Techniques for Overlay Design of Content-Based Publish/Subscribe Systems

by

Naweed Tajuddin

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Techniques for Overlay Design of Content-Based Publish/Subscribe Systems

Naweed Tajuddin

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2010

Mission-critical distributed applications, such as Internet advertising platforms, increasingly utilize distributed publish/subscribe systems as a messaging substrate for information dissemination. These applications require low latency performance from the substrate, as the timely delivery of messages can have a direct on impact revenue. The cost of managing and operating publish/subscribe systems, however, can be prohibitive due to system size and scale. It is, therefore, critical to derive low latency message delivery from a minimal set of system resources.

To this end, this thesis presents a solution for designing low latency, minimal-broker overlay networks for content-based publish/subscribe systems. The solution includes a framework for quantifying the similarity of clients and brokers, and algorithms for constructing overlay topologies where brokers sharing similar interests are assigned a direct overlay connection. Additionally, a load model and algorithms are presented for designing overlays that utilize a minimal number of brokers in order to reduce system cost.

Acknowledgements

This thesis is dedicated to my dear parents, Mr. Nazim Tajuddin and Mrs. Noorbanu Tajuddin. It is through their endless love, support, and encouragement that I was able to get through even the most challenging times.

I would like to extend my deepest gratitude to my supervisor, Professor Hans-Arno Jacobsen, for inspiring this research, and for his invaluable guidance and feedback all throughout my Masters studies.

I am indebted to Dr. Balasubramaneyam Maniymaran, truly my partner in this work, for teaching me how to get to the core of a problem, and for letting me tap into his extensive reservoir of knowledge whenever I found myself stuck.

Lots of thanks to everyone in and around the Middleware Systems Research Group for providing a friendly and intellectually stimulating research environment. In particular, I would like to acknowledge the contributions of Serge Mankovski of CA Labs and (soon-to-be Dr.) Vinod Muthusamy.

Last but not least, many thanks to Noreen, Saleem, HVPS, Mr. Imran Ibrahim, The Fung, MKs, T.M.A., and other family and friends for keeping life enjoyable outside the lab.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Motivating Examples 2

1.2.1 GooPS 2

1.2.2 Yahoo Message Broker 4

1.2.3 From Topic-based to Content-based 4

1.3 Problem Statement 4

1.4 Contributions 6

1.5 Thesis Structure 7

2 Background **8**

2.1 Overview of Publish/Subscribe Systems 8

2.2 Advertisement-Based Publish/Subscribe Model 10

3 Related Work **13**

3.1 Terminology and Background 13

3.2 Topic-based Publish/Subscribe 16

3.3 Content-based Publish/Subscribe 17

3.4 Performance Modeling and Capacity Planning 20

4 Similarity Techniques **22**

4.1 Commonality 22
4.1.1 Overlap-Based Commonality 23
4.1.2 Covering-Based Commonality 25
4.2 Interest 27
4.3 Extending Similarity with Publications 28
4.4 Chapter Summary 30

5 Fixed-Broker Overlay Design 31
5.1 Problem Formulation 32
5.2 Overlay Topologies and Performance Metrics 33
5.3 Overlay Topology Construction 35
5.3.1 Overlap-based and Covering-based Commonality Overlays 36
5.3.2 Hybrid Topologies 37
5.3.3 Complexity Analysis 40
5.4 Evaluation 41
5.4.1 Simulation Setup 41
5.4.2 Workload Specification 42
5.4.3 Performance Results 44
5.4.4 Benefit of Advertisements 50
5.5 Chapter Summary 53

6 Minimal-Broker Overlay Design 54
6.1 Problem Formulation and Analysis 54
6.1.1 Problem Formulation 55
6.1.2 Proof of NP-Completeness 56
6.2 Load Modeling Framework 58
6.3 Broker Allocation and Overlay Design 60
6.3.1 Client-Broker Allocation Algorithm 62

6.3.2 Overlay Topology Construction 67
6.3.3 Complexity Analysis 67
6.3.4 Algorithm Assumptions 69
6.4 Evaluation 69
6.4.1 Simulation Setup 70
6.4.2 Workload Specification 70
6.4.3 Performance Metrics 71
6.4.4 Overlay Design Cost 72
6.4.5 Overlay Design Performance 73
6.4.6 Load Modeling and Broker Congestion Effects 75
6.4.7 Load Compensation Factor 77
6.5 Chapter Summary 79

7 Conclusions and Future Work **80**

8 Appendices **82**
8.1 Appendix A: Simulator Implementation 82
8.1.1 Implementation Details 83
8.1.2 Configuration Parameters 84

Bibliography **86**

List of Figures

2.1 A graphical example of a publish/subscribe deployment 9

2.2 An example of intermediate brokers 10

2.3 A content-space with two attribute spaces. 12

3.1 Overlay topology reconfiguration to reduce message traffic 15

4.1 Calculating similarity using overlap 24

4.2 Calculating similarity using covering 26

4.3 The effect of publication rates on the interest metric 29

4.4 Supporting non-uniform publication distributions 30

5.1 Star-like and chain-like overlay topologies 33

5.2 Interest vs. commonality trade-off 38

5.3 Performance implications of subscription regionalism 45

5.4 Overlay topologies that produce similar latencies but different message counts . 47

5.5 Node degree distribution 48

5.6 Broker load distribution 48

5.7 Performance improvements with advertisements 50

5.8 Interest vs. commonality trade-off 52

6.1 Measuring the load impact of subscribers 59

6.2 Phases of MBODP 61

6.3 Overlay cost measured in terms of broker count 72

6.4 Effect of clustering on message count 74
6.5 Effect of clustering on message latency 74
6.6 Effect of clustering on pure forwarding traffic 74
6.7 Effectiveness of load modeling framework at reducing broker congestion 76
6.8 Impact of LCF value on performance and load 78

Chapter 1

Introduction

1.1 Motivation

Distributed publish/subscribe systems are increasingly the technology of choice for information dissemination in large-scale distributed applications [29, 26]. Large enterprises leverage the technology for spreading information in an expressive, scalable, and decoupled manner, across a variety of disparate services that span a range of geographic locations [26]. The increasing adoption of publish/subscribe (pub/sub) systems for mission-critical applications has led to the current situation where system performance can have a direct impact on an organization's bottom-line [26, 14]. In such scenarios, the timely delivery of messages is of the utmost importance.

The scope, scale, and performance requirements of distributed pub/sub systems, however, comes at a cost: since application users and system resources span a wide area, resource and administration costs can be excessive. It is, therefore, critical to minimize the costs associated with system operation, without sacrificing performance. One way of maintaining performance and reducing costs is to make optimal use of a given set of resources.

Distributed pub/sub systems consist of publishers, subscribers, and a federated network of brokers connected in an overlay. Publishers and subscribers inject and receive publications,

respectively, and brokers route publications from publishers to interested subscribers. The overlay network of brokers plays a critical role in both the performance and cost of a given publish/subscribe deployment, with both the number of brokers deployed, and the design of the overlay connecting the brokers having a direct impact on both metrics. For example, a network with too few brokers may reduce costs associated with system operation, but decrease end-to-end delivery performance due to processing delays caused by broker congestion. Similarly, a poor overlay design can increase delivery delays due to lengthy routing paths and result in an inefficient use of system resources.

In this thesis, we present techniques for designing broker overlay networks for distributed publish/subscribe systems that require low latency performance at minimal cost. We reduce costs by designing overlays that utilize a minimal number of brokers. Our work is specifically geared towards content-based publish/subscribe, a powerful breed of pub/sub where information processing is based on the content of messages, as opposed to message type or category. There is, therefore, an increased processing demand at the brokers, making the design of the overlay particularly important in terms of both performance and cost.

1.2 Motivating Examples

The work presented in this thesis is motivated by two practical publish/subscribe systems that support mission-critical operations at leading Internet companies. The Google publish/subscribe system, GooPS, is a topic-based pub/sub system at Google, and Yahoo Message Broker, YMB, is a topic-based pub/sub system at Yahoo. The relevant details of each system is described in turn, followed by the context through which our work is motivated.

1.2.1 GooPS

GooPS is a topic-based pub/sub middleware used for information dissemination by a broad range of Google services and applications. The system operates across a WAN, and inter-

connects applications across a global set of data centers. GooPS deploys an overlay network consisting of hundreds of brokers to facilitate communication between thousands of publishers and subscribers. The system supports messaging rates of tens of thousands of messages per second.

GooPS is a distributed pub/sub system, however, a key component of the infrastructure is a *logically centralized* service with which applications communicate in order to achieve objectives (such as synchronization and failure recovery [6]). In addition, GooPS is a *managed* system, where all resources are owned and operated by Google. Since costs associated with operating and maintaining a managed system can be excessive, a goal of algorithms employed by the system is to utilize a minimal number of resources for each task.

In GooPS, applications wishing to use the system present themselves as publishers and subscribers. We have learned that a key requirement of GooPS is to provide low latency message transfer, as the timely delivery of messages within and across data centers has a direct impact on revenue for certain applications [26]. A number of techniques are employed in order to satisfy this low latency requirement, one of which is an overlay design process, that functions as follows.

An application wishing to use GooPS must specify, to the logically centralized service, its messaging rates and topics of interest. The service uses this information to estimate the resources required to support the application, and computes an initial overlay topology to connect the appropriate end points in a way that satisfies latency requirements. The service computes a best overlay for each supported application, however, the same overlay link is used for multiple applications when routing paths overlap. In addition, while an application is operational, the service periodically computes alternative overlay topologies for the purpose of improving performance. If a better overlay is found, appropriate routing table changes are pushed to the affected brokers.

1.2.2 Yahoo Message Broker

Yahoo Message Broker (YMB) is a pub/sub component of PNUTS, a distributed database system for web applications at Yahoo [11]. YMB is used for a variety of purposes, one of which is data replication: a web application will publish a data update to YMB, which will then asynchronously propagate the update to the appropriate replication data stores. In addition, YMB is used to send notifications that invalidate cached copies of ads once the advertising budget expires [11]. In Internet advertising, where ad selection algorithms operate at a millisecond scale [33], timely removal of expended ads is critical as they no longer generate revenue.

1.2.3 From Topic-based to Content-based

Both GooPS and YMB are deployed as topic-based pub/sub systems, where messages are processed according to the group, or *topic*, to which they belong. Topic-based systems facilitate simpler routing protocols and overlay design algorithms because the interest relation that maps publishers to interested subscribers is static. Thus, the set of subscribers interested in the messages of a publisher is known in advance, and moreover, subscribers receive all messages produced by publishers with which they have interest. In contrast, the interest relationship between publishers and subscribers in content-based systems is dynamic: the set of subscribers interested in the messages of a particular publisher can only be determined per-message. Accordingly, content-based systems place greater processing capabilities within the broker network, and thereby facilitate thinner clients, and additionally may reduce the number of messages processed in the system due to a greater number of messages being filtered.

1.3 Problem Statement

Motivated by the low latency requirements of GooPS and YMB, we seek to construct low latency broker overlay networks that utilize a minimal number of brokers for managed content-based pub/sub systems. Similar to the overlay design computation process in GooPS, where

a central service is responsible for computing the initial and improved overlay topologies, we approach overlay design from a design-time perspective. In other words, we assume the information needed to compute a low latency, minimal broker overlay design is available at a central point. Accordingly, we devise algorithms that accept a set of input parameters that characterize the expected workload, and output a broker overlay design. The overlay network of a content-based pub/sub system may then be configured according to the computed overlay design.

The contributions of the thesis are split into three key sections, with each subsequent section using techniques described in the previous section. The first such section examines techniques for measuring similarity of clients and brokers in content-based pub/sub systems. We define general concepts for measuring similarity that describe the likelihood that the parties of interest, be they publisher-subscriber pairs, client-broker pairs, or broker-broker pairs, may be involved in substantial communication. The problem statement for this section is: *Given a set of publishers, subscribers, and brokers, how can we estimate the likelihood of communication among any given pair of entities.*

The second section presents algorithms that use the similarity techniques to construct overlay topologies for a given set of brokers. Under this circumstance, low latency overlays are developed by clustering brokers that have strong similarity. In addition, we pay attention to estimating the load distribution of a given overlay design. The problem statement for this section is: *Given a set of brokers, how can we construct an overlay topology that minimizes message latencies and message traffic.*

The third section builds on the previous section by adding the additional constraint of producing overlay designs that utilize a minimal set of brokers, without sacrificing performance objectives. To this end, we introduce a load modeling framework and client allocation techniques to measure broker load levels, with the goal of building an overlay that deploys only as many brokers as needed in order to avoid broker congestion. The problem statement for this section is: *Given a set of publishers and subscribers, how can we design a broker overlay net-*

work that maximizes performance (message latency) and minimizes costs (number of brokers deployed).

1.4 Contributions

In our investigation of designing low latency, minimal broker overlay networks for content-based pub/sub systems, we make the following contributions.

1. We present a framework for measuring the similarity of interests of publishers, subscribers, and brokers. We present multiple measurement techniques and conduct a thorough evaluation of the benefits and disadvantages of each approach. We show the flexible nature of the framework by utilizing it for client allocation and overlay topology construction, as well as for load modeling of content-based pub/sub systems.

2. We present an analytical model for measuring load on content-based pub/sub systems. The model predicts the load impact of publishers and subscribers on brokers, and is used for designing overlays where broker processing delays due to congestion is eliminated or significantly reduced. We are among the first to explicitly incorporate load and capacity considerations into the overlay design process.

3. We present a client-broker allocation algorithm that simultaneously determines a minimal number of brokers needed to support a given set of clients, and allocates the clients to the brokers. We additionally prove that the problem of allocating clients to a minimal set of brokers is NP-hard.

4. We present and evaluate algorithms for constructing overlay networks of content-based pub/sub systems. We examine techniques for overlay design where the number of brokers is fixed, as well as when the aim is to minimize the number of brokers deployed without sacrificing performance due to broker congestion. In addition, we present a hybrid algorithm that utilizes advertisements in content-based pub/sub to further improve

performance. We perform extensive analysis of our algorithms to evaluate their effectiveness.

1.5 Thesis Structure

This thesis is organized as follows. We have motivated and defined our work in this chapter. Chapter 2 provides background information about content-based pub/sub systems. Chapter 3 presents related work. Chapter 4 describes techniques for measuring the similarity of any combination of publisher, subscriber, and broker pairs. Chapter 5 describes and evaluates an algorithm for constructing low latency broker overlay networks where the number of brokers deployed is assumed fixed. Chapter 6 describes and evaluates techniques for constructing overlay networks with a minimal number of brokers. We conclude in Chapter 7.

Chapter 2

Background

2.1 Overview of Publish/Subscribe Systems

Publish/subscribe (pub/sub) is a widely-used communication paradigm for the dissemination of information in distributed applications. Pub/sub systems consist of publishers, subscribers, and a federated network of brokers. *Publishers* are interfaces for the injection of information into the system in the form of *publications* or *events*. *Subscribers* are recipients of publications, and inject messages called *subscriptions* that specify information that is of interest to them. Publications and subscriptions are submitted to a messaging substrate, comprised of a federation of *brokers* organized in an *overlay topology*, that are collectively responsible for delivering publications to interested subscribers. This is accomplished through a process of *matching* publications to subscriptions, and *routing* publications along the appropriate path of the overlay towards one or more destinations. Since publishers and subscribers are unaware of other publishers and subscribers, the broker overlay allows for the decoupling of producers of information from consumers of information in both time and space. Note that publishers and subscribers are collectively referred to as *clients*, and the group of publishers, subscribers, and brokers are referred to as *entities* of a pub/sub system.

Figure 2.1 shows a graphical example of a typical pub/sub deployment. The system consists

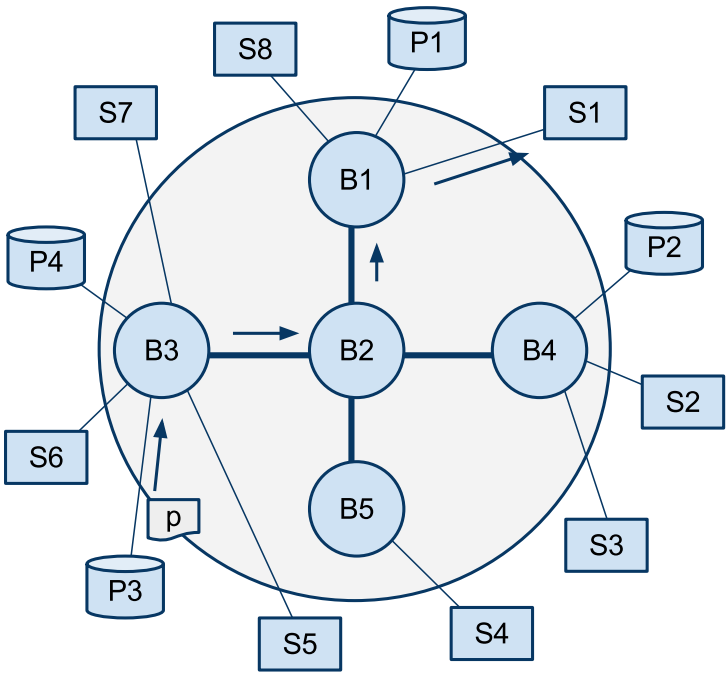


Figure 2.1: A graphical example of a publish/subscribe deployment

of four publishers and eight subscribers, which each connect to one of five brokers. The brokers are connected to other brokers through a set of overlay links. The arrows indicate a publication traveling from publisher P3 to subscriber S1.

The two most common types of pub/sub systems are *topic-based* and *content-based*. In topic-based pub/sub systems, publications are categorized into logical groups called *topics*. Subscribers of a topic receive all messages published to that topic. In content-based pub/sub, subscriptions express filters on the content of publications, thereby providing greater expressiveness to subscribers to control which messages are received. Publications are expressed as a set of attribute–value pairs, and subscriptions are expressed as a set of Boolean predicates that define constraints on these attribute–value pairs. Content-based systems provide a higher level of expressiveness for filtering information; however, at the cost of increased processing requirements at the brokers.

In traditional content-based pub/sub systems, brokers are connected in a tree-based overlay topology [7, 22, 13]. In order to enable our algorithms to be used by these systems, and for simplicity, we also assume tree-based overlay topologies. It is an aspect of future work to relax

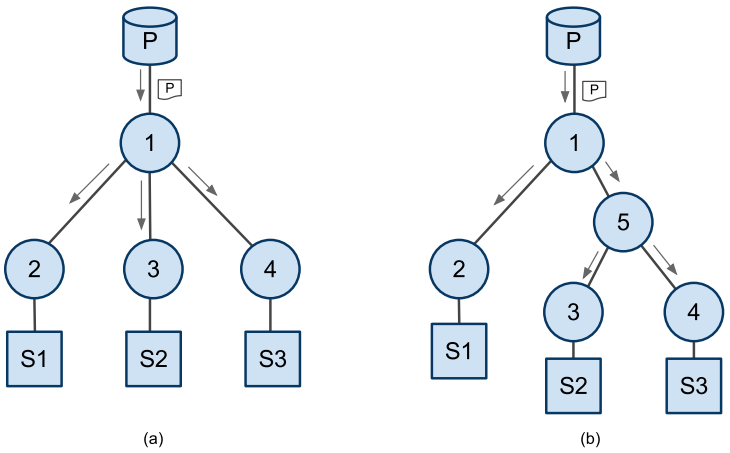


Figure 2.2: An example of intermediate brokers

this assumption.

A tree-based overlay may contain intermediate brokers that do not maintain any client connections, and simply connect to other brokers. These brokers do not match any messages; they simply forward messages between neighboring brokers. Although intermediate brokers increase the number of hops a message must travel from the publisher to interested subscribers, they are useful in preventing brokers from getting overloaded by excessive traffic. For example, Figure 2.2 shows a publication *P* that is to be forwarded to subscribers S1, S2, and S3. When the overlay is structured as in Figure 2.2 (a), Broker 1 has to create three forwarding messages. However, if they are connected as in Figure 2.2 (b) with an intermediate broker (Broker 5), the number of forwarding messages for Broker 1 is reduced to two.

2.2 Advertisement-Based Publish/Subscribe Model

We assume pub/sub systems employ an advertisement-based routing protocol to deliver messages. Such a routing protocol is common in many existing pub/sub systems, including SIENA [7], REBECA [22], and PADRES [13]. In this model, publishers advertise the type of information they publish, in messages called *advertisements*. The protocol works as follows. Advertisements are injected into the system and forwarded to every broker. Incoming sub-

scriptions are then matched against advertisements and forwarded along the reverse path of matching advertisements. Publications are then forwarded along the reverse path of matching subscriptions. Formerly, subscriptions were flooded in the network, with publications forwarded along the reverse path of matching subscriptions. Thus, the advertisement-based routing protocol works best when the number of subscriptions is much greater than the number of advertisements, and the number of publications is much greater than the number of subscriptions. [7].

In addition, we assume that advertisements, subscriptions, and publications can be mapped to a *content-space*. The content-space consists of multiple *attribute spaces*, each belonging to a type attribute, referred to as a *class*. An attribute space is a multi-dimensional space where each axis belongs to a particular attribute. A publication, described as a set of attribute–value pairs, becomes a point in a particular attribute space of the content space. Advertisements and subscriptions are modeled as conjunctions of *range-based* predicates and, therefore, occupy a volume[1] in the space. A publication matches a subscription if it is a point within the region defined by the subscription.

