$_M$ and TIN$_M$ are reduced because casts are handled by many RPs.



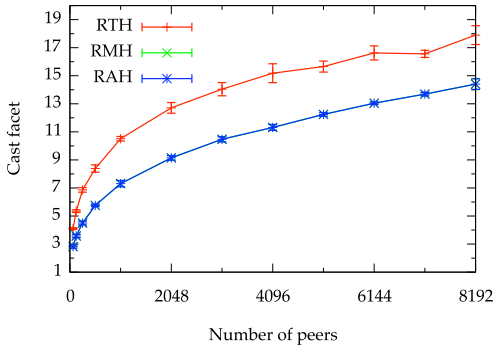
(b) STOR$_M$ increases because more peers register common tags stored at few RPs.



(c) Network metrics shadow the peer facet.

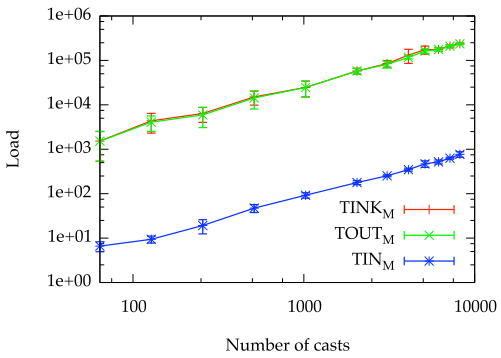


(d) TIN$_G$ is fair because many peers on the ICE surface route casts.

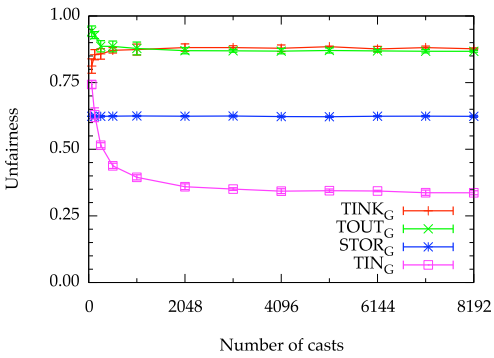


(e) RAH/RMH are identical.

Figure 7.8: Peer scalability in the basic SPICE implementation.



(a) TIN_M is quite reduced because many RPs handle the casts.



(b) Incoming loading is much fairer than CENTRALISED.

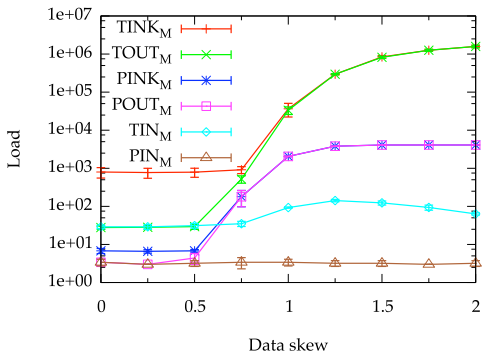
Figure 7.9: Cast scalability in the basic SPICE implementation.

required to forward payloads to consumers for each cast much like a server. TOUT_M also increases with the number of peers but is a fraction of the CENTRALISED implementation because the total outgoing load is divided over several RPs rather than a single server. TIN_M is likewise reduced. PIN_M, however, is slightly increased because the ICE algorithm occasionally routes the same cast through the same peers. This can occur when a route is branching but the next best hop for each branch is the same neighbour, for example. Network loading is also similar to CENTRALISED, since the links around the most loaded peers are also loaded (Figure 7.8(c)).

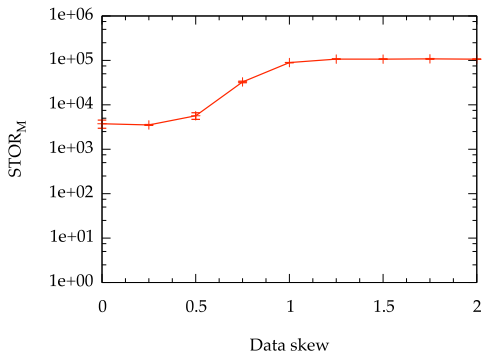
Figure 7.8(b) shows that STOR_M is reduced from the CENTRALISED implementation because registries are stored at distinct RPs. If a single tag were registered by every peer, however, STOR_M would be much closer.

The basic SPICE implementation is unfair in terms of TOUT_G and STOR_G (Figure 7.8(d)). This is because a few tags are very common in registrations (leading to high concentrations of registrations at few RPs), and appear frequently in target expressions (meaning a few RPs must handle and forward many casts). TIN_G, however, is much fairer because many peers contribute to routing messages over the ICE surface. Network load is also unfair due to heavy loading around the commonly selected RP peers.

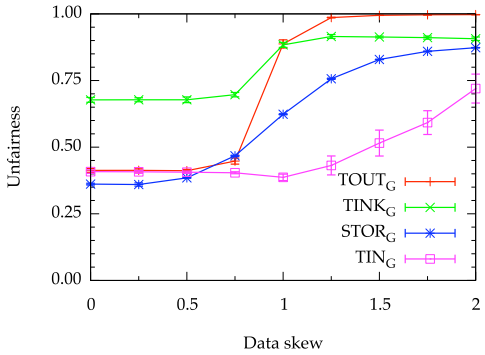
Figure 7.8(e) shows the RAH/RMH to precisely follow the expected trend. The higher number of hops to each consumer is a result of routing over the ICE surface. The total cost of casting to many small groups is dominated by routing to an RP, raising RTH generally.



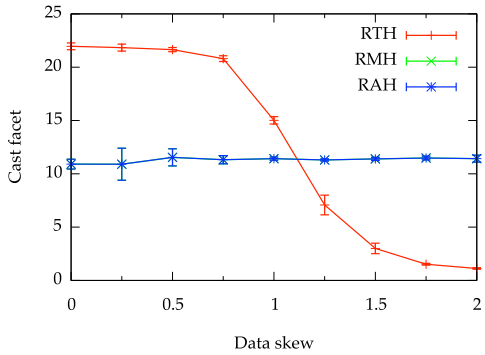
(a) POUT_M and TOUT_M plateau once large implicit groups are selected in high skew scenarios.



(b) STOR_M plateaus when all peers register the same tags (high skew).



(c) The design is very unfair with skewed data.



(d) RTH is highly dependent on the variety and sizes of implicit groups caused by data skew.

Figure 7.10: Data skew in the basic SPICE implementation.

7.6.2 Cast scalability

The skewed distribution of casts means that as more messages are cast to implicit groups, particular RP peers are more heavily loaded than others. Figure 7.9(a) shows that TOUT_M and TIN_M therefore increase with the number of peers, as they do in CENTRALISED. The actual load is only a fraction because many RPs are used over all casts, and TIN_M is particularly reduced because many RPs receive the casts, rather than a single server. Figure 7.9(b) confirms that TIN_G is quite fair because of the distribution of casts over many RPs, and the fact that many peers aid routing.

7.6.3 Data skew

Figure 7.10(a) shows that apart from a fractional reduction in TIN_M and TOUT_M caused by the involvement of many RPs over all casts, the basic SPICE implementation behaves much like CENTRALISED. The reduction in TIN_M is counter-intuitive. When data skew is high, the same implicit groups are targeted repeatedly and the same RP peers receive the casts leading to a generally high number of incoming mes-

sages. However, the high data skew also results in many peers registering those tags selected by casts, including the RP peers. Because these casts are expected by the RPs (as consumers), they are not considered incoming load, and TIN_M is therefore low.

STOR_M increases with the peer skew (Figure 7.10(b)) because as some tags become more common, individual RPs are required to store registrations for more peers.

Figure 7.10(c) shows basic SPICE to be generally more unfair with higher data skew. TOUT_G is particularly unfair because the same RP peers must frequently notify large implicit groups. TINK_G shadows this trend due to the links around the RPs carrying multiple copies of the casts.

Figure 7.10(d) shows that RAH/RMH do not vary with skew because each cast still requires a route across the surface followed by a notification to the consumer. RTH falls dramatically, however, because the sizes of implicit groups tends to grow significantly with higher skew, and the cost of the route to the RP becomes insignificant in comparison to the number of implicit group members that must be notified.

7.7 Summary

This chapter has presented and evaluated the basic SPICE implementation, which is based on a distributed structured P2P design (Contributions 7 and 8). A major component of SPICE is the novel ICE substrate, which combines an hierarchical tesseral addressing scheme with an amortised routing algorithm enabling applications to efficiently multicast large groups of peers (Contribution 9).

Without load distribution techniques, SPICE performs similarly to the CENTRALISED implementation. However, storage load and total incoming and outgoing load on peers is generally reduced, even without the ancillary load distribution mechanisms of SPICE, by virtue of the fact that more peers act like servers. On a per cast basis, however, there is no improvement. In fact, because RP peers are randomly selected from possibly under-provisioned peers rather than servers designed to cope with a high load, the basic SPICE implementation is actually less capable of supporting a large-scale IGM system.

The registry distribution and replication mechanisms are designed to alleviate this drawback, permitting peers to limit their involvement in casts. The next chapter details these techniques. The evaluation separately investigates their ability to distribute load, before revisiting the common loading experiments.

Chapter 8

SPICE Unloaded

Previous chapters have presented increasingly distributed IGM implementations aimed at fairly and effectively reducing the load on peers and the physical network. This chapter broadens the theme of decentralisation to its logical extreme by extending the SPICE P2P design with load distribution techniques utilising the features of ICE (Contribution 10). This chapter directly addresses the hypothesis that a distributed structured overlay design can effectively manage the highly skewed loading characteristics inherent to IGM.

8.1 Design

Due to its homogenous design, the basic SPICE implementation is sufficiently fair when peer registration and cast distributions are uniform. However, as claimed in Section 3.4, these distributions are more likely to follow Zipfian distributions of varying skew. In such conditions, the basic SPICE implementation presented in Chapter 7 suffers loading problems comparable to the CENTRALISED implementation. This chapter therefore extends the basic design by making use of the novel features offered by the ICE surface: its hierarchical tesseral addressing and amortised routing algorithm.

The load distribution techniques presented are motivated primarily by the results of the data skew evaluation of the basic SPICE implementation (Section 7.6.3). These experiments demonstrate that as larger implicit groups are more frequently cast (i.e., as data skew increases), POUT_M, TOUT_M and STOR_M increase very significantly, and all metrics become extremely unfair. In the basic SPICE implementation, network facet metrics tend to shadow peer facet metrics. Intuitively, curbing load and unfairness with respect to the peer facet should improve the network facet also.

Table 8.1 enumerates four extreme points on the continuum of possible casts and shows their effect on the incoming and outgoing loads of RPs. The work of any one peer should be minimised despite this variety of loading possibilities. For small

Table 8.1: Various types of casts that cause load imbalance.

	Frequent cast	Seldom cast
Common tag	↑ incoming, outgoing	↑ outgoing
Rare tag	↑ incoming	↓ incoming, outgoing

implicit groups, it is best to deliver casts directly from RPs to each member as this minimises delay without greatly affecting the network. For large groups, notification places a high outgoing load on RPs and the links surrounding them, so a better solution is to spread the notification load to other peers. Finally, RP peers for tags frequently appearing in casts may have a high incoming load. This incoming cost should be shared with other peers. These are the primary guiding principles of the load distribution techniques presented in this chapter.

Two new parameters are needed to capture these principles. The *storage limit* (SL) is the maximum number of registrations a rendezvous peer is willing to store for a particular registry. The *frequency limit* (FL) is the maximum frequency of casts (per second) a rendezvous peer is prepared to service. The intention is to reduce storage and outgoing load on RPs that hold registries for common tags, and reduce incoming load on RPs that hold registries for tags that appear frequently in target expressions. Note that although the designs presented below make it possible for each peer to maintain individual storage and frequency limits, the evaluation in this chapter assumes the parameters are global across peers.

The storage and frequency limits guide the operation of two complementary load distribution techniques called *registry distribution* and *registry replication*. The former adaptively distributes the registrations held in a registry over many peers surrounding the original RP, such that each is required to store and forward only a fraction of the total outgoing load for a cast. The latter makes complete copies of registries at other parts of the surface that can be independently used to resolve casts, thereby reducing the total incoming load on any single RP. These techniques can be combined for the registries of tags that are both commonly registered by peers and frequently used in target expressions.

8.1.1 Registry distribution

The basic SPICE registration algorithm stores registries at deep rendezvous extents on the ICE surface. When many peers register the same tags, these registries become large and the RPs responsible for them must store many registrations and notify many consumers.

Registry distribution is designed to reduce the storage and outgoing load of these

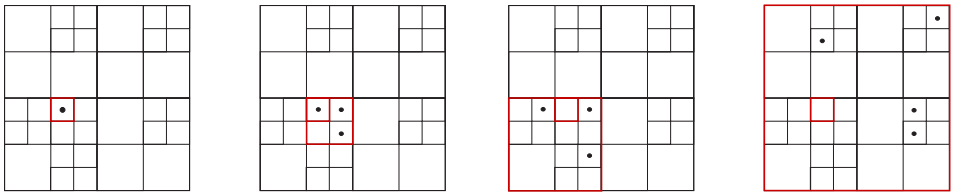


Figure 8.1: Registry distribution—a registry's rendezvous extent scales as it is overloaded with registrations. Extent 210 scales to 21, then 2, and finally $\mathcal{U}$.

RPs. It works by incrementally scaling the address of where a registry is stored from the initially deep rendezvous extent to the entire ICE surface, using the hierarchical property of the tesseral addressing scheme. The initial rendezvous extent found by hashing a tag can be thought of as the deepest possible *container* for all registrations. At this depth it is covered by a single peer which is solely responsible for the registry. When an RP reaches its storage limit, the rendezvous extent is scaled by omitting a digit from the end of its address. The new container is 2^d times the size and intersects with the tracts of more peers, each of which becomes an RP for that registry and is responsible for storing a fraction of the registrations. The number of times a rendezvous extent is scaled is called the *notch* of the registry.

Take Figure 8.1 as an example. Suppose that on a 2-dimensional surface the `jazz` tag hashes to the rendezvous extent `210` (the small highlighted extent in the leftmost subfigure). This is referred to as notch 0. All peers that register `jazz` route their registrations to the RP that owns that extent. If the tag is common, the storage limit of the RP will soon be reached so it will increase the rendezvous extent to `21` by omitting the trailing `0`. This extent, at notch 1, is $2^d = 4$ times the size and encompasses four peers (assuming the ICE surface is equally partitioned), each of which becomes an RP for the `jazz` registry and stores a fraction of the registrations. This process can continue as needed to comfortably accommodate all registrations stored in the registry. Eventually the rendezvous extent can reach the universal extent which contains the entire surface, such that every peer in the system may store a fraction of the registrations. The benefit of this approach is that the available storage naturally scales with the number of peers registering in the system.

The scaling of a rendezvous extent does not occur simultaneously across all peers, because SPICE is a distributed system. Each peer acting as an RP waits until it reaches its storage limit before distributing registrations to the peers in the next container. Hence, some parts of the container scale ahead of others. Figure 8.2 shows a tract within a container scaling to the next notch.

When a cast arrives at the original rendezvous peer, it is efficiently routed to peers at the next notch using the ICE routing algorithm. This continues until all registrations are found, whereupon consumers may be notified as they normally would. By

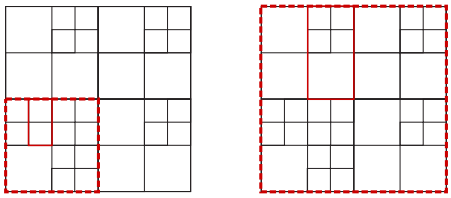


Figure 8.2: A tract is scaled from {201,203} to {01,03} (solid line) within the context of container extent 2 (dashed line).

distributing the registrations over larger extents, individual peers need store fewer registrations, and perform fewer notifications.

8.1.2 Registry replication

While registry distribution reduces the storage load and outgoing load per cast on RPs, registry replication is designed to reduce the incoming load on peers caused by casts frequently using the same tags in their target expressions. Each peer records the frequency of casts it receives. When this exceeds the frequency limit, the peer replicates its registrations to other parts of the surface using the ICE routing algorithm. These replicas then resolve a fraction of new casts in the same way as the original RP, reducing its incoming load.

The extents to which registries are copied are found by replacing the head of the rendezvous extent address with a number of "wildcards". For example, Figure 8.3 shows a 2-dimensional surface with the RP at `210` that holds the registry for the `jazz` tag handling many casts (at left). Level 1 replication of this rendezvous extent, `*10`, copies the registry to corresponding extents in the other three quadrants, i.e., `010`, `110` and `310`. Level 2 replication results in a total of 16 copies at corresponding extents within the next deepest set of extents, `**0`.

When a peer determines that its frequency limit is exceeded by incoming casts, it begins the replication process. If the registry to be replicated is not distributed, the peer simply routes a copy of it to the set of extents that will act as the roots of the replicas. Casts may be resolved by the replicas once the routed registry is received and stored by the new RPs. If the registry to be replicated is distributed, a replication command traverses the whole registry in the same way as casts. Each distributed RP that receives the command routes its fraction of the registry to the set of new replica roots. As the new replicas receive the casts, registry distribution is employed as normal to create newly distributed replicas of the registry. Note that this approach, although conceptually straightforward, is not ideal for larger distributed registries. See Section 8.3.2 for optimisations and complexity analysis.

Every level of replication reduces the probability of a particular replica being chosen by a factor of 2^d, since replication replaces the first digits of an address with wild-

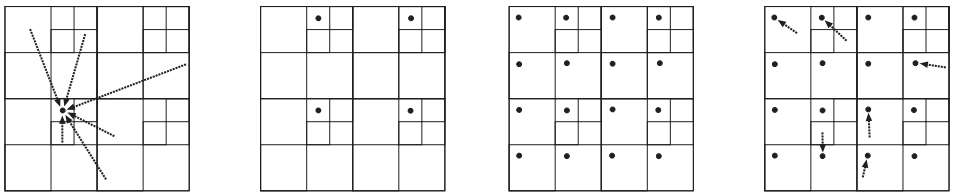


Figure 8.3: Registry replication—0, 1 and 2 levels of replication. Registries are replicated to extents corresponding to the original RP.

cards and is thus dependent on the surface dimensionality. A peer can calculate the replication level needed to reduce a tag's selection frequency to a value less than its frequency limit. Replication level r for a tag with selection frequency f and peer frequency limit m is defined by Inequality 8.1. Equation 8.2 finds the smallest integer r that satisfies this inequality, which is the level to which a peer replicates the registry.

$$\frac{f}{2^{dr}} \leq m \tag{8.1}$$

$$r = \left\lceil \frac{\log \frac{f}{m}}{\log 2^d} \right\rceil \tag{8.2}$$

To actually spread load to the replicas, it is necessary to adjust the cast algorithm slightly. Ordinarily, casts are routed from the publisher to the distant rendezvous extents of tags in the target expression. Replicas may be found during this route by perturbing slightly from the most direct path so as to pass through extents where a replica may exist. This is practical since it is possible to calculate the locations of potential replicas from the tag alone. If a replica is found, the route is terminated, resulting in a shortened path and reduced total incoming load on the original rendezvous peer. If no replicas exist for a tag, the message will still reach the original rendezvous extent, having deviated only slightly from the optimum route.

Specifically, the cast is first routed to the nearest extent that may hold the most heavily replicated version of the registrations. If no replica is found at this extent upon arrival, it is known that the registrations have not been replicated to that level, so a lesser level of replication is selected and the process is repeated. Figure 8.4 shows the process. For those rendezvous extents which have been replicated, this modification to the cast algorithm means that casts will always be delivered to the closest possible replica (illustrated in Figure 8.3). Therefore, replication combined with the modified cast algorithm divides the amount of incoming traffic for a tag evenly over all replicas, assuming publishers are randomly spread through the network.

The altered cast algorithm may seem to promote heavy loading of routes between potential replica sites. In fact this is not a problem because any cast that is sufficiently popular to create loading between replica sites naturally triggers registry replication. Hence replicas are found quickly on these routes, and no worn paths appear.

