OVERLAY NETWORK DESIGN FOR PUBLISH/SUBSCRIBE SYSTEMS

by

Chen Chen

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
in the Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2014 by Chen Chen
Abstract

Overlay Network Design for Publish/Subscribe Systems

Chen Chen
Doctor of Philosophy
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
2014

This thesis focuses on the fundamental problem of constructing a topic-connected overlay (TCO) for a topic-based publish/subscribe system, i.e., all nodes interested in the same topic are organized in a directly connected dissemination sub-overlay, referred to as TCO.

There are two kinds of approaches that aim to construct TCOs while minimizing the node fan-out: (a) centralized algorithms and (b) decentralized protocols. Centralized algorithms suffer from several key shortcomings: 1) prohibitively high runtime cost, 2) require global knowledge and centralized operation, 3) construction of the overlay from scratch only, 4) lack of fault tolerance mechanism, and 5) no provision for churn handling. Decentralized protocols tend to yield overlays with high node degrees. Finally, little work has explored overlay design for content-based pub/sub, which a TCO cannot adequately model.

We develop more efficient and scalable static algorithms and dynamic protocols for pub/sub TCO design. Our contributions include: (1) a general indexing data structure that provides a significantly faster implementation for different state-of-the-art TCO construction algorithms, (2) divide-and-conquer algorithms for the MinAvg-TCO problem, which minimizes the average node degree of TCOs, (3) use of divide-and-conquer techniques for the MinMax-TCO problem that optimize the maximum node degree of TCOs, (4) incorporation of fault tolerance into the pub/sub overlay by extending a TCO to a kTCO, i.e., all nodes interested in the same topic are organized in one k-connected sub-overlay, (5) the design and development of ElastO – a distributed system for constructing and maintaining scalable churn-resistant pub/sub TCOs, (6) finally, we embark on overlay network design for content-based pub/sub systems.
# Contents

1 Introduction .......................... 1
   1.1 Overview .......................... 1
   1.2 Publish/Subscribe Systems ........ 2
   1.3 Problem Statement ................. 3
   1.4 Notation .......................... 4
   1.5 Existing Solutions and Limitations . 6
      1.5.1 TCO Design for Topic-Based Pub/Sub ... 6
      1.5.2 Overlay Design for Content-Based Pub/Sub ... 7
   1.6 Contributions ........................ 9
   1.7 Organization ........................ 10

2 Related Work .......................... 12
   2.1 Overlay Design for Topic-Based Pub/Sub ... 12
      2.1.1 GM algorithm for MinAvg-TCO .... 14
      2.1.2 MinMax-ODA algorithm for MinMax-TCO ... 15
      2.1.3 LowODA algorithm for Low-TCO ... 17
      2.1.4 Decentralized approaches .......... 19
   2.2 Overlay Design for Content-Based Pub/Sub ... 20
   2.3 Peer-to-Peer Overlay Networks .......... 21
   2.4 Distributed Hash Tables ............... 22
4.8 Conclusions ......................................................... 70

5 Scaling Construction of Low Fan-out Pub/Sub TCO .............................. 71
5.1 Introduction ....................................................... 71
5.2 Divide-and-Conquer for MinMax-TCO .................................. 73
  5.2.1 MinMax-ODA as a Building Block for Divide-and-Conquer .......... 73
  5.2.2 Divide and Conquer Phases of the Solution .......................... 74
  5.2.3 Combine Phase of the Solution ................................... 79
  5.2.4 Decentralizing the DCBR-M Algorithm .............................. 86
5.3 Selecting Parameters .............................................. 87
5.4 Evaluation ......................................................... 89
  5.4.1 Partitioning for Lightweight Nodes .................................. 90
  5.4.2 Impact of Coverage Factor ....................................... 92
  5.4.3 Effects Under Different Distributions ............................... 92
  5.4.4 Impact of Number of Nodes ..................................... 93
  5.4.5 Impact of Number of Topics .................................... 93
  5.4.6 Impact of Average Subscription Size ............................. 94
  5.4.7 Comparison with Ring-Per-Topic .................................. 94
5.5 Conclusions ......................................................... 95