An example content-space is shown in Figure 2.3. The figure shows a two-dimensional and three-dimensional attribute space, the first with $class = sale$ and the second with $class = purchase$. The $sale$ attribute space contains one subscription and one advertisement. Publication $p2$ matches subscription $s2$.

[1] Our later examples assume a two-dimensional space, in which case an area is occupied in the content-space, instead of a volume.

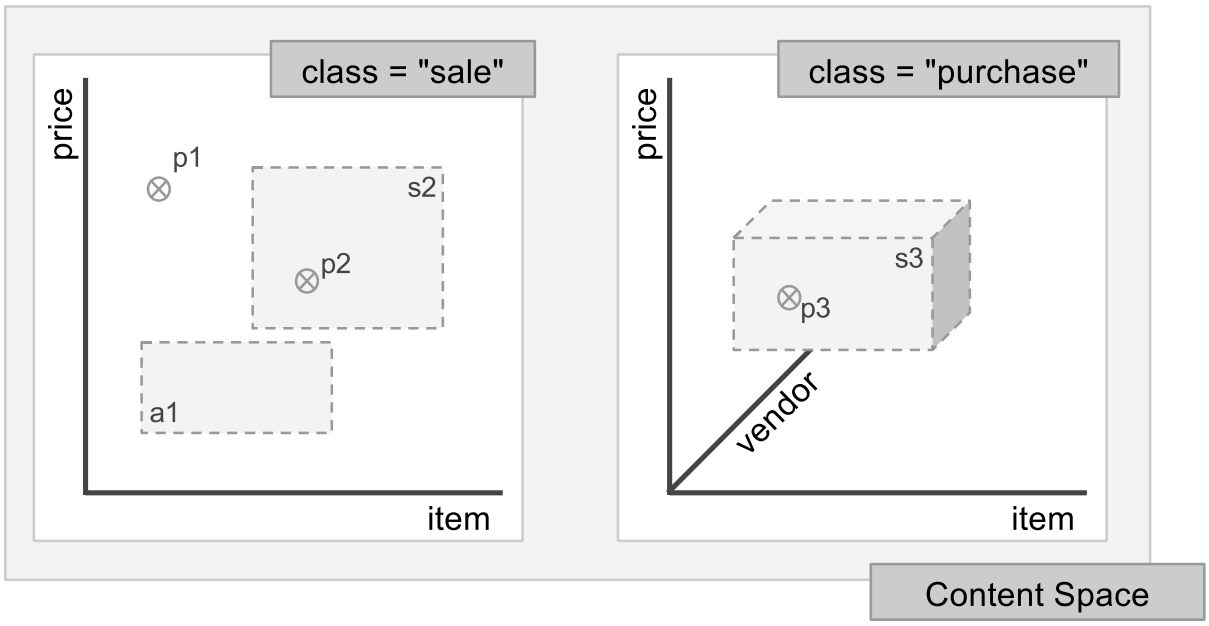


Figure 2.3: A content-space with two attribute spaces.

Chapter 3

Related Work

In this section, we describe related work with respect to the three primary chapters of this thesis. We provide an overview and discuss the limitations of existing overlay design techniques and relate to it as motivation for our work. The points of discussion that aid in understanding and framing the techniques described by related work are presented first, followed by a survey of relevant overlay design techniques for topic-based and content-based pub/sub systems.

We note that overlay design of the broker network is a recent trend for pub/sub systems. The traditional practice has been to connect brokers in an ad-hoc fashion without regard for the performance impact of the broker overlay network. As pub/sub systems have garnered greater adoption for large-scale distributed applications in recent years, the performance of the broker overlay has become increasingly important.

3.1 Terminology and Background

The scope of this survey is limited to overlay design techniques that aim to improve the performance of a given pub/sub system, as opposed to improving its reliability. The *performance objective* of an overlay design technique describes the primary goal of producing the overlay design. The most common performance objective is to reduce *messaging traffic* in a network, which is defined as the number of messages generated for a publication as it traverses the over-

lay to all destinations. A message is generated by each broker that forwards a publication to a neighboring broker. For example, in Figure 3.1 (a), as a publication travels from Broker 1 to Broker 6, it is forwarded by four brokers, resulting in four messages being generated to deliver the publication. Part (b) of the figure shows an alternate overlay configuration, which results in only two message generated. A reduction in message traffic may produce several benefits, including improved delivery latencies, lower processing overhead at the brokers, and lower bandwidth consumption in the communication links.

Most overlay design techniques attempt to reduce message traffic by establishing direct overlay links between brokers that are predicted, or known, to match a similar set of messages. Algorithms identify favorable overlay partners by measuring the *similarity* of the subscription sets of a given pair of brokers, and establish links between those brokers that are deemed to have high similarity. The process of designing an overlay based on similarity measurements is called *interest clustering* [25]. The goal of interest clustering is to build an overlay that clusters brokers with similar interests, thereby increasing the likelihood that a message matching at one broker will also match at neighboring brokers. The similarity of a given pair of brokers is measured using a *similarity metric*. Clustering brokers with high similarity aims to reduces the incidence of *pure forwarding brokers* along routing paths. These are brokers that do not contain a matching subscription, and simply forward the message. In the example in Figure 3.1 (a), Broker 3, Broker 4, and Broker 8 are pure forwarding brokers.

Overlay design algorithms can be classified as *design-time* or *runtime*. Design-time approaches assume the information necessary for making intelligent overlay modification decisions is available at a central point, where improved overlay designs can be computed and implemented. Runtime approaches modify the structure of an overlay while the pub/sub system is operational using distributed protocols, and require an *information exchange* among the set of brokers such that they may locally decide if, and with whom, an overlay modification is appropriate. If brokers in runtime systems do indeed decide to restructure the overlay, a *rewiring protocol* is used to implement the change.

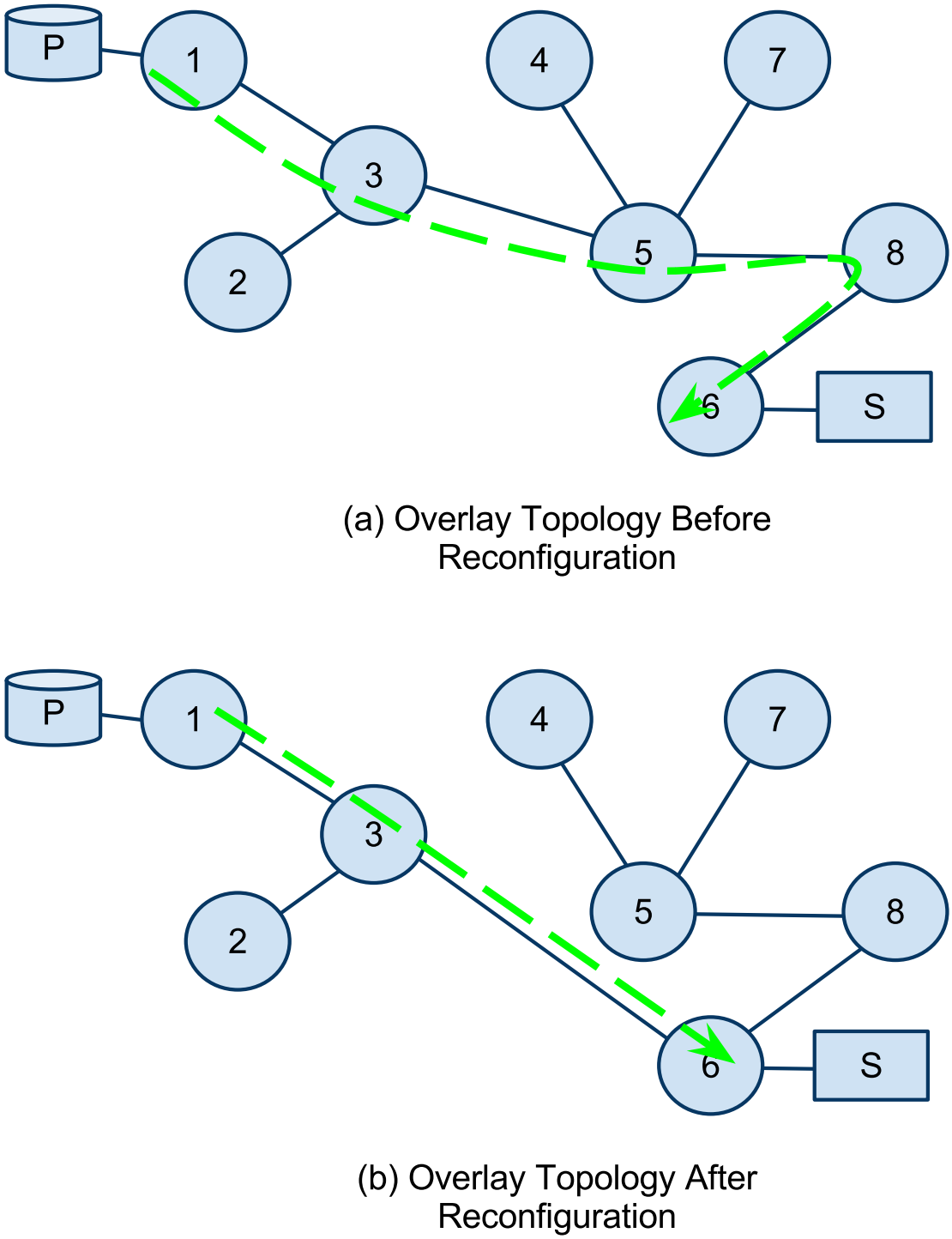


Figure 3.1: Overlay topology reconfiguration to reduce message traffic

3.2 Topic-based Publish/Subscribe

Overlay design for topic-based pub/sub systems has been a popular area of research [3, 10, 23, 8], with much of the work focusing on large scale peer-to-peer networks that implement the pub/sub communication paradigm [3, 10, 23]. The primary performance objective for these systems has been to construct overlays that reduce message traffic. An advantage of topic-based pub/sub is that the similarity metric for topic-based systems is straightforward to define and measure: a given pair of topics are either identical or different. Thus, the clustering process aims to connect nodes that subscribe to the same topics.

Early works clustered similar nodes by forming a dedicated overlay for each topic, called a *topic overlay* [3]. Publications could therefore be routed to the appropriate topic overlay and flooded to all nodes. The performance objective then became efficiently routing messages to the appropriate topic overlay, a process called *outer-cluster routing* [3]. The authors of [3] addressed this issue by having nodes randomly exchange local subscription sets, thereby making other nodes aware of possible access points to the topic overlay. Thus, a node that received a publication on a particular topic could simply disseminate it to the topic overlay itself, if it was a member, or otherwise forward it to the appropriate access point for the topic. A significant amount of information exchange was required, however, to inform nodes of access points for each topic.

A limiting factor of having a dedicated overlay per topic was the excessive number of overlay links that had to be managed by nodes interested in many topics. To both reduce message traffic and the number of links in peer-to-peer topic-based pub/sub systems, Chockler et al. [10] introduced the notion of a *topic-connected* overlay, which requires the sub-graph consisting of nodes with at least one common topic to be connected. Constructing topic-connected overlays with a minimum number of overlay links was shown to be an NP-complete problem, and a heuristic algorithm, called GM, was presented to solve the problem. GM starts with a network with no overlay links, and then iteratively establishes a link between the pair of nodes that maximally increases the number of topic-connected topics. The total number of links is reduced as

nodes with multiple topics in common are connected through a single overlay link.

Whereas GM, in effect, reduced the average node degree of a network, Onus et al. [23] sought to create topic-connected overlays where the maximum node degree was minimized. They recognized that networks with large maximum node degree (relative to the average node degree of the network) may hinder scalability due to uneven load distribution. A greedy heuristic was presented to solve this issue, but at the cost of a much larger running time than GM.

An additional work on topic-connectivity was presented by Chen et al., that uses divide and conquer techniques to connect multiple topic-connected overlays [8]. However, as with GM, the proposed algorithm does not limit the maximum degree of the network.

Much of the work presented in this thesis brings overlay design contributions for topic-based systems into the content-based realm. For example, the advertisement-subscription similarity metric in Chapter 4 and hybrid overlay construction algorithm in Chapter 5 are analogous to efficient outer-cluster routing [3], as they reduce the overlay distance between publishers and subscribers. In addition, the discussion and evaluation of star-like and chain-like overlays in Chapter 5 is motivated by the need to reduce the maximum node degree of a network. We also extend the notion of limiting the maximum node degree by explicitly considering broker processing capacity constraints in Chapter 6.

3.3 Content-based Publish/Subscribe

For content-based pub/sub systems, overlay design has primarily been investigated as a runtime problem [16, 5, 21, 4] for systems that implement a tree-based overlay topology, as is typical of most pub/sub research implementations [7, 22, 13]. The problem thus translates into devising distributed protocols to first determine a new overlay connection that better meets the performance objective, and second to perform the rewiring. The various runtime techniques all execute the following steps, with slight variation, to establish new connections.

First, there is an evaluation phase where a broker evaluates potential overlay connections

with other brokers in its local neighborhood to determine if the rewiring protocol should be initiated. A broker will communicate with other brokers in its neighborhood to evaluate potential overlay links with respect to the similarity metric of the system. It will subsequently propose a new link if one is found that produces greater value of the similarity metric than an existing link. Second, there is a consensus phase, where neighboring brokers agree if a proposed rewiring will be beneficial for the system as a whole. Since the evaluation and consensus protocols are distributed, these stages require a significant amount of information exchange among the brokers to measure similarity and organize the consensus process. Next, if there is indeed consensus for the proposed new link, the third phase is executed, where a link to be torn down is identified. This is necessary in order to maintain a tree-based overlay, since the addition of a new link will create a cycle. From the cycle that results with the addition of the proposed link, the link that contributes the least similarity is the one that is removed. Fourth, the routing tables are updated to reconfigure the network. This last phase requires a synchronization mechanism where traffic is halted among the affected brokers in order to avoid message loss during the rewiring process.

The primary difference in each of the runtime overlay design approaches is the similarity metric used in the evaluation and consensus phases to determine if a new link should be established. In [4], the author's define a "zone of interest" for a broker to be the union of all its local subscriptions. The similarity metric is comprised of the overlapping region of the zone of interests of a given broker pair, divided by the size of the zone of interest of one broker (the broker evaluating the new overlay connection). Each broker measures its similarity with all direct neighbors, and attempts to establish a new link if it finds a broker that raises its overall similarity. This metric is similar to our overlap technique described in Section 4.1.1. The difference is that our overlap metric is symmetric, as we aim to capture the equivalence of a given pair of subscriptions, which in other words corresponds to the likelihood that a publication will match both subscriptions or neither. The asymmetric metric in [4], however, is from the perspective of a particular broker, since the broker evaluating a new overlay link considers

the likelihood that a publication that matches it will also match the new neighbor, but not vice versa. We also differ from [4] in that we consider a covering-based similarity metric, as well as advertisement-subscription similarity when forming an overlay.

The similarity metrics used in [16], [21], and [5] follow a different methodology than that of [4]. Instead of defining similarity in terms of the region of intersection of a set of subscriptions, the metric is defined in terms of the number of recent publications that were matched by subscriptions at both brokers. Thus, in these systems, two brokers have high similarity if, of the recent N publications matched by one broker, many of them were also matched by the other broker. The similarity metrics in [16] and [21] are additionally augmented with node and link cost functions, intended to model the cost of processing a message on a given node and the cost of transmitting a message across a link. Thus, overlay links are selected on the basis that the brokers to be connected have strong similarity, but also with consideration that utilizing one of the brokers or the interconnecting link does not result in excessive cost. It should be noted, however, that costs are left as abstract metrics, without indication of how they may be measured.

Whereas the approaches discussed so far for content-based systems are runtime approaches, our focus is a design-time overlay construction algorithm. Runtime solutions may not be appropriate for certain applications that require low latency performance due to the time required for the four-step distributed protocol to execute, and for the overlay to converge to a near-optimal solution. In addition, a design-time approach may avoid much of the message overhead generated by the information exchange process in runtime systems. Indeed, for the runtime solution in [5], it is reported that the reconfiguration overhead is worthwhile only if the publication rate is at least ten times greater than the rate at which subscription sets change.

Whereas most content-based pub/sub systems are managed systems that implement a broker overlay network, a peer-to-peer content-based pub/sub system is Sub-2-Sub [32], which employs an epidemic-based clustering algorithm for grouping nodes using a similarity metric based on the overlapping regions of subscriptions. A potential scalability issue with Sub-2-Sub

is link management: for each subscription, a node maintains a link for each unique overlapping region. For example, given three nodes with one subscription each, each node must maintain links for (1) the overlapping region common to all three subscriptions, and (2) the region common only to its own subscription and one of the others. In addition, links are managed under the assumption of one subscription per node, therefore nodes with multiple subscriptions are considered multiple "virtual" nodes, thereby increasing the link count even further. In contrast, our techniques are for managed systems where the number of overlay links is explicitly restricted such that a tree-based topology is formed. Thus, our similarity metrics must aggregate the subscription sets per broker.

As an alternative to reducing overlay distance through rewiring the overlay, Cheung and Jacobsen present algorithms for relocating publishers closer to interested subscribers [9]. However, they do not consider relocating subscribers closer to other subscribers that share similar interests. The clustering of subscribers may, in fact, provide a significant performance improvement over the relocation of publishers, as indicated by our evaluation in Section 5.4.4.

3.4 Performance Modeling and Capacity Planning

The overlay design techniques we have considered until now have all assumed a fix set of brokers in the overlay. Ideally a design-time overlay algorithm deploys only as many brokers as necessary, in order to make efficient use of resources. The same holds true in a runtime scenario, where reorganization may lead to an inefficient use of resources if a single broker attracts many overlay connections. In order to avoid congestion, traditional practice is to place a limit on the maximum number of neighbors a broker may have [5]. This can also limit performance if the load imposed by the many neighbors is sufficiently low compared to broker processing capacity. Thus, an overlay algorithm should explicitly consider broker load and capacity constraints when constructing (design-time) or reorganizing (runtime) an overlay network. We are among the first to address this problem with our work in Chapter 6.

A related approach to overlay design with a minimal number of brokers is performance modeling and capacity planning of pub/sub systems [30, 28, 17]. The goal of these works is to estimate the performance of a system based on its configuration and expected workload. In this way, system parameters such as overlay topology and the number of deployed resources can be tuned to optimally satisfy performance objectives. The primary mode of research in addressing these issues has been to develop analytical models, which can then be applied to a given pub/sub system to estimate system performance [28, 17].

Kounev et al. present a methodology for workload characterization and performance modeling of pub/sub systems in order to approximate average message delivery latencies [17]. They present analytical equations and measurement techniques for estimating system parameters such as message service times at a broker and publication routing probabilities. However, some of the techniques require instrumentation of the system under consideration for data collection, therefore the system needs to be available for modification and testing in order to utilize the techniques presented.

Schröter et al. extend the techniques in [17] by providing analytical models for stochastic performance analysis of pub/sub systems [28]. The models incorporate a comprehensive range of system variables including the routing algorithm employed by the system, routing table sizes, and the time taken for a subscription to propagate through the network. These models can then be used to devise algorithms for a variety of pub/sub system design scenarios.

Our approach in Chapter 6 can be seen as complementary to the performance modeling techniques. The purpose of performance modeling is to provide a general framework for quantitatively characterizing the properties of a system, with the intention to use that information for solving design problems; one of which may be overlay design with a minimal set of brokers. In contrast, our goal is to develop algorithms for the specific task of designing a minimal broker overlay network. Accordingly, we provide concrete frameworks and algorithms for doing so.

Chapter 4

Similarity Techniques

As discussed in Chapter 2, the clients in content-based pub/sub systems express interest in information that maps to regions of the content-space. For example, subscribers, through their subscriptions, express interest in receiving information that maps to a specific region of the content-space. Similarly, publishers, through their advertisements, express interest in publishing information that maps to a specific region of the content-space. A given pair of clients are *similar* if they share interest in common regions of the content-space.

Accordingly, this chapter presents concepts and metrics for estimating the similarity between publishers, subscribers, and brokers in content-based pub/sub systems. We first define and motivate the concepts of *commonality* and *interest* to describe the similarity of subscriptions and advertisement-subscription pairs, respectively. We then describe several metrics for calculating similarity. Our work in subsequent chapters uses the concepts and metrics presented here to form low latency and cost-effective overlay designs by clustering clients and brokers with a high degree of similarity.

4.1 Commonality

The term *commonality* is used to describe the similarity of subscriptions. Specifically, the commonality between two subscriptions describes the likelihood that a publication matching

one subscription will also match the other subscription. Similarly, the commonality between two brokers describes the likelihood that a publication matching a subscription at one broker will also match a subscription at the other broker. When calculated for the subscription sets maintained by a set of brokers **B**, a *commonality matrix*, **C**, of size $|\mathbf{B}| \times |\mathbf{B}|$ is produced in which each element, c_{ij}, is a measure of the commonality between brokers b_i and b_j.

4.1.1 Overlap-Based Commonality

As opposed to topic-based pub/sub systems, where topic similarity is a binary condition (two topics are either identical or different), in content-based pub/sub systems, subscriptions may express interest for a range of possible values. Thus, two subscriptions may have only a portion of the range in common, and publications may match both subscriptions in some instances and only one in other instances. Thus, for content-based systems, subscription similarity must be considered in terms of *degree* of similarity.