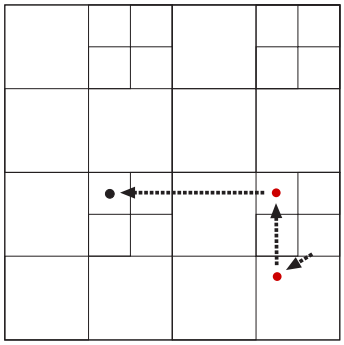


Figure 8.4: The modified cast algorithm perturbs a route towards an RP through potential replicas (first level 2, then level 1, then 0).

8.1.3 Distribution and replication algorithms

Table 8.2 and Listings 9–12 present the extended SPICE Unloaded objects and algorithms. A new object, `Registry`, is introduced which holds a set of registrations and is associated with a particular combination of `tag`, `notch` and `replica_root`. Registry objects are stored by all peers that are part of a registry, distributed or otherwise. The `replica_root` is the extent that is used as the root of the registry (which will vary according to the placement of replicas). The depth of this extent indicates the level of the replication. A root at the universal extent (i.e., of depth 0) indicates a registry is the original version, not a replica.

Peers register by constructing a `Registry` object for each tag and route it to the corresponding RPs as before (Line 10 of Listing 9). When it arrives it is concatenated to the existing registry object (Line 9 of Listing 10). If the registry is full, it is marked as *closed* and all its registrations are routed to the next notch (Line 16) recursively until enough space is available. These notches are found by scaling the peers' tracts within the next largest container extent (illustrated in Figure 8.2, and implemented at Line 22).

Registrations are routed to these tracts using a modified ICE routing algorithm. Recall that ICE ordinarily routes a payload to all peers within a destination tract. However, it also provides two hooks that allow an application (such as SPICE) to alter these payloads at key points during the routing process. Listing 11 presents the two callbacks used to alter the registry payload as it is routed. Essentially, these ensure that the registrations contained within the registry payload are uniformly divided among the peers in a destination tract. The premise is that if a routed message is branching to a number of clusters, the registry payload should be proportionally split among them, according to their size. Similarly, when a registry is to be delivered to a peer for storage, only a fraction of registrations proportional to the remaining destination tract are kept. The rest are routed to the remainder of the destination. No registration is

delivered to more than one peer, and no registrations are omitted.

Casts are handled similarly. A peer initiates a cast by handling it as though it were just received from another peer (Line 19 of Listing 9). Because the cast is not initially assigned to any particular replica, the nearest potential replicated rendezvous extent is calculated (Line 29 of Listing 12). This continues until a replica is found (Line 6). If it is a non-distributed, non-closed registry, notifications can begin immediately (Line 19). If the registry is closed, then it has been distributed to higher notches, and the cast is routed to the scaled notch in the same way as the registration algorithm (Line 17). Eventually, the open registries will be reached, and consumers may be notified.

Section 8.1.4 includes further details of aspects of the algorithms.

Table 8.2: Extended object structure for SPICE Unloaded algorithms.

(a) A peer object.

Peer	registration	tract	registries

(b) A registry object.

Registry	tag	notch	replica_root	registrations	*closed?()*

(c) A registration object.

Registration	peer	tags	registry

(d) A cast object.

Cast	target	term	tag	notch	replica_root	destination_tract	payload

Listing 9 The SPICE Unloaded publisher/consumer algorithms.

```
1  # Peer p registers.
2  def reg(p)
3     p.registration.tags.each do |tag|
4        registry = Registry.new
5        registry.registrations.push(p.registration)
6        registry.tag = tag
7        registry.notch = 0
8        registry.replica_root = Extent.universal
9        p.registration.registry = registry
10       p.route(tag.rp,p.registration)
11    end
12 end
13
14 # Peer p casts c.
15 def cast(p,c)
16    c.target.or_terms.each do |term|
17       c.term = term
18       c.tag = term.first
19       handle_cast(p,c)
20    end
21 end
```

Listing 10 The SPICE Unloaded register algorithm.

```
# Peer p handles routed registration r.
def handle_reg(p,r)
   # Look up an existing registry at this peer.
   if (rp_registry = p.registries.find(r.registry.tag,r.registry.notch,
                           r.registry.replica_root))
      if rp_registry.closed?  # The registry is closed; route up.
         route_reg_up(p,r.registry)
      else  # Add new registrations to existing registry.
         rp_registry.registrations.concat!(r.registry.registrations)
      end
   else  # Couldn't find an existing registry, so create one.
      rp_registry = p.registries.new(r.registry)
   end
   # If registry is full, close it and scale up.
   if rp_registry.size > p.storage_limit and rp_registry.notch < max_notch
      route_reg_up(p,rp_registry)
      rp_registry.close!
   end
end

# Peer p routes registry to the next notch for random storage.
def route_reg_up(p,registry)
   # Find the registry's current container.
   rp = registry.tag.rp.translate_to(registry.replica_root)
   container = rp.omit_tail(registry.notch)
   # Scale up the part of this peer's tract in the container.
   destination = p.tract.scale_all_within(container)
   # Route the registry to that part of the scaled container.
   r = Registration.new
   r.registry = registry
   r.registry.notch += 1
   p.route(destination,r)
end
```

Listing 11 The ICE callbacks for the SPICE Unloaded register algorithm.

```
# Deliver only a portion of the registrations to a peer.
def callback_deliver(p,route)
   if (route.destination_tract - p.tract).empty?
      # If this peer is the last in the destination tract,
      # give it the remaining registrations.
      yield route.payload
   else
      registrations = route.payload.registry.registrations
      # Keep some registrations for this peer, proportional to
      # its size compared to the entire tract being visited.
      k = (p.tract.volume / route.destination_tract.volume) *
         registrations.size
      new_payload = route.payload
      new_payload.registry.registrations = registrations[0...k]
      yield new_payload
      # And route the remaining registrations on.
      route.payload.registry.registrations = registrations[k..-1]
   end
   return route.payload
end

# Route only a portion of registrations to each cluster.
def callback_branch(p,route,clusters)
   registrations = route.payload.registry.registrations
   total_volume = clusters.inject(0) { |sum,cluster| sum+cluster.volume }
   # Send a portion of registrations to each cluster.
   clusters[0...-1].each do |cluster|
      # Make new payload with partial registrations.
      new_payload = route.payload
      k = (cluster.volume / total_volume) * registrations.size
      new_payload.registry.registrations = registrations.slice!(0..k)
      # Route it to a cluster.
      yield new_payload,cluster
   end
   # And send the remainder to the final cluster.
   final_payload = route.payload
   final_payload.registry.registrations = registrations
   yield final_payload,clusters.last
end
```

Listing 12 The SPICE Unloaded cast algorithm.

```
# Peer p handles routed cast c.
def handle_cast(p,c)
   # Test whether this peer is storing a replica of the registry.
   if c.replica_root.nil?
      # Find the maximum possible replica root. TODO.
      root = max_replica_root(p.registries,c.tag,c.notch)
      unless root.nil?
         c.replica_root = root
         c.destination_tract = Tract.new(c.tag.rp.translate_to(c.replica_root))
      end
   end

   # A replica has been found; notify consumers or route up the notches.
   if c.replica_root
      if (registry = p.registries.find(c.tag,c.notch,c.replica_root))
         if registry.closed?
            route_cast_up(p,c)
         else
            registrations = registry.registrations.find_all do |r|
               c.target.selects?(r)
            end
            registrations.each do |r|
               p.notify(r.peer,c) if minimum_term?(r,c)
            end
         end
      end
   else
      # Looking for replica. Find next possible replica on path.
      p.route(calc_next_replica(p,c),c)
   end
end

# Peer p routes cast c to the next notch to find registrations.
def route_cast_up(p,c)
   root = c.tag.rp.translate_to(c.replica_root)
   container = root.omit_tail(c.notch)
   destination = Tract.new(c.destination_tract.intersect(p.tract))
   destination.scale_all_within!(container)
   c_dup = c
   c_dup.notch += 1
   c_dup.destination_tract = destination
   p.route(destination,c_dup)
end
```

8.1.4 Peer flux and self-stabilisation

Registry distribution and replication necessarily complicate SPICE's handling of peers joining, registering, unregistering, leaving and failing. However, the two techniques are independent so their behaviour can be considered separately.

Registry distribution

A new peer claiming part of a tract owned by a peer participating in a distributed registry is handled by copying all closed registries to the new peer, and splitting any open registries equally. Each peer is then capable of handling casts as normal.

Registrations can be removed by routing unregister requests to an RP, which then routes it through the distributed registry in the same ways as casts. To improve efficiency, RPs may periodically forward these casts in bundles, or piggy back them on casts, rather than route each as it arrives. Alternatively, registrations could be *leased* meaning they automatically expire after a period of time. Unregistration would not be required, but consumers would need to reregister periodically.

The departure or failure of a peer that is part of a distributed registry is handled as it is in basic SPICE. Peers are seeded with copies of neighbours' data when registrations first occur, so a neighbour is able to take over both a failed peer's tract and its role as part of the distributed registry. If both peers have open registries for the same combination of tag, notch and replica root, they are merged. If either is already closed, both are closed and any registrations are routed to the next notch contained by the remaining peer's new total tract. This ensures that the remaining peer still has exactly one registry for each combination of tag, notch and root. In essence, local repairs to the surface and delegation of roles result in the remaining peer acting as if it had always been the only peer at that location.

General peer flux and the removal of registrations by consumers may eventually lead to distributed registries becoming fragmented and only partially full. It is inefficient for casts to be routed to all RPs when a lesser number would suffice. For this reason, peers can initiate a redistribution process if the number of registrations they store falls below a threshold. A peer does this by routing a redistribution command to those RPs above it, which propagates to the registry's leaves in the same way as a cast. Upon receipt of this command, each RP routes its registrations to the original RP where they are treated as new registrations and redistributed as normal. During this period, old RPs maintain copies of the registrations so that casts can continue to be resolved before they have been redistributed. After a short period of time, the registrations are presumed to have been stored and the old RPs delete their registries.

This algorithm potentially permits consumers to receive duplicate casts or miss casts completely, thereby failing to satisfy both the safety and liveness conditions of

IGM. The liveness condition can be satisfied by using a distributed commit protocol between the old RPs and the original. Only once the original has agreed to take responsibility for consumer notifications does the old RP delete its registry. Note that although consumers will not miss any casts, this still permits some to be notified twice if a cast occurs during this commit. Such duplicates can be detected by the SPICE code running on the consumer and eliminated to ensure applications do not receive them.

Registry replication

Registry replication can be applied to both distributed and non-distributed registries. Each replica behaves exactly like the original registry, distributing as needed to efficiently store incoming registrations. The algorithms for maintaining distributed registries described above apply equally to all replicas.

A new peer may register at any replica. Because the IGM liveness condition only requires that registrations are eventually (not immediately) active, replicas need only be weakly consistent. I.e., registrations can be collected for a period before they are routed to the other replicas. Similarly, requests to unregister can be collected and copied to other replicas periodically. Note, however, that since a peer that has received a cast must receive all subsequent casts selecting the same registration, a registration must be fully replicated at all roots before any cast targeting it is honoured.

When a replica's incoming cast frequency drops below a threshold, it tries to reduce the level of replication in the system to avoid the maintenance traffic required to synchronise registries. It does this by routing a suggestion to all other replicas that the replication level be reduced. If no replica vetoes the suggestion within a certain period, each replica reduces its level. In most cases this means the registries stored by the replica can be deleted, although replicas situated at the rendezvous extents for the reduced replica level continue to resolve casts.

8.2 Conformance

The basic SPICE implementation has already been shown to conform to the IGM specification (Section 7.4), so it is sufficient to demonstrate that registry distribution and replication do not invalidate those arguments.

In basic SPICE, registrations are stored by routing them to RPs and casts are resolved by routing to a subset of the RPs comprising a target expression and notifying all selected consumers. This section shows that the distributed and replicated registries in SPICE Unloaded have identical semantics for these operations.

This analysis assumes the system is in a stable state, with all replication and distribution complete. See Section 8.1.4 for discussion of how dynamism affects the liveness and safety conditions.

8.2.1 Liveness

The liveness condition (Theorem 3.7) states that if p registers, then from some point in the future onwards, p will eventually be notified of each cast that selects its registration.

The modified cast algorithm is guaranteed to eventually reach either a replica or the original registry RP, because it is routed through extents where replicas must be stored if they exist, before finally arriving at the original rendezvous extent (Line 29 of Listing 12). These extents are deterministically calculated using the same algorithm that calculates the replica extents during registry replication. A replicated registry contains identical registrations to the original (by design), and the case of a non-distributed registry has been previously shown to satisfy liveness. It is sufficient, therefore, to show that every registration stored in a distributed registry will be discovered by a cast.

Firstly, it must be shown that every registration is stored by a peer in the distributed registry. Initially the registration is routed to the root. If the registry fills, it is routed to the next notch, and the registry is closed. The tract to which registrations are routed is determined by scaling the tract of the peer that closes the registry (Line 22 of Listing 10).

The ICE routing algorithm ordinarily routes an identical copy of a payload to every peer intersecting a destination tract. SPICE modifies this behaviour using two routing callbacks (Listing 11). These callbacks partition the registrations such that each is routed to exactly one peer in the destination tract. When a route branches, the set of registrations is equally split between each cluster (Line 31). Clusters cannot intersect, so different peers must receive different registrations. When a payload is delivered to a peer, only some of the registrations are retained. The rest are delivered to the remainder of the destination tract (Line 14). However, if the peer is the last in the remaining destination tract, all registrations are delivered to it. Thus, although the set of registrations is divided as the route progresses, each registration will be delivered to exactly one peer.

Registrations are recursively routed in this way if peers exceed their storage limits. However, distributed registries may only scale to a finite size limited by the number of hash bits used in the system (see Section 7.3.1). There must exist a peer that does not close its registry (Line 15 of Listing 10). I.e., there must exist a peer with an open registry storing each registration.

Finally, it is necessary to show that a cast will be routed to all of these open registries. Casts are initially routed to the root of a registry (Line 29 of Listing 12). If a closed registry is found, it is routed to the same scaled destination tract as registrations (Line 17). Casts are routed to all peers in this destination tract using default ICE routing which guarantees delivery to every peer in the tract (and therefore every peer that

may hold registrations previously routed from a lower notch). As in the registration algorithm, this process recursively continues when closed registries are encountered. The distributed registry is of finite size, so eventually the cast must be routed to peers with open registries which together contain all of the registrations. When open registries are found, the algorithms used in basic SPICE ensure that all selected consumers are notified.

Therefore the SPICE Unloaded design satisfies the liveness condition.

8.2.2 Safety

The safety condition (Theorem 3.8) states that when a consumer is notified of a cast: it will not be notified again; the message was previously cast by some publisher; and its registration is selected by the cast's target expression. Additionally, the condition requires that the consumer has completed a registration in order to agree with the adjusted definition of an implicit group.

As with the other IGM implementations, it suffices to show all components of the IGM safety condition separately to show it as a whole.

$\Box[notify(p, c) \Rightarrow \circ\Box\neg notify(p, c)]$ This component says that a consumer must never be notified more than once for a particular cast. The validity of the basic cast algorithm has already been shown in Chapter 7, including the use of a total ordering of disjunctive terms to prevent duplicates.

In SPICE Unloaded, a cast is always resolved by exactly one registry replica (possibly the original level 0 replica). The modified cast algorithm searches for a replica by routing to their potential locations (Line 29 of Listing 12). This amounts to a sequential search and the route is essentially point-to-point, so it is not possible for more than one replica to receive and process the cast.

Each replica is a registry that may or may not be distributed. If it is not distributed, the RP will not notify the same consumer twice (since this instance collapses to the basic SPICE implementation). If it is distributed, each of the constituent RPs notifies the owner of every selected registration it stores (Line 23). Hence, it is necessary to show that a registration is stored by no more than one peer in a distributed registry.

A registration is routed once to the root of a registry (Line 10 of Listing 9) where it may be added to an existing registry (Line 9 of Listing 10) or to a new one (Line 12). If the registry is closed or overloaded, the registration is routed to the next container (Line 22).

The destination tract for the route is calculated by scaling the peer's tract within a container extent. As Figure 8.2 illustrates, any mutually exclusive tracts within the container scale to mutually exclusive tracts at the next notch. Since a peer's tract is exclusively owned, the destination tract cannot intersect with the destination tract of

another peer's scaled tract. Therefore, the same registrations cannot be routed to any common tract, nor any common peers.

The ICE routing algorithm ordinarily routes an identical copy of a payload to every peer intersecting a destination tract. SPICE modifies this behaviour using two routing callbacks (Listing 11). These callbacks partition the registrations such that none is routed to more than one peer in the destination tract. When a route branches, the set of registrations is proportionally split between each cluster (Line 31). Clusters cannot intersect, so different peers must receive different registrations. When a payload is delivered to a peer, only some of the registrations are retained. The rest are delivered to the remainder of the destination tract (Line 14).

This argument can be inductively applied to the whole distributed registry, and therefore no registration is ever stored by more than one peer within a single registry. Combined with the fact that only a single registry is used to resolve a cast, no peer is ever notified of a cast more than once.

$\Box[notify(p, c) \Rightarrow \exists q.c \in C(q)]$ This component requires that a message must have been cast before a consumer is notified of it. Consumers are only notified when a peer processes a cast (Line 23 of Listing 12). Casts are only processed by a peer when received through the ICE routing mechanism, or when a peer initiates the cast (Line 19). Casts are only passed to ICE for routing when searching for a replica (Line 29), or routing to the next notch in a distributed registry (Line 42). Registries may only be distributed to a finite notch, limited by the number of bits in the rendezvous extent hash function. Therefore every routed cast must originate while searching for a replica (Line 29). This case is only invoked when a replica root has not been found. At such a time, the next potential replica root is calculated, which is necessarily at a lesser level of replication (by design). The number of levels of replication is also dependent on the number of bits in the hash function (and therefore finite), so there must exist a point before which the cast cannot have been searching for a higher level of replication. The only other way for a cast message to be processed by a peer is when it initiates a cast (Line 19).

$\Box[notify(p, c) \Rightarrow c_t \triangleright p_r]$ This component requires that only consumers with registrations selected by a cast's target expression are notified, which is expressly specified in Line 20 of Listing 12). No peers besides RPs notify consumers of casts.

$\Box[notify(p, c) \Rightarrow p \in R]$ This component requires that a consumer has registered before it is notified of any casts. RPs only notify peers if an associated registration is found in its registry (Line 23 of Listing 12). Registries are only modified when RPs receive registrations (Lines 9 and 12 of Listing 10). RPs receive registrations routed

from consumers (Line 10 of Listing 9), but also during registry distribution (Line 32 of Listing 10) and replication.

It is necessary to show therefore that any registration distributed or replicated must have originated with a consumer. Replicated registries are copies of existing distributed or non-distributed registries (by design). Therefore, it suffices to show that distributed registries contain only registrations registered by consumers. A registration is only added to a registry when it has been routed by a consumer (Line 10 of Listing 9) or from a lower notch (Line 32 of Listing 10). Distributed registries are of finite depth (limited by the hash function). There must exist a root of the registry, from beyond which no registrations can be routed. Therefore, every registration must originate from a consumer.

Because the SPICE Unloaded design satisfies these components individually, it satisfies the safety condition generally.

8.3 Analysis

This section extends the analysis of the basic SPICE implementation from Chapter 7 by incorporating the load distribution techniques described in this chapter.

8.3.1 Registration

The cost of registration is generally the same in SPICE Unloaded as the basic implementation although it is slightly increased by registry distribution and replication. Registrations for popular tags must be routed to RPs in a distributed registry that have not closed, before they can be stored. If the registry is replicated, it must also be routed to the roots of the replicas. However, these costs can be reduced by collating several new registrations before distributing them within a registry, or to other replicas. See Section 8.1.4 for a more detailed discussion of this.

8.3.2 Replication

The replication algorithm presented above is conceptually straightforward, but may be costly for large distributed registries. Each RP stores at most s registrations, where s is the storage limit parameter. Therefore, a distributed registry containing m registrations will be distributed over approximately $\frac{m}{s}$ RPs. Each of these must route its registrations to the new replica extents. If m is large and s is small, this could result in thousands of routes to replicate an entire distributed registry.