6 Constructing Fault-Tolerant Overlays for Topic-based Pub/Sub ............ 96
6.1 Introduction ....................................................... 96
6.2 Parameterized MinAvg- kTCO Problem and its Complexity .................. 98
6.3 GM2 Algorithm to Build 2TCO ...................................... 100
6.4 Approximation Ratio of GM2 .......................................... 101
  6.4.1 Progress Measure Towards the Construction of 2TCO ................ 102
  6.4.2 Estimate of Edge Contribution ................................... 105
  6.4.3 Comparison Against the Unknown Optimum ......................... 107
6.5 HararyPT Algorithm to Build kTCO .................................... 109
6.6 Evaluation ......................................................... 110
8.2.1 Recall and Precision for Pub/Sub ............................................. 144
8.2.2 The Overlay Construction Problem ........................................... 148
8.3 Pub/Sub Routing on δ-Local-View .................................................. 148
8.4 The Overlay Design Algorithm ....................................................... 152
8.5 Evaluation ..................................................................................... 154
  8.5.1 Impact of neighborhood parameter δ ......................................... 155
  8.5.2 Node Degrees ........................................................................... 157
  8.5.3 Recall, Precision and F1-score ................................................... 158
  8.5.4 Runtime cost of overlay construction ........................................ 159
  8.5.5 Runtime cost of message routing .............................................. 160
8.6 Conclusion .................................................................................... 160

9 Conclusions ................................................................................... 162
  9.1 Summary ...................................................................................... 162
  9.2 Future Work ................................................................................ 163

A Analysis of MinAvg-$k$TCO Problem and GM2 Algorithm .................. 166
  A.1 Complexity of the Parameterized MinAvg-$k$TCO Problem .............. 166
  A.2 Correctness and Running Time of the GM2 Algorithm .................... 169
  A.3 Building Blocks to Complete the Proof of Lemma 36 ....................... 169

Bibliography ...................................................................................... 178
## List of Tables

1.1 Approaches to construct pub/sub TCO ........................................... 7

3.1 Algorithms for Solving the TCO Problems ................................. 28

4.1 Random Partitioning Under Uniform Distribution ......................... 64

5.1 Algorithms for the MinMax-TCO Problem .................................. 89
5.2 Random Partitioning Under Exponential Distribution ....................... 91

7.1 Algorithms and protocols that we evaluate ................................ 133

8.1 Example of confusion matrix for pub/sub communication .................. 146
8.2 \( v_2, L'(\delta) \) for Fig. 8.1 ..................................................... 149
8.3 F1-score, Recall, and Precision ................................................. 149
8.4 Example: 4 nodes and 5 messages .............................................. 153
List of Figures

1.1 Topic-connected overlay ........................................ 5

3.1 Illustration of EdgeContrib data structure ...................... 29

3.2 F-MinMax-ODA vs. MinMax-ODA .......................... 40

3.3 F-Low-ODA vs. LowODA .................................... 42

4.1 Bad case for incremental addition of edges ................... 46

4.2 Illustration of star set ....................................... 49

4.3 SM vs. NM vs. GM wrt. avg degree .......................... 63

4.4 SM vs. NM vs. GM wrt. running time ........................ 63

4.5 DC vs. GM wrt. $p$ ........................................ 65

4.6 Inner vs. Outer of DC ....................................... 65

4.7 DC vs. GM wrt. $|V|$ ......................................... 65

4.8 Random partitioning vs. $k$-means clustering for DC on Unif ....... 66

4.9 Random partitioning vs. $k$-means clustering for DC on Zipf ........... 66

4.10 DC vs. DCBC vs. GM vs. RingPT .............................. 67

4.11 DCBC vs. DC vs. GM ....................................... 68

5.1 (a) $TCO_{OPT}$ with $D_{OPT} = 3$. (b) $TCO$ for random partitioning. ... 76

5.2 Poor performance of naive divide-and-conquer .................. 77

5.3 Illustration of representative set ................................ 82

5.4 DCBR-M wrt. $p$ ........................................ 90
5.5 DCBR-M wrt. $\lambda$ ......................................................... 90
5.6 DCBR-M wrt. dist. ......................................................... 90
5.7 DCBR-M wrt. $|V|$ ......................................................... 93
5.8 DCBR-M wrt. $|T|$ ......................................................... 93
5.9 DCBR-M wrt. $|T_{v}|$ ......................................................... 93
5.10 DCBR-M vs. MinMax-ODA vs. RingPT ................................. 95