Overlap-based similarity measures the volume of intersection between a pair of subscriptions. For example, Figure 4.1 shows two subscriptions, s_1 and s_2, in a two-dimensional attribute space. The intersecting, or *overlapping*, region α_{12} represents the content space where a publication will match both subscriptions. The area of overlap in Figure 4.1 (a) is $5 \times 3 = 15$. Intuitively, the larger the intersecting region, the greater the similarity of the subscriptions. However, the absolute value of the region can be misleading. For example, Figure 4.1 (a) provides a larger overlap area ($\alpha_{12} = 15$) than the one shown in 4.1 (b) ($\alpha_{34} = 10$). However, for the same set of publications, only 40% of the publications that match subscription s_1 also match subscription s_2. In Figure 4.1 (b), however, 67% of the publications that match s_3 also match s_4. The subscriptions in part (a) of the figure occupy a larger region of the content space, relative to the intersecting region, thereby leading to increased likelihood of a publication matching one subscription but not the other. It is, therefore, important to consider the ratio of the size of the overlapping area to the size of the subscriptions under consideration. Formally, we define

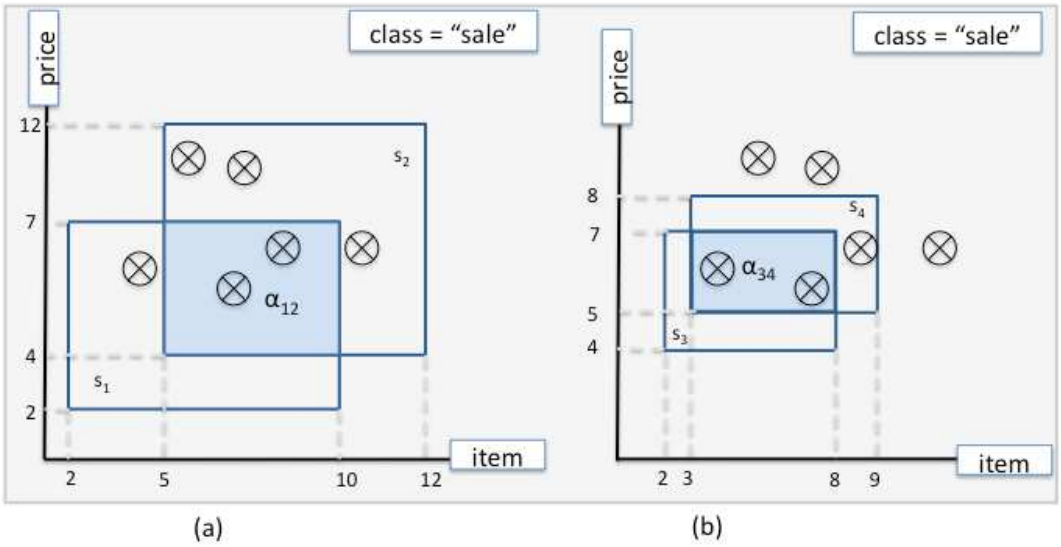


Figure 4.1: Calculating similarity using overlap

the value of overlap between two subscriptions as:

$$o_{xy} = \frac{\alpha_{xy}^2}{\mathcal{A}_x \mathcal{A}_y}, \tag{4.1}$$

where $\mathcal{A}_x$ and $\mathcal{A}_y$ are the volumes of subscriptions s_x and s_y, respectively. The value of overlap ranges from 0 for two disjoint subscriptions, to 1 for two identical subscriptions. The values of overlap for the example shown in Figures 4.1(a) and 4.1(b) are 0.10 and 0.31, respectively.

Thus far we have discussed similarity between two subscriptions, but in order to calculate commonality among a pair of brokers, we have to consider the entire subscription sets maintained by both brokers. We calculate this by summing Equation 4.1 over all subscription pairs for the entire subscription sets for a given pair of brokers. Formally, the overlap-based similarity value c_{ij} between two brokers i and j is defined as:

$$c_{ij} = \sum_{s_x \in |\mathbb{S}_i|} \sum_{s_y \in |\mathbb{S}_j|} \frac{\alpha_{xy}^2}{\mathcal{A}_x \mathcal{A}_y}, \tag{4.2}$$

where $\mathbb{S}_i$ and $\mathbb{S}_j$ represent the local subscriptions of brokers b_i and b_j. The other terms are as defined for Equation 4.1.

The notion of measuring similarity based on the geometric intersection of subscriptions has been utilized in related work [4]. As discussed in Section 3.3, however, we differentiate from the metric used in [4] by employing a symmetric metric, where the magnitude of the overlapping region for a given pair of subscriptions, s_x and s_y, is the same as in the reverse direction. In this way, we measure the equivalence, or "sameness", of a given pair of subscriptions, as opposed to the similarity from the perspective of a particular broker.

4.1.2 Covering-Based Commonality

In this section, we describe a covering-based similarity metric. Covering is an important optimization of content-based pub/sub that describes the concept of subscription subsumption, where a subscription s_y covers a subscription s_x if s_x is entirely encompassed within the attribute ranges of s_y. The reader is referred to [19] for a formal definition of covering, but it may be visualized using the example illustrated in Figure 4.2, where we see the ranges of subscriptions s_2, s_3, and s_4 fall within the range of subscription s_1 along both attribute dimensions. Thus, s_1 covers s_2, s_3, and s_4. In the following, a subscription that covers is referred to as the *coverer*, with a subscription that is covered referred to as the *coveree*.

Prior to describing our covering-based similarity metric, we describe two properties of covering that differentiate it from the overlap technique. First, the value of covering, ζ_{xy}, is a binary relationship. Thus, for a pair of subscriptions s_x and s_y,

$$\zeta_{xy} = \begin{cases} 1 & \text{if } s_x \text{ covers } s_y \\ 0 & \text{otherwise} \end{cases}$$

Second, the covering relationship between a pair of subscriptions is asymmetric: by definition, a coveree can not cover a coverer unless both subscriptions are identical. In developing a covering-based similarity metric, we wish exploit these two aspects of the covering semantic that differentiate it from the geometric intersection technique of the overlap metric.

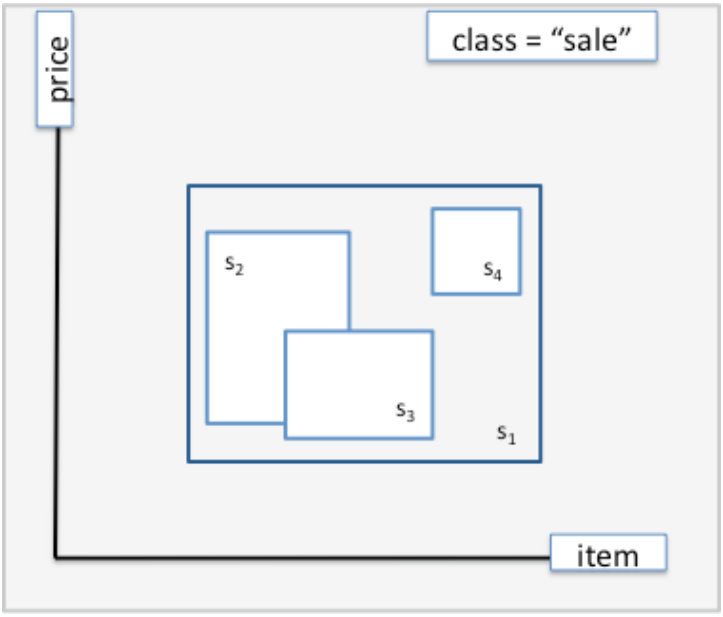


Figure 4.2: Calculating similarity using covering

Accordingly, the covering metric is developed as follows. Figure 4.2 shows a coverer, s_1, and three coverees, s_2, s_3, and s_4. The figure shows that a single coverer may have multiple coverees, and moreover, the coverees can together fill much of content-space occupied by the coverer. Thus, there can be a high degree of similarity among a coverer and its coverees.

To capture the notion of multiple coverees combining to fill the region occupied by the coverer, it is necessary to iterate over the entire subscription set of the coveree broker. Therefore, the value of covering similarity, $\overrightarrow{u_{xj}}$, between a single subscription coverer s_x and the subscription set of broker b_j is defined as

$$\overrightarrow{u_{xj}} = \frac{\mathcal{A}_j^x}{\mathcal{A}_x} \frac{\gamma_{xj}}{|\mathbb{S}_j|} \tag{4.3}$$

where,

$$\gamma_{xj} = \sum_{s_y \in \mathbb{S}_j} \zeta_{xy} \tag{4.4}$$

Here, $\overrightarrow{u_{xj}}$ is the similarity measurement and ranges in value from 0 to 1. The vector notation indicates the asymmetric nature of the calculation. $\mathcal{A}_j^x$ is the volume of the subscriptions

in broker b_j that are covered by subscription s_x. Note that $\mathcal{A}_j^x$ represents the merged volume, therefore the region common to the overlapping coverees is counted once (cf. s_2 and s_3 in Figure 4.2). $\mathcal{A}_x$ is the volume of subscription s_x. The γ_{xj} term provides a count of the number of subscriptions that are covered by the coveree. We divide γ_{xj} by the total number of subscriptions in the coveree set, $|\mathbb{S}_j|$, in order to give a higher similarity value when a larger percentage of the total number of subscriptions are covered.

As in the case of the overlap metric, the value of covering between two brokers, $\overrightarrow{c_{ij}}$, is defined as the aggregated value over the entire subscription sets.

$$\overrightarrow{c_{ij}} = \sum_{s_x \in |\mathbb{S}_i|} \frac{\mathcal{A}_j^x}{\mathcal{A}_x} \frac{\gamma_{xj}}{|\mathbb{S}_j|} \tag{4.5}$$

4.2 Interest

Thus far we have defined similarity in terms of subscriptions, which will enable us to cluster clients and brokers based on the similarity of their subscription sets in later chapters. Advertisements, when measured in conjunction with subscriptions, offer an additional tool for estimating similarity. The region of the content space common to both an advertisement and a subscription provides an estimate of the likelihood that a publication induced by the advertisement will match the subscription. For example, if an advertisement and subscription are disjoint, no publications on the advertisement will match the subscription. Conversely, a large region of intersection between an advertisement and subscription is likely to result in many publications induced by the advertisement to match the subscription.

We use the term *interest* to describe advertisement-subscription similarity. Specifically, the interest of an advertisement-subscription pair describes the likelihood that a publication induced by an advertisement will also match the subscription. The interest between two brokers describes the likelihood that a publication emanating from one broker will match a subscription at the other broker. When calculated across a set of brokers **B**, we get an *interest matrix*, **I**, of

size $|\mathbf{B}| \times |\mathbf{B}|$, where each element i_{ij} indicates the interest between brokers b_i and b_j.

In contrast to the commonality measurement, where we aim to capture the equivalence of a given pair of subscriptions (and thus the size of both subscriptions are important), the interest measurement aims to capture the likelihood that a publication will match a given subscription. Since a publication must fall within the range specified by an advertisement, we calculate interest by measuring the overlapping region as a ratio of the size of the advertisement. Therefore, the interest, $\overrightarrow{v_{xy}}$, between a single advertisement, a_x, and subscription, s_y, can be calculated as follows.

$$\overrightarrow{v_{xy}} = \frac{\alpha_{xy}}{\mathcal{A}_x}, \tag{4.6}$$

where α_{xy} is the area of the intersecting region between a_x and s_y. This value is divided by the size of the advertisement, $\mathcal{A}_x$.

The interest between a pair of brokers, b_x and b_y, can be computed by summing the interest over the set of advertisements of b_x and set of subscriptions of b_y as follows. Let $\mathbb{A}_x$ be the set of advertisements of b_x, and let $\mathbb{S}_y$ be the set of subscriptions of b_y.

$$\sum_{a_x \in |\mathbb{A}_x|} \sum_{s_y \in |\mathbb{S}_y|} \frac{\alpha_{xy}}{\mathcal{A}_x} \tag{4.7}$$

Similar to the covering-based commonality metric, the interest metric is asymmetric, since we are measuring the intersecting region of advertisements of one broker with subscriptions of another broker.

4.3 Extending Similarity with Publications

Currently, the commonality and interest similarity metrics do not assume knowledge of publication distributions or publication rates. The metrics are most accurate, therefore, when publications are uniformly distributed across the advertisement and subscription space. The full

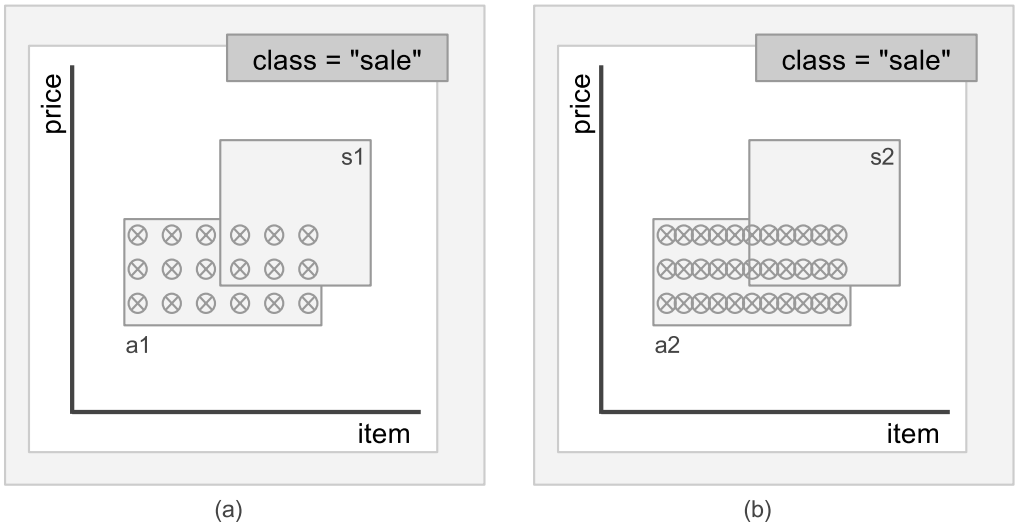


Figure 4.3: The effect of publication rates on the interest metric

details of incorporating publication information into our similarity metrics is outside the scope of this work, however, in this section we consider the feasibility of doing so.

To examine the impact of publication rates, consider the example in Figure 4.3, which shows two sets of advertisement-subscription pairs. The advertisements and subscriptions in the figure occupy identical regions of the content-space, and publications are uniformly distributed across the advertisement space. The advertisement in part (b) of the figure, however, has a greater publication rate than in part (a). Currently, our interest metric would assign identical values to both advertisement-subscription pairs, however, if publication rates are considered, part (b) of the figure should be assigned a greater value as it results in more publications matching a subscription.

To examine the impact of non-uniform publication distributions, consider the example in part (a) of Figure 4.4. The majority of publications induced by advertisement $a1$ in the figure are concentrated in a specific region. Part (b) of the figure shows one technique for supporting non-uniform publication distributions: the advertisement or subscription can be cropped to expose only the concentrated region ($a1'$ in the figure), and the similarity metric can be calculated over this region. A difficulty may arise in computing the concentrated region when each attribute has a separate distribution, since the intersecting region may not form a hyper-rectangle

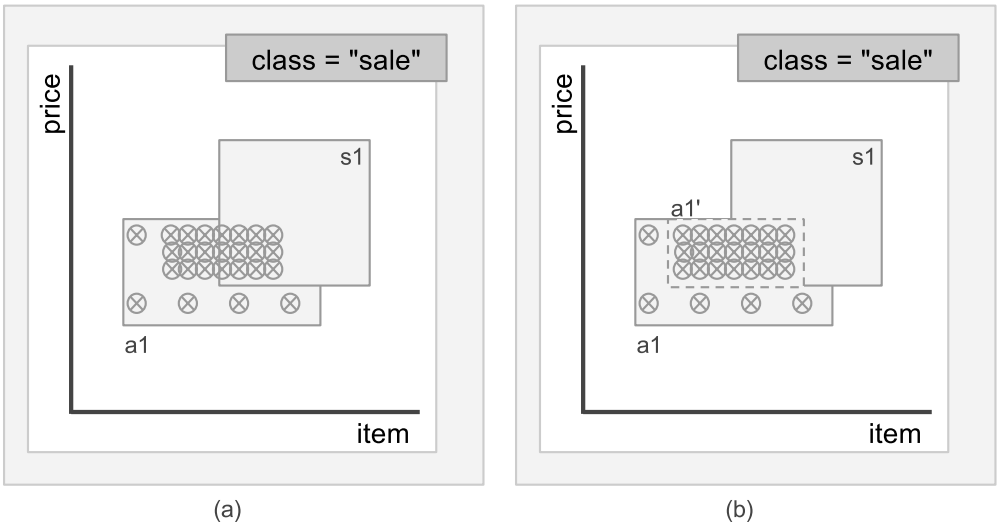


Figure 4.4: Supporting non-uniform publication distributions

in the content-space.

4.4 Chapter Summary

This chapter has presented definitions and metrics for two types of similarity. First, commonality, defined as subscription similarity, measures the likelihood that a publication matching one subscription (or subscription set) will also match another. Second, interest, defined as advertisement-subscription similarity, measures the likelihood that a publication belonging to an advertisement will match a subscription. The similarity metrics serve as tools for the overlay design process. As we show in subsequent chapters, the performance of overlays can be improved by clustering brokers with strong similarity. In addition, the similarity metrics are used for estimating the load imposed on a broker by its local clients.

Chapter 5

Fixed-Broker Overlay Design

This chapter presents algorithms for constructing broker overlay topologies for content-based pub/sub systems. The objective is to determine the overlay links between brokers that reduce message traffic and message latency. In order to achieve this, the algorithms cluster brokers based on the similarity of their subscription sets, using the commonality metrics described in Chapter 4. In this way, message count and message latencies are improved through a reduction in the number of pure forwarding brokers as publications are routed to interested subscribers.

Also presented is an overlay design algorithm that constructs topologies by combining the commonality and interest metrics discussed in Chapter 4. This algorithm clusters brokers based on advertisement-subscription similarity in addition to subscription similarity, and thereby reduces the overlay distance between publishers and interested subscribers. The evaluation shows that overlays constructed using both interest and commonality improve performance over commonality-based overlays alone.

Lastly, the discussion and evaluation in this chapter are geared towards measuring the quality of an overlay, not only in terms of performance, but also the distribution of load. In overlays with uneven load distribution, where the majority of messages are routed through a small subset of brokers, latencies may increase due to processing delays if brokers become congested. The discussion serves as a preamble to the next chapter where overlays are designed with explicit

consideration for the load imposed on a broker by its local clients.

5.1 Problem Formulation

Given a set of brokers containing the advertisements and subscriptions of locally connected clients, the goal is to construct a broker overlay topology that minimizes the number of messages generated per publication, referred to as message count, as well as end-to-end delivery latencies. Formally, the problem is to construct a tree-based overlay network $T = (\mathbf{B}, \mathbf{E}')$ from a fully connected graph $G = (\mathbf{B}, \mathbf{E})$, where $\mathbf{B}$ is the set of brokers, and $\mathbf{E}$ and $\mathbf{E}'$ are the sets of available and selected overlay links, respectively. The edge set $\mathbf{E}$ includes all the communication links between every pair of brokers, while $\mathbf{E}'$ consists of the edges that form a tree-based overlay, where $\mathbf{E}' \subset \mathbf{E}$. The input and output parameters are defined as follows.

Input

1. Set of brokers, $\mathbf{B} = \{b_i\}$, publishers, $\mathbf{P} = \{p_j\}$, and subscribers, $\mathbf{S} = \{s_k\}$
2. Set of publishers and subscribers connected to each broker b_i, $\mathbf{P}_i$ and $\mathbf{S}_i$, respectively
3. Set of advertisements, $\mathbb{A} = \{\mathfrak{a}_x\}$, and subscriptions, $\mathbb{S} = \{\mathfrak{s}_y\}$
4. Set of advertisements produced by each publisher p_j, $\mathbb{A}_j$, and set of subscriptions submitted by each subscriber s_k, $\mathbb{S}_k$

Output

Broker overlay topology defined as a $|\mathbf{B}| \times |\mathbf{B}|$ adjacency matrix, $T_{|\mathbf{B}|\times|\mathbf{B}|}$

Objectives

1. Minimize average message count per publication delivery
2. Minimize average end-to-end delivery latencies

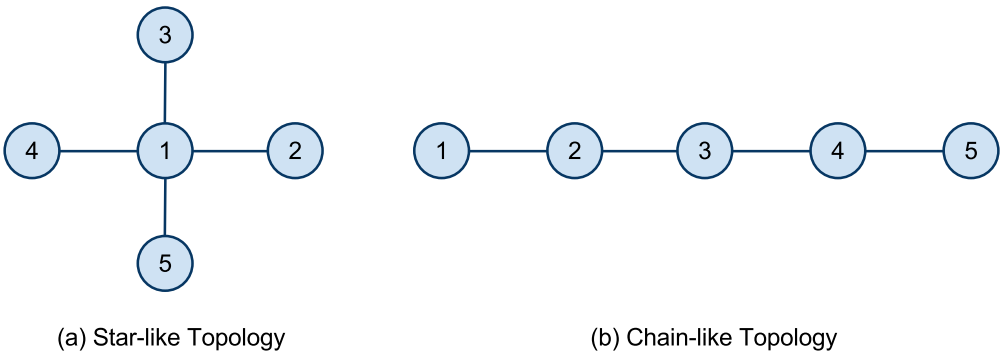


Figure 5.1: Star-like and chain-like overlay topologies

5.2 Overlay Topologies and Performance Metrics

For the purpose of optimally reducing message traffic and message latencies, an ideal content-based pub/sub system consists of a single broker, to which all clients connect. In this way, no extra messages are produced, and publications do not experience overlay link latencies. However, if message rates are sufficiently high, the single broker may become overloaded, and publications may be delayed due to processing delays caused by broker congestion. In order to resolve this scalability issue, the concept of distributed pub/sub was introduced [7], with the aim of splitting clients across multiple brokers in order to better balance the load. Accordingly, clients connected to different brokers, but sharing similar interests, communicate through overlay links.