This can be optimised by bundling the registrations. Instead of each RP routing its registrations directly to the new replica extents, it routes them to a nearby marshalling peer which collects many such messages before routing them together to the replicas.

The marshalling peers are simply the owners of nearby *potential* replica extents for the tag of the registry. Note that these are not actual replicas, but extents where replicas could potentially exist at a sufficiently deep level. The level chosen determines the tradeoff between the number and size of the messages eventually routed to the replicas. A high level means more marshalling peers route smaller messages and a low level generates a small number of large messages.

Each marshalling peer is responsible for bundling $O(2^{jd})$ messages, for some j, because of the hierarchical nature of ICE. Therefore, the total number of messages that must be routed to replicate a registry can be reduced to $O\left(\frac{m}{s2^{jd}}\right)$, with a corresponding increase in the size of each message. The value of j must therefore be chosen to balance these two extremes. Note also that a registry need only be fully replicated once. After it has been replicated, any new registrations are routed to all replicas as they occur (or soon after).

As with the other implementations, the simulations in the following sections measure loading and fairness during the cast phase only. It should be noted that the levels of replication in the experiments are reasonably low (specified in the introduction of each experiment), and as this section has shown, registry replication can be relatively inexpensive even for very large distributed registries.

8.3.3 Peer facet

A peer's storage limit, s, ensures that it does not store more than a certain number of registrations for each tag, so STOR_M is $O(s)$. POUT_M is also $O(s)$ because the limited number of registrations reduces the number of consumers a peer must notify for any single cast. TOUT_M is similarly reduced by distribution, but also by replication since registries must resolve only a fraction of casts. TIN_M is also reduced by replication for this reason, but it is increased by distribution because the peers storing parts of distributed registries are required to respond to more casts. As distribution and replication increase, TIN_G, TOUT_G and STOR_G become more fair, simply because more peers participate in the processing of casts, notification of consumers and storage of registrations.

8.3.4 Network facet

Basic SPICE tends to load some RPs very heavily. This also results in high load on surrounding network links. SPICE Unloaded aims to spread load fairly over many peers, so the intuition is that network load will not be concentrated around any particular peers. However, the structure of physical networks often emphasises some transit links over others that connect segments of the network. Although the ICE overlay attempts to map closely to the physical network, it cannot effectively prevent all un-

necessary messages traversing these bottleneck links. Thus, while network load is expected to be generally lower and more fair under SPICE Unloaded, the physical topology has some bearing on the extent to which this is true.

8.3.5 Cast facet

Casting takes more hops when registry distribution is used because additional hops are needed to discover registrations surrounding an RP. The maximum hops to reach a group member may be particularly increased if the level of distribution is extremely high. Registry replication may increase or decrease the average number of hops, depending on whether the tag being sought has been replicated. If it has, the path length is approximately reduced by 2^i times where i is the level of replication, because nearby replicas will be found more quickly. If replication is not used, the number of hops will be slightly greater because the route is perturbed from the optimum.

The total number of hops can be expected to increase significantly with high distribution because many more messages must be routed before all registrations are found.

8.4 Evaluation of load distribution

This section investigates the load distribution mechanisms of the SPICE implementation (Section 8.1), imparted by the tesseral addressing scheme and amortised routing algorithm of ICE.

Registry distribution is designed to reduce storage load and per cast outgoing load on rendezvous peers by distributing the work to other peers. Replication is designed to reduce total incoming load on rendezvous peers by allowing any of several replicas to handle each cast. Replication also serves to reduce total outgoing load on peers because rendezvous points need only forward a fraction of the casts to consumers. These techniques can be combined such that the registrations for a tag are both distributed and replicated, reducing all forms of load on the original rendezvous peer.

The peer and cast data used for these experiments are designed to clearly illustrate the benefits of the two techniques. In each, 4096 peers register tags from a Zipfian distribution with skew 1.0. Section 8.4.1 first investigates how the implementation performs under extremely difficult casting conditions to gauge its "worst case" effectiveness. Section 8.4.2 then demonstrates SPICE under more realistic conditions using a Zipfian distribution of cast frequencies.

Because the casts to be used in a simulation are known in advance, it is possible to calculate the level of replication needed for each tag in advance, rather than during the simulation. Casts in the cast phase of the simulation occur once per second (see Section 4.3.3). The RPs to replicate are calculated based on the frequency of tags ap-

pearing in the cast list, the frequency limit parameter and the surface dimensionality parameter for the simulation (Inequality 8.1 describes this relationship). Replication occurs during the replication phase of the simulations, before the casts are initiated. This is true of all simulations of SPICE Unloaded.

8.4.1 Extreme cast load

This experiment tests how the implementation performs when 1024 casts are made to the same very large implicit group selected by a target expression consisting of just one tag. The expression is chosen so as to select a large group: approximately one quarter of all peers in the network (≈ 1024 members).

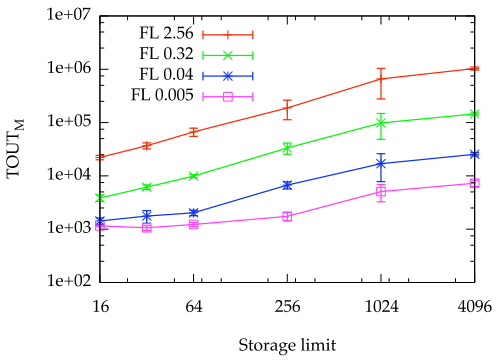
This configuration of peers and casts loads the network in a very specific way, and is a "worst case" scenario to test the load-balancing features of SPICE. Without distribution or replication, the same peer is required to handle and notify all group members for every cast. In effect it is behaving like the server in the CENTRALISED implementation and incurs similarly high storage load (STOR_M), total incoming load (TIN_M), and per cast and total outgoing load (POUT_M, TOUT_M). Indeed, it is required to receive 1024 casts and forward more than a million copies to consumers (approximately 1024 per cast, being a quarter of the 4096 peers in the network).

The storage limit is varied from 4096 (no distribution) to 16 (heavy distribution) and the frequency limit from 2.56 casts/s (no replication) to 0.005 casts/s (level 3 replication). The benchmark for these experiments is the basic SPICE implementation with no distribution or replication, which is equivalent to SPICE Unloaded with very high storage and frequency limits.

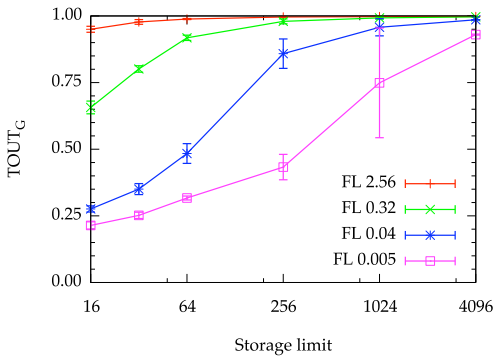
Note that the degree of distribution increases from right to left in these plots as the storage limit decreases.

Figure 8.5(a) shows that TOUT_M is worst when there is no replication or distribution. A very large reduction in TOUT_M is observed by reducing the storage limit, because individual peers store only a small number of registrations each. Replication also greatly reduces TOUT_M because each registry replica is only used to resolve a fraction of all casts. TOUT_M converges to a minimum when replication is combined with distribution, because the scaled rendezvous extents used by registry distribution begin to coincide with those from other replicas and load the same peers.

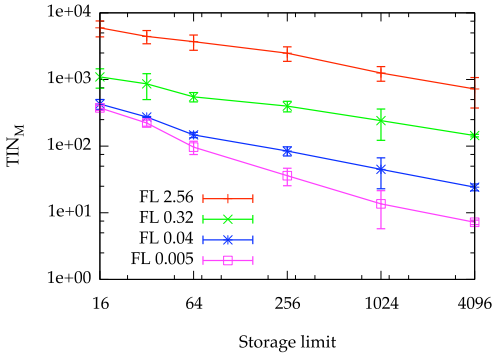
Figure 8.5(c) shows that TIN_M is lowest when there is no distribution. This is a consequence of the routing algorithm; by not needing to route a cast to all RPs in a distributed registry, fewer total messages are needed. This is confirmed in Figure 8.5(e) which shows that RTH is highest when registries are distributed. Replication significantly reduces TIN_M, as it is designed to do, though the reduction is less dramatic as the degree of replication increases. Again, collision of distributed registries limits the benefits of heavy distribution and replication beyond a certain point.



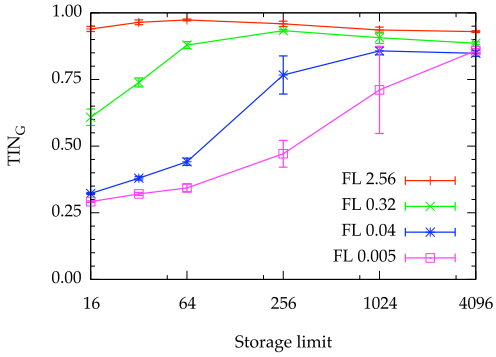
(a) TOUT$_M$ decreases with heavier registry distribution and replication.



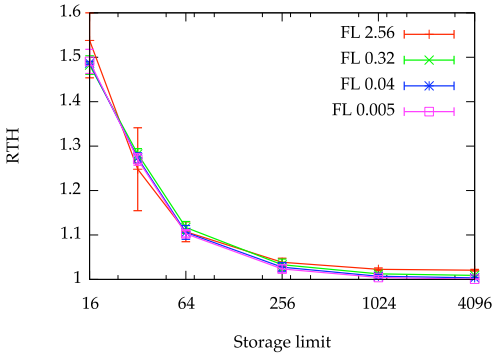
(b) The extremely skewed casts make fairness difficult, but heavy distribution and replication have a remarkable effect on TOUT$_G$.



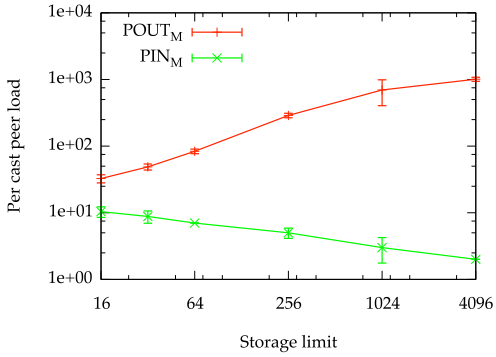
(c) TIN$_M$ increases with heavier registry distribution but can be reduced by also applying registry replication.



(d) The extremely skewed casts make fairness difficult, but heavy distribution and replication have a remarkable effect on TIN$_G$.



(e) RTH is highest with heavy distribution since more peers must be contacted. Replication has no significant effect.



(f) POUT$_M$ decreases with registry distribution at the cost of increased PIN$_M$. Replication does not affect per cast metrics.

Figure 8.5: Extreme cast loading of SPICE Unloaded.

With no distribution, the total number of messages needed to publish a single cast is comparable to the CENTRALISED implementation (Figure 8.5(e)), since a single RP is acting as a server. When heavy distribution is applied, the RTH increases to just twice that of the CENTRALISED implementation, while greatly reducing the load any single peer endures. Replication has little effect on RTH because each cast is handled by a single replica, and the slight reduction of messages provided by routing to a nearby replica is vastly outweighed by the total number of messages needed to notify all group members.

Figure 8.5(f) shows that registry distribution is capable of greatly reducing per cast outgoing load (POUT) (in addition to total outgoing load), at the cost of slightly increased per cast incoming load (PIN). The slight increase in PIN_M is due to a small number of duplicate messages routed through the same RPs when searching for registrations in distributed registries. As these metrics measure per cast load, registry replication does not affect them.

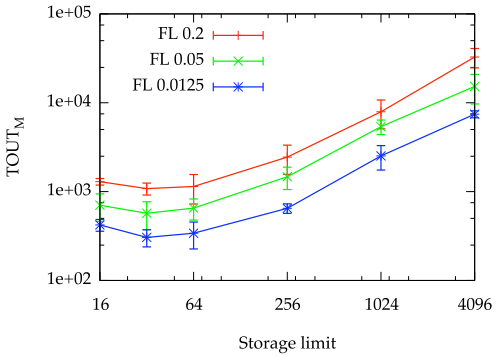
It is very difficult to achieve global fairness with an extremely skewed cast set such as this, but the combination of registry distribution and replication is decidedly effective (Figures 8.5(b) and 8.5(d)). The benchmark basic SPICE implementation with no distribution or replication has TOUT_G of 0.99 and TIN_G of 0.93. With heavy distribution and replication, RTH increases but TOUT_G and TIN_G are reduced to exceptionally fair values of 0.21 and 0.29, respectively.

8.4.2 Realistic cast load

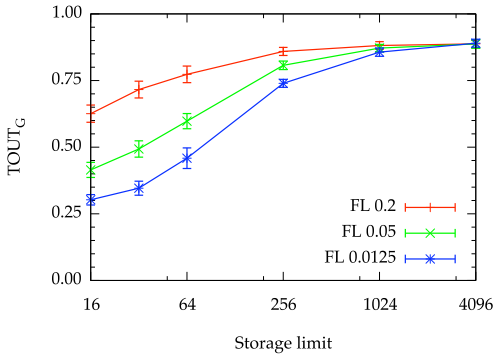
This section tests the implementation using a set of casts with a Zipfian distribution. This is a more realistic type of access load than that used in Section 8.4.1. Consequently, the range of the frequency limit is modified to produce appropriate levels of replication, from 0.2 (no replication) up to 0.0125 (level 1 replication of the six most frequent tags). The general trends seen mirror those in Section 8.4.1 and most of the observations remain valid. However, there are several differences worth noting.

TOUT_M is generally more than an order of magnitude lower with Zipfian casts than the extreme casts of the previous experiment (Figure 8.6(a)). This is due to the far more varied mix of implicit group sizes selected by the casts; many of the groups have no members at all. TIN_M is also an order of magnitude lower than the extreme case (Figure 8.6(c)), since even the most popular tag is only accessed relatively seldom in comparison.

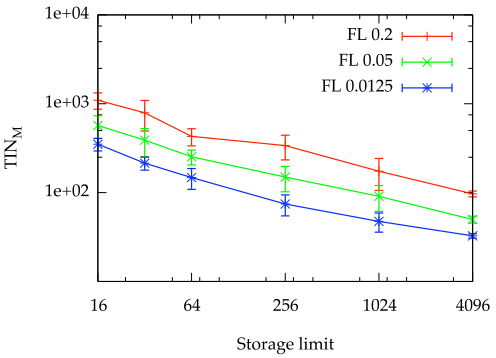
RTH is dramatically higher for the Zipfian cast set on the whole (Figure 8.6(e)). The total number of SPICE messages for a cast using a registry that is not distributed is 14 times that of the CENTRALISED implementation. This is because many of the groups in the Zipfian cast data have very few or no members, but this is unknown until the cast has been routed from the publisher to the RP. Over a 4096 peer 3-dimensional



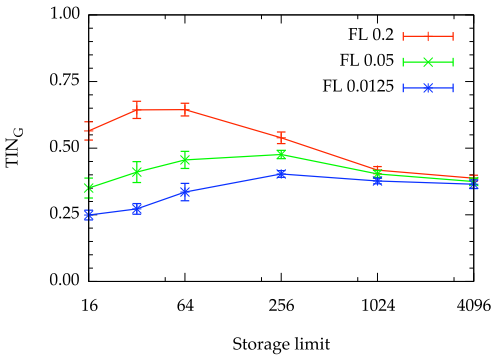
(a) Zipfian casts result in much lower TOUT_M, but distribution/replication still shows a drastic improvement.



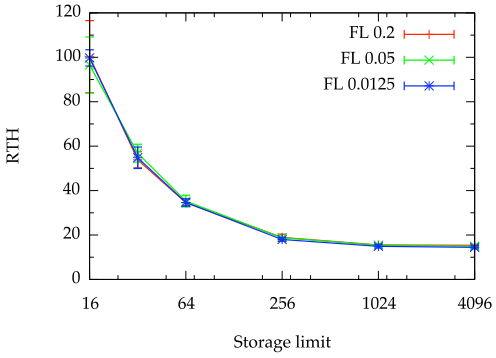
(b) With Zipfian casts, SPICE shows exceptional fairness in terms of TOUT_G.



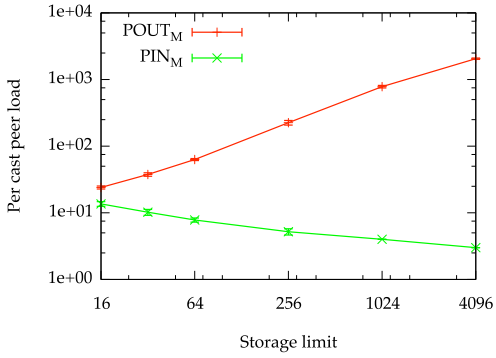
(c) TIN_M is generally lower for Zipfian casts, but registry distribution still leads to increased incoming peer load.



(d) TIN_G is less fair with heavy distribution, unless replication is also employed.



(e) RTH is higher for Zipfian casts since they must still be routed to rendezvous peers, despite many groups being memberless.



(f) POUT_M decreases with registry distribution at the cost of increased PIN_M. Replication does not affect per cast metrics.

Figure 8.6: Realistic cast loading of SPICE Unloaded.

ICE surface, this is approximately 24 hops (Section 7.2.2), compared to the single hop needed to cast to an empty group in the CENTRALISED implementation. Although the RTH seems high, the actual number of messages required is still quite low, and could be reduced further by increasing the dimensionality of the surface, shortening the average path length across the surface.

Figure 8.6(f) shows that POUT_M and PIN_M follow much the same trend as the previous experiment because there are still some casts published to large implicit groups. As before, registry distribution reduces by two orders of magnitude the outgoing load on any single peer for any single cast, at the cost of a slight increase in incoming load.

Generally, SPICE is slightly fairer for the Zipfian cast distribution (Figures 8.6(b) and 8.6(d)). With no replication or distribution, TOUT_G is 0.89. More prominently, TIN_G is just 0.39. As the degree of distribution increases, TOUT_G only improves and as replication is added it falls to a very fair 0.30. As before, TIN_G increases slightly due to the increased number of packets transmitted around distributed tag registries. Replication keeps TIN_G fair despite this by dividing the load between replicas, reducing it to just 0.25.

8.5 Evaluation of SPICE Unloaded

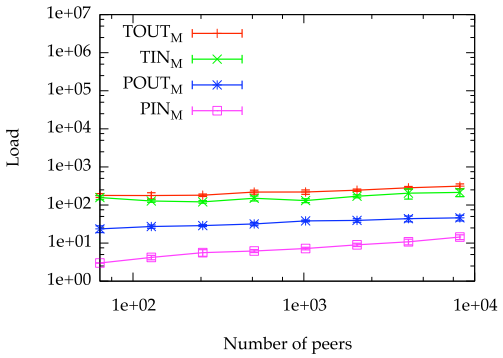
This section returns to analysing the SPICE Unloaded implementation using the same set of experiments testing peer, cast and data skew scalability. The default parameters are those shown in Table I.

8.5.1 Peer scalability

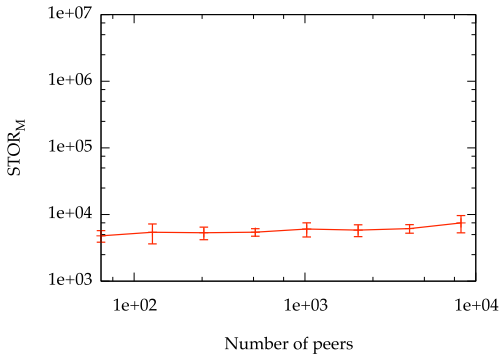
SPICE's registry distribution and replication techniques have a striking effect on the maximum loading and fairness across peers, both per cast and over many casts.

The frequency limit for these experiments (0.0125) does not result in particularly heavy replication; the six most frequently used tags in casts are only replicated to level 1. Yet instead of linearly soaring with the number of peers, as basic SPICE does, TOUT_M and POUT_M barely increase at all (Figure 8.7(a)). In particular, POUT_M is $O(s)$ (i.e., the storage limit), which also reduces the total outgoing load over many casts. The penalty for the reduction in outgoing load is an increase in incoming load. PIN_M rises slightly with the number of peers because messages are occasionally routed to the same neighbours when distributed registries are traversed. This also increases TIN_M significantly, though not excessively. Storage load is also low (Figure 8.7(b)) due to registry distribution.