6.1 An edge sequence forms 2TCO on $t$. ................................ 102
6.2 Example of the $E$-ear-decomposition ................................ 104
6.3 GM$^2$ vs. HararyPT vs. GM vs. CyclePT wrt. $|V|$ ............... 111
6.4 GM$^2$ vs. HararyPT vs. GM vs. CyclePT wrt. $|T|$ ............... 112
6.5 GM$^2$ vs. HararyPT vs. GM vs. CyclePT wrt. the subscription size 113
6.6 Diameters of GM$^2$ vs. GM wrt. $|V|$ ................................ 114

7.1 Illustration of nodeId circle in ElastO .................................. 122
7.2 Modules in ElastO .......................................................... 123
7.3 CCDF of followers and followees – Twitter and Facebook .......... 134
7.4 buildBackupsByPD() ....................................................... 136
7.5 buildBackupsGreedily() ..................................................... 136
7.6 Node degrees ............................................................... 136
7.7 ElastO-L with $\chi$’s - FB 1K ............................................. 137
7.8 Node degrees under churn ................................................. 138
7.9 Runtime cost on FB ......................................................... 139
7.10 Runtime cost on TW ......................................................... 139
7.11 Diameter - TW 1K .......................................................... 141
7.12 Communication - TW 1K .................................................. 141

8.1 Message routing on an overlay ........................................... 149
8.2 ConfODA vs. LowODA under ROL($\delta$) with different $\delta$ .... 155
8.3 (ConfODA, ROL($\delta$)) vs. (LowODA, ROL($\delta$)) under different distributions 156
8.4 Node degrees .......................................................... 157
8.5 Recall, precision and F1-score ........................................... 158
8.6 Runtime cost of overlay construction .................................... 159
8.7 Routing cost ................................................................. 160

A.1 The construction of a 2TCO instance from a 1TCO instance .......... 167
Chapter 1

Introduction

1.1 Overview

Publish/Subscribe (pub/sub) systems constitute an attractive choice as the communication paradigm and messaging substrate for building large-scale distributed systems. Many real-world applications are using pub/sub for message dissemination, such as, online delivery of notifications due to social interaction \[83\], application integration across data centers \[78, 38\], file synchronization in distributed storage systems \[12\], financial data dissemination \[10\], RSS feed aggregation, filtering, and distribution \[64\], business process management \[61\], and algorithmic trading \[81\]. More specifically, Google’s GooPS \[78\] and Yahoo’s YMB \[38\] constitute the distributed messaging substrates for online applications operating worldwide. Microsoft Live Mesh \[12\] uses a pub/sub service to support file synchronization applications. TIBCO RV \[10\] has been used extensively for NASDAQ quote dissemination and order processing. Global Data Synchronization Network (GDSN) \[3\] is a global pub/sub network enabling suppliers and retailers to exchange timely and accurate supply chain data.

Besides, many industrial standards have adopted pub/sub as part of their specifications, including WS Notifications, WS Eventing, the OMG’s Real-time Data Dissemination Service, the Active Message Queuing Protocol, and Extensible Messaging and Presence Protocol (XMPP).

A distributed pub/sub system is often organized as an application-level overlay of nodes (e.g.,
brokers, servers or routers) connected in a federated or in a peer-to-peer manner. The overlay infrastructure directly impacts the pub/sub system’s performance and scalability, such as the message routing cost. Constructing a high-quality broker overlay is a fundamental problem that has received attention both in industry [78, 38] and academia [35, 70, 69, 49, 19].

The scope of this thesis is to provide solutions for pub/sub overlay network design problems. We look at a variety of desirable properties for the overlay, including (a) graph connectivity (e.g., topic-connectivity, see §1.3), (b) low node degree (e.g., average node degree and maximum node degree), (c) efficiency to construct, (d) ease of distributed implementation, (e) fault tolerance, and (f) adaptability to churn. We mainly focus on the overlay network design for topic-based pub/sub, and then suggest an extension to content-based pub/sub.

1.2 Publish/Subscribe Systems

Pub/sub defines two different roles for entities in the system. Publishers are sources that inject their information using publication messages. Subscribers are information sinks and act as consumers of publications that were produced by publishers. Each subscriber needs to issue its subscriptions to specify the types of publications that it would like to consume. The language and data model to subscribe and publish vary among systems. Based on the expressiveness of the language used to represent subscribers interests, pub/sub systems are commonly classified into two types, namely topic-based pub/sub and content-based pub/sub.

In the topic-based pub/sub model, a publisher associates its publication message with a specific topic and subscribers register their interest in a subset of all topics. Topic-based pub/sub is adopted by