The existence of overlay links, however, causes messages to experience link latencies. Accordingly, the *diameter* of an overlay, or the largest end-to-end path in the overlay, provides an indication of the performance of a network, measured in terms of message traffic and message latencies. Since one message is produced for each overlay link a publication traverses, a longer diameter will produce more messages. Similarly, longer overlay paths lead to longer end-to-end delivery latencies. Thus, it is desirable to reduce the diameter of an overlay.

For a given set of brokers, the diameter of an overlay can be reduced by connecting more brokers to a single broker. For example, in the overlay topology in Figure 5.1 (a), all brokers connect to a single broker (Broker 1). Thus, a message can be sent from any one broker to

any other in a maximum of two hops. An overlay in which the diameter is much smaller than the number of brokers is defined as a *star-like overlay*. While a star-like overlay may reduce message count and latency, it is vulnerable to an uneven load distribution, as a small subset of brokers must process a disproportionately larger number of messages compared to other brokers. Thus, in Figure 5.1 (a), Broker 1 will be required to process all messages in the system since it is connected to all other brokers. Accordingly, a star-like overlay is highly dependent on the performance of a few brokers in the network; if one of the *central* brokers (Broker 1 in the figure) experiences congestion, the majority of messages will experience delivery delays.

Part (b) of Figure 5.1 shows an alternate configuration. In this example, the diameter of the network is larger, where in the worst case, a message will traverse four hops (to get from Broker 1 to Broker 5). An overlay with a large diameter is referred to as a *chain-like overlay*. A chain-like overlay may be less dependent on the performance of a single broker, but may increase delivery latencies due to messages traversing an increased number of hops. It should be noted that a chain-like overlay does not necessarily avoid broker congestion. For example, if there is frequent communication between Brokers 2 and 5 in part (b) of the figure, Brokers 3 and 4 must process a large number of messages, possibly congesting either broker.

Thus, when designing broker overlay networks, it is important to consider the trade-off between performance and load distribution across the set of brokers. The metrics below are used in this chapter to evaluate overlay designs produced by our algorithms in light of the discussion above. The first two metrics define our performance objectives, and the last two help to estimate the likelihood that a particular overlay design will induce broker congestion based on whether it is more star-like or chain-like.

Message Count: As mentioned previously, this metric indicates the number of messages generated for a publication as it travels from its source broker to a destination broker. A message is generated by a broker for every publication it forwards along the network. Since messages consume processing cycles at brokers and network bandwidth across the links, a good overlay reduces message count.

Message Latency: The metric measures the time taken by a message to travel from its source broker to a destination broker. A good overlay is expected to provide a low value for this metric.

Node Degree: The node degree of a broker is the number of other brokers with which it has a direct overlay connection. Nodes with higher degree can be expected to process more messages, and therefore experience greater load, possibly leading to congestion. In a connected, tree-based overlay network with N brokers, the overlay topology consists of $N-1$ bidirectional overlay links. Because there are a fixed number of overlay links and a fixed number of brokers, the average number of overlay links per node is constant. Accordingly, the distribution of degrees is of importance in a tree-based overlay, and indicates whether the overlay is more star-like or chain-like. Overlays where a small number of brokers have a degree that is much larger than that of the majority of brokers are star-like. In contrast, overlays in which the maximum node degree is near the median node degree are chain-like.

Number of Messages Processed: This metric measures the number of messages disseminated by a broker over a period of time, and thus provides an indication of the load on a broker. A good overlay produces a low average value for this metric by reducing the message count. Additionally, the maximum value of this metric across the set of brokers indicates whether the topology is more star-like or chain-like. A maximum that is much greater than the average (thus the maximum value contributes significantly to the average) results in a star-like topology with uneven load distribution, as one broker is disseminating a large proportion of the messages.

5.3 Overlay Topology Construction

This section describes algorithms for constructing broker overlay topologies. Two types of overlay construction algorithms are presented. First, a commonality-based algorithm is presented that constructs overlay topologies based on the similarity of the subscription sets of the brokers. Second, a hybrid algorithm is described that considers interest in addition to common-

ality when measuring similarity.

5.3.1 Overlap-based and Covering-based Commonality Overlays

In order to construct overlay topologies, the broker overlay network is modeled as a graph, with brokers represented by vertices, and communication links represented by edges (cf. Section 5.1). The overlay construction methodology then follows a two-step process. First, similarity is calculated between every pair of brokers to determine the broker pairs that have the most similar subscription sets. Second, a maximum spanning tree algorithm is executed using the similarity measurements as weights, to determine the overlay configuration that maximizes overall similarity. The overlay links are then set equal to this maximum spanning tree.

The pseudo code for the overlay construction algorithm is presented in Algorithm 1. The algorithm accepts as input the list of brokers in the overlay, *DeployedBrokers*, and the commonality metric with which to compute similarity between the broker pairs, *commonalityMetric*. This metric may be either the overlap or covering metric from Chapter 4. Then, for every broker pair b_x and b_y, the $computeCommonality()$ function in line 8 applies the appropriate commonality equation (Equation 4.2 for the overlap metric or Equation 4.5 for the covering metric) to determine the weight between brokers b_x and b_y. Once all brokers have been processed and the commonality matrix, $\mathbf{C}$, is computed, a maximum spanning tree algorithm is applied to the weights to determine the overlay topology that maximizes similarity. Since in a tree-based overlay with N brokers, the total number of overlay connections is limited to $N-1$ links, by maximizing similarity, we create direct overlay links between brokers that share the most similar subscription sets, subject to the constraint that the overlay does not contain a cycle.

The maximum spanning tree algorithm used for building the tree depends on the commonality metric. If the commonality metric is bidirectional (thus the weights are bidirectional and the commonality matrix, $\mathbf{C}$, is symmetric), traditional spanning tree algorithms such as Prim's algorithm [24] may be used to efficiently build the tree. In the case of a directional common-

Algorithm 1 Commonality Overlay Construction

```
1: Input DeployedBrokers, commonalityMetric
2: C ← ∅
3: for all b_x ∈ DeployedBrokers do
4:    for all  b_y ∈ DeployedBrokers do
5:       if (b_x == b_y) then
6:          continue
7:       end if
8:       C(b_x, b_y) ← computeCommonality(commonalityMetric, b_x.subs, b_y.subs)
9:    end for
10: end for
11: T = computeMaximumSpanningTree(C)
```

ality metric, Edmond's algorithm [12] for computing optimal directed spanning trees is used. In this latter case, the overlay is configured according to the maximum directed spanning tree, although the overlay links are indeed bidirectional.

5.3.2 Hybrid Topologies

In this section, we introduce a hybrid overlay design algorithm that improves system performance through the inclusion of interest (in addition to commonality) when constructing an overlay topology. As discussed in Chapter 4, conceptually, interest can improve system performance by reducing the distance between publishers and subscribers that share a significant amount of similarity.

It is important to consider that a topology based exclusively on either commonality or interest may not provide the best performance, as the choice between commonality and interest represents a trade-off. Consider the scenario in Figure 5.2, where the dashed links represent high commonality between the respective brokers, and the dotted links represent high interest. To avoid cycles, only three of the four links can be chosen. If commonality is favoured over interest, either messages published from Broker A that match at Broker B, or messages published from Broker D that match at Broker C, will have to be routed along a longer part of the network. Conversely, if the interest links are favoured, then messages matching subscriptions

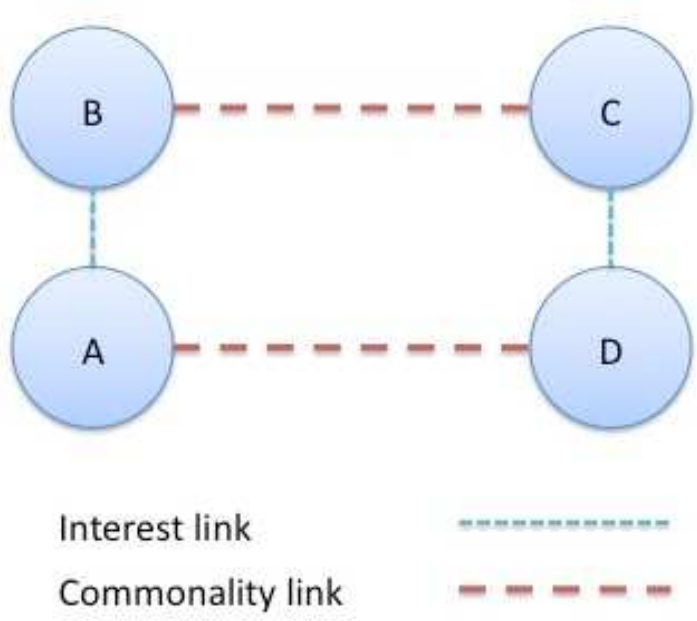


Figure 5.2: Interest vs. commonality trade-off

at both Brokers B and C (perhaps from another node in the network), will require additional routing. Clearly, a compromise is required, where neither approach is given exclusive preference.

In order to facilitate this compromise, and also to use the optimal spanning tree methodology described earlier in this section, we compose a commonality-interest hybrid matrix. Let **C** and **I** represent the commonality and interest matrices, respectively. The commonality matrix may be calculated using either of the commonality techniques from the previous chapter, with the interest matrix calculated using the technique described in Section 4.2. A hybrid matrix, H, is defined as follows.

$$\mathbf{H} = \alpha\mathbf{I} + \beta\mathbf{C}, \tag{5.1}$$

where $\beta = 1 - \alpha$. α and β are priority factors that specify the relative preference given to the interest and commonality matrices, respectively. These factors may be determined empirically, but our experiments (Section 5.4.4) show that performance is most improved when α is between 0.05 and 0.2. This finding–that commonality requires higher priority than interest–is consistent

with our pub/sub model where the number of subscriptions is much greater than the number of advertisements. Subscription weights, therefore, should play a greater role in the design of the overlay.

Algorithm 2 Hybrid Overlay Construction

```
 1: Input DeployedBrokers, α, commonalityMetric
 2: I ← ∅
 3: C ← ∅
 4: H ← ∅
 5: for all b_x ∈ DeployedBrokers do
 6:    for all b_y ∈ DeployedBrokers do
 7:       if (b_x == b_y) then
 8:          continue
 9:       end if
10:       I(b_x, b_y) ← computeInterest(b_x.advs, b_y.subs)
11:       C(b_x, b_y) ← computeCommonality(commonalityMetric, b_x.subs, b_y.subs)
12:    end for
13: end for
14: normalize(I)
15: normalize(C)
16: β ← 1 − α
17: H ← αI + βC
18: computeMaximumDirectedSpanningTree(H)
```

The hybrid overlay construction algorithm is described in Algorithm 2. Hybrid topologies are constructed by first computing **C** and **I** independently of each other given the subscription and advertisements sets of brokers (lines 10 and 11). Next, after the matrices have been normalized, Equation 5.1 is applied to form the hybrid matrix **H** (line 17), followed by computation of the maximum directed spanning tree on **H**. In our evaluation, we only consider the case where the commonality metric is the bidirectional overlap technique. It is left for future work to consider other possible combinations, such as a hybrid composed of a cover-based **I** and overlap-based **C**. This combination potentially leverages the case where interest is given priority over commonality when a publication is guaranteed to match a subscription (as is the case when a subscription covers an advertisement), and favours commonality otherwise.

5.3.3 Complexity Analysis

The complexity of the overlay construction algorithms presented in this section is determined as follows. The notation used below is in accordance with definitions in the problem formulation in Section 5.1. The commonality overlay design algorithm computes weights for a given broker pair by determining the commonality of each subscription in one broker with all the subscriptions in the other broker. The process is repeated for every pair of brokers in order to compute the commonality matrix of the system. The effect is that every subscription in the system is compared with all other subscriptions (minus local subscriptions). Thus, the average time for the algorithm to determine weights between every pair of brokers is $O(|\mathbb{S}|^2)$.

To compute the overlay topology given the weights, a spanning tree algorithm must be executed. If a symmetric similarity metric is used, such as the overlap metric, a bidirectional spanning tree algorithm can be used. Prim's algorithm [24] executes in $O(|\mathbf{B}|^2)$ time. For a asymmetric metric, such as the cover metric, a directional spanning tree algorithm must be used to construct the overlay topology. Edmond's algorithm for directional spanning trees also has a time complexity of $O(|\mathbf{B}|^2)$.

Therefore, the total time complexity for the commonality overlay construction algorithm is $O(|\mathbb{S}|^2 + |\mathbf{B}|^2)$. The memory footprint of the algorithm is dependent on the number and size of the subscriptions. A large number of subscriptions, and subscriptions with many attributes, will induce greater memory utilization during algorithm execution.

The time complexity of the hybrid algorithm is similar to that of the commonality algorithm, but with the additional requirement to compute the interest matrix. The interest matrix can be computed in $O(|\mathbb{S}|\,|\mathbb{A}|)$ as every subscription is compared with every advertisement. Accordingly, the time complexity of the hybrid algorithm is $O(|\mathbb{S}|^2 + |\mathbb{A}|\,|\mathbb{S}| + |\mathbf{B}|^2)$.

Note that in our pub/sub model, the number of subscriptions is greater than both the number of advertisements and the number of brokers, therefore, the computation of the commonality matrix dominates both time and space complexity.

5.4 Evaluation

In this section, we evaluate the performance of our overlay topology construction algorithms. First, commonality overlay topologies constructed using Algorithm 1 are evaluated. The overlays are built using the overlap and covering commonality metrics from Chapter 4, and compared against one another. The overlays are evaluated for performance as well as for predicted load distribution (cf. Section 5.2). Next, the hybrid algorithm (Algorithm 2) is examined to determine the advantages of considering interest in addition to commonality. We begin this section by describing our simulation framework and workload details.

5.4.1 Simulation Setup

Our algorithms were implemented in Java and evaluated on a distributed content-based pub/sub system built using the JiST discrete event simulator [1]. The experimentation process consisted of two-steps. First, an overlay topology was computed using Algorithm 1 or 2, given a workload (cf. input parameters in Section 5.1). Second, the pub/sub simulator was configured according to the overlay topology and the simulation was executed by disseminating generated publications. Data was collected during the simulation, followed by the computation of results. The performance metrics presented are based on the evaluation criteria discussed in Section 5.2. The experimentation process was repeated for each commonality metric, with an identical workload provided. All results are averaged over five independent runs per overlay topology. For the latency results, all link latencies were set to 100 ticks. Note that a tick is intended to model a fraction of a second in a live system, such as a microsecond or millisecond, depending on system parameters.

For additional comparison, a baseline overlay topology was constructed by randomly connecting the brokers in a tree structure. The baseline topology is intended to model traditional pub/sub implementations that do not give explicit consideration to the design of the broker overlay. It mimics an approach taken by an administrator of a system who may be unaware of

Table 5.1: Workload Summary

Parameter	Specification
Number of Brokers	450
Number of Advertisements	1500
Number of Subscriptions	4500
Number of Publications	10,000
Number of Classes	4
Number of Predicates	2–5
Attribute Value Range	0–1000

performance enhancements possible by clustering brokers that have a high degree of similarity. Accordingly, the baseline topology is used as a reference point to gauge the effectiveness of our similarity metrics and overlay construction algorithms.

5.4.2 Workload Specification

The workload used to conduct our experiments is summarized in Table 5.1, and described in more detail below. As there is no formal workload specification for content-based pub/sub systems, and industry data sets are kept private, our workload is modeled around that used by related work [5, 27].

Central to our evaluation methodology is the concept of the regionalism of the workload, which measures the degree of similarity of the subscriptions of a particular broker, as well as the subscription sets of a group of brokers. Specifically, a highly regionalized subscription set implies that the majority of local subscriptions are for similar types of information. For example, in an algorithmic trading system implemented using pub/sub, the majority of the local subscriptions at a particular broker would express interest in stock information. Moreover, the majority of the brokers in the system would also have subscriptions that express interest in stock information. Furthermore, the publications disseminated in the system would also be of stock information. Thus, the system is highly regionalized as the cluster of brokers that comprise the system are interested in similar types of information. It has been shown in [27] that message dissemination using pub/sub is most efficient when the regionalism of the workload is high,

otherwise it may be more beneficial to simply broadcast a message.

We use the following technique to control the regionalism of the workload. First, a *preference class*[1] is assigned to each broker using a uniform distribution. Thus the set of brokers is partitioned into groups, where all brokers in the group have a common preference class. Next, a *bias factor* is introduced, which specifies: of all the subscriptions assigned to a broker, what percentage have the same class as the broker's preference class. Thus, for a bias factor of 0.8, of all the subscriptions assigned to a particular broker, 80% will be of the same class as its preference class. By varying the range of the bias factor from 0 to 1.0, it is possible to control the regionalism of the workload. In the extreme case where the bias factor is set to 1.0, there will be clusters of brokers that have commonality only with other brokers that have the same preference class. In other words, any publications matched at a particular broker can only be matched by other brokers with the same preference class.

We now define in detail our evaluation scenario.

Content-space and Attributes The content-space consisted of four classes and five attributes. Each attribute was randomly assigned a mean value in the range of [0,1000], and was assigned a center point according to a Gaussian function around this mean, with standard deviation of 3.0. The range of each attribute was determined using a Zipf distributed function, with a minimum range of of 10.0, and a skew value of 0.8. Attribute endpoints were capped if the range exceeded the boundaries of the content space.

Subscriptions 4500 range-based subscriptions were generated, with each assigned a minimum of two and a maximum of five predicates, one of which was a class attribute. Classes were distributed across the 4500 subscriptions using a uniform distribution. Of the non-class predicates, each had a fifty percent chance of being assigned a specific range or being designated a *don't-care-attribute*. Don't-care-attributes match for all values in the attribute value

[1] Please refer to Chapter 2 for a description of attribute class values.

range. Subscriptions were distributed to brokers as follows. Each broker was assigned a preference class (uniformly selected from the set of all class values in the content-space), and a generated subscription was assigned to a broker within the set of brokers having the same preference class with probability equal to the bias factor. For example, a subscription with $class = stock$ and bias factor $b = 0.75$ had a 75% probability of being assigned to a broker with a preference class of type *stock*. Otherwise, the subscription was deposited at a broker with a different preference class.

Advertisements 1500 range-based advertisements were generated following similar conventions as the set of subscriptions, unless otherwise specified in this section. Brokers were also assigned a preference class for advertisements, which was identical to the subscription preference class. A minimum of one advertisement was assigned to each broker, with the remaining being distributed using the same bias scheme as for subscriptions. Once the set of brokers among which to deposit the advertisement was selected (i.e., either the set with preference class equal to the class of the broker, or a different preference class), a broker within that group was selected using a Gaussian function with a mean of one-third of the number of brokers in the group and a standard deviation of 3.0.

Publications 10000 point publications were generated by selecting a broker using a Gaussian distribution with a mean equal to one-third the total number of brokers and a standard deviation of 3.0. An advertisement was uniformly selected from the set of advertisements of the selected broker, and the publication was defined as a point within the range of the advertisement. The location of the point within the advertisement was randomly selected for each attribute.

5.4.3 Performance Results

In this section, we evaluate commonality overlay topologies constructed using Algorithm 1 and the overlap and covering similarity techniques. In addition to each other, the topologies are

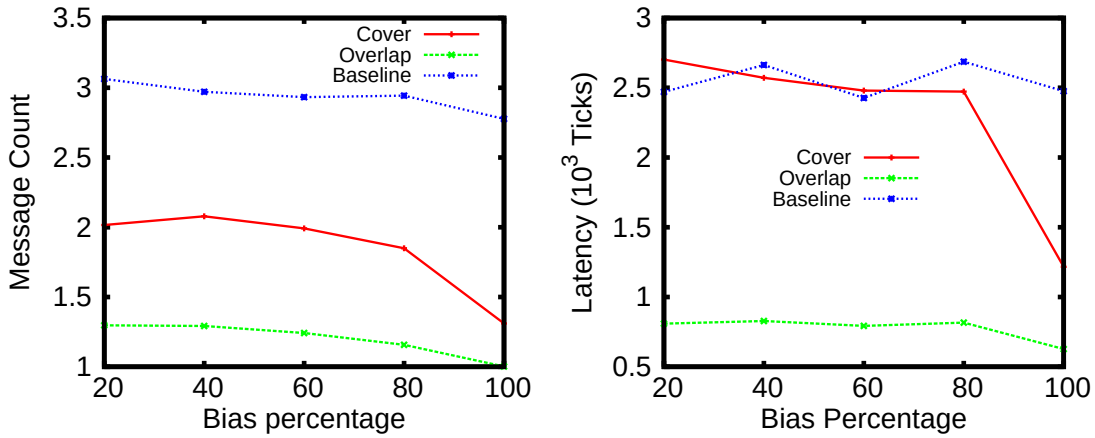


Figure 5.3: Performance implications of subscription regionalism

compared to the baseline technique described previously, that randomly constructs an overlay topology without regard for similarity. We examine the performance, degree distribution, and broker load for each overlay, as per our discussion in Section 5.2. The algorithms are compared by varying the regionalism of the workload using the bias parameter. Note that the term "cover topology" refers to a topology constructed using the cover similarity metric. Similarly, an "overlap topology" refers to a topology constructed using the overlap similarity metric.