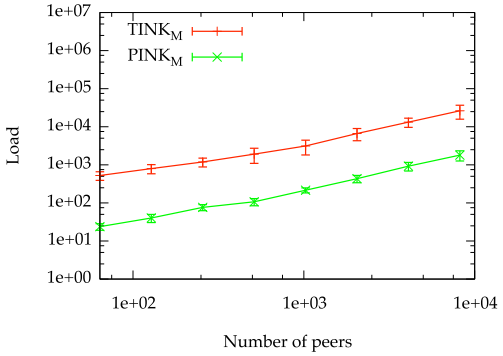
Network loading also improves, but not to the same extent as peer facet metrics. The lack of highly loaded peers means surrounding links are not as highly loaded, leading to TINK_M and PINK_M roughly half that of basic SPICE. It is not reduced more



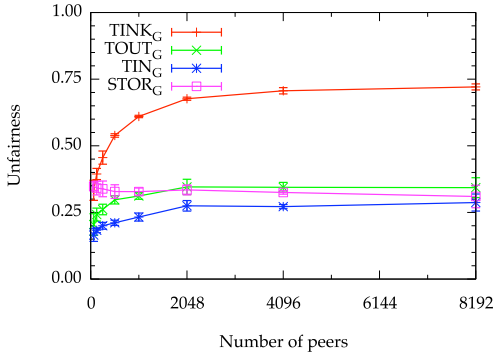
(a) Outgoing load is greatly reduced at the cost of a small increase in incoming load.



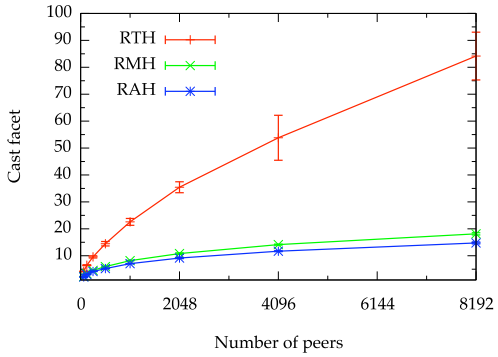
(b) Registry distribution effectively limits storage loading.



(c) Network load increases linearly, but is less than half that under basic SPICE.

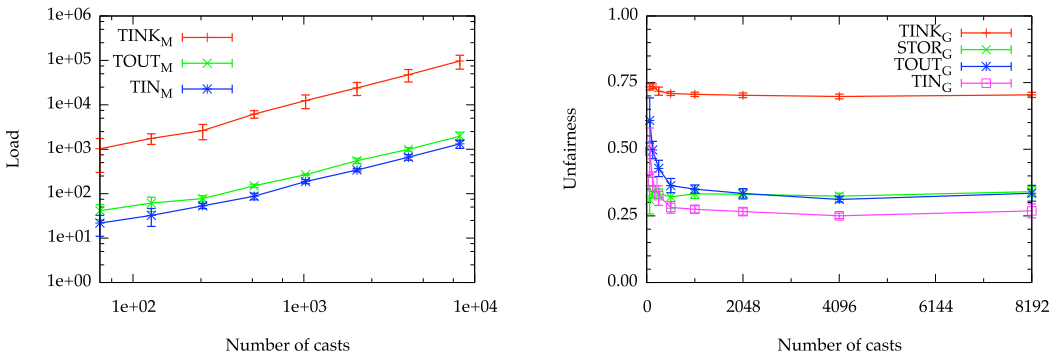


(d) All metrics are significantly fairer with distribution/replication.



(e) Hops to each consumer are comparable, but far more total hops are needed with distribution.

Figure 8.7: Peer scalability in the SPICE Unloaded implementation.



(a) TOUT_M is much reduced over basic SPICE.

(b) SPICE is consistently fair over many casts.

Figure 8.8: Cast scalability in the SPICE Unloaded implementation.

substantially because some links in the physical network are bottlenecks that are required to carry many messages between various parts of the ICE surface. Figure 8.7(d) confirms that TINK_G is still not particularly fair, though fairer than basic SPICE and the BROKER and CENTRALISED implementations. Peer facet metrics are all extremely fair.

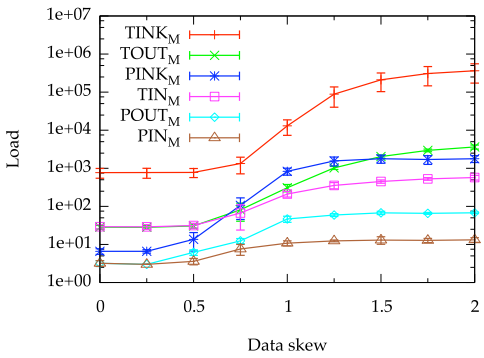
Another large difference between SPICE Unloaded and the basic implementation is the trend of RTH. Although the average and maximum hops from the publisher to each consumer are only slightly higher with distribution, the total number of hops is greatly increased. This is because registry distribution involves a significant amount of routing over the ICE surface when searching for registrations, in addition to the subsequent notifications.

8.5.2 Cast scalability

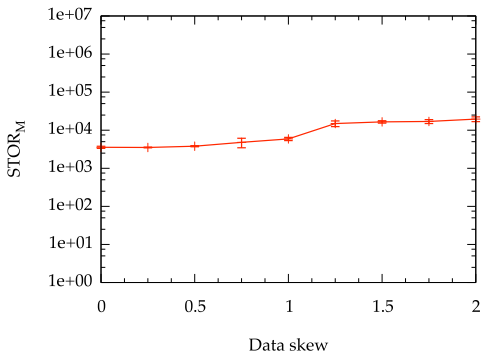
Figure 8.8(a) shows that TIN_M is slightly higher than under the basic SPICE implementation because more peers are involved in routing casts around distributed registries. TINK_M is slightly reduced because less heavily loaded peers tend to load the physical network less. However, TOUT_M is greatly reduced over the basic implementation due to distribution and replication. The loading is also fair over many casts (Figure 8.8(b)), indicating many peers are handling casts and notifying consumers, unlike the basic implementation.

8.5.3 Data skew

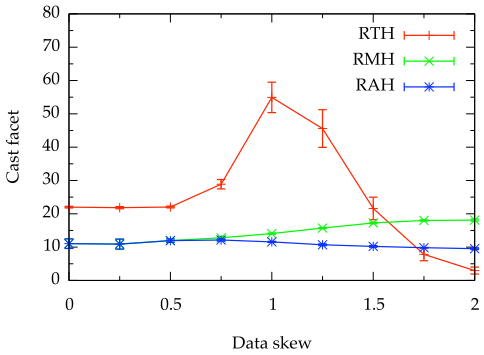
This experiment reveals the effect of data skew on loading in the SPICE Unloaded implementation, and is the most capable of accentuating the differences between basic SPICE and SPICE Unloaded.



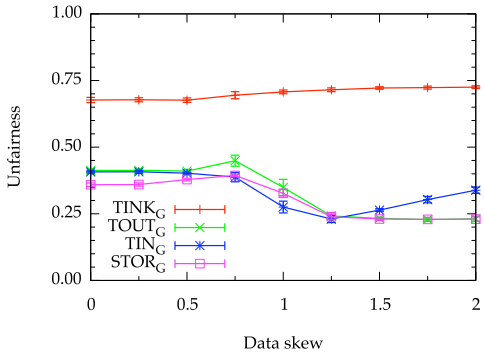
(a) Outgoing load is greatly reduced even with high data skew.



(b) Distribution is able to curtail extreme storage requirements of any single peer.



(c) RTH is highly dependent on the variety and sizes of implicit groups caused by data skew.



(d) Fairness improves with skew because the greater level of activity caused by distribution/repication is spread over very many peers.

Figure 8.9: Data skew in the SPICE Unloaded implementation.

Unlike the basic implementation which causes POUT_M to plateau quickly with increased peer and cast skew, SPICE Unloaded is able to retard this process enormously (Figure 8.9(a)). TOUT_M follows a similar trend. By employing registry distribution and replication, these metrics can be kept extremely low, especially when peer and cast skew are around the natural Zipfian value of 1.0. Even under extreme skew, these mechanisms result in two orders of magnitude less outgoing load on peers over any peer in the basic implementation. Similarly, STOR_M is an order of magnitude lower than basic SPICE because of distribution (Figure 8.9(b)).

The cost of this outgoing load reduction is an increase in incoming load. Basic SPICE shows a significantly different trend for TIN_M than SPICE Unloaded. In the former, the highest incoming load is caused by repeated casts to the same empty implicit groups. In the latter, however, many peers handle incoming messages as part of the distribution and replication mechanisms needed to effectively cope with such high peer and cast skew.

Figure 8.9(c) shows that RAH is slightly reduced with increased skew. This is

because registry replication allows casts to reach RPs (and hence consumers) more quickly. However, RMH is increased because a high level of distribution means more hops are needed to find all registrations. For very high or low skew, RTH in SPICE Unloaded is the same as the basic implementation. However, it is much higher when skew is Zipfian. This is due to a greater variety in the sizes of implicit groups that are selected by casts. Although some are very large, many are empty. At this skew, the popularity of some tags means some registries are highly distributed. Messages cast to implicit groups involving these tags are thus required to find all distributed registrations, but as the skew is not high enough to ensure many of these registrations are actually selected, some of the routing is wasted effort.

Overall fairness is much improved in SPICE Unloaded (Figure 8.9(d)). Interestingly, fairness actually improves for peer facet metrics with increased data skew. This is because the distribution and replication mechanisms are activated only once peers reach their limits. When skew is low, no registries are distributed, resulting in a higher variance of loading (though lower total loading). As registries become distributed and replicated, more peers are involved in storing data and participating in the delivery of casts. The overall amount of activity in the system increases because of this, but its highly distributed nature loads peers similarly.

8.6 Summary

This chapter has presented the design and evaluation of two complementary load distribution techniques for the SPICE IGM implementation: registry distribution; and registry replication (Contribution 10).

Distribution is designed to reduce storage and outgoing load on peers by storing registrations at peers near to the original rendezvous points. The technique is based on the hierarchical nature of the tesseral addressing scheme afforded by the ICE P2P substrate upon which SPICE is built.

Replication reduces incoming load and total outgoing load on rendezvous peers by making complete copies of registries at corresponding extents on the ICE surface. These replicas can be discovered by casts in the course of routing towards a rendezvous extent without any effort on the part of the rendezvous peers, due to the property of extent translation on the ICE surface.

At the cost of slightly increasing incoming load on peers, distribution significantly reduces POUT_M, TOUT_M and STOR_M. Replication is also capable of reducing TIN_M and TOUT_M. When judiciously combined, all forms of IGM loading can be greatly reduced and fairly distributed over peers.

This thesis has posited that a distributed structured P2P network supplemented by adaptive load distribution techniques can efficiently and fairly support implicit group

messaging, irrespective of scale or skew. This chapter has shown that under both realistic and extreme peer registration and cast skew, SPICE can very fairly distribute incoming, outgoing and storage loads over all peers without excessively loading any single peer. Furthermore, SPICE has been shown to scale very gracefully with an increasing number of casts and peers.

Chapter 9

A case study: special interest messaging

This chapter is a case study of using IGM as the delivery component of a collaborative research tool. The purpose of the chapter is to sketch how IGM could be used in a real system (Contribution 11), and to evaluate the proposed implementations against a real data source and real network topologies (Contribution 12).

9.1 Motivation

In early 2007, the web site Technorati [9] tracked more than 80 million discrete blogs. Thousands were highly technical, focussing on data mining and artificial intelligence, for example, and several thousand articles were posted in 2006 regarding these particular topics alone.

Already mainstream among typical Internet users, blogging is burgeoning in institutional and special interest domains. In the last few years, many institutions have started creating both client-facing and internal blogs for purposes ranging from branding and customer service to project management and team collaboration [152]. Some of the world's largest companies have employees that regularly write public articles, such as Robert Scoble [153] at Microsoft (until June 2006). IBM in 2005 requested its 320 000 employees to consider blogging on a regular basis [154], and Verizon uses blogs as a tool for internal knowledge management [155]. Many libraries and universities also use blogs as a means of communicating internally to staff and students [156]. Blogs have several features suited to institutional use: they allow every employee to voice an opinion; they act as a stimulus for dialogue; and they serve as a constantly evolving repository of ideas and pieces of information that are often lost or unheard in large organisations.

9.2 The research tool

There are many existing ways to connect blog authors and readers in institutional and special interest domains, e.g., RSS and search engines. However, as argued in Chapter 1, the serendipitous publisher-selected-described approach of IGM could be used to improve the experience of readers as publishing becomes more pervasive and specialised.

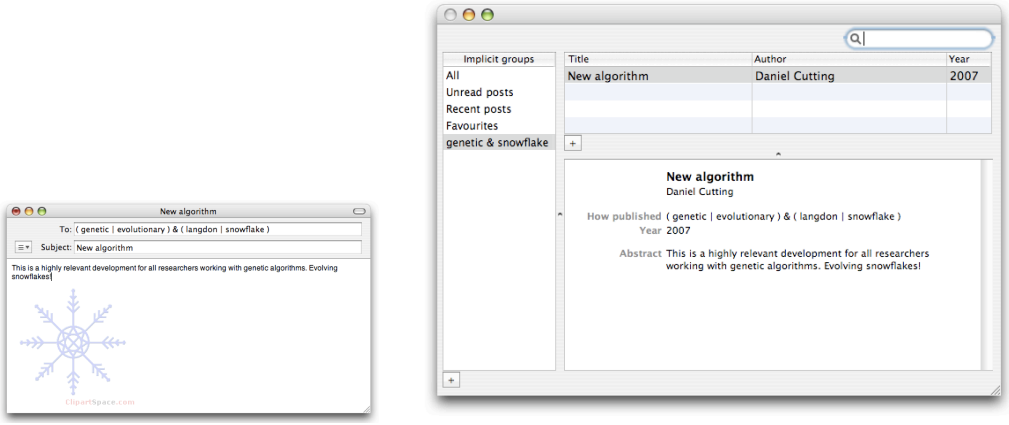
This section outlines an application specifically designed for researchers in computer science, though also applicable to business domains. It is inspired by the rise of institutional and special interest blogging and could be used within a research laboratory or over the Internet. Its primary functionality is to assist collaboration using IGM: researchers can keep a public journal that is automatically published to interested peers; appropriate collaborators for new work can be found by messaging implicit groups of people in a specific field; conference organisers can direct calls for papers to relevant audiences; literature searches can be constrained to peers with similar interests; and conversations can be held between like-minded colleagues in implicit group forums and chat rooms.

Consider in particular an implicit group blogging component with an interface similar to an email client. Authors are able to compose posts comprising rich text, images and attachments (Figure 9.1(a)). Instead of entering recipients' email addresses, however, the message is addressed to an implicit group by way of a target expression. Consumers are notified of incoming messages in the same way as new email or RSS feeds, with the additional advantage that messages can be automatically classified according to the target group (Figure 9.1(b)). If a reader finds the message interesting, they are able to either reply solely to the author via ordinary email, or "reply all" to the same implicit group. This latter option permits the possibility of ongoing exchanges between members of a group that is never explicitly reified.

The research tool client may also combine or integrate with a literature management component that records which papers a researcher is reading. In such a case, the tool itself could determine the interests of the researcher by observing their reading patterns. Common keywords could be extracted from the most recently read papers, and used to automatically register tags in the IGM network on their behalf. Registrations could be adjusted over time by the tool as the researcher's interests shift.

This notion could be extended further by interfacing with other components such as calendars and "to do" lists. The software could then act as a news ticker while the reader is browsing the world wide web, or as an adaptive chat room suited to the project the user is currently working on. Such *context-awareness* has been explored at length in the form of context modelling [157, 158] and toolkits [159, 160].

Synchronising and interfacing with portable devices would also allow the software



(a) Publishing a new article.

(b) Browsing received articles.

Figure 9.1: A GUI for publishing and reading articles via IGM.

to recognise what has already been seen by a researcher (e.g., when commuting with a PDA) and tailor the message display. By extending the context-awareness capabilities of the client software and adjusting registrations accordingly, IGM becomes a vehicle for *context-based messaging*, where users receive messages based not just on their interests but also their current activities.

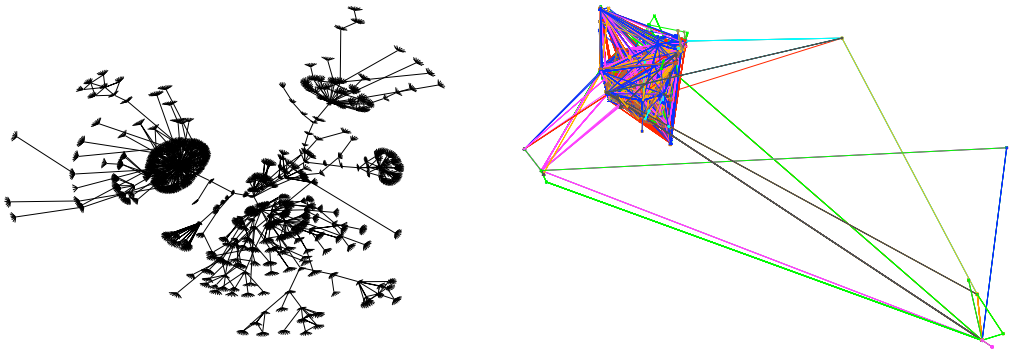
Such a tool would be useful in domains beyond academic research. For instance, a physician at a hospital having trouble diagnosing a patient could send messages to the implicit group of colleagues with expertise in the exhibited symptoms.

The distribution of articles in this application is achieved via an IGM component. Chapters 5–8 describe three possible implementations: the CENTRALISED client/server, decentralised BROKER backbone, and the distributed, structured P2P implementation called SPICE. Section 9.4 comparatively evaluates these implementations using real data and network topologies appropriate to a computer science research tool.

9.2.1 Related work

Several similar tools designed to aid researchers have been proposed and implemented, including literature management applications [161, 162, 163, 164], social networking sites [108, 165] and mailing lists [166]. Those that offer collaborative features generally fall into the consumer-selected category, offering forums and web pages for browsing by researchers.

Some P2P-based tools also exist. Bibster [167] is a P2P network for sharing and finding bibliographic information, that offers semantic searches (such as finding papers discussing a specific topic). Often these types of networks focus on building networks mirroring the actual small-world relationships between researchers. In Scien-



(a) Laboratory topology. Peers are randomly connected via LANs (with latencies of 1–5ms).

(b) Worldwide topology. Peers are randomly connected via broadband links (Table 9.1).

Figure 9.2: The physical network topologies used in the evaluation scenarios.

tific Collaboration Networks [71], groups are formed by collaborators explicitly inviting others to join, noting that many researchers collaborate with few others while some work with many. Objects (such as datasets or articles) are replicated within a group and queries are flooded between members of appropriate groups until objects are found. A similar approach is taken in Chirita et al [72] where paper coauthorships form the basis of how peers are topologically arranged.

Conversely, an IGM-based research tool seeks to reach undiscovered implicit groups of researchers, more closely akin to Khambatti's interest-based communities [49]. However, that work is based on unstructured P2P networks which, due to their inherent "best effort" nature, cannot make strong guarantees about delivering to all group members in the way IGM requires. This precludes features such as IGM chat, where such a condition is necessary for the continuity of a conversation between peers.

9.3 Experimental design

This section details the experimental design for evaluating the IGM component of the research tool. The evaluation is a comparative analysis of the various implementations, driven by OMNeT++/INET simulations. All experiments are run five times with different random seeds and peer/cast sets. The error bars on all figures represent a 95% confidence interval of the standard error. The default parameters used are those in Table I.

9.3.1 Physical network topologies

The IGM implementations are based on connectionless datagram protocols, and this evaluation assumes network links are reliable (meaning no packets are corrupted, lost or delivered out of order). In order to clearly isolate the performance of the imple-

Table 9.1: Broadband links in the Worldwide topology.

Technology	% of peers	Latency (ms)
LAN	2	1–5
DSL	66	5–70
Cable	30	5–40
Satellite	2	100–220

mentations, the links are also assumed to have no congestion or cross-traffic. However, realistic link latency is modelled in order to explore the time taken to deliver messages.

This evaluation is concerned with router-level statistics for a network typical of large institutions and Internet-based special interest groups. Two scenarios are specifically considered: a *Laboratory* topology which is a regional homogenous network such as may be operated by a university or research laboratory; and a *Worldwide* topology comprising national/continental links and peers connected via broadband links. The physical topologies used are derived by Liljenstam et al [168] from many sources of real router-level network measurements.

The Laboratory topology (Figure 9.2(a)) represents a national research laboratory and consists of 447 routers and 612 links. Link latency is 5ms, 3ms or 1ms depending on the link type. Peers are connected to random routers with LAN links which have a latency of 1–5ms. The Worldwide topology (Figure 9.2(b)) covers a much greater area and is built around a core of six major U.S. ISP networks with additional autonomous systems in Denmark, Taiwan, Singapore and Indonesia. The topology comprises 3279 routers and 11 952 links. Link latency between routers is determined largely by their physical distance, and ranges from a few milliseconds to 144ms for the longest transcontinental links. Broadband connectivity to users is increasingly ubiquitous (two-thirds of all users in 2006[1] [5]), and such peers tend to be the major generators of content [6]. Hence in these experiments, peers are randomly connected to routers with broadband links. The proportions of broadband links are based on figures published by the OECD from 1999–2004 extrapolated forward to 2006 (Table 9.1). In the CENTRALISED and BROKER implementations, the server and brokers are always connected to the network core via links with a latency of 1ms.

9.3.2 Data source

Implicit group messaging is a novel approach to content distribution and hence there are no directly applicable sources of real data. However, similar systems do exist,

[1] Based on forecasts from 2004 data.

including topic-based publish/subscribe systems and even ordinary web search engines. Search engines are a particularly good source of data as they are selection-described like IGM, and generally support a language similar to the tag-based modelling language used in this paper. Conceptually, a search engine indexes objects (documents) by associated keywords, and search queries describe "implicit groups" of these objects. The general availability and similarity of search engine data allows creation of suitable peer profiles and casts.

The data in these experiments come from a web site called the Collection of Computer Science Bibliographies (CCSB) [169]. The site is essentially a search engine that allows users to query a collection of almost 3.5 million computer science papers covering fields as diverse as artificial intelligence, computational mathematics and typesetting.

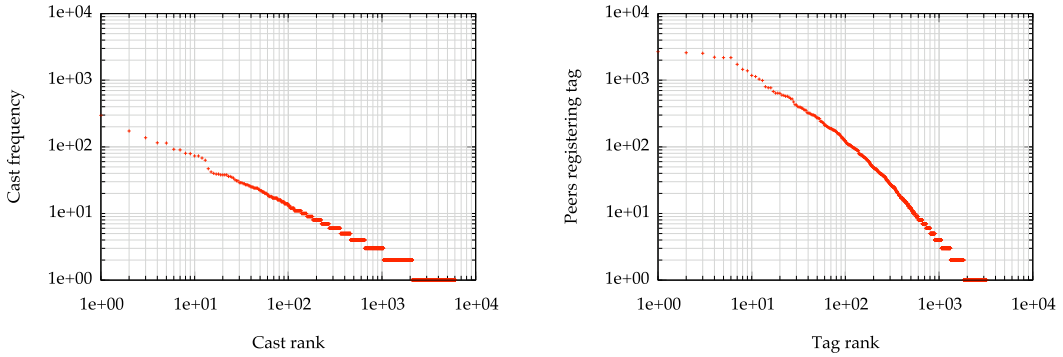
The site is divided into several categories. Within each category is a set of bibliographies, maintained by separate computer scientists and librarians, each pertaining to a specific institution, conference, journal or field. Users can search over the entire collection of papers, or just particular categories or bibliographies. The bibliographies are stored in the plain text BIBTEX format commonly used to store and format metadata for referencing articles in scientific papers. The main descriptive fields for each paper are title, authors, abstract, journal/booktitle, annotations, notes, keywords and topics.

The data used in this case study are chosen from a single category within the CCSB corpus: artificial intelligence (AI). This particular category is chosen because it suits a special interest research group, and has the largest available search query log, ≈16 000 raw queries, spanning the period 4th September 2005 to 25th July 2006. The corresponding BIBTEX data are a snapshot of the bulk of the content indexed on 7th August 2006 (≈92 000 paper references or 72% of the AI category).

Note that it is important to obtain the queries *and* the index to which the queries pertain. It is not sufficient to obtain just one and generate the other from the same distribution, since it is unclear that the two will correlate. For instance, many people may search for very rare tags as well as very common tags. The popularity of a search query is not necessarily related to the popularity of the indexed data.

The CCSB data must be converted to sets of casts and peer registrations in order to be used in the simulations. The raw search queries are first normalised by treating each keyword as a tag in a conjunctive expression. This results in ≈15 000 total casts, ≈6000 of which are unique. This heavy-tailed cast distribution (Figure 9.3(a)) is used as the basis for constructing random sets of casts for use in the simulations. Half of the distinct casts use a single tag to select implicit groups. One third are conjunctions of two tags, and the remaining sixth combine three tags.

Generating the peer registrations is more complex, since there is no trivial way



(a) CCSB casts are Zipfian with exponent ≈0.7.

(b) CCSB peer registrations are approximately Zipfian.

Figure 9.3: CCSB cast and peer registration distributions.

Listing 13 Algorithm to generate a peer registration from a bibliography.

```
def generate_registration(num_papers,num_tags)
   bag = Hash.new(0) # Every value defaults to 0.
   bib = get_random_bibliography
   num_papers.times do
      # Tags are harvested from the title, abstract, journal, booktitle
      # annote, note, topics and keywords BibTeX fields.
      bib.get_random_paper.tags.each do |tag|
         bag[tag] += 1
      end
   end
   # Rank the bag from most to least common.
   ranked_bag = bag.sort { |x,y| y[1] <=> x[1] }
   # Take the most common tags.
   top_bag = ranked_bag[0...num_tags]
   # Return the tags from the bag (sans the counter).
   top_bag.map { |x| x[0] }
end
```

to convert the BIBTEX bibliographies. In a real system, the research tool could observe the reading patterns of the researcher (as outlined in Section 9.2), and extract the most commonly appearing keywords to be used for registrations. Because the CCSB bibliographies are structured according to themes, fields or conferences, they are naturally grouped according to common threads of interest. Hence, a registration may reasonably be defined by the most common keywords from the descriptive fields of a set of papers chosen from a single bibliography. Specifically in these experiments, each peer is assigned 30 random entries from a random AI bibliography representing those papers most recently read by the researcher. The 18 most common keywords appearing in the papers' metadata determine the researcher's registration tags. Very common stop words are omitted from the peer registrations and casts, using an augmented version of the Glasgow stop list [170]. 18 keywords are chosen because the registration is compact enough to prevent too much similarity between peers while allowing a considerable range of interests. Furthermore the results in this chapter may be directly compared to those in previous chapters. Listing 13 presents the algorithm and Figure 9.3(b) shows the approximately Zipfian distribution of tags registered by a sample of 4096 peers applying this strategy. Although roughly Zipfian, the flat head reveals a greater number of very common tags than would be expected by a purely Zipfian distribution. This mirrors the distribution of words used by Shakespeare seen in Section 3.4. Note that these experiments demonstrate load distribution when peer registrations and casts are skewed differently, unlike the previous chapters in which they were bound.

Table 9.2 is a sample of some of the casts and peer registrations created from the CCSB data.

9.4 Evaluation

This section presents the results of simulations comparing the IGM implementations of Chapters 5–8. These implementations form the distribution component of the research tool used in the Laboratory and Worldwide scenarios. The purpose of the evaluation is to explore what impact the choice of implementation has on the consumer, publisher and network facets in a realistic scenario. Due to simulator limits, it is only possible to measure the SPICE implementation under the Worldwide scenario to a maximum of 4096 peers, and the BROKER implementation to 6144 peers.

9.4.1 Peer scaling

The number of early adopters of a laboratory or worldwide research tool incorporating special interest messaging could be expected to be quite low, increasing with popularity. These experiments show how the IGM implementations scale as the number of

Table 9.2: Sample CCSB peer registrations and casts.

Peer registrations
garis evolutionary evolvable papers late using algorithm network fitness genetic breaking gecco wu computation dinerstein search algorithms strategies
belief knowledge artificial action preliminary nonmonotonic systems base management intelligence proc minker formal theories data distributed logic theory
mining genetic algorithm graduate simulation design data systems parallel optimization programming problem using evolution student evolutionary algorithms immune

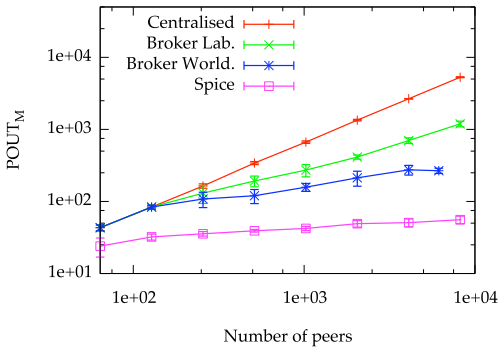
Casts
advances & automobiles applications & real & world langdon

peers increases from a small core to several thousand.

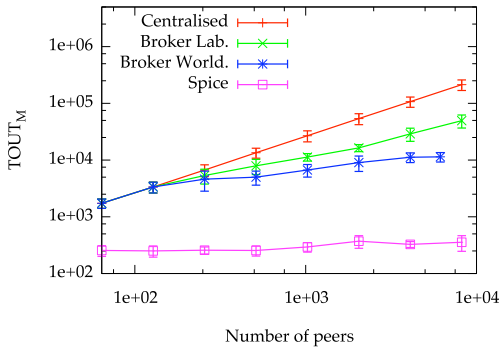
In the CENTRALISED and SPICE implementations, peer and cast facet metrics are the same for both scenarios because the overlays are formed independently of the physical network topology. However, the BROKER implementation is dependent on the physical topology, since it is built around a minimal spanning tree backbone. The following analysis notes the differences when they are significant, otherwise results are taken from the Laboratory scenario.

The load on peers is of foremost interest. Figure 9.4(a) shows the maximum per cast outgoing load for any single peer (POUT_M). Clearly the CENTRALISED implementation has a high POUT_M that increases linearly with the number of peers, since the server peer forwards each cast to all group members. The BROKER implementation follows a similar trend, though at a slower rate, because each broker assumes a fraction of the overall workload required by a single server. BROKER's POUT_M disparity between the two scenarios is due to a small number of brokers in the Laboratory scenario handling more clients than average. Recall that clients register with the closest broker; the clustered nature of the Laboratory topology and consequent backbone (Figures 9.2(a) and 6.2) can lead to many clients selecting the same broker. However, when brokers have approximately equal numbers of clients (as in the Worldwide scenario), POUT_M is generally lower, as expected. Note that this imbalance of broker clients also increases the maximum total outgoing load (TOUT_M) in the Laboratory scenario (Figure 9.4(b)).

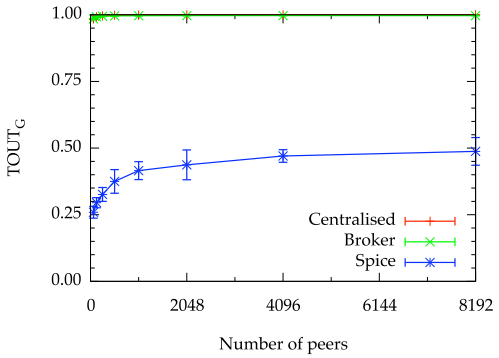
In SPICE, POUT_M increases with the number of peers but at a very slow rate,



(a) Maximum per cast outgoing load.



(b) Maximum total outgoing load.



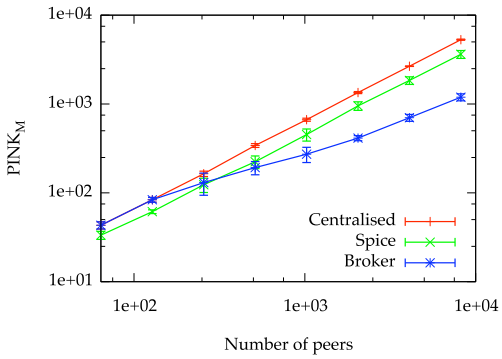
(c) Total outgoing load fairness. CENTRALISED and BROKER are almost identical.

Figure 9.4: Peer facet under increasing numbers of peers.

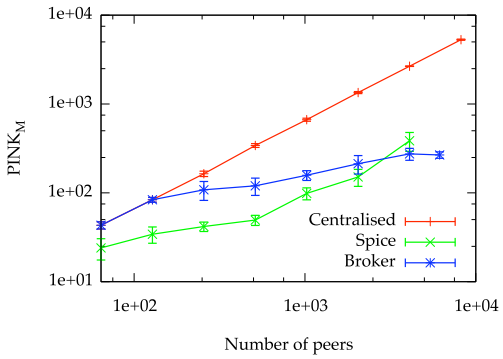
mainly constrained to the storage limit by registry distribution. The same phenomenon is clearly evident in Figure 9.4(b) which shows the maximum total outgoing load on any single peer after all 1024 casts (TOUT_M). SPICE is again extremely low in comparison to other implementations. Its consistency indicates that outgoing load is well spread not only for individual casts but also over time, despite many casts selecting the same groups. Figure 9.4(c) agrees with this analysis, showing that the total outgoing load on peers in SPICE is remarkably fair (TOUT_G). This is due to three mechanisms: the hashing of tags onto the ICE surface (essentially creating a distinct "server" for each tag); the distribution of registries to peers around the RP; and the replication of whole registries to other parts of the surface.

SPICE is undoubtedly the fairest implementation in terms of the peer facet. Outgoing load is very low and fair, both per cast and over many casts.

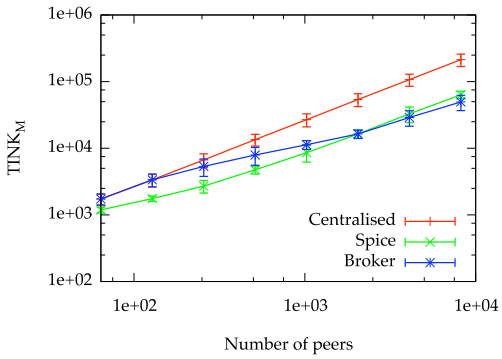
Figure 9.5 presents the network facet metrics under an increasing number of peers in the Laboratory and Worldwide scenarios. The maximum link loading is strikingly different between the two scenarios, which is mainly caused by their differing scales. The Worldwide topology has an order of magnitude more routers and links, yet both are supporting overlays of the same size. Secondly, as Figures 9.2(a) and 9.2(b) show,



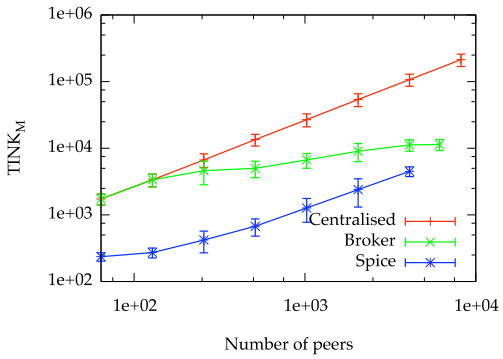
(a) Maximum per cast link load (Laboratory).



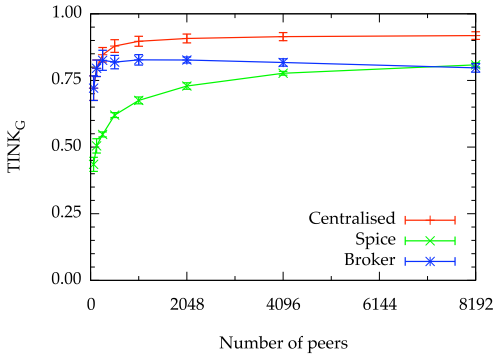
(b) Maximum per cast link load (Worldwide).



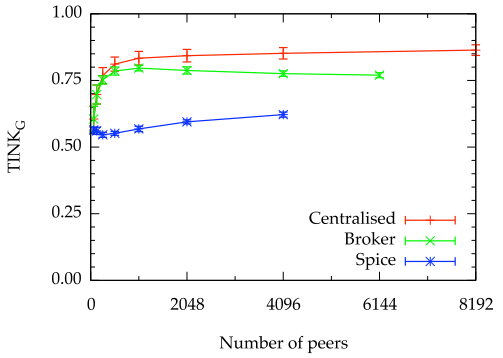
(c) Maximum total link load (Laboratory).



(d) Maximum total link load (Worldwide).



(e) Total link load fairness (Laboratory).



(f) Total link load fairness (Worldwide).

Figure 9.5: Network facet under increasing numbers of peers.

the Laboratory topology is partitioned into a few major components connected by individual links while the Worldwide topology is densely concentrated in the U.S.A., with several redundant links to other areas of the network. Clearly the Laboratory topology will impose greater loads on those links connecting partitions, as messages between any peers in the partitions must be funnelled through them.

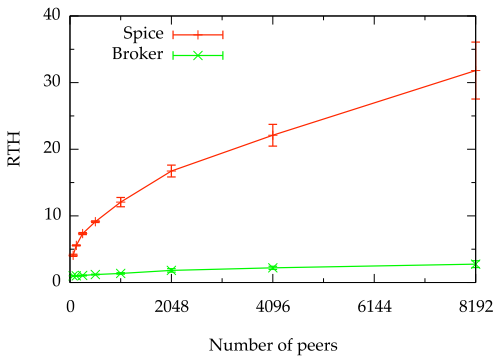
In both scenarios, the CENTRALISED implementation most heavily loads the link from the server to the network, as this must carry a copy of a cast for each selected consumer. Thus, PINK$_M$ (Figures 9.5(a) and 9.5(b)) precisely follows the maximum implicit group size that is selected. Due to its minimum spanning tree backbone, the BROKER implementation will generally minimise the number of packets traversing the same link per cast, especially as the number of peers increases. It is encouraging to see that SPICE is relatively close for networks of this scale, primarily due to the distribution of registries preventing any single peer forwarding too many casts to consumers.

The same trends are seen for total link load (TINK$_M$) in Figures 9.5(c) and 9.5(d). As the number of peers increases, the BROKER implementation's backbone again results in the most efficient use of network links. However, SPICE performs very well in comparison due to its use of multiple RPs and registry replication technique which directs casts to different regions of the network for each cast.

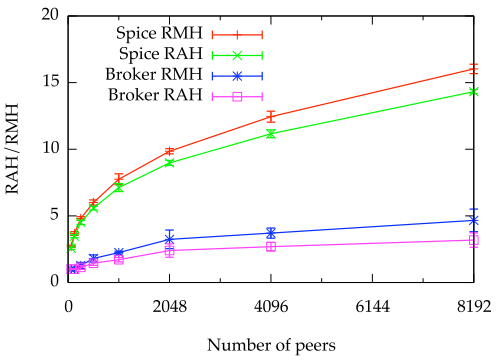
Figure 9.5(e) shows that no implementation is especially fair in the Laboratory scenario. Again, this is due to the partitioned nature of the topology and the higher density of peers to links. Note, however, that BROKER grows increasingly fair with the number of peers, unlike the other implementations. This benefit is caused by the growing backbone better approximating the physical network and attenuating the number of duplicate packets. The Worldwide topology is generally fairer (Figure 9.5(f)), particularly for SPICE. Again, BROKER's fairness increases with the number of peers.

In general, BROKER is the gentlest implementation with respect to loading the physical network, although ICE's amortised routing enables SPICE to perform comparably, especially over time.