The plots in Figure 5.3 depict the performance of the algorithms in terms of average message count and average message latency, which measure the average number of messages generated per publication and average time taken per publication delivery, respectively. Overlap topologies provide superior performance for both metrics across the entire bias range. This is expected as the overlap metric provides an exact measure of similarity, i.e., all overlapping regions are factored into the similarity measurement, and brokers are ranked accurately according to the similarity of their subscription sets. In contrast, covering-based similarity is an "all-or-nothing" approximation technique. Thus, two subscriptions may have a large common region in the content-space, but if they do not satisfy the covering relationship, the similarity will not be accounted for by the algorithm. Accordingly, overlap provides a more accurate measure of similarity, and produces better broker clustering.

When at least eighty percent of the subscriptions and advertisements are regionalized, how-

ever, the cover-based topologies experience substantial improvements. This is due to the fact that increasing regionalism creates a greater distinction between brokers that have similarity, and those that do not, which effectively negates some of the error in approximation produced by the cover metric. For example, at a bias factor of 80%, a given pair of brokers will either be very similar (if they share the same preference class), or very disimilar (if they have a different preference class). Thus, when the algorithm computes commonality with the cover metric, the similar brokers can withstand greater error in approximation since the disimilar brokers do not have much similarity in the first place.

It is interesting to note that for low to moderate regionalism, the cover topology produces lower average message count than the baseline topology, but performs equal in terms of average message latency. To explain this counter-intuitive result, consider the two topologies in Figure 5.4. Both contain one publisher, two subscribers, and five brokers. Additionally, each link delays a message by value L. The two topologies are different in the placement of Broker 1. In the lower part of the figure, Broker 1 has been connected to Broker 4 instead of Broker 2.

Consider a publication traveling to both subscribers S1 and S2. In the top figure, the message will be duplicated at Broker 3 and must travel two hops in each direction, thus the message count will be 4. In the bottom figure, the message will be duplicated at Broker 4, one hop closer to the destination than in the top figure. Thus the message count will be 3. The latencies for both topologies, however, will be constant. Since latency is a measure of the delay experienced by each message delivery from its point of publication, both topologies will produce latencies of 2L for each message.

Therefore, we can explain both graphs of Figure 5.3 as follows. Given that the cover topology produces a lower message count, the cover topology exhibits clustering of similar brokers, thus the cover topology is structurally more similar to the bottom part of Figure 5.4. The baseline topology, however, has not been constructed based on similarity, and therefore is not clustered, resulting in larger message counts. In addition, we can conclude that the cover topology is more star-like than the baseline topology, again due to it producing lower message

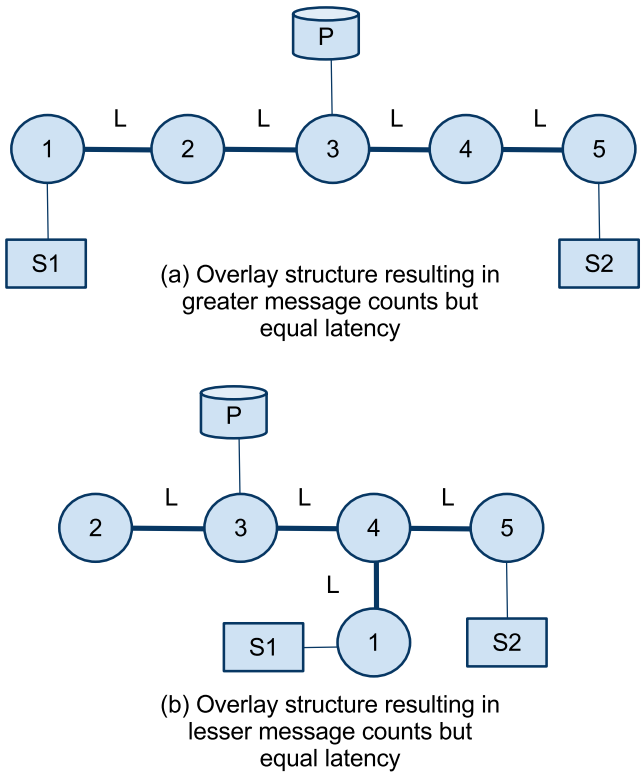


Figure 5.4: Overlay topologies that produce similar latencies but different message counts

counts.

This observation is further confirmed by the plots in Figure 5.5. The error bars in part (a) present the range of node degrees of the ten percent of brokers with the largest number of neighbors. The high and low points represent the maximum node degree and 90 percentile, respectively. The figures show that the overlap topology generally produces larger maximum node degrees than the other algorithms, thus leading to more star-like overlays, and thus a greater reliance on a single broker. Indeed, by using the overlap similarity technique, a broker with a large and diverse subscription set naturally exhibits strong similarity with a large number of brokers in the overlay, leading to a network with a large maximum node degree. The topologies produced by the cover metric result in lower maximum node degree than overlap, but higher maximum node degree than baseline. The baseline algorithm is therefore more chain-like than cover because all brokers have a similar node degree.

Part (b) of Figure 5.5 shows the cumulative distribution function of the node degrees. The

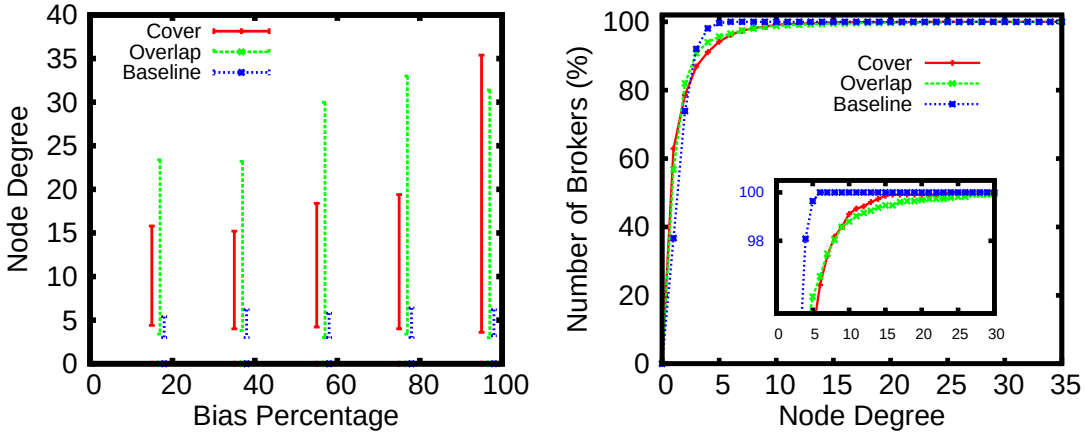


Figure 5.5: Node degree distribution

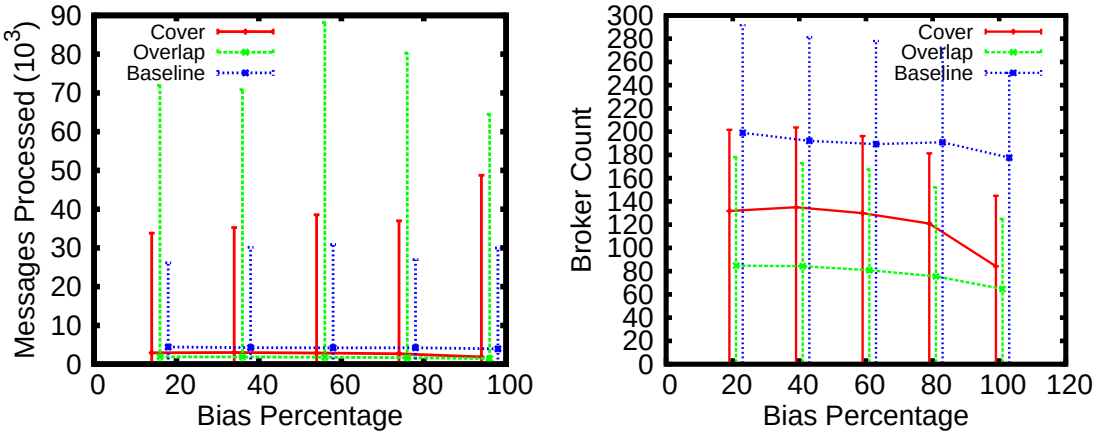


Figure 5.6: Broker load distribution

bends of the respective curves provide insight into the degree distributions of each overlay. The baseline topology quickly ramps to the full set of brokers, thus indicating that the majority of brokers have a similar node degree, with a low maximum. The baseline topology is therefore highly chain-like. The curve for the cover topology, alternatively, has a large number of brokers with few neighbors, but has a longer approach to the maximum set of brokers. Thus, a greater number of brokers have a higher node degree, compared to the baseline network. The cover topology, therefore, offers a different type of distribution than the baseline topology; one where there are many brokers with low degrees, but which are distributed across several brokers with moderately high degrees. In comparison, the overlap curve has a higher bend than the cover curve (thus there are a greater number of brokers with lower node degree compared to the cover topology) but reaches the full set of brokers at a later point, as shown in the inset. This implies that there are fewer brokers with a moderate number of neighbors, and one broker with a very large number of neighbors, and thus the overlap topology is more star-like.

We see the load impact of the topologies generated in Figure 5.6 (a), which shows the average number of messages processed by a broker. In addition, the error bars show the maximum number of messages that were processed by a single broker. The overlap topology produces the best average, but at the expense of a single broker processing a disproportionately large number of messages. Thus, the average message count and latency are superior for the overlap topology (as shown in Figure 5.3), due to the fact that a single broker is connected to a large number of other brokers and is able to deliver messages in fewer overlay hops. The cover topology provides better load distribution at the cost of a slightly higher average. The baseline topology has yet a higher average and better load distribution, and correspondingly, poorer performance due to longer network paths.

Part (b) of Figure 5.6 shows the number of brokers that a message passes through on its path to delivery to all interested subscribers. The overlap topology limits the number of brokers that process a given message to a greater extent than the other topologies, due to the central broker(s) handling the majority of the messages. More brokers are involved in the dissemina-

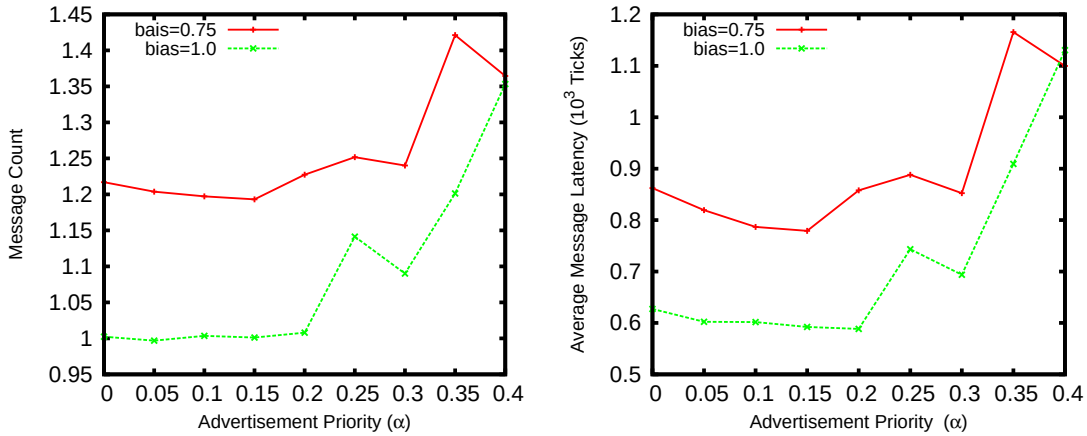


Figure 5.7: Performance improvements with advertisements

Table 5.2: Data points for Figure 5.7

α	Message Count		Message Latency	
	b=0.75	b=1.0	b=0.75	b=1.0
0.00	1.217	1.002	862.246	627.229
0.05	1.204	**0.997**	819.434	602.351
0.10	1.197	1.003	786.766	601.712
0.15	**1.193**	1.001	**779.030**	592.265
0.20	1.227	1.008	857.775	**588.490**
0.25	1.252	1.141	888.017	743.416
0.30	1.240	1.090	852.421	693.766
0.35	1.421	1.201	1165.598	909.304
0.40	1.364	1.353	1099.695	1130.367
Improvement (%)	1.953	0.549	9.651	6.176

tion of messages in the cover topology due to it being more chain-like compared to the overlap topology. We see the negative impact of the baseline topology in that the average number of brokers involved in the baseline topology is greater than or near the maximum of the cover and overlap topologies.

5.4.4 Benefit of Advertisements

In this section, we investigate the benefits of considering interest in addition to commonality when designing overlays.

The plots in Figure 5.7 show average message count and average message latencies of hybrid overlays for advertisement priority (α of Equation 5.1) ranging from 0 to 0.4, in increments of 0.05. The overlap similarity metric was used to produce the commonality matrix in Equation 5.1. Experiments were conducted for bias values of 0.75 and 1.0. The data points of Figure 5.7 are presented in Table 5.2, with the best performance results for each bias value shown in bold text. Note that when $\alpha = 0$, the hybrid overlay is equal to a commonality overlay. Accordingly, the last row of the table indicates the percentage by which the hybrid overlay improves performance over the commonality overlay.

We can infer the following from Table 5.2. First, the data shows that incorporating advertisements into the overlay design process does indeed improve system performance, as the results for both message count and latency are lowest when $\alpha > 0$. Indeed, performance may be increased by nearly 10% with the hybrid overlay design algorithm. For both values of the bias factor, however, the improvement is more pronounced for message latencies than for message count. As per the discussion regarding Figure 5.4, latency is an end-to-end metric measured from source broker to destination broker, independent of the total number of broker hops a publication traverses. Accordingly, the inclusion of advertisements produces overlay links that shorten the distance between interested publishers and subscribers, and therefore results in improved latencies.

When interest accounts for greater than 30% of the similarity calculation (i.e., $\alpha > 0.3$), the overlays experience significant reduction in performance. Given that our content-based model assumes the number of subscriptions is much greater than the number of advertisements, publications generated from advertisements will likely match many subscriptions. Thus, when interest links are favoured over commonality links (cf. discussion in Section 5.3.2), the clustering of brokers is less effective, as messages matching at multiple brokers may traverse many additional hops.

For example, consider the pub/sub deployment in Figure 5.8, which depicts five brokers, two publishers and three subscriptions. Part (a) of the figure shows an overlay that favours

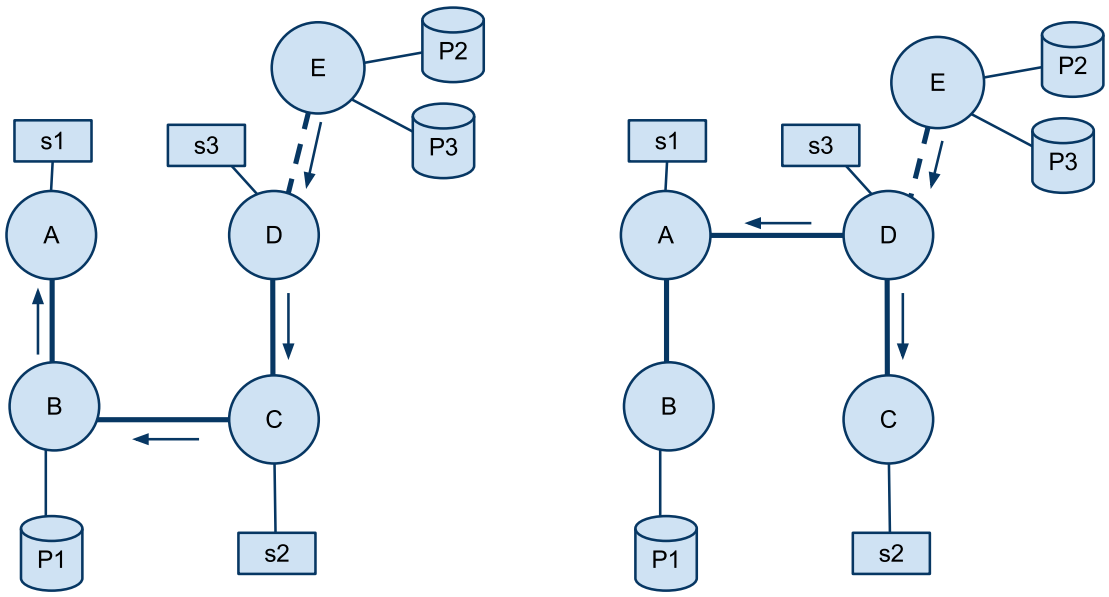


Figure 5.8: Interest vs. commonality trade-off

interest over commonality, thus the link between Broker B and Broker C that connects publications from P1 to subscription s2 is established in favour of the link between Broker A and Broker D. Part (b) of the figure shows the reverse situation, where commonality is favoured, and thus the link connecting Broker A and Broker D that connects similar subscriptions s1 and s3 is established. The dashed link between Broker D and Broker E represents a connection to a different segment of the overlay network.

When interest is given high priority, the link between Broker B and Broker C will be established, as in part (a). However, this may occur at the expense of messages coming from elsewhere in the network that match both s1 and s3. If the aggregated rate of messages coming from elsewhere in the network that match both s1 and s3 is larger than the publication rate of P1 (a likely scenario), the interest link between Broker B and Broker C reduces system performance, since the messages coming from other segments of the network must be routed through a longer overlay path (indicated by the arrows in part (a)). Thus, while interest reduces the distance between some publishers and subscribers, it may increase the distance for publications that match many subscriptions, due to reduced subscription clustering. Since there are more

subscriptions than advertisements, it is better to give greater preference to commonality.

5.5 Chapter Summary

In this chapter, we have presented two algorithms for computing tree-based overlay topologies for content-based pub/sub systems. In doing so, we have examined the effectiveness of our similarity techniques presented in the previous chapter. In addition, we have discussed techniques for evaluating the quality of an overlay topology in terms of performance and load distribution.

Our evaluations show that the overlap similarity metric produces overlays with superior message count and latency results, but at the cost of a topology that is star-like, with the central broker experiencing a disproportional amount of load compared to the other brokers. The covering technique provides poorer performance, but produces a more chain-like topology, and better load distribution. Finally, we show that performance may be improved by considering interest in addition to commonality, however interest should be restricted to less than 25% of the overall similarity measurement.

Currently, the scope of our work is limited to tree-based overlay topologies. There may be benefits, however, to constructing graph-based, or general, overlay topologies. Li et al. [20] have shown that well connected general overlays can improve delivery times up to 20% over tree-based overlays through adative routing protocols that utilize alternate routing paths depending on network conditions (for example, to avoid congested brokers). It is an aspect of future work to devise algorithms for building general overlay networks.

Chapter 6

Minimal-Broker Overlay Design

This chapter provides algorithms for the *Minimal-Broker Overlay Design Problem*, which seeks to construct an overlay network for a content-based pub/sub system that utilizes of a minimal number of brokers. The performance objective in designing the overlay is to minimize message latencies, however, a constraint is added requiring broker capacity limitations not be violated. To this end, we first prove that determining a minimal set of brokers for a given workload is an NP-hard problem, and then describe a heuristic to solve the problem. Central to our approach is a load modeling framework that utilizes concepts from Chapter 4 to estimate the load a given client will impose on a broker. The chapter concludes with an extensive evaluation to measure the effectiveness of our techniques at producing overlay designs that (1) provide good end-to-end performance, and (2) deploy a minimal set of brokers that do not succumb to excessive processing delays caused by broker congestion.

6.1 Problem Formulation and Analysis

In this section, we present a formal problem definition of the Minimal Broker Overlay Design Problem (MBODP). The goal of the MBODP is to design a broker overlay network for a content-based pub/sub system that minimizes average message delivery latency, and deploys a minimal number of brokers in the overlay. The purpose of reducing both metrics is to pro-

duce a low-latency network at minimal cost, where cost is measured by the number of brokers deployed. An overlay network with an excess number of brokers can increase costs due to inefficient resource utilization and result in increased message counts and higher delivery latencies due to longer routing paths. Conversely, an overlay network with too few brokers can increase latencies due to processing delays caused by broker congestion.

6.1.1 Problem Formulation

The MBODP is formally defined as follows.

Input

The input parameters define the workload and available resources.