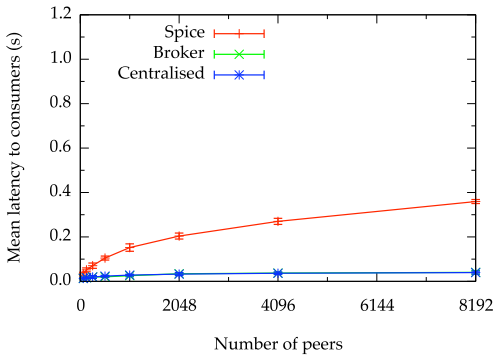
The cast facet examines how well individual casts are delivered to consumers from publishers. Figure 9.6(a) shows the ratio of the total number of overlay hops needed to reach every group member (RTH) as compared to the CENTRALISED implementation (which, by definition, always has an RTH of 1). SPICE has a somewhat high RTH because more peers are involved in delivering a cast by routing it across the ICE surface. The BROKER implementation's low RTH is due to the relatively low diameter backbone that distributes single copies of casts. The routing Bloom Filters between brokers also serve to limit cast propagation. Figure 9.6(b) presents the average and maximum ratio of overlay hops needed to reach a consumer from the publisher of a cast (RAH/RMH). The amortised routing algorithm used by SPICE increases the num-



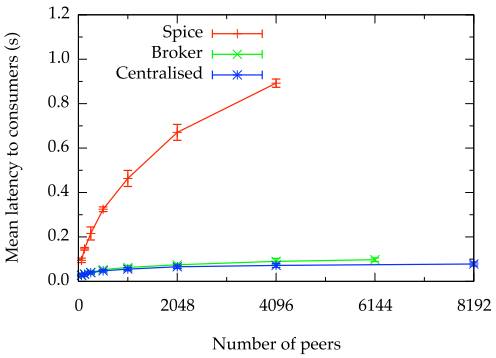
(a) Ratio of total hops per cast.



(b) Ratio of average/maximum hops per cast.



(c) Latency to each consumer (Laboratory). CENTRALISED and BROKER are almost identical.



(d) Latency to each consumer (Worldwide).

Figure 9.6: Cast facet under increasing numbers of peers.

ber of hops, but judicious branching somewhat diminishes the difference to BROKER.

Finally, Figures 9.6(c) and 9.6(d) show the actual latency of the IGM implementations. BROKER is able to keep apace of CENTRALISED despite significant additional overlay hops because the backbone forwards casts to consumers as they are encountered in the backbone, rather than first traversing a potentially distant server. The higher latency for SPICE is due to the relatively large number of overlay hops required to traverse the surface as the number of peers increase. Limiting the branch factor would reduce this latency by routing casts more directly to each consumer rather than following circuitous amortised routes, at the cost of additional network and peer load.

SPICE is more susceptible than other implementations to the high latency links in the Worldwide topology, as they may need to be traversed several times during a route. Hence its average latency is approximately eight times that of CENTRALISED in the Laboratory scenario, but eleven times in the Worldwide scenario.

9.4.2 Loading over time

The research tool is intended to operate over long periods of time as a persistent service for an institution or special interest group. Hence it is important to investigate how the IGM implementations load peers and the physical network over time. This is achieved by fixing the number of peers to 4096 and varying the number of casts (while maintaining the same frequency of casts). These experiments focus on the Laboratory scenario.

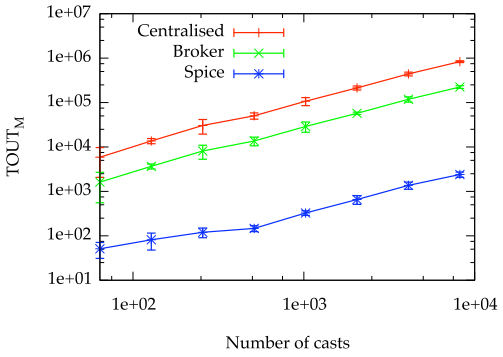
Figures 9.7(a) and 9.7(b) show the maximum total outgoing peer (TOUT_M) and network (TINK_M) loads. Since every cast in the CENTRALISED implementation goes through the server, TOUT_M and TINK_M are equal to the sum of the sizes of all implicit groups selected by all casts. By contrast, BROKER and particularly SPICE employ several different peers to handle each cast, leading to lower total loads. Indeed, SPICE's TOUT_M is two orders of magnitude lower than CENTRALISED or BROKER and TINK_M is similar to the network-optimised BROKER implementation. This demonstrates that the amortised routing algorithm of ICE can effectively reduce network loading. Although SPICE shows great fairness for TOUT (Figure 9.7(c)), no implementation is able to fairly distribute total link load in the dense Laboratory scenario (Figure 9.7(d)).

9.4.3 Group size scaling

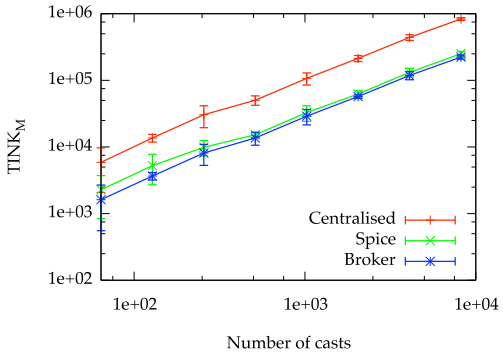
This experiment is designed to test how the implementations scale with the size of the implicit groups that need to be messaged. In a real system, many groups would have few or no members and some would have very many members.

Although it is likely that many participants will express a common tag, it is unlikely in practice that any cast would need to reach *every* member of an institution. Indeed the purpose of implicit group messaging is diluted if it is used to broadcast rather than discriminate. Consequently in this experiment, the group sizes range from 0.1–50% of the network, which is substantial enough to demonstrate network effects, but not so large as to negate the need for IGM.

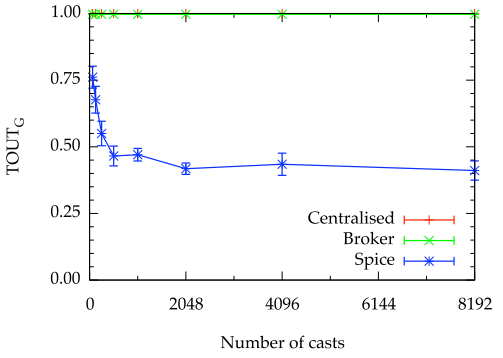
Large groups are of concern for the CENTRALISED implementation since the server must send a message to each member (high POUT_M), saturating the links around the server with myriad redundant copies of each cast (high PINK_M). The BROKER implementation aims to improve this situation by spreading POUT over more servers. SPICE is designed to take this to an extreme, employing enough outgoing "servers" as necessary to fairly deliver the message, using registry distribution. The advantage of this approach is evident in Figure 9.8(a), which shows that the SPICE network produces a maximum per cast outgoing load on any single peer (POUT_M) barely exceeding 32 (the storage limit used for these experiments), even for casts selecting every second peer.



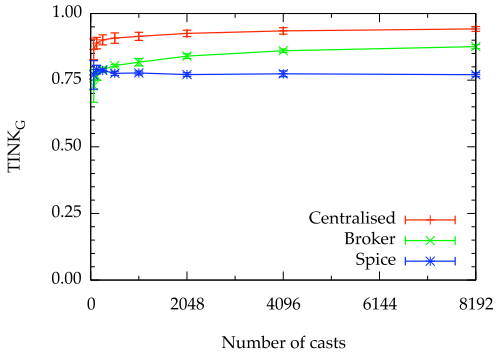
(a) Maximum total outgoing load.



(b) Maximum total link load.

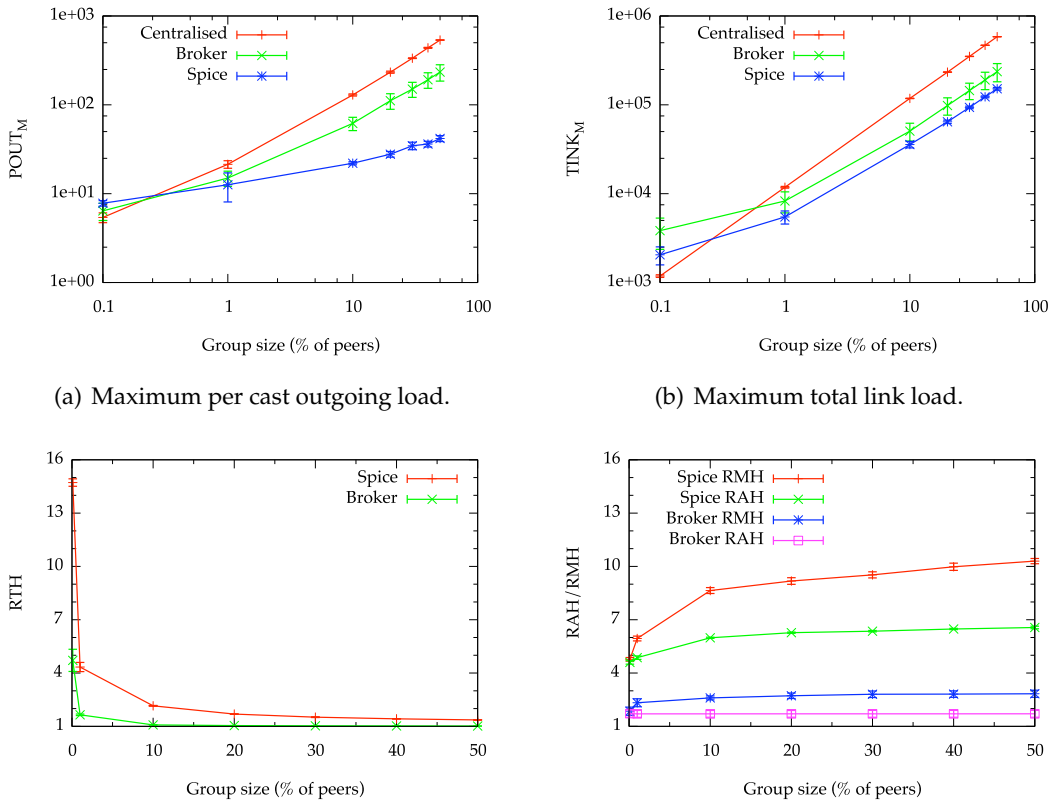


(c) Total outgoing load fairness. CENTRALISED and BROKER are almost identical.



(d) Total link load fairness.

Figure 9.7: Peer and network facets under increasing numbers of casts.



(a) Maximum per cast outgoing load.

(b) Maximum total link load.

(c) Ratio of total hops per cast.

(d) Ratio of average/maximum hops per cast.

Figure 9.8: Peer, network and cast facets with increasing group size.

Figure 9.8(b) presents the maximum link load after all casts have been delivered. Again, the CENTRALISED implementation heavily loads the same outgoing server link. The other implementations spread the load to several major links in the physical network. SPICE actually imposes a lower load than BROKER because it is not constrained to forwarding separate casts along the same backbone links, instead routing each cast to different parts of the ICE surface, and physical network.

As group size increases, RTH generally converges towards unity (Figure 9.8(c)). This is because the overheads of BROKER and SPICE (backbone propagation, and routing to RPs) become smaller relative to the number of messages needed to reach every member of larger groups. SPICE takes longer to converge because of the larger initial routing overhead, and the additional hops needed to discover summaries that have been distributed for the registries of common tags. Figure 9.8(d) shows that the average and maximum overlay hops to each consumer increase slightly in SPICE, but remain almost constant in BROKER. In SPICE, summary distribution of common tags increases the path length from publisher to consumer. In BROKER, there is only a relatively small backbone segment that must be traversed in the worst case, regardless of how many consumers are selected.

9.5 Discussion

Overall, each IGM implementation has strengths and weaknesses. The CENTRALISED implementation is easy to implement and deploy and would suffice for small institutions. The BROKER implementation is also reasonably straightforward to implement, though more work is required for deployment and administration, as some machines must be configured as brokers. In latency-critical domains which require scalability, this implementation is the best option. Even on a global network it can deliver a cast to thousands of group members in approximately 100 milliseconds. It also incurs generally low link load on the physical topology, except when many large groups are selected, causing abnormal loads on the links supporting the backbone. SPICE shares properties of both of these systems. Like the CENTRALISED implementation, administrative overhead is low, since the network is self-configuring. Like the BROKER implementation, network link load is quite low on a per cast basis, and actually the lowest over many casts to large implicit groups. Finally, participating peers do not need to dedicate excessive network resources to the system. The delivery latency of SPICE may preclude its use in some demanding global applications, but it is acceptable for the research tool presented in this chapter. In the Laboratory scenario, the latency would certainly permit the possibility of real-time implicit group chats.

9.6 Summary

With blogging finding increasing usage in institutions for team collaboration and internal communication, and hundreds of special interest blogs related to diverse niche topics also appearing, novel distribution mechanisms from publishers to consumers are desirable. This chapter has proposed using implicit group messaging, and outlined a context-aware research tool integrating special interest messaging (Contribution 11).

The focus of the chapter has been a detailed comparative simulation of the three IGM implementations driven by a real data source and two actual network topologies (Contribution 12). The results have shown that for Internet applications requiring high performance chat capabilities, the BROKER implementation's low latency is preferable. However, for many of the collaborative features desired of the research tool, SPICE offers exceptionally efficient and load-balanced special interest messaging.

Chapter 10

Conclusions and future directions

This thesis has introduced a new messaging approach called implicit group messaging (IGM), motivated by the ever-increasing role of Internet-based communication and the overwhelming amount of content appearing. IGM is based on the idea that publishers should specify the consumer demographic for each article of content such that messages are delivered from publishers to implicit groups of consumers based upon their characteristics, rather than their name. Implicit groups are potentially unlimited in size, undefined until the moment of publication, and may cross-cut the population of consumers along any number of dimensions.

This thesis has posited that a distributed structured P2P network supplemented by adaptive load distribution techniques can efficiently and fairly support implicit group messaging, irrespective of scale or skew. The SPICE implementation, when augmented with load distribution techniques enabled by the ICE substrate, very fairly distributes incoming, outgoing and storage loads over all peers without excessively loading any single peer. Furthermore, SPICE has been shown to scale very gracefully with an increasing number of casts and peers, and to operate well in situations where IGM could be employed.

Through a number of contributions, this thesis has advanced the state of the art in load distribution in P2P networks and the analysis of load distribution in distributed systems. This chapter summarises these contributions (Section 10.1), and concludes with an overview of directions for future research (Section 10.2) and applications (Section 10.3).

10.1 Contributions

Chapter 2 has classified and discussed existing Internet messaging techniques according to whether they are publisher- or consumer-selected, and selection-named or -described (Contribution 1). It has followed with a review of load distribution techniques in similar systems (Contribution 2).

Chapter 3 has formally specified implicit groups and implicit group messaging (Contribution 3) and defined a concrete tag-based modelling language (Contribution 4). The chapter has also described the types of implicit groups expected and attendant loading problems caused by frequent casts to large implicit groups (Contribution 5).

Chapter 4 has defined a framework for analysing implementations of IGM systems according to peer, network and cast facets (Contribution 6). Fairness across peers is measured with the Gini coefficient, a measure commonly used in other fields to quantify inequality.

Three IGM implementations have been presented (Contribution 7): a baseline CENTRALISED implementation (Chapter 5); a decentralised BROKER implementation optimised to minimise network load and latency (Chapter 6); and a P2P design called SPICE (Chapters 7 and 8).

The implementations have been evaluated against the analysis framework in a set of common experiments that test peer, cast and data skew scalability (Contribution 8). The CENTRALISED implementation does not scale naturally with the number of peers or the number of casts, and is particularly poor at handling casts selecting large implicit groups. Decentralising the responsibility for forwarding casts to consumers over several peers in the BROKER implementation reduces the maximum outgoing load on any single peer but does not address total incoming load, or fairness generally. SPICE in its basic form has been shown to perform comparably to CENTRALISED.

The tesseral addressing and amortised routing features contributed by the ICE P2P substrate (Contribution 9) enable the SPICE implementation to employ very effective load distribution techniques (Contribution 10). Registry distribution reduces storage and outgoing load on peers by storing registrations at peers near to the original rendezvous points. Registry replication reduces incoming load and total outgoing load on rendezvous peers by making complete copies of registries at corresponding extents on the ICE surface. SPICE very fairly distributes incoming, outgoing and storage loads over all peers without excessively loading any single peer. SPICE has also been shown to scale well under an increasing number of casts and peers.

Chapter 9 has presented a case study of how IGM could be effectively used as the basis of a communication tool for implicit groups of researchers working in specialised domains (Contribution 11). The chapter has also corroborated the evaluation of the implementations using a real data source and actual network topologies (Contribution 12). SPICE has been shown to incur a relatively high delivery latency, but is capable of low and fair network and peer loading without the need for specialised peers.

10.2 Future research

This thesis opens several avenues for additional work relating to implicit group messaging, including novel application domains, advanced modelling languages, new overlay designs and extensions to the presented implementations.

10.2.1 Fairness

Using the Gini coefficient as a measure of fairness implies that it is desirable to balance load uniformly over the network. This is not always the case. For example, some P2P systems make use of "superpeers" that offer additional services to the system or act as proxies for less capable peers. While this thesis made the reasonable assumption that every peer is approximately equal in the types of domains considered, it may be desirable in some cases to design systems for very heterogeneous networks. A measure such as the Gini coefficient may still be applicable if measured over the ratios of load to ability rather than directly to load.

10.2.2 Modelling languages

Chapter 3 specifies IGM in terms of an arbitrary modelling language, allowing different domains to use the most applicable language. This thesis examines only one such model: tagging.

Although tagging has sufficient expressiveness to support a range of applications, distinct communities within a system are unlikely to share a common vocabulary. For instance, technical jargon is now so specialised to niche domains that interdisciplinary research can be baffling. Hence it may be beneficial to extend tagging with a way for participants to express relationships between tags that the system can use during casting.

Such a scheme could allow *tag rules* modelled on thesaurus operators such as: UF (Use For), USE, RT (Related Term), NT (Narrower Term) and BT (Broader Term). Rules could be created by any participant at any time, and registered with servers or RPs. When casts are published, these rules could cause target expressions to be widened, for example. To prevent abuse, rules could be weighted according to how many participants have expressed them, and only applied if they exceed a threshold.

10.2.3 Malicious publishers

There are some immediately evident open problems with implicit group messaging perhaps foremost of which is unsolicited and unwanted messages, or *spam* as it is popularly known.

In fact, spam is a problem for all publisher-directed messaging, not just IGM. The most apparent and obnoxious example is email spam, where a publisher indiscrimi-

nately pushes a message to millions of email addresses in the hope of reaching a few interested consumers. This shotgun approach to marketing relies on volume, since there is no obvious way to know ahead of time whether or not a recipient is interested in the message. Indeed, one reason spam is so irritating is that it is high volume and of virtually no interest to the average consumer.

It could be countered that IGM, unlike email which is selection-named, allows advertisements to be far more targeted to consumers who may genuinely be interested. It could even be argued that as IGM is "opt-in" and *designed* to deliver content to interested consumers from many unknown publishers, the amount of *perceived* spam may be low.

Clearly, however, this would not account for all cases. Spam would likely still be a legitimate and irritating concern to some degree. Many of the numerous anti-spam techniques developed in the email space, such as Bayesian filtering [171], could be easily adapted to work with IGM clients.

While this protects consumers, it does not protect the network itself, which may be required to deliver frequent messages to very large, general implicit groups. In such cases, servers, brokers or rendezvous peers determining an excessive number of selected peers may choose to silently drop messages, or reply with an error message. In SPICE, for example, peers comprising a large distributed registry may choose not to continue routing a cast if its target expression selects just the tag in that registry.

A more subtle problem than generic spamming is *trolling*: the act of deliberately agitating consumers. Such messages are typically not sent in bulk, but specifically tailored to an individual or small group. Feedback mechanisms built into client software could allow consumers to "opt-out" from some sorts of messages. For example, consumers may wish to block certain publishers or messages with specific characteristics. Such feedback could either be entirely managed by the client software or flow from the consumer peer back towards publishers, allowing the system to limit their impact. In SPICE, complaints could be returned to those RPs that notified consumers. If enough complaints were registered, peers near RPs could be told not to route casts from certain publishers or matching certain profiles. Using such a technique, implicit groups could become self-censoring if enough members responded negatively to publishers or types of messages.

10.2.4 Other concrete implementations

Other concrete implementations beyond BROKER and SPICE could also be explored. Although SPICE is well-suited to a tag-based modelling language, it may be less suited to other modelling languages, and its complexity may be unnecessary for some more limited domains.

A distributed, unstructured P2P implementation relying on ad hoc connections

between neighbours, in the style of Gnutella, may be suitable for some domains, but would have several attendant difficulties that would oblige weaker IGM semantics in practice. In particular, it would be practically impossibly to efficiently guarantee delivery of casts to all implicit group members. The lack of inherent structure in such P2P networks makes it difficult to find all implicit group members without resorting to either excessive storage costs or flooding of casts. If the semantics were relaxed such that only a fraction of group members needed to be reached with a given probability, an approach based on gossiping [76] or percolation search [73] may enjoy some limited success. Casting to small groups may still be problematic, however.

The SPICE implementation treats individual end-clients as first-class peers in the network with each owning a part of the surface and being responsible for routing and maintenance. An advantage of the BROKER implementation is that the bulk of this work can be delegated to servers, freeing individual peers but requiring additional administrative costs and potentially overloading brokers. A hybrid model using *proxy peers* may benefit from the inherent load-balancing features of SPICE and the reduced peer emphasis of BROKER. Proxy peers could join a SPICE network and act as servers for several ordinary clients. Thus to SPICE, a proxy could act like a peer with many registrations, and to consumers it could act like a server. Such an approach has the advantage of potentially improving performance for organisations with many clients and additionally, proxy peers can act as "always on" clients collecting messages on behalf of intermittently connected consumers.

10.3 Future applications

This thesis has suggested and evaluated a scientific research tool incorporating a special interest messaging component based on IGM (Chapter 9). However, IGM may also be applied to more adventurous scenarios such as Indie TV [32] and music distribution.

10.3.1 Indie TV

It is generally expensive to create, programme and broadcast a traditional television channel, which means the content must be targeted at a broad range of viewers in order to recoup the costs and make a profit. However, with the cost of creation and dissemination dropping substantially, in-depth channels catering to niche interests and demographics are now practical. Consumer-level equipment is of excellent quality, and the tools necessary for production have become affordable and capable of running on a standard desktop PC or laptop. Distribution too is becoming cheaper. With the rapid growth of the Internet and mass adoption of broadband connectivity, a spectrum of specialised TV channels has flourished: Sail TV [172] and YUKS TV [173]

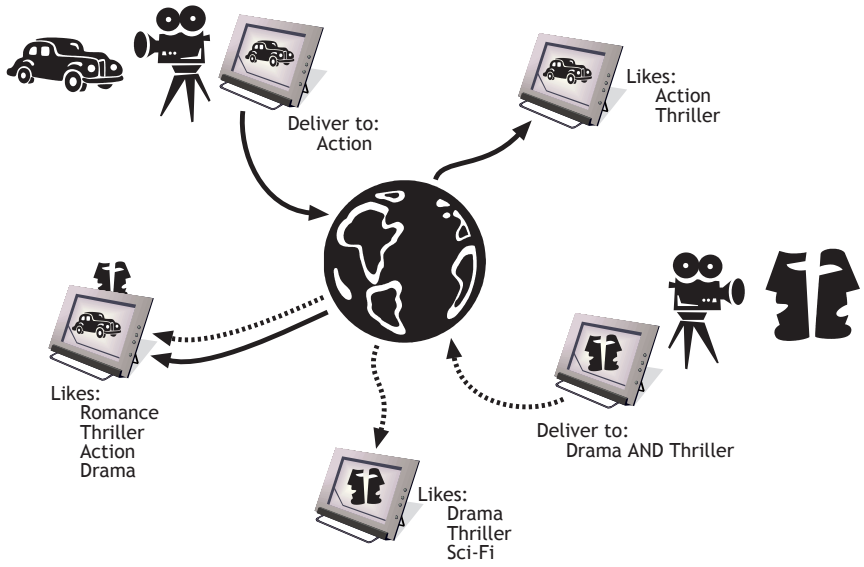


Figure 10.1: The Indie TV design.

stream video to small audiences of highly interested viewers; platforms such as Miro [174] allow people to make their own channels based on RSS feeds; and hardware such as Apple TV [175] brings broadband media into the living room.

Using IGM, a model of independent ("indie") programming can be taken to an extreme—a tailored TV channel for each viewer based specifically on their distinct and variable interests. An agent on each consumer's device takes content published to implicit groups and presents it as a continuous television channel (Figure 10.1).

10.3.2 Indie Music

Similarly, indie music artists could use IGM to publish original music to a receptive worldwide audience. Artists often need the support of music labels to succeed; but with production costs falling, labels are mainly needed to help with the distribution and marketing of music in a competitive environment. An IGM network may supplant labels for some indie artists by enabling them to directly reach consumers who may be interested in their music, achieving marketing and distribution simultaneously.

Such a system may initially seem open to abuse such as piracy or spamming. However, it is generally not to a publisher's advantage to push copyrighted music to other people in the network who may or may not wish to receive it. Piracy is a "consumer-selected" act, which IGM does not directly support. Furthermore, artists who spam their music to imprecise groups are unlikely to achieve their objective of building a credible name for themselves; listeners may reject indiscriminately published music.

This arrangement benefits publishers and consumers. Publishers are able to reach

an audience of listeners who are likely to be interested in their work and consumers serendipitously receive a stream of fresh and free independent music. When combined with proxies, IGM networks could also support mobile consumers either through wireless access points or even via mobile ad hoc networks [176]. IGM could then unite boundless combinations of people not just by their interests, but by their context and physical activities.

Appendix A

How to read the IGM pseudocode

The algorithms in this dissertation are presented as pseudocode resembling Ruby. Ruby is a clear and concise object-oriented scripting language, well-suited to high-level descriptions of algorithms. Everything is an object, including literal numbers and strings. Methods are invoked using the common "." notation. Parentheses are optional when the intention is unambiguous. Comments begin with the hash symbol.

```
# Returns the number 5.
"Hello".length
```

A variable's scope is denoted by its prefix.

```
# Local variables have no prefix.
a = 3
# Instance variable.
@a = 3
# Class variable.
@@a = 3
# Global variable.
$a = 3
```

Arrays and ranges are easy to manipulate and can hold any sorts of objects.

```
# Create an array containing three numbers.
a = [1,2,3]
# Create an array containing a number and a string.
a = Array.new
a.push 3
a << "Hello"    # Same as push.
a[0]            # Arrays are 0-based so returns 3.
# Arrays can be read from the end using negative indices.
a[-1]    # Returns the last element, "Hello".
# Create a hash.
a = { "Hello" => 3, "World" => 45, 3 => "Goodbye" }
a["World"]      # Returns 45.
```

Ranges are like arrays but need just a start and end point.

```
# Create a range from 0 to 5 (two dots means inclusive).
r = (0..5)
# Create a range from 0 to 4 (three dots means exclusive).
```

```
r = (0...5)
# Ranges can be used to take a slice from an array.
a = ['a','b','c','d']
a[0..2]  # Returns ['a','b','c'].
a[0...2] # Returns ['a','b'].
# Ranges and slices can also be used with negative indices.
a[0..-1] # Returns ['a','b','c','d'].
a[0...-1]   # Returns ['a','b','c'].
```

Ruby has idioms that increase intelligibility, such as naming a method with a trailing question or exclamation mark: a question mark means the method returns a Boolean value; an exclamation mark means the method alters the object (rather than the usual default of returning a modified copy).

```
a = ["bb","aaaa","ccc"]
a.empty?    # False.
b = a.sort  # b is ["aaaa","bb","ccc"]. a is unaffected.
a.sort!     # Modifies a to be ["aaaa","bb","ccc"] .
```

Like Perl, Ruby includes a negated *if* operator called *unless*. Either can be appended to an expression to make it conditional.

```
puts "Sender is not a client." unless sender.client?
```

One of the most interesting and useful of Ruby's features is the ability to pass inline blocks of code to methods. This allows methods to be very generic, and adapted as needed for each situation. Blocks can be enclosed in *do*...*end* clauses or braces {...}, and accept arguments from the called method.

```
# sort can be called with or without a block.
a = ["aaaa","cc","bbb"]
# Here a block is used to modify the sort order.
a.sort { |x,y| x.length <=> y.length } # Returns ["bb","ccc","aaaa"].
```

Blocks, combined with the fact that everything in Ruby is an object, enable appealing constructs.

```
# Print "Hello" five times.
5.times { puts "Hello" }
```

Blocks are used throughout Ruby. E.g., iterating over collections with the *each* method.

```
a = ["bb","aaaa","ccc"]
a.each do |x|
   x.upcase! unless x == "ccc"
end
# a is now ["BB","AAAA","ccc"].
```

When *each* is used to iterate over a hash, the key and value for each pair are passed to the block as arguments.

```
a = { "Hello" => 3, "World" => 45, 3 => "Goodbye" }
a.each do |key,value|
```

```
    puts "#{key} is #{value}"
end
```

There are a plethora of other methods that act on collections such as *map* and *find_all*.

```
[1,2,3].map { |x| x*2 }                 # Returns [2,4,6].
(0..10).find_all { |x| x % 2 == 0 }     # Returns [0,2,4,6,8,10].
```

inject works by iterating over a collection, passing in the accumulated value derived from the last iteration.

```
(1..6).inject(0) { |h,x| h + 1.0/x } # Returns the sixth harmonic number.
```

Methods are created using the *def* keyword, and return the last evaluated expression.

```
def hello
    "Hello"
end
a = hello    # a is now the string "Hello".
```

For further information, the book "Programming Ruby" (a.k.a. "the PickAxe") [139] provides an excellent introduction and reference manual for Ruby.

Bibliography

[1] Miniwatts Marketing Group, "Internet usage statistics," Internet World Stats, [Online], Available: http://www.internetworldstats.com/stats.htm, [Accessed: March, 2007]. 1

[2] T. Berners-Lee, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. HarperCollins Publishers, September 1999, ISBN: 978-0062515865. 1

[3] Google Inc., Blogger, [Online], Available: http://blogger.com, [Accessed: March, 2007]. 1

[4] Yahoo! Inc., Flickr, [Online], Available: http://flickr.com, [Accessed: March, 2007]. 1

[5] Organisation for Economic Co-operation and Development (OECD), "OECD Broadband Portal," OECD, [Online], Available: http://www.oecd.org/sti/ict/broadband, [Accessed: March, 2007]. 1, 132

[6] Pew Internet & American Life Project, "Content creation online: 44% of U.S. Internet users have contributed their thoughts and their files to the online world," Pew Internet, [Online], Available: http://www.pewinternet.org/PPF/r/113/report_display.asp, February 2004, [Accessed: March, 2007]. 1, 132

[7] C. Lindahl and E. Blount, "Weblogs: simplifying web publishing," *Computer*, vol. 36, no. 11, pp. 114–116, November 2003. 1

[8] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins, "On the bursty evolution of blogspace," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM Press, May 2003, pp. 568–576. 1

[9] Technorati Inc., Technorati, [Online], Available: http://technorati.com, [Accessed: March, 2007]. 1, 26, 128

[10] OhmyNews, OhmyNews International, [Online], Available: http://english.ohmynews.com, [Accessed: March, 2007]. 1

[11] H. Rheingold, *Smart Mobs: The Next Social Revolution*. Perseus Books Group, October 2002, ISBN: 978-0738206080. 1

[12] A. F. Farhoomand and D. H. Drury, "Managerial information overload," *Communications of the ACM*, vol. 45, no. 10, pp. 127–131, October 2002. 1

[13] SourceForge, Inc., Slashdot: News for nerds, stuff that matters, [Online], Available: http://www.slashdot.org, [Accessed: March, 2007]. 2

[14] Electric Pulp, FeedRinse, [Online], Available: http://www.feedrinse.com, [Accessed: March, 2007]. 2

[15] KnowNow, Inc., KnowNow, [Online], Available: http://www.knownow.com, [Accessed: March, 2007]. 2

[16] P. Morville, *Ambient Findability*. Sebastopol, CA: O'Reilly Media, Inc., September 2005, ISBN: 0596007655. 2

[17] Google Inc., Google, [Online], Available: http://google.com, [Accessed: March, 2007]. 4

[18] 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 the 2001 conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press, August 2001, pp. 149–160. 4, 5, 12, 19, 88

[19] S. P. Ratnasamy, "A scalable content-addressable network," Ph.D. dissertation, University of California at Berkeley, October 2002. 4, 5, 12, 18, 77, 80

[20] Y. J. Pyun and D. S. Reeves, "Constructing a balanced, (log(N)/loglog(N))-diameter super-peer topology for scalable P2P systems," in *4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*. IEEE Computer Society, August 2004, pp. 210–218. 4

[21] C. Ding, C.-H. Chi, J. Deng, and C.-L. Dong, "Centralized content-based web filtering and blocking: How far can it go?" in *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, October 1999, pp. 115–119. 4

[22] A. Mowshowitz and A. Kawaguchi, "Bias on the web," *Communications of the ACM*, vol. 45, no. 9, pp. 56–60, September 2002. 4

[23] J. Lee, "An end-user perspective on file-sharing systems," *Communications of the ACM*, vol. 46, no. 2, pp. 49–53, February 2003. 5

[24] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, November 2001, pp. 329–350. 5, 12, 88

[25] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, January 2004. 5, 12, 15, 88

[26] I. Barland, J. Greiner, and M. Vardi, "Using temporal logic to specify properties," Connexions, [Online], Available: http://cnx.org/content/m12317/1.13/, October 2005, [Accessed: March, 2007]. 5

[27] D. Cutting, D. J. Corbett, and A. Quigley, "Context-based messaging for ad hoc networks," in *Third International Conference on Pervasive Computing (Pervasive 2005), Munich, Germany, Adjunct Proceedings*, A. Ferscha, R. Mayrhofer, T. Strang, C. Linnhoff-Popien, A. Dey, A. Butz, and A. Schmidt, Eds. Oesterreichische Computer Gesellschaft, May 2005, pp. 33–36, ISBN: 3-85403-191-2. 5

[28] D. Cutting, "Middleware in pervasive computing environments," in *Fifth Association of Pacific Rim Universities (APRU) Doctoral Students Conference, Sydney, Australia*. APRU, August 2004. 5

[29] D. Cutting and A. Quigley, "Mapping context to situations in a pervasive computing environment," in *First International Workshop on Software Aspects of Context (IWSAC'04), ACS/IEEE International Conference on Pervasive Services (ICPS'04), Beirut, Lebanon*, July 2004. 5

[30] D. Cutting, B. Landfeldt, and A. Quigley, "Implicit group messaging over peer-to-peer networks," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P2006), Cambridge, United Kingdom*, A. Montresor, A. Wierzbicki, and N. Shahmehri, Eds. IEEE Computer Society, September 2006, pp. 125–132. 6

[31] D. Cutting, A. Quigley, and B. Landfeldt, "SPICE: Scalable P2P implicit group messaging," *Computer Communications*, 2007, doi:10.1016/j.comcom.2007.08.026. 6

[32] D. Cutting, B. Landfeldt, and A. Quigley, "The long, interesting tail of Indie TV," in *International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB) at the Fourth International Conference on Pervasive Computing (Pervasive 2006), Dublin, Ireland (invited paper)*, May 2006. 7, 149

[33] D. Cutting, A. Quigley, and B. Landfeldt, "Special Interest Messaging: A Comparison of IGM Approaches," *The Computer Journal*, 2007, doi:10.1093/comjnl/bxm076. 7

[34] S. Deering, "Host extensions for IP multicasting," Network Working Group, Internet Engineering Task Force, Stanford University, RFC 1112, [Online], Available: http://www.ietf.org/rfc/rfc3170.txt, August 1989, [Accessed: March, 2007]. 8

[35] Y. hua Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM Press, June 2000, pp. 1–12. 8

[36] Y. D. Chawathe, "Scattercast: An architecture for Internet broadcast distribution as an infrastructure service," Ph.D. dissertation, University of California, Berkeley, December 2000. 8

[37] P. Francis, "Yoid: Extending the Internet multicast architecture," [Online], Available: http://www.icir.org/yoid/docs/index.html, April 2000, [Accessed: March, 2007]. 8

[38] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, October 2002. 9, 13

[39] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. London, UK: Springer-Verlag, November 2001, pp. 14–29. 9

[40] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, June 2001, pp. 11–20. 9

[41] P. T. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, June 2003. 9

[42] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a

wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, August 2001. 9, 57

[43] B. Segall and D. Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching," in *Proceedings of Australian Unix and Open Systems User Group Conference (AUUG 97), Brisbane, Australia*. Distributed Systems Technology Centre, University of Queensland, Australia, September 1997. 9

[44] G. Mühl, "Large-scale content-based publish/subscribe systems," Ph.D. dissertation, Darmstadt University of Technology, Darmstadt, Germany, August 2002. 9, 11, 23, 25, 57

[45] S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach, "Exactly-once delivery in a content-based publish-subscribe system," in *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, June 2002, pp. 7–16. 9, 11, 25, 57

[46] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery," in *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*. London, UK: Springer-Verlag, August 2002, pp. 195–210. 9

[47] P. Busetta, M. Merzi, S. Rossi, and M. Zancanaro, "Group communication for real-time role coordination and ambient intelligence," in *Proceedings of Workshop on AI in Mobile Systems (AIMS 2003), 5th International Conference on Ubiquitous Computing (UbiComp 2003)*, A. Kruger and R. Malaka, Eds. Seattle, WA, USA: Springer-Verlag, October 2003, pp. 9–16. 9

[48] T. Chambel, C. Moreno, N. Guimaraes, and P. Antunes, *Concepts and Architecture for Loosely Coupled Integration of Hyperbases*, ser. Broadcast Technical Report Series, ISSN: 1350-2042. Broadcast Secretariat, Department of Computing Science, University of Newcastle-upon-Tyne, UK, 1994, vol. 3, ch. 4. 9

[49] M. Khambatti, "Peer-to-peer communities: architecture, information and trust management," Ph.D. dissertation, Arizona State University, December 2003. 9, 10, 11, 12, 131

[50] A. P. Yu and S. T. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, June 2005, pp. 99–104. 10

[51] K. L. Morse, "Interest management in large-scale distributed simulations," Department of Information & Computer Science, University of California, Irvine, CA, USA, Tech. Rep. ICS-TR-96-27, 1996. 10

[52] A. Bozdog, R. van Renesse, and D. Dumitriu, "SelectCast: a scalable and self-repairing multicast overlay routing facility," in *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*. New York, NY, USA: ACM Press, October 2003, pp. 33–42. 10

[53] J. Lave and E. Wenger, *Situated Learning : Legitimate Peripheral Participation (Learning in Doing: Social, Cognitive & Computational Perspectives)*. Cambridge University Press, September 1991, ISBN: 0521423740. 10

[54] G. Fischer, "Communities of interest: Learning through the interaction of multiple knowledge systems," in *24th Annual Information Systems Research Seminar in Scandinavia (IRIS'24), Ulvik, Hardanger Fjord, Norway*, A. M. S. Bjornestad, R. Moe and A. Opdahl, Eds., August 2001, pp. 1–14. 10

[55] H. Rheingold, *The Virtual Community: Homesteading on the Electronic Frontier*, revised ed. The MIT Press, November 2000, ISBN: 978-0-262-68121-6. 10

[56] Y. Zhang and M. Weiss, "Virtual communities and team formation," *Crossroads*, vol. 10, no. 1, pp. 5–5, Fall 2003. 10

[57] W. Gu and W. Wei, "Automatic community discovery in peer-to-peer systems," in *Fifth International Conference on Grid and Cooperative Computing Workshops, 2006. GCCW '06*. Los Alamitos, CA, USA: IEEE Computer Society, October 2006, pp. 110–116. 10

[58] S. Castano and S. Montanelli, "Semantic self-formation of communities of peers," in *Proc. of the ESWC Workshop on Ontologies in Peer-to-Peer Communities*, Heraklion, Greece, May 2005. 10

[59] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," *SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 285–298, Winter 2002. 11

[60] M. Landers, H. Zhang, and K.-L. Tan, "Peerstore: Better performance by relaxing in peer-to-peer backup." in *4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*. IEEE Computer Society, August 2004, pp. 72–79. 11

[61] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: a read/write peer-to-peer file system," *SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 31–44, Winter 2002. 11