1. Set of brokers available for use, $\mathcal{B} = \{b_i\}$, and associated processing capacities, $\mathcal{P}_i$, measured in message rate, indicating the number of messages the brokers are capable of processing per unit time.
2. Set of publishers, $\mathbf{P} = \{p_j\}$, and subscribers, $\mathbf{S} = \{s_k\}$
3. Set of advertisements, $\mathbb{A} = \{\mathfrak{a}_x\}$, and subscriptions, $\mathbb{S} = \{\mathfrak{s}_y\}$
4. Set of advertisements produced by each publisher p_j, $\mathbb{A}_j$, and set of subscriptions submitted by each subscriber s_k, $\mathbb{S}_k$
5. Publication rate r_x per advertisement $\mathfrak{a}_x$

Outputs

1. Set of brokers chosen $\mathbf{B} = \{b_i\}$, where $\mathbf{B} \subseteq \mathcal{B}$
2. Set of publishers and subscribers connected to each broker b_i, $\mathbf{P}_i$ and $\mathbf{S}_i$ respectively
3. Broker overlay topology, T, defined as an adjacency matrix of size $|\mathbf{B}| \times |\mathbf{B}|$

Objective

1. Minimize average end-to-end delivery latencies, d

2. Minimize the total number of brokers deployed in the overlay, N

Constraints

The processing load on each broker, measured in terms of message rate, should not exceed its processing capacity.

$$\omega_i \leq \mathcal{P}_i \tag{6.1}$$

where ω_i is the processing load at broker b_i. The processing load measures the number of messages a broker must process at a given instance of time.

6.1.2 Proof of NP-Completeness

In this section, we prove that the problem of determining a minimum (i.e., optimally minimal) set of brokers to satisfy a given workload, such that processing capacity thresholds are not exceeded at any broker, is an NP-complete problem. Since a broker must process the messages of its local publishers and subscribers, the set of clients local to a broker impact its processing load at a given instance of time. Thus, the problem of determining a minimum set of brokers is directly related to the problem of identifying the optimal allocation of clients to brokers such that the number of brokers is minimized. In terms of our problem formulation, we are proving that computing the first and second outputs is possible in polynomial time only if P=NP. We refer to the problem of determining the client allocation that minimizes the number of brokers as the *client-broker allocation problem*.

The decision version of the client-broker allocation problem is stated as follows.

Definition 1. Client-Broker Allocation Decision Problem. Given a set of clients (publishers

and subscribers) that send and receive messages at a rate of $\mathcal{R} = r_1, ..., r_n$ messages per unit time, and given a set of V brokers capable of processing $\mathcal{P}$ messages per unit time, is it possible to allocate all clients to the set of V brokers without exceeding the processing capacity of any brokers?

Theorem 1. The decision version of the client-broker allocation problem is NP-hard.

First, without loss of generalization, consider a content-based pub/sub scenario consisting of a set of clients where half are publishers and the other half are subscribers. Let each publisher and subscriber offer one advertisement and subscription, respectively. Additionally, let each advertisement occupy a distinct area of the content-space, such that no two advertisements overlap. Moreover, let there be a publisher-subscriber pairing, such that the advertisement of the publisher and subscription of the subscriber occupy equal regions of the content-space. Thus, each subscriber will receive all messages, and only those messages, from its paired publisher. The rate at which messages are sent and received is equal to the publication rate of the publisher.

We now show that the client-broker allocation decision problem is NP-hard by reducing an instance of the classical bin-packing problem to an instance of the client-broker allocation problem. The bin-packing problem states [18] that n items with sizes $s_1, s_2, ..., s_n \in (0, 1]$ must be packed into unit-sized bins so as to minimize the number of bins used. The bin-packing problem has been proven to be NP-complete [15].

Let the n items of the bin-packing problem correspond to a publisher-subscriber pairing. Let the size of the items correspond to the publication rate of the publisher. Let a bin correspond to a broker, and let the capacity of the bin correspond to the processing capacity of the broker, namely the number of messages it is capable of processing per unit time. Thus, the bin packing problem has been reduced to allocating publisher-subscriber pairings to a minimal set of brokers. Accordingly, the client-broker allocation problem is NP-hard.

Note that the algorithms presented in Section 6.3 for solving the MBODP are not able to leverage existing bin-packing heuristics. As described in the problem formulation, the objec-

tive of the MBODP is two-fold: in addition to allocating clients to a minimal set of brokers, message delivery latencies must also be minimized. Thus, we present algorithms that allocate clients to a minimal set of brokers but also give due attention to the performance impact of the allocation.

6.2 Load Modeling Framework

In this section, we define our client load and broker capacity models. These models are used for determining a minimal number of brokers that can satisfy a given workload without inducing congestion delays–one of our objectives from Section 6.1. In other words, the models help us estimate (1) the load that a client will impose on a broker and (2) the set of clients that a broker can support without exceeding its capacity limitations. Broker capacities are considered in terms of *message throughput*, the number of messages a broker is capable of processing per unit time.

The term *load impact* is used to describe the load a client will impose on a broker. It is defined as the number of messages a broker must process in unit time due to the client. For a publisher, the load impact is equal to its publication rate, since a broker must process all of the messages produced by a local publisher, regardless of whether the publications are forwarded or dropped.

The load impact of a subscriber, which may receive messages from multiple publishers, is calculated using the interest metric described in Section 4. We explain using an example. Consider the broker overlay in Figure 6.1, which consists of two brokers, b_1 and b_2, two publishers, p_1 and p_2, and one subscriber, s_1. Let each publisher contain one advertisement and the subscriber contain one subscription. The publication rates of p_1 and p_2 are $r_1 = 5$ and $r_2 = 4$, respectively. To determine the load impact of subscriber s_1 on broker b_1, we compute the interest between s_1 and each publisher using Equation 4.6, and then multiply the interest with the publication rate of each advertisement. The load impact of the subscriber is equal to

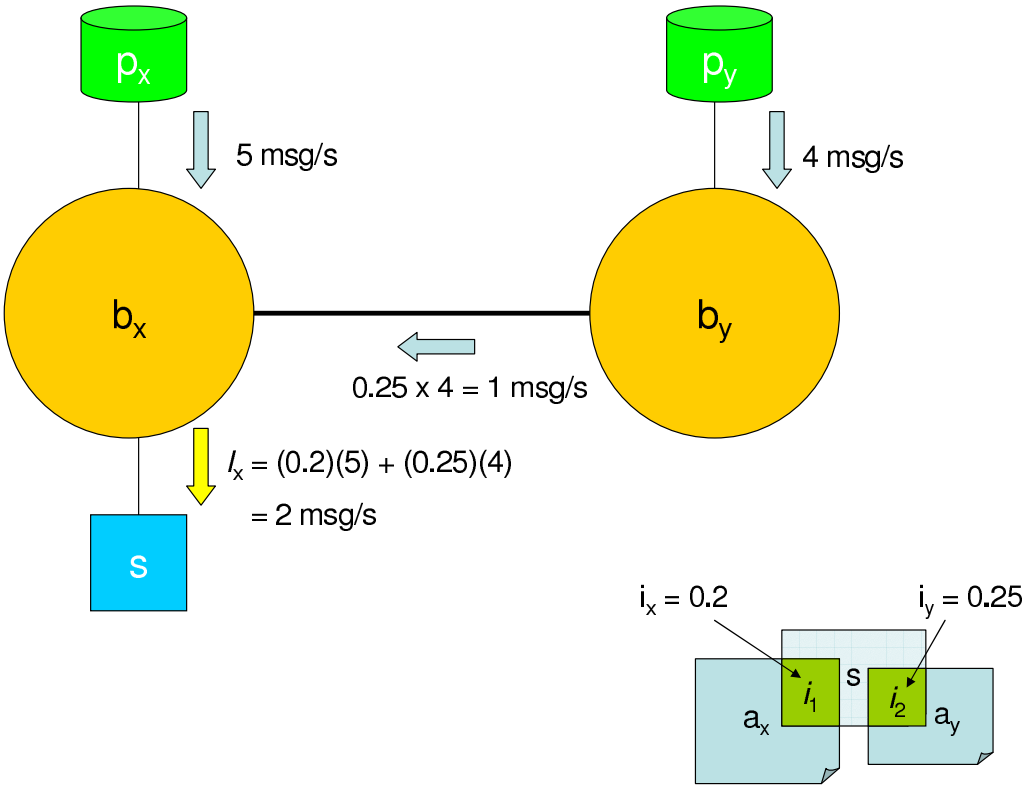


Figure 6.1: Measuring the load impact of subscribers

the sum of these values.

Formally, we can calculate the load impact, l_{s_x} of a subscriber s_x as follows.

$$l_{s_x} = \sum_{p_j \in \mathbf{P}} i_{xj} r_j,$$

where i_{xj} is the interest between s_x and publisher p_j.

The load on a broker is equal to the sum of the load impacts of its local publishers and subscribers minus a correction factor. We deduce the need for a correction factor as follows. The load impact of a publisher models the load on the broker for messages that are forwarded into the network, as well as messages that are delivered locally. The load impact of a subscriber estimates the number of messages the broker must process due to the subscriber for messages published locally and elsewhere in the network. We must add a correction term to account for

messages that are both published locally and matched locally, as this load is counted twice; both by the publisher and subscriber.

The load on a broker b_x, $\mathfrak{L}_x$, can thus be calculated as follows.

$$\mathfrak{L}_x = \sum_{p_j \in \mathbf{P}_x} l_{p_j} + \sum_{s_k \in \mathbf{S}_x} l_{s_k} - \sum_{s_u \in \mathbf{S}_x} \sum_{p_v \in \mathbf{P}_x} i_{uv} r_v, \tag{6.2}$$

where $\mathbf{P}_x$ and $\mathbf{S}_x$ represent the set of local publishers and subscribers of broker b_x, l_x is the load impact of client x, i_{uv} is the interest between local clients s_u and p_v, and r_v is the publication rate of p_v.

6.3 Broker Allocation and Overlay Design

In this section, we present our heuristic algorithm to solve the MBODP. The algorithm is composed of two phases, as depicted in Figure 6.2. The first phase, the *client-broker allocation phase*, determines a minimal number of brokers needed to satisfy a given workload and simultaneously allocates clients to the minimal broker set. In fact, as we will see, the minimal broker count is determined through the client allocation process. The set of brokers is deemed to be *minimal* when all brokers are projected to operate near capacity given the load impacts of their individual client sets. The second phase, the *overlay topology construction phase*, determines the overlay links between the brokers, such that the overlay network forms a connected tree.

While determining a minimal set of brokers is handled by the first phase, both phases are designed to improve end-to-end message performance through the clustering of clients (in the first phase) and brokers (in the second phase) that share strong similarity. The clustering of clients enables a larger number of messages to be contained within a broker, thereby reducing delays associated with messages propagating through the network. The clustering of brokers improves performance through a reduction in the number of pure forwarding brokers.

Note that, due to our dual objective of low-latency performance and minimal broker count, both phases of our algorithm are inter-related. To see this, consider that the load imposed on

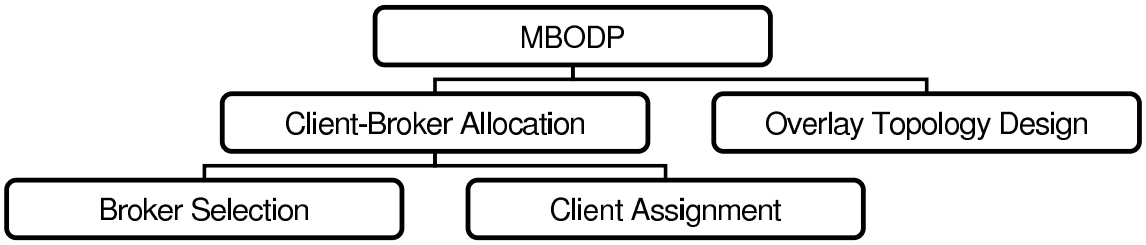


Figure 6.2: Phases of MBODP

a broker is due to two causes. First, brokers must process messages both produced by, and intended for, locally connected clients. This traffic may or may not be forwarded into the network. The load imposed by this traffic is accounted for by our load estimation framework. Second, brokers must process messages that are required to be forwarded along the overlay, but are not matched by local subscriptions (i.e., the broker acts as a pure forwarding broker). We refer to these messages as *pure forwarding traffic*. The load imposed on a broker by the pure forwarding traffic is determined by the layout of the overlay topology. However, the overlay topology can not be constructed until the broker count has been determined and the clients have been allocated. The inter-relation between the two phases requires that overlay designs be computed iteratively, where each phase continues to make adjustments on top of the modifications of the previous phase until the solution converges to a stable state.

In order to simplify our algorithm, however, we employ a buffer at each broker, that reserves

broker capacity for pure forwarding traffic. We define a parameter called a *load compensation factor* (LCF) to control the amount of a broker's capacity that is reserved for pure forwarding traffic. LCF is configured in the range of 0 to 1. For example, for an LCF of 0.75, only 75% of the original capacity of a broker is used by the first phase of the algorithm for allocating clients; thus the remaining 25% of the capacity is reserved for handling pure forwarding traffic. The LCF allows for keeping the two phases of our algorithm disjoint, thereby permitting singular execution of the algorithm. As we will see in the evaluation results presented in Section 6.4.6, the LCF technique is sufficient to build a low-latency overlay through a single execution of our algorithm.

6.3.1 Client-Broker Allocation Algorithm

The general flow of the client-broker allocation algorithm is to sequentially assign a client to an existing broker in the overlay that has available capacity. When no existing brokers are able to handle the load imposed by the next unallocated client, a new broker is deployed. We refer to existing brokers in the overlay as *deployed brokers*. The clients are allocated in a particular order, determined by a *client ranking function* (CRF). We describe four ranking functions shortly.

The pseudo code for client-broker allocation algorithm is presented in Algorithm 3. The inputs to the algorithm are the set of available brokers, the set of clients (publishers **P**, and subscribers **S**), and a client ranking function. The output will be the set of deployed brokers and a client-broker allocation.

Lines 2–7 describe the algorithm initialization process. The algorithm starts with an overlay network containing zero brokers (line 2), with no client allocations (line 3). The algorithm next computes the $|\mathbf{P}| \times |\mathbf{S}|$ interest matrix, **I**, and $|\mathbf{S}| \times |\mathbf{S}|$ commonality matrix, **J**, which are used to select the best broker for client allocation. Next, in line 6, the set of brokers is sorted in descending order of capacity, which determines the order in which available brokers are deployed to the overlay. In sorting by capacity, the algorithm uses the more powerful brokers

first, which enables a greater number of clients to be allocated to a smaller set of brokers, thus reducing the overall number of brokers in the overlay and enabling stronger clustering per broker, as a more powerful broker can support a larger number of clients.

Algorithm 3 Client-Broker Allocation

```
1:  Input Brokers, Clients, CRF
2:  DeployedBrokers ← ∅
3:  DeployedClients ← ∅
4:  I ← computeInterestMatrix(Clients)
5:  J ← computeCommonalityMatrix(Subscribers)
6:  BrokerStack ← Brokers sorted by descending capacity
7:  ClientStack ← sort(Clients, I, CRF)
8:  while ClientStack ≠ ∅ do
9:      c_x ← ClientStack.pop()
10:     for all c_i ∈ c_x do
11:         if c_i ∉ DeployedClients then
12:             b_x ← findBrokerForClient(c_i, I, J, DeployedBrokers)
13:             if b_x == null then
14:                 B ← BrokerStack.pop()
15:                 DeployedBrokers.add(B)
16:                 b_x ← B
17:             end if
18:             connectClientToBroker(c_i, b_x, ClientStack, algoType)
19:             DeployedClients.add(c_i)
20:         end if
21:     end for
22: end while
```

The last step of the initialization process is to sort clients according to the specified client ranking function. The client rankings determine the order in which clients are allocated, and thus effect the set of clients allocated to a particular broker. The four ranking functions are described and motivated as follows.

Greatest Load Impact (GLI): This client ranking function sorts clients in descending order of load impact (Section 6.2), independent of whether the client is a publisher or subscriber. This sorting is motivated by a greedy heuristic for solving the knapsack problem [2], where items to be packed in a knapsack are inserted in descending order of value. Since a knapsack contains a finite capacity, the greedy heuristic aims to maximize the overall value of the items

in the knapsack. Similarly, this technique assigns the clients with the greatest load impact first, under the assumption that clients with large load impacts have strong similarity, thus indirectly maximizing similarity at the most powerful set of brokers. This CRF places a batch of high load impact subscribers at the top of the list, followed by an interspersing of publishers and subscribers, followed by a batch of low load impact subscribers.

Greatest Interest (GI): This CRF does not sort individual clients, but rather sorts publisher–subscriber pairs in descending order of the interest value between them. It therefore places the publishers and subscribers that have the greatest interest, and therefore are projected to be heavy communication partners, at the top of the list. Note that individual clients can appear multiple times in the sorted list, but are assigned only once (the first time they appear). Since this function clusters subscribers with their most interested publishers, it favours low latency communication. During the client allocation process, the publisher is allocated first, followed by the client.

Greatest Interest with Group Allocation (GIg): This function sorts groups comprised of a single publisher and all the subscribers with which it has non-zero interest. Similar to the GI function, the groups are sorted in descending order of highest interest value in the group. By clustering a publisher with a group of subscribers, this CRF incorporates a measure of commonality into the rankings, thus encouraging a client allocation where a single publication matches multiple subscriptions per hop. Per group, the allocation process assigns the publisher first, followed by the subscribers in descending order of interest with the publisher. Thus, whereas the GI function allocates a single publisher and subscriber in turn (unless either or both have already been allocated), GIg assigns a single publisher followed by a batch of subscribers.

Baseline: This function sorts clients in a random order, with no regard for load impact or similarity. This technique models typical pub/sub deployments where clients are allocated in an ad-hoc manner. Note that when this scheme is used, the list of available brokers is also randomly sorted (Algorithm 3, line 6).

Once the initialization is complete, the algorithm begins by assigning the top ranked client

(or client set) to the first broker. The algorithm exits when all clients have been assigned. Each iteration of the algorithm (inside the **while** loop, Algorithm 3, lines 8–22) does the following: the client or client group is popped from the sorted client stack (Algorithm 3, line 9) and a broker from the existing set of brokers `DeployedBrokers`, is chosen per client using the `findBrokerForClient()` function.

This function, described in Algorithm 4, determines the best broker for the client assignment based on two criteria: first, the broker must have available capacity to accommodate the load impact of the new client (Algorithm 4, line 19, and Section 6.2), and second, the client set already assigned to the broker must have the highest similarity value compared to the client sets of the other deployed brokers. The method for determining similarity between a client, c_x, and the client set of a broker is shown Algorithm 4, lines 4–15. If c_x is a publisher, the similarity is equal to the sum of the interest between the client and all local subscribers (Algorithm 4, line 8). If c_x is a subscriber, the similarity is equal to the sum of the commonality between the subscriber and all local subscribers (Algorithm 4, line 10).

Once the similarity per broker has been determined for c_x, the set of deployed brokers are ranked according to greatest similarity (Algorithm 4, line 16). The selected broker is the first that is able to handle the additional load imposed by c_x (Algorithm 4, line 17–22). In the case where the broker with the greatest similarity is unable to accept the additional load, the broker with the next greatest similarity is tested, and so down the list until a capable broker is found. If all brokers are unable to accept the additional load, the function returns *null* and a new broker is popped from the broker stack (Algorithm 3, lines 14–15)).

To determine if a broker can support a client, the load impact of the client is added to the load on the broker. If this value is less than broker capacity reduced by the LCF factor (Algorithm 4, line 19), the client will be assigned to the broker. Once a broker has been selected, the load on the broker is updated by applying Equation 6.2. When all clients have been deployed, client allocation is complete and the chosen broker set and client assignment is fed into the overlay construction algorithm described in Section 6.3.2.

Algorithm 4 *findBrokerForClient()*

```
1:  Input c_x, I, J, DeployedBrokers
2:  totalSimilarity ← 0
3:  SimilarityStack ← ∅
4:  for all B ∈ DeployedBrokers do
5:      S ← B.subscribers
6:      for all s_x ∈ S do
7:          if c_x is publisher then
8:              totalSimilarity += I(c_x, s_x)
9:          else
10:             totalSimilarity += J(c_x, s_x)
11:         end if
12:     end for
13:     SimilarityStack.add(B, c_x)
14:     totalSimilarity ← 0
15: end for
16: SimilarityStack ← sort()
17: while SimilarityStack ≠ ∅ do
18:     b_x ← SimilarityStack.pop()
19:     if (b_x.currentLoad() + c_x.loadImpact() < LCF × b_x.capacity()) then
20:         return b_x
21:     end if
22: end while
23: return null
```

The idea behind the algorithm we have presented is to cluster each client to an existing broker with which it has strongest similarity, and to deploy a new broker only when the capacity of the existing overlay becomes inadequate. In this way, we are able to deploy a minimal number of brokers and also reduce average message latencies of the system.

6.3.2 Overlay Topology Construction

Once the client-broker allocation has been performed and the number of brokers has been determined, the problem reduces to the fixed-broker overlay design problem that we addressed in Chapter 5. Therefore, Algorithm 1 from Section 5.3.1 is executed to construct the overlay topology. The overlap similarity metric is used since it provides the best performance, and our explicit consideration of broker capacity constraints prevents a star-like network (cf. discussion regarding Figure 5.6 in Section 5.4.3).

Note that the hybrid algorithm is not used as the client-broker allocation process already incorporates a measure of interest when allocating clients. The hybrid algorithm, therefore, does not improve performance.

6.3.3 Complexity Analysis

The complexity of the client-broker allocation algorithm can be determined as follows. The notation used below is in accordance with the definitions in the problem formulation (Section 6.1.1). Note, we assume a single advertisement per publisher and single subscription per subscriber. Thus, the number of publishers and subscribers is equal to the number of advertisements and subscriptions, respectively, and we use the terms interchangeably.

As discussed in Section 5.3.3, the commonality and interest matrices may be computed in time $O(|\mathbb{S}|^2)$ and $O(|\mathbb{S}||\mathbb{A}|)$, respectively. Additionally, the set of available brokers can be sorted in time $O(|\mathcal{B}|\,log(|\mathcal{B}|))$ using an efficient sorting algorithm.

Of the client ranking functions, the most complex is the GIg technique. This function ranks

groups consisting of a single publisher and all subscribers with non-zero interest. Thus, the size of the client list when this function is employed is equal to the number of publishers, $|\mathbb{A}|$. The list is ranked according to the greatest interest in the group, which can be executed in time $O(|\mathbb{A}|\,log(|\mathbb{A}|))$. In addition, each group within the list is also ranked in descending order of interest. Ranking all $|\mathbb{A}|$ groups requires a time of $O(|\mathbb{A}|\,|\mathbb{S}|\,log(|\mathbb{S}|))$. Therefore, the time complexity of the GIg CRF is $O(|\mathbb{A}|\,log(|\mathbb{A}|) + |\mathbb{A}|\,|\mathbb{S}|\,log(|\mathbb{S}|))$.

Since the GI technique ranks publisher-subscriber pairs, the client list will be of size $|\mathbb{A}|\,|\mathbb{S}|$. The list can be ranked in $O(|\mathbb{A}|\,|\mathbb{S}|\,log(|\mathbb{A}|\,|\mathbb{S}|))$. Similarly, the GLI technique creates a list of size $|\mathbb{A}| + |\mathbb{S}|$, and can be ranked in time $O((|\mathbb{A}| + |\mathbb{S}|)log(|\mathbb{A}| + |\mathbb{S}|))$.

Since the GIg and GI algorithms place clients in multiple groups, the choice of client ranking function determines the number of client allocation attempts that take place. When the GIg function is used, the number of groups in the client list is equal to the number of advertisements, however, within each group is a number of subscribers, many of which belong to multiple groups. Thus, the CRF technique will require $|\mathbb{A}|\,|\mathbb{S}|$ clients to be processed. The same holds true for the GI ranking function, since all possible publisher-subscriber pairings are considered. If the GLI technique is used, each client is processed only once, therefore the number of allocation attempts is $|\mathbb{A}| + |\mathbb{S}|$.

When allocating a client to a broker, the list of deployed brokers is traversed to find the most similar broker, and the set of subscribers at each broker is accessed to identify the broker with the greatest similarity to the client. The set of deployed brokers is subsequently ranked according to degree of similarity with the client. Thus, the time required to allocate a single client is $O(|\mathbf{B}|\,|\mathbb{S}| + |\mathbf{B}|\,log(|\mathbf{B}|))$.

The complexity analysis is summarized in Table 6.1. The dominant contributor to the time complexity is the process of allocating client to brokers. In the worst case, it can reach a running time proportional to $O(|\mathbf{B}|\,|\mathbb{A}|\,|\mathbb{S}|^2)$.

Table 6.1: Complexity breakdown of client-broker allocation algorithm

CRF	Complexity Breakdown				
	Interest	Commonality	Broker Ranking	Client Ranking	Client Allocation
GLI	$O(\|\mathbb{S}\|\|\mathbb{A}\|)$	$O(\|\mathbb{S}\|^2)$	$O(\|\mathcal{B}\|log(\|\mathcal{B}\|))$	$O((\|\mathbb{A}\|+\|\mathbb{S}\|)log(\|\mathbb{A}\|+\|\mathbb{S}\|))$	$O(\|\mathbb{A}\|+\|\mathbb{S}\|)(\|\mathbf{B}\|\|\mathbb{S}\|+\|\mathbf{B}\|log(\|\mathbf{B}\|))$
GI	$O(\|\mathbb{S}\|\|\mathbb{A}\|)$	$O(\|\mathbb{S}\|^2)$	$O(\|\mathcal{B}\|log(\|\mathcal{B}\|))$	$O(\|\mathbb{A}\|\|\mathbb{S}\|log(\|\mathbb{A}\|\|\mathbb{S}\|))$	$O(\|\mathbb{A}\|\|\mathbb{S}\|)(\|\mathbf{B}\|\|\mathbb{S}\|+\|\mathbf{B}\|log(\|\mathbf{B}\|))$
GIg	$O(\|\mathbb{S}\|\|\mathbb{A}\|)$	$O(\|\mathbb{S}\|^2)$	$O(\|\mathcal{B}\|log(\|\mathcal{B}\|))$	$O(\|\mathbb{A}\|log(\|\mathbb{A}\|)+\|\mathbb{A}\|\|\mathbb{S}\|log(\|\mathbb{S}\|))$	$O(\|\mathbb{A}\|\|\mathbb{S}\|)(\|\mathbf{B}\|\|\mathbb{S}\|+\|\mathbf{B}\|log(\|\mathbf{B}\|))$

6.3.4 Algorithm Assumptions

The models we have outlined thus far make several assumptions. First, we assume that publications are uniformly distributed across the advertisement space. This has an effect on the interest metric, as well as the load impact calculation. We have previously addressed this issue Chapter 4, and note that the algorithm presented here is independent of the specific interest calculation. Similarly, the algorithm is independent of the specific load impact computation method, so long as a load measurement is provided to update broker load.

Second, in minimizing the number of brokers, we assume that the cost of deployment of all brokers is identical. In practice, however, this assumption may not hold as resources have differing costs that are dependent on a number of factors. Accordingly, there should be a specialized cost function that incorporates several parameters including capacity and monetary costs associated with broker utilization. It is an aspect of future work to develop such a model. In such a scenario, the core of the algorithms presented would remain the same. The broker ranking function may need modification, however.

Third, we assume unlimited network capacity. It is left for future work to extend our models and algorithms to account for link bandwidth constraints.

Fourth, we assume constant broker throughput. In practice, throughput would depend on factors such as incoming and outgoing messaging rates and subscription sizes.

6.4 Evaluation

This section evaluates the techniques presented in this chapter. Overlays are constructed using each of the four client ranking functions and compared against one another. The overlay

designs are examined in terms of cost (number of brokers deployed), as well as performance (message count and message latencies). In addition, the effectiveness of the load modeling framework at reducing congestion effects is evaluated. The section ends with an analysis of the effect of the load compensation factor on performance and load.

6.4.1 Simulation Setup

The algorithms presented in this chapter were implemented in Java and evaluated via simulations conducted on a pub/sub system built using the JiST [1] discrete event simulator. The experimentation process consisted of executing the Client-Broker Allocation algorithm (Algorithm 3) on a generated workload to assign clients to a minimal set of brokers. Next, an overlay topology was constructed by executing the commonality overlay construction algorithm (Algorithm 1 from Chapter 5) using the overlap similarity metric. The topology was programmed into the simulator, and results were collected as the simulations were executed. The results presented are averaged over five independent runs.

A simulation *time unit* was considered to be 1000 ticks. Thus, a publisher with a rate of five messages per time unit would publish a message every 200 ticks. A broker processing load delay mechanism was implemented to model the effects of congestion, where for every message a broker processed in excess of its capacity in the previous time unit, the next message forwarded by the broker was delayed by that amount. For example, if a broker with a capacity of 1000 messages per time unit processed 1250 messages in the previous time unit, the next message sent would be delayed by 250 ticks. Lastly, link latencies were fixed at 100 ticks.

6.4.2 Workload Specification

The workload used to conduct our experiments is summarized in Table 6.2. The workload was configured in keeping with the assumptions of content-based pub/sub systems that utilize advertisements, namely the number of publications in the system is much greater than the number

Table 6.2: Workload Summary

Number of Advertisements	200 - 1000
Number of Subscriptions	600 - 3000
Publication Duration (time units)	10
Broker Capacity (messages per time unit)	1000
Max Publication Rate	10
Distribution of Publication Rate	Zipf
Number of Attributes	2
Attribute Value Range	0 - 1000
LCF	0.75

of subscriptions, which is much greater than the number of advertisements. Accordingly, the number of subscriptions was set equal to three times the number advertisements. Note that we assumed one advertisement per publisher and one subscription per subscriber. In doing so, we assume publishers and subscribers with multiple advertisements and subscriptions are able to connect to multiple brokers, as is common practice in related work [9].

In terms of publications, each advertisement was assigned a publication rate up to 10 messages per time unit, based on a Zipf distribution. Publications were produced for the first ten time units of the simulation, and were uniformly distributed across the advertisement space, in keeping with our assumption of uniformly distributed publications as outlined in Section 6.3.4.

6.4.3 Performance Metrics

The following performance metrics were calculated to evaluate each overlay design.

- *Average Message Count*: The average number of messages generated per publication. A lower value favours better client and broker clustering, as messages travel reduced overlay hops.

- *Average Message Latency*: The average latency (in simulation ticks) required for a message to be delivered.

- *Pure Forwarding Message Ratio*: The ratio of the number of messages that passed through a broker and the total number of messages processed. It indicates the percentage

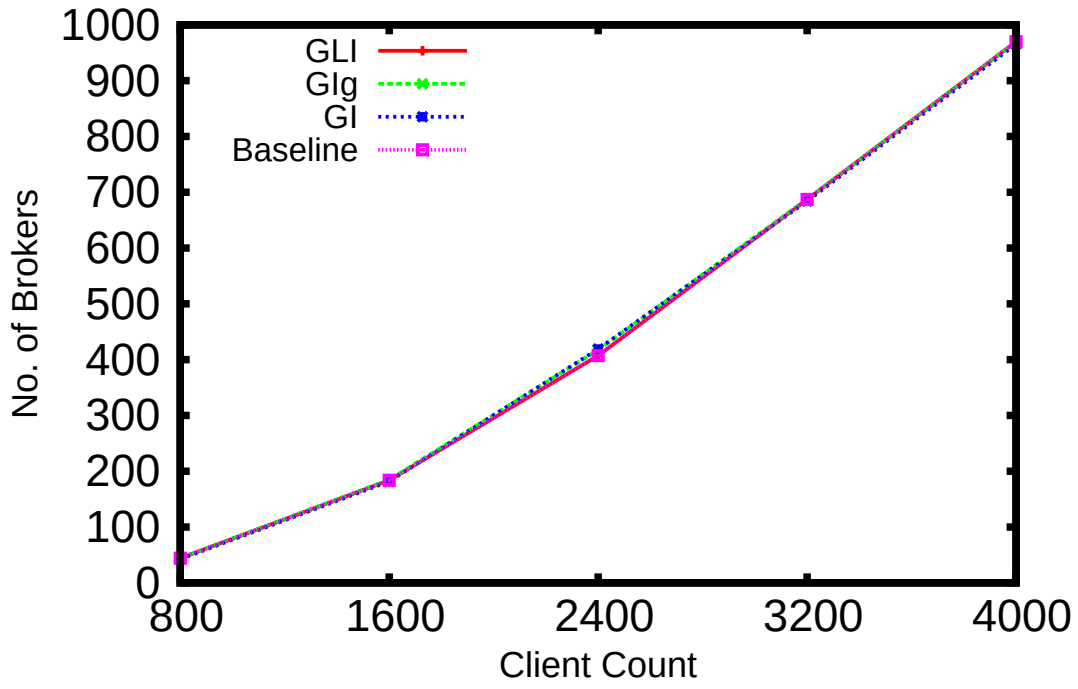


Figure 6.3: Overlay cost measured in terms of broker count

of messages a broker processed that were not matched by a local subscription. The result is averaged over all brokers.

- *Average Peak Load*: Peak load is defined as the maximum number of messages processed by a broker in any 1000 tick interval. Average peak load is the average of the peak load of all brokers in the overlay. It is an estimate of the maximum load experienced per broker.

- *Average Processing Delay*: The average delay experienced by a message due to broker congestion.

6.4.4 Overlay Design Cost

Figure 6.3 shows the number of brokers deployed by the algorithm for each client ranking function as the number of clients is increased from 800 to 4000. The figure shows that all ranking functions utilize a similar number of brokers. This is a consequence of the algorithm

scanning all existing brokers for available capacity prior to deploying a new broker. Accordingly, a client is allocated to a broker even if there is low similarity with local clients, so long as the broker can support the additional load. Thus, while the ranking function affects the client clustering per broker, and therefore has an effect on performance and broker load, the overall load impact of the entire client set remains the same and all clients can be accommodated by a similar number of brokers.

There is, however, a slight variation in the number of brokers deployed by each ranking scheme. The figure shows that on occasion, such as when the client count is 2400, the GLI technique allocates clients to a slightly fewer set of brokers. GLI is more efficient in its client allocation because it assigns the heavily loaded clients first, leaving the clients with low load impact for allocation near the end of algorithm execution. The situation is analogous to packing marbles and sand into a set of containers. If marbles are packed first, the sand can occupy gaps within the container that are too smaller for other marbles. However, if sand is packed first, or is packed intermittently with the marbles, the last batch of marbles may not fit in the gaps of containers that are near capacity. Similarly, since the GLI technique allocates the clients with high load impact first, the clients with low load impact can be efficiently supported by existing brokers that are near capacity. The other techniques intermittently allocate clients of all load impacts, and are therefore less effective at fully utilizing all of a broker's capacity.

6.4.5 Overlay Design Performance

Figures 6.4 and 6.5 show the average message count and average message latency results for overlays constructed using each of the four client ranking functions. The metrics are evaluated as the workload is increased from 800 clients (200 publishers, 600 subscribers) to 4000 clients (1000 publishers, 3000 subscribers). Through the plots we can evaluate the impact of the frequency with which publishers are allocated, as follows. The GLI technique performs better than the baseline approach, but worse than the GI and GIg techniques. Since GLI allocates clients with high load impact first, all of which are subscribers, the GLI technique creates an

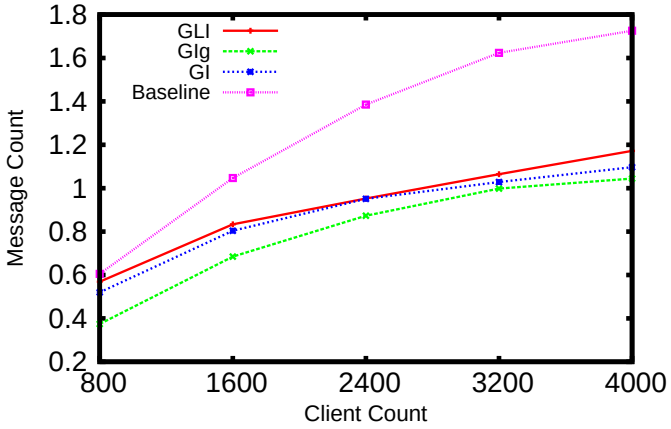


Figure 6.4: Effect of clustering on message count

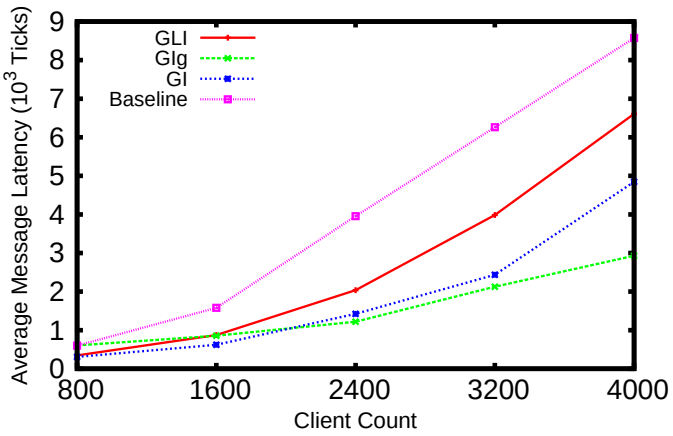


Figure 6.5: Effect of clustering on message latency

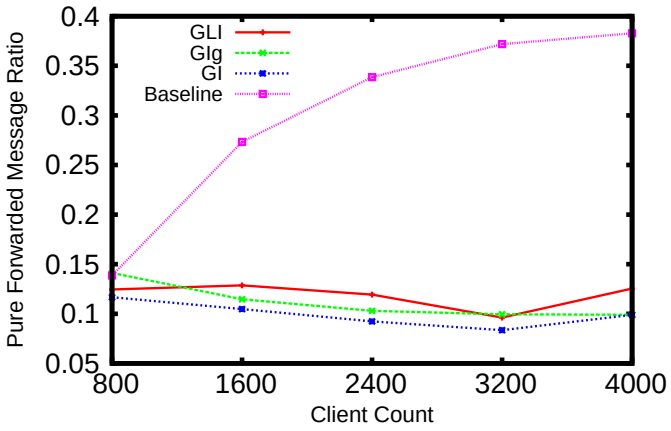
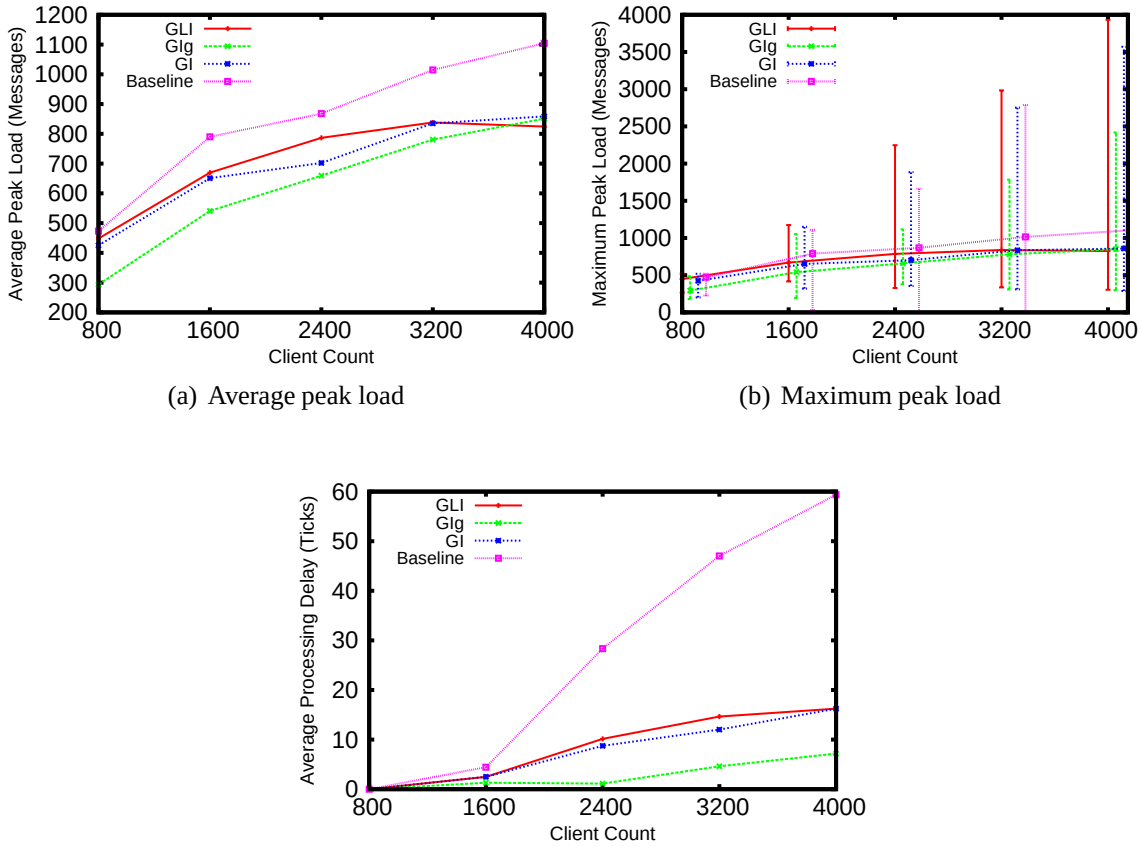


Figure 6.6: Effect of clustering on pure forwarding traffic

overlay where the majority of subscribers are clustered to the initial set of brokers. When it is turn for publishers to be allocated, there is less capacity at the deployed brokers for the clustering publishers and subscribers that share interest. This results in increased overlay distance between publishers and subscribers, and therefore larger message counts and latencies. The GI technique performs second best. As it allocates publishers and subscribers in turn, the overlay distance is reduced for many interested publishers and subscribers. However, the frequency with which publishers are allocated favours interest at the expense of commonality. In other words, as publishers are allocated frequently, there is less capacity at the brokers for additional subscribers to connect. This hinders a clustering where a single publication matches many subscriptions at the same broker. The GIg technique provides the lowest message counts and latencies, and therefore produces the most effective client clustering. Since it allocates a single publisher followed by all interested subscribers, the overlay contains a subset of brokers that connect very many subscribers. This is in fact similar to the GLI technique. However, since a publisher is allocated along with the group of subscribers, the initial set of brokers contains publishers in addition to many subscribers. Thus, there is greater occurrence of a single publication matching multiple subscriptions at the same hop, which GLI is not able to produce since it allocates publishers much later in the allocation process. Figure 6.6 shows the fraction of all messages that are pure forwarding messages. Since the baseline technique avoids clustering of similar clients entirely, it produces a large number of pure forwarding messages. The ranking routines, through the clustering of clients are brokers, are able to limit the number of pure forwarding messages to 10–15% of all messages processed. Moreover, the percentage of pure forwarding traffic remains constant as the workload increases, despite the increased number of brokers.