[62] G. J. Fakas and B. Karakostas, "A peer to peer architecture for dynamic workflow management using web services," *Information and Software Technology Journal*, vol. 46, no. 6, pp. 423–431, 2004. 11

[63] S. Waterhouse, "JXTA search: Distributed search for distributed networks," Sun Microsystems, Inc., Palo Alto, CA, USA, Tech. Rep., May 2001, [Online], Available: http://gnunet.org/papers/JXTAsearch.pdf, [Accessed: January, 2008]. 11

[64] P. R. Eaton, "Caching the web with oceanstore," UC Berkeley, Computer Science Division, Tech. Rep. UCB/CSD-02-1212, November 2002. 11

[65] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. S. Wallach, X. Bonnaire, P. Sens, J.-M. Busca, and L. B. Arantes, "POST: A secure, resilient, cooperative messaging system," in *Proceedings of HotOS'03: 9th Workshop on Hot Topics in Operating Systems, Lihue (Kauai), Hawaii, USA*, M. B. Jones, Ed. USENIX, May 2003, pp. 61–66. 11

[66] S. Kulkarni, "Video streaming on the Internet using split and merge multicast," in *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, September 2006, pp. 221–222. 11

[67] P. R. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," in *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02), Vienna, Austria*. Washington, DC, USA: IEEE Computer Society, July 2002, pp. 611–618. 11

[68] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: Content-based publish/subscribe over P2P networks." in *Proceedings of Middleware 2004, ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada*, H.-A. Jacobsen, Ed. New York, NY, USA: Springer, October 2004, pp. 254–273. 11, 15

[69] T. Xue, B. Feng, and Z. Zhang, "P2PENS: Content-based publish-subscribe over peer-to-peer network," in *Proceedings of Third International Conference on Grid and Cooperative Computing (GCC 2004), Wuhan, China*, H. Jin, Y. Pan, N. Xiao, and J. Sun, Eds. Germany: Springer-Verlag, October 2004, pp. 583–590. 11

[70] P.-A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl, "Publish/subscribe for RDF-based P2P networks," in *Proceedings of the European Semantic Web Symposium, Heraklion, Greece*, May 2004. 11

[71] J. Mitre and L. Navarro-Moldes, "P2P architecture for scientific collaboration," in *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling*

Technologies: Infrastructure for Collaborative Enterprises (WETICE'04). Washington, DC, USA: IEEE Computer Society, June 2004, pp. 95–100. 11, 131

[72] P. A. Chirita, A. Damian, W. Nejdl, and W. Siberski, "Search strategies for scientific collaboration networks," in *P2PIR '05: Proceedings of the 2005 ACM workshop on Information retrieval in peer-to-peer networks*. New York, NY, USA: ACM Press, November 2005, pp. 33–40. 11, 131

[73] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury, "Percolation search in power law networks: Making unstructured peer-to-peer networks scalable." in *4th International Conference on Peer-to-Peer Computing (P2P 2004), Zurich, Switzerland*. IEEE Computer Society, August 2004, pp. 2–9. 12, 149

[74] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proceedings of INFOCOM 2002: Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA*, June 2002, pp. 1248–1257. 12

[75] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks," in *Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria*, July 2002. 12

[76] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, no. 4, pp. 319–349, 1988. 12, 149

[77] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," in *Proceedings of the First Workshop on Sensor Networks and Applications (WSNA '02), Atlanta, GA*. New York, NY, USA: ACM Press, October 2002, pp. 22–31. 12

[78] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, Boston Massachusetts, USA*. New York, NY, USA: ACM Press, August 2000, pp. 56–67. 12

[79] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. Washington, DC, USA: IEEE Computer Society, July 2002, pp. 23–32. 12

[80] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proceedings of the ACM SIGCOMM '04 Conference, Portland, Oregon, USA*. New York, NY, USA: ACM Press, August 2004, pp. 15–26. 13, 44

[81] H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A structural approach to latency prediction," in *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*. New York, NY, USA: ACM Press, October 2006, pp. 99–104. 13

[82] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, August 2001, pp. 161–172. 13, 88

[83] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: multi-dimensional queries in P2P systems," in *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, Paris, France*. New York, NY, USA: ACM Press, June 2004, pp. 19–24. 14

[84] J. Aspnes, J. Kirsch, and A. Krishnamurthy, "Load balancing and locality in range-queriable data structures," in *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, July 2004, pp. 115–124. 14

[85] Y. Choi and D. Park, "Mirinae: A peer-to-peer overlay network for large-scale content-based publish/subscribe systems," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, June 2005, pp. 105–110. 14

[86] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," *SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 353–366, October 2004. 14

[87] T. Pitoura, N. Ntarmos, and P. Triantafillou, "Replication, load balancing and efficient range query processing in DHTs," in *Proceedings of 10th International Conference on Extending Database Technology (EDBT06)*, Y. E. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, A. Kemper, T. Grust, and C. Böhm, Eds. Munich, Germany: Springer, March 2006, pp. 131–148. 15, 35

[88] C. Wang, B. A. Alqaralleh, B. B. Zhou, F. Brites, and A. Y. Zomaya, "Self-organizing content distribution in a data indexed DHT network," in *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, September 2006, pp. 241–248. 15

[89] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," in *NSDI'04: Proceed-*

ings of the 1st conference on Symposium on Networked Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, March 2004, pp. 99–112. 15

[90] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), Hachioji, Tokyo, Japan*. Washington, DC, USA: IEEE Computer Society, March 2004, pp. 360–369. 15

[91] M. Cai, A. Chervenak, and M. Frank, "A peer-to-peer replica location service based on a distributed hash table," in *Proceedings of the ACM/IEEE SC2004 Conference on High Performance Networking and Computing, Pittsburgh, PA, USA*. Washington, DC, USA: IEEE Computer Society, November 2004, pp. 56–56. 15

[92] J. Gao, "A distributed and scalable peer-to-peer content discovery system supporting complex queries," Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, October 2004, CMU-CS-04-170. 16

[93] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970. 17, 59

[94] A. Broder and M. Mitzenmacher, "Network applications of Bloom Filters: A survey," in *Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing, Illinois, USA*, October 2002. 17, 18

[95] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, June 1984. 18

[96] E. Tanin, A. Harwood, and H. Samet, "A distributed quadtree index for peer-to-peer settings," in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan*. Washington, DC, USA: IEEE Computer Society, April 2005, pp. 254–255. 18

[97] Z. Xu and Z. Zhang, "Building low-maintenance expressways for P2P systems," HP Labs, Tech. Rep. HPL-2002-41, March 2002. 18

[98] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of Internet content delivery systems," *5th Symposium on Operating Systems Design and Implementation SIGOPS*, vol. 36, no. SI, pp. 315–327, Winter 2002. 21

[99] W. E. Weihl, "Beyond content delivery: applications to the edge," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video, Cork, Ireland*, V. N. Padmanabhan and C. J. Sreenan, Eds. New York, NY, USA: ACM Press, June 2004, pp. 1–1. 21

[100] Akamai Technologies, Akamai, [Online], Available: http://akamai.com, [Accessed: January, 2008]. 21, 56

[101] T. Strang, C. Linnhoff-Popien, and K. Frank, "CoOL: A Context Ontology Language to enable Contextual Interoperability," in *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, J.-B. Stefani, I. Dameure, and D. Hagimont, Eds. Paris/France: Springer Verlag, November 2003, pp. 236–247. 26

[102] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *Knowledge Engineering Review*, vol. 18, no. 3, pp. 197–207, September 2003. 26

[103] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American Magazine*, May 2001. 26

[104] M. Wolfe, "Metadata, knowledge management, and communications," *Canadian Journal of Communication*, vol. 25, no. 4, pp. 539–552, Autumn 2000. 26

[105] N. Guarino, "Some ontological principles for designing upper level lexical resources," in *First International Conference on Language Resources and Evaluation*, Rubio, Gallardo, Castro, and Tejada, Eds., Granada, Spain, September 1998, pp. 527–534. 26

[106] M. Uschold and M. Grüninger, "Ontologies: principles, methods, and applications," *Knowledge Engineering Review*, vol. 11, no. 2, pp. 93–155, 1996. 26

[107] S. Pawar, "Taxonomic chauvinism and the methodologically challenged," *BioScience*, vol. 53, no. 9, pp. 861–864, September 2003. 26

[108] B. Lund, T. Hammond, M. Flack, and T. Hannay, "Social bookmarking tools (II): A case study - Connotea," *D-Lib Magazine*, vol. 11, no. 4, April 2005, [Online], Available: http://www.dlib.org, doi:10.1045/april2005-lund, [Accessed: January, 2008]. 26, 130

[109] S. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *Journal of Information Science*, vol. 32, no. 2, pp. 198–208, April 2006. 26, 29

[110] T. Hammond, T. Hannay, B. Lund, and J. Scott, "Social bookmarking tools (I): A general review," *D-Lib Magazine*, vol. 11, no. 4, April 2005, [Online], Available: http://www.dlib.org, doi:10.1045/april2005-hammond, [Accessed: January, 2008]. 26

[111] A. Mathes, "Folksonomies - cooperative classification and communication through shared metadata," [Online], Available: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html, December 2004, [Accessed: January, 2008]. 26

[112] G. Zipf, *Selective Studies and the Principle of Relative Frequency in Language*. Cambridge, MA, USA: MIT Press, 1932. 28

[113] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, pp. 143–150, June 2002. 28

[114] Wikipedia, the free encyclopedia, "Most linked to categories: Wikipedia," [Online], Available: http://en.wikipedia.org/wiki/Special:Mostlinkedcategories, June 2007, [Accessed: 17 June, 2007]. 28

[115] H. Halpin, V. Robu, and H. Shepherd, "The complex dynamics of collaborative tagging," in *WWW '07: Proceedings of the 16th international conference on World Wide Web, Banff, Canada*. New York, NY, USA: ACM Press, May 2007, pp. 211–220. 29

[116] C. Cattuto, "Semiotic dynamics in online social communities," *The European Physical Journal C - Particles and Fields*, vol. 46, no. 2, pp. 33–37, August 2006. 29

[117] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001), Prague, Czech Republic*, June 2001. 38

[118] D. Symonds and D. Cutting, "ROMNeT: a Ruby-based simulation harness for OMNeT++," April 2007, ROMNeT, [Online], Available: http://romnet.worldjourney.com, [Accessed: January, 2008]. 38

[119] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communication*, vol. 6, no. 9, pp. 1617–1622, December 1988. 39

[120] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959. 39

[121] M. B. Doar, "A better model for generating test networks," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, November 1996, pp. 86–93. 39

[122] K. Calvert, J. Eagan, S. Merugu, A. Namjoshi, J. Stasko, and E. Zegura, "Extending and enhancing GT-ITM," in *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research,*

Karlsruhe, Germany. New York, NY, USA: ACM Press, August 2003, pp. 23–27. 39

[123] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, Cambridge, Massachusetts, USA*. New York, NY, USA: ACM Press, October 1999, pp. 251–262. 39, 40

[124] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*. Washington, DC, USA: IEEE Computer Society, August 2001, pp. 346–353. 39

[125] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator," Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor, MI, USA, Tech. Rep. CSE-TR-433-00, September 2000. 39

[126] T. Bu and D. Towsley, "On distinguishing between Internet power law topology generators," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), New York, NY, USA*, June 2002, pp. 638–647. 39

[127] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, pp. 2–16, February 2004. 39, 40

[128] Y. Shavitt and E. Shir, "DIMES: let the Internet measure itself," *SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 71–74, October 2005. 40

[129] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: degree-based vs. structural," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, Pittsburgh, Pennsylvania, USA*. New York, NY, USA: ACM Press, August 2002, pp. 147–159. 40

[130] L. Li, D. Alderson, W. Willinger, and J. Doyle, "A first-principles approach to understanding the Internet's router-level topology," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, Portland, Oregon, USA*. New York, NY, USA: ACM Press, September 2004, pp. 3–14. 40

[131] D. Alderson, L. Li, W. Willinger, and J. C. Doyle, "Understanding Internet topology: principles, models, and validation," *IEEE/ACM Transactions on Networking*, vol. 13, no. 6, pp. 1205–1218, December 2005. 40

[132] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz, "Generating realistic ISP-level network topologies," *IEEE Communications Letters*, vol. 7, no. 7, pp. 335–337, July 2003. 40

[133] L. Lao, J.-H. Cui, and M. Gerla, "A framework for realistic and systematic multicast performance evaluation," *Computer Networks*, vol. 50, no. 12, pp. 2054–2070, August 2006. 40

[134] IAC Search & Media, Excite, [Online], Available: http://www.excite.com, [Accessed: January, 2008]. 41

[135] S. Ozmutlu, A. Spink, and H. C. Ozmutlu, "Trends in multimedia web searching: Excite queries," in *ITCC '02: Proceedings of the International Conference on Information Technology: Coding and Computing, Las Vegas, Nevada, USA*. Washington, DC, USA: IEEE Computer Society, April 2002, pp. 40–45. 41

[136] A. Spink and J. L. Xu, "Selected results from a large study of Web searching: the Excite study," *Information Research*, vol. 6, no. 1, October 2000, [Online], Available: http://InformationR.net/ir/6-1/paper90.html, [Accessed: January, 2008]. 41

[137] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys (CSUR)*, vol. 38, no. 2, July 2006. 48

[138] Ruby, Ruby, [Online], Available: http://www.ruby-lang.org, [Accessed: January, 2008]. 48

[139] D. Thomas, C. Fowler, and A. Hunt, *Programming Ruby: The Pragmatic Programmers' Guide*, 2nd ed. Releigh, North Carolina, Dallas, Texas: Pragmatic Bookshelf, January 2006, ISBN 0-9745140-5-5. 48, 154

[140] T. M. Przytycka and L. Higham, "Optimal cost-sensitive distributed minimum spanning tree algorithm," in *SWAT '96: Proceedings of the Fifth Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland*. London, UK: Springer-Verlag, July 1996, pp. 246–258. 58

[141] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*. Rio de Janeiro, Brazil: Springer, June 2003, pp. 21–40. 73

[142] M. Zhong, J. Moore, K. Shen, and A. L. Murphy, "An evaluation and comparison of current peer-to-peer full-text keyword search techniques," in *Eighth International Workshop on the Web and Databases (WebDB 2005)*, A. Doan, F. Neven, R. McCann, and G. J. Bex, Eds., Baltimore, Maryland, June 2005, pp. 61–66. 73

[143] T. Lu, S. Sinha, and A. Sudan, "Panaché: A scalable distributed index for keyword search," Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, Tech. Rep., 2002. 73

[144] M. Cai and M. Frank, "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network," in *Proceedings of the 13th international conference on World Wide Web (WWW 2004), New York, NY, USA*. New York, NY, USA: ACM Press, May 2004, pp. 650–657. 73

[145] L. Liu, K. D. Ryu, and K.-W. Lee, "Keyword fusion to support efficient keyword-based search in peer-to-peer file sharing," in *Proceedings of the Fourth International Workshop on Global and Peer-to-Peer Computing*, Chicago, IL, USA, April 2004. 74

[146] O. Gnawali, "A keyword set search system for peer-to-peer networks," Master's thesis, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, June 2002. 74

[147] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," IBM Canada Ltd., Ottawa, Canada, Tech. Rep., 1966. 85

[148] M. Schneider, "Self-stabilization," *ACM Computing Surveys (CSUR)*, vol. 25, no. 1, pp. 45–67, March 1993. 86

[149] A. Ghodsi, L. O. Alima, and S. Haridi, "Low-bandwidth topology maintenance for robustness in structured overlay networks," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), Big Island, HI, USA*. Washington, DC, USA: IEEE Computer Society, January 2005. 86

[150] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *ACM Wireless Networks*, vol. 7, no. 6, pp. 609–616, November 2001. 87

[151] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *First International Workshop on Peer-to-Peer Systems at IPTPS 2002, Cambridge, MA, USA*, P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds. Springer, March 2002, pp. 53–65. 88

[152] Backbone Media Inc., "Corporate blogging: is it worth the hype?" [Online], Available: http://www.backbonemedia.com/blogsurvey, 2005, [Accessed: March, 2007]. 128

[153] R. Scoble, Scobleizer, [Online], Available: http://scobleizer.com, [Accessed: March, 2007]. 128

[154] J. Snell, "Blogging@IBM," [Online], Available: http://www-03.ibm.com/developerworks/blogs/page/jasnell?entry=blogging_ibm, May 2005, [Accessed: March, 2007]. 128

[155] J. McGregor, "It's a blog world after all," [Online], Available: http://www.fastcompany.com/magazine/81/blog.html, April 2004, [Accessed: March, 2007]. 128

[156] A. Etches-Johnson, Blogging Libraries Wiki, [Online], Available: http://www.blogwithoutalibrary.net/links, [Accessed: March, 2007]. 128

[157] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Proceedings of First International Workshop on Advanced Context Modelling, Reasoning and Management at UbiComp 2004, Nottingham, England*. Springer, September 2004, pp. 34–41. 129

[158] G.-C. Roman, C. Julien, and J. Payton, "A formal treatment of context-awareness," in *Proceedings of Fundamental Approaches to Software Engineering, Seventh International Conference (FASE 2004), part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2004), Barcelona, Spain*. Springer, March 2004, pp. 12–36. 129

[159] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction (HCI)*, vol. 16, no. 2–4, pp. 97–166, 2001. 129

[160] F. Siegemund, "A context-aware communication platform for smart objects," in *Second International Conference on Pervasive Computing, Linz/Vienna, Austria*. Heidelberg, Germany: Springer-Verlag, April 2004, pp. 69–86. 129

[161] Mekentosj, Papers, [Online], Available: http://mekentosj.com/papers/, [Accessed: January, 2008]. 130

[162] BibDesk, BibDesk, [Online], Available: http://bibdesk.sourceforge.net, [Accessed: January, 2008]. 130

[163] The Thomson Corporation, EndNote, [Online], Available: http://www.endnote.com, [Accessed: March, 2007]. 130

[164] D. Cutting, Cloister, [Online], Available: http://cloister.sourceforge.net, [Accessed: January, 2008]. 130

[165] R. Cameron, CiteULike, [Online], Available: http://citeulike.org, [Accessed: January, 2008]. 130

[166] Software Engineering Research Laboratory (SERL) at the University of Colorado, Boulder, SEWORLD, [Online], Available: http://serl.cs.colorado.edu/~serl/seworld/, [Accessed: January, 2008]. 130

[167] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich, "Bibster — a semantics-based bibliographic peer-to-peer system," in *Proceedings of the Third International Semantic Web Conference (ISWC 2004), Hiroshima, Japan*, S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds. Springer-Verlag, November 2004, pp. 122–136. 130

[168] M. Liljenstam, J. Liu, and D. M. Nicol, "Simulation of large scale networks II: development of an Internet backbone topology for large-scale network simulations," in *WSC '03: Proceedings of the 35th conference on Winter simulation, New Orleans, Louisiana, USA*. Winter Simulation Conference, December 2003, pp. 694–702. 132

[169] A.-C. Achilles and P. Ortyl, Collection of Computer Science Bibliographies, [Online], Available: http://liinwww.ira.uka.de/bibliography/index.html, [Accessed: March, 2007]. 133

[170] Department of Computing Science, University of Glasgow, The Information Retrieval Group, [Online], Available: http://ir.dcs.gla.ac.uk, [Accessed: January, 2008]. 135

[171] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A bayesian approach to filtering junk E-mail," in *AAAI-98 workshop on Learning for Text Categorization, Madison, Wisconsin*, July 1998. 148

[172] Sailing Networks, Sail TV, [Online], Available: http://sail.tv, [Accessed: January, 2008]. 149

[173] Dailey Pike, YUKS TV, [Online], Available: http://yuks.tv, [Accessed: March, 2007]. 149

[174] Participatory Culture Foundation, Miro, [Online], Available: http://www.getmiro.com, [Accessed: January, 2008]. 150

[175] Apple Inc., Apple TV, [Online], Available: http://apple.com/appletv, [Accessed: January, 2008]. 150

[176] I. F. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Communications Magazine*, vol. 43, no. 9, pp. S23–S30, September 2005. 151