6.4.6 Load Modeling and Broker Congestion Effects

The plots in Figures 6.7(a) – 6.7(c) show the effectiveness of our load modeling framework. Figure 6.7(a) shows the average peak load as the number of clients is increased. Recall that



(a) Average peak load

(b) Maximum peak load

(c) Processing delays due to broker congestion

Figure 6.7: Effectiveness of load modeling framework at reducing broker congestion

peak load measures the maximum load experienced by a broker. Whereas the baseline overlay produces an average peak load above the broker capacity threshold (1000 messages per time unit) for a sufficiently large workload, the overlays produced by our algorithm are able to contain the load, and thereby reduce processing delays. In addition, as the workload increases, the peak load values level-off, indicating that the solution is scalable.

Figure 6.7(b) shows the maximum peak load experienced in the overlay (by a single broker). We see that the maximum peak load for the GLI ranking function is significantly larger than that of the other techniques. Since GLI initially assigns the clients with the largest load impacts to the first set of brokers, these brokers are a destination for a disproportionately large number of messages, and therefore experience large loads. The GIg overlay has the lowest

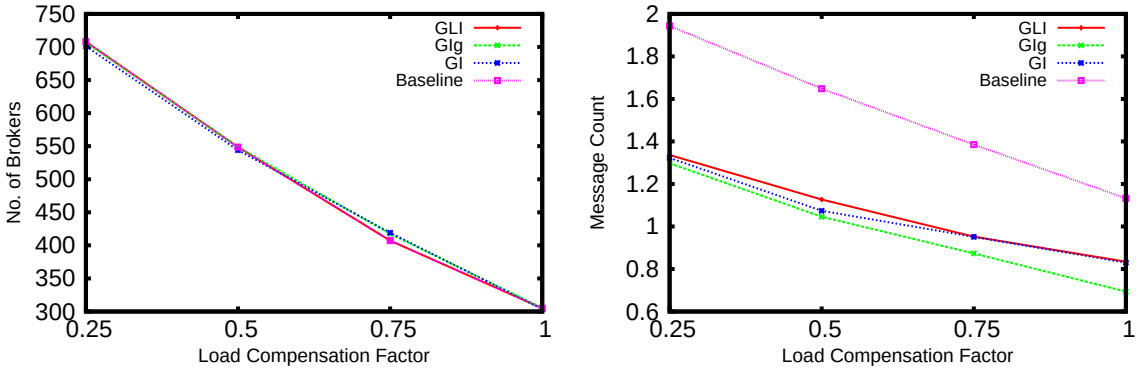
maximum peak load, and therefore produces the overlay with both the lowest load (through improved clustering) and the most even load distribution.

Figure 6.7(c) shows the average processing delay that messages experience due to broker congestion. Since the graph shows that none of the overlays produce zero processing delays, all overlays do indeed experience some congestion. This is also evident from Figure 6.7(b) in that all maximum peak loads are above the broker capacity threshold of 1000 messages per time unit. The congestion is due to pure forwarding traffic, which is not explicitly accounted for by our load modeling framework. However, by comparing our algorithms to the baseline approach in Figure 6.7(c), it is evident that the clustering aspects of our algorithm, along with the load modeling framework, are sufficient for significantly reducing congestion effects.

6.4.7 Load Compensation Factor

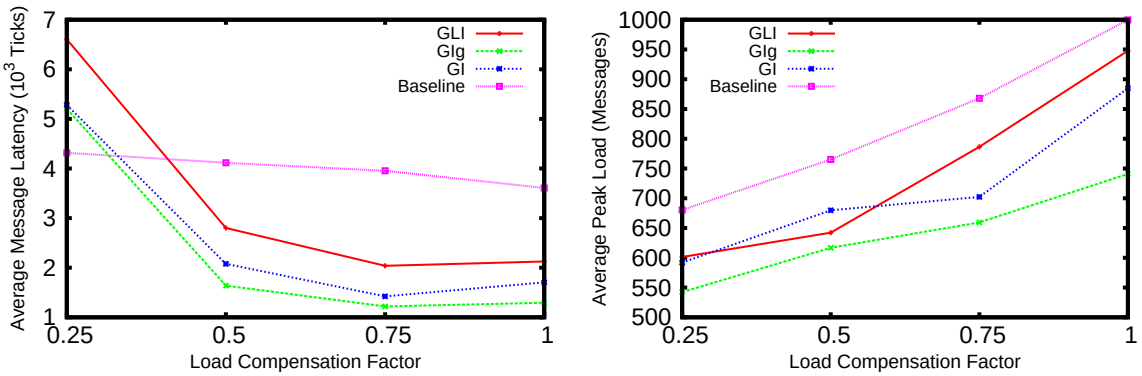
In this section, we examine the effect of the load compensation factor in terms of overlay design performance and broker congestion reduction. The results presented are for a workload consisting of 2400 clients, and otherwise as described in Table 6.2. Figure 6.8 (a) shows that LCF value has a significant impact on the number of brokers deployed. Since a lower LCF reduces the amount of capacity reserved for client allocation, more brokers are required. Part (b) shows that message count decreases with higher LCF, which is expected since a higher LCF value enables more clients to be allocated per broker, and thereby reduces the total number of brokers in the overlay, enabling publications to traverse shorter overlay hops to reach all interested subscribers.

Part (c) of the figure shows the effect of the LCF on average message latency. As with message count, the latency is affected by the number of brokers in the overlay: an increased number of brokers results in longer overlay distances, and thus greater link latencies. In addition, latencies are also affected by broker congestion. From part (d) of the figure, we see that a higher LCF results in greater peak loads: since an increase in the LCF value results in more broker capacity reserved for client allocation, there is less space reserved for pure forwarding



(a) Effect of LCF on overlay cost

(b) Effect of LCF on message count



(c) Effect of LCF on message latency

(d) Effect of LCF on broker load

Figure 6.8: Impact of LCF value on performance and load

traffic, and therefore there are greater peak loads. Accordingly, we see form part (c) that the best balance between latency due to overlay hops and congestion is when LCF is configured to 0.75. It should be noted, however, that latencies increase only marginally when LCF is 1.0; but the cost of the overlay increases significantly from 300 brokers when LCF is 0.75 to 400 brokers when LCF is 1.0.

6.5 Chapter Summary

This chapter has addressed the Minimum Broker Overlay Design Problem, which seeks to construct overlay topologies for content-based pub/sub systems that minimize delivery latencies and utilize a minimal number of brokers. First, we proved the problem of computing a minimal number of brokers for a given client set is an NP-hard problem. This was followed by the description of a load modeling framework, that utilized concepts from Chapter 4 to estimate the load imposed on a broker by a given set of clients. The framework was used by our heuristic algorithm to allocate clients to a minimal set of brokers.

The heuristic algorithm decomposed the MBODP into two phases. The client-broker allocation phase allocates clients to a minimal set of brokers. The overlay topology construction phase builds an overlay using the commonality overlay construction algorithm from Chapter 5. The evaluation showed that our algorithm is effective at constructing low latency overlays that reduce broker congestion.

Chapter 7

Conclusions and Future Work

This thesis has studied the problem of constructing broker overlay topologies for content-based publish/subscribe systems. First, the challenge of measuring similarity between entities of publish/subscribe systems was examined. To this end, a framework was developed for measuring similarity based on the subscription and advertisement sets of publishers, subscribers, and brokers.

Next, algorithms for constructing broker overlay topologies were presented, with the objective of reducing message traffic and delivery latencies. The similarity framework was leveraged to construct overlay topologies based on the similarity of the subscription and advertisement sets of brokers. In this way, overlay distances are reduced, leading to improved performance. In addition, overlay topologies were examined in terms of the dependence of the overlay on a small subset of brokers, in order to identify topologies that are vulnerable to broker congestion effects.

Lastly, the overlay design problem was extended to explicitly consider broker capacity constraints. This led to the introduction of the Minimal Broker Overlay Design problem, that has the dual-objective of constructing broker overlay networks where performance is maintained and the number of brokers utilized is reduced. After proving the problem of determining a minimal set of brokers is NP-hard, a heuristic was developed to solve the problem.

The contributions of this thesis serve as stepping-stones for the problem of designing broker overlays for content-based publish/subscribe systems. Our work can therefore be extended in several ways. A key aspect of future work is to relax many of the assumptions we have made. Of particular importance is removal of the assumption of unlimited network bandwidth. A promising technique for relaxing this assumption is to extend our similarity metrics to account for link metrics such as the cost of link utilization and available bandwidth of the overlay link connecting the entities.

A long term aspect of future work is to add support for additional constraints. For example, we hope to produce an optimization engine that can answer queries such as: design an overlay that has a maximum diameter of four hops; design an overlay where publisher X is a maximum of three hops away from subscribers A, B, and C; and are 10 brokers enough to support a given set of clients.

Another key future work theme is to develop runtime solutions to the Minimum Broker Overlay Design Problem. Current runtime techniques assume a fixed number of brokers, which may lead to inefficient resource utilization. An overlay design technique that adds and removes brokers from an operation system may be beneficial for large-scale distributed systems that are not operated by a single entity.

Lastly, we hope to extend our work by incorporating knowledge of the physical network on top of which the overlay operates. One path forward is to devise algorithms for optimal broker placement in the physical network based on similarity with other neighboring brokers.

Chapter 8

Appendices

8.1 Appendix A: Simulator Implementation

This section describes the implementation of a pub/sub simulator that was used to evaluate the work presented in this thesis. The system simulates a distributed content-based pub/sub system and is powered by the JiST discrete event simulator [1]. The system operates in one of two modes: fixed-broker mode and minimal-broker mode. The former corresponds to the work presented in Chapter 5, where the number of brokers is assumed fixed, and the objective is to construct an overlay topology. The latter refers to the work presented in Chapter 6, where the objective is to construct a broker overlay network consisting of a minimal set of brokers for a given workload. The mode of operation can be toggled by setting the `spanning_tree_algo` variable in the configuration file (cf. Section 8.1.2).

Simulations are conducted via a two-step process. First, an overlay design is computed by executing the algorithms presented in this thesis. Second, the overlay design is programmed into the simulator and a simulation is executed. The following sections describe the relevant details of the implementation, and provide a description of the configuration parameters through which simulation properties can be set.

8.1.1 Implementation Details

The core overlay design algorithms and pub/sub functionality are implemented in the `overlay` package. The implementations of the similarity metrics and maximum spanning tree algorithms are in `Overlay.java`. It creates an `overlay` object that models a pub/sub overlay network. Accordingly, the `overlay` object is responsible for computing the overlay topology in both fixed-broker and minimal-broker modes. The object then performs initialization of broker simulation objects, wires the broker overlay network according to the overlay design, and launches the simulation by instructing the broker simulation objects to start publishing.

A `padresNode` object represents a broker simulation object. It utilizes the JiST API for advancing simulation time using the `JistAPI.sleep()` function [1]. A `padresNode` object issues publications at a rate according to its configured parameters, and employs interfaces for sending and receiving messages. The `sendMessage()` function is used for sending a message to a neighboring broker. A message is sent by invoking the `receiveMessage()` interface of the destination broker. In order to model link latencies, prior to a message being forwarded by one broker to another, the simulation time is increased by an amount equal to the value of the link latency that connects the two brokers. Link latencies are fixed at 100 simulation ticks.

The algorithms for constructing a minimal-broker overlay network are in the `overlay.capacity` package. `VariableBrokersSpanningTree.java` contains the implementation of the client-broker allocation algorithm and client ranking functions. The `overlay.dmst` package contains the implementation of Edmond's algorithm for optimal directed spanning trees. It is derived from the implementation described in [31].

The `sim` package contains the main file through which the simulator is launched (`Simulation.java`) and the workload generator (`WorkloadGenerator.java`). The `simulation` object is responsible for loading the configuration parameters and initializing the `overlay` object. It also initializes the `workloadGenerator` object and passes it to the `overlay` object, where workloads are produced. The `workloadGenerator` object produces ranged-based advertisements and subscriptions, and point publications according to the quantities and dis-

tributions specified in the configuration file. In addition, it assigns generated messages to the set of brokers, and performs a matching operation, where generated publications are matched with subscriptions. This centralized matching technique facilitates simpler routing by enabling publication destinations to be embedded within publication messages. Accordingly, brokers do not have to match messages; they may simply forward messages to the appropriate next hop for each message destination.

The routing mechanism functions as follows. Each broker keeps a routing table describing its next hop to all other brokers in the overlay. The routing table is determined using a shortest path algorithm after the overlay design has been programmed into the simulator. Upon receiving a message, a broker performs a lookup on the message destinations, and forwards the message to its next hop towards each destination. If there are no additional destinations, the message is dropped.

As messages are routed among brokers, at each hop, broker and message statistics are collected. The `StatsCollector.java` file in the `sim` package contains helper functions for aiding in statistics collection. Although message and broker stats are collected during the simulation, they are processed by the `sim` object after the simulation has expired. The statistics are subsequently written to a file using methods in `Utility.java` in the `utils` package.

8.1.2 Configuration Parameters

The following parameters can be specified in the configuration file, `sim_config.txt`, to specify the properties of the simulation.

Simulation Parameters

Parameter	Description
random_seed	A seed for the random function used by the workload generator and simulator.
no_of_nodes	The number of brokers, or `PadresNode` objects, to be deployed in fixed-broker mode. Not applicable in minimal-broker mode.
no_of_subs	The total number of range-based subscriptions to be generated by the workload generator.

no_of_advs	The total number of range-based advertisements to be generated by the workload generator.
no_of_pubs	The total number of point publications to be generated by the workload generator. Only applicable in fixed-broker mode.
link_weight_type	The similarity metric to be used to compute broker weights. Possible values: OVERLAP_WITH_TIGHTNESS (overlap metric), COVER_WITH_TIGHTNESS (cover metric), HYBRID (hybrid weights), and RANDOM (baseline)
spanning_tree_algo	The algorithm to use to construct a maximum spanning tree. In fixed-broker mode, possible values are: MAX_SPANNING_TREE (Prim's algorithm [24]) and DIRECTED_SPANNING_TREE (Edmond's algorithm [12]). In minimal-broker mode, the parameter must be set to VARIABLE_SPANNING_TREE. In the latter case, the directed spanning tree algorithm is used to construct the overlay topology.
adv_factor	The value of the advertisement priority factor to be used when the hybrid algorithm is executed. The value must be in the range of [0.0,1.0]. Only applicable when link_weight_type=HYBRID.
client_rank_type	The client ranking function employed by the client-broker allocation algorithm in minimal-broker mode. Possible values: 1 (GLI), 2 (GIg), 3 (GI), 4 (baseline).
pub_interval	The duration at the start of a simulation in which publications are generated. Measured in 10^3 simulation ticks. Only applicable in minimal-broker mode.
max_pub_rate	The maximum number of publications any given publisher will generated per 1000 ticks in minimal-broker mode. Together with pub_interval, specifies the number of publications that will be generated in minimal-broker mode.
max_broker_cap	The processing capacity of a broker measured as the number of messages a broker is capable of processing per 1000 simulation ticks without becoming overloaded. Only applicable in minimal-broker mode.
diff_broker_cap	Specifies if all brokers should have identical processing capacities. Possible values: true (different capacities), and false (identical capacities). If true, broker processing capacities are uniformly distributed in the range [0,max_broker_cap]. If false, all processing capacities are set equal to the value of max_broker_cap. Only applicable in minimal-broker mode.
apply_load_delay	Specifies if a load delay should be applied to outgoing messages if a broker has processed messages in excess of its processing capacity in the previous 1000 tick interval. Possible values: true (apply load delay), and false (ignore load delay). If true, an outgoing message is delayed by one tick for every message that was processed in excess of the capacity value. Only applicable in minimal-broker mode.
load_comp_factor	The value of the load compensation factor (LCF) in minimal-broker mode. The value must be in the range of [0.0,1.0].

Content-Space Parameters

Parameter	**Description**
no_of_attribs	The number of different attributes in the workload.
no_of_classes	The number of different class attributes in the workload.
low_value	The lowest possible value for an attribute.
high_value	The highest possible value for an attribute.

pref_class_bias	The bias factor to be used when distributing subscriptions and advertisements to brokers. The value must be in the range [0,100]. A value of 0 implies none of the subscriptions and advertisements assigned to a broker will be of its preference class. A value of 100 implies all local subscriptions and advertisements will be of the preference class. Only applicable in fixed-broker mode.
pref_class_same	Specifies whether the preference class assigned for advertisements should be the same as the preference class assigned for subscriptions. Possible values: true (same preference class), and false (different preference class). Only applicable in fixed-broker mode.
value_dist_sub	The distribution of values assigned to attributes. Possible values: UNIFORM and ZIPF. Only applicable in fixed-broker mode.

Workload Parameters

Parameter	**Description**
sub_over_nodes_dist	The distribution function for assigning subscriptions to brokers. Possible values: UNIFORM and GAUSSIAN.
sub_over_nodes_std_dev_factor	The standard deviation factor of the Gaussian function if sub_over_nodes_dist=GAUSSIAN.
adv_over_nodes_dist	The distribution function for assigning advertisements to brokers. Possible values: UNIFORM and GAUSSIAN.
adv_over_nodes_std_dev_factor	The standard deviation factor of the Gaussian function if adv_over_nodes_dist=GAUSSIAN.
pub_over_nodes_dist	The distribution function for assigning publications to brokers. Possible values: UNIFORM and GAUSSIAN.
pub_over_nodes_std_dev_factor	The standard deviation factor of the Gaussian function if pub_over_nodes_dist=GAUSSIAN.

Output Parameters

Parameter	**Description**
write_graph	Indicates if both the similarity matrix describing broker weights and adjacency matrix describing the overlay topology should be output to a file. Possible values: true (output to file) and false (do not output to file).
link_weights	File name of similarity matrix output file.
adjacency_matrix	File name of adjacency matrix output file.

Bibliography

[1] JiST discrete event simulator. http://jist.ece.cornell.edu/.

[2] The knapsack problem. http://en.wikipedia.org/wiki/knapsack_problem.

[3] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. TERA: Topic-based event routing for peer-to-peer architectures. In *DEBS '07*, 2007.

[4] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, and Antonino Virgillito. Subscription-driven self-organization in content-based publish/subscribe. Technical report, 2004.

[5] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni, Antonino Virgillito, and Roma Italia. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *The Computer Journal*, 2007.

[6] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI '06*, 2006.

[7] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001.

[8] Chen Chen, Hans-Arno Jacobsen, and Roman Vitenberg. Divide and conquer algorithms for publish/subscribe overlay design. In *ICDCS '10*, 2010.

[9] Alex King Yeung Cheung and Hans-Arno Jacobsen. Publisher placement algorithms in content-based publish/subscribe. In *ICDCS '10*, 2010.

[10] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *PODC '07*, 2007.

[11] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 2008.

[12] Jack R. Edmonds. Optimum branchings. In *J. Research of the National Bureau of Standards*, 1967.

[13] Eli Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski. Distributed publish/subscribe for workflow management. In *ICFI'05*, 2005.

[14] Katherine Heires. Budgeting for latency: If I shave a microsecond, will I see a 10x profit?, 2010.

[15] Dorit S. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., 1997.

[16] Michael A. Jaeger, Helge Parzyjegla, Gero Mühl, and Klaus Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *SAC '07*, 2007.

[17] Samuel Kounev, Kai Sachs, Jean Bacon, and Alejandro Buchmann. A methodology for performance modeling of distributed event-based systems. *ISORC '08*, 2008.

[18] Fei Li. End-to-end service quality measurement using source-routed probes. In *INFOCOM '06*, 2006.

[19] Guoli Li, Shuang Hou, and Hans-Arno Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS '05*, 2005.

[20] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware '08*, 2008.

[21] Matteo Migliavacca and Gianpaolo Cugola. Adapting publish-subscribe routing to traffic demands. In *DEBS '07*, 2007.

[22] Gero Mühl. *Large-scale content-based publish/subscribe systems.* PhD thesis, Department of Computer Science, Darmstadt University of Technology, 2002.

[23] Melih Onus and Andrea W. Richa. Minimum maximum degree publish-subscribe overlay network design minimum maximum degree publish-subscribe overlay network design. In *INFOCOM '09*, 2009.

[24] Robert. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 1957.

[25] Leonardo Querzoni. Interest clustering techniques for efficient event routing in large-scale settings. In *DEBS '08*, 2008.

[26] John Reumann. "Pub/Sub at Google", lecture and personal communication at CANOE Summer School, Oslo, Norway, 2009.

[27] Anton Riabov, Zhen Liu, Joel L. Wolf, Philip S. Yu, and Li Zhang. Clustering algorithms for content-based publication-subscription systems. In *ICDCS '02*, 2002.

[28] Arnd Schröter, Gero Mühl, Samuel Kounev, Helge Parzyjegla, and Jan Richling. Stochastic performance analysis and capacity planning of publish/subscribe systems. In *DEBS '10*, 2010.

[29] Christoph Schuler, Heiko Schuldt, and Hans-Jörg Schek. Supporting reliable transactional business processes by publish/subscribe techniques. In *TES '01*, 2001.

[30] Sasu Tarkoma. Dynamic content-based channels: meeting in the middle. In *DEBS '08*, 2008.

[31] Ali Tofigh. Optimum Branchings and Spanning Arborescences, 2009.

[32] Spyros Voulgaris, Etienne Rivi, Anne-Marie Kermarrec, and Maarten Van Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *IPTPS '06*, 2006.

[33] Ramana Yerneni. "Internet Advertising", lecture and personal communication at CANOE Summer School, Toronto, Canada, 2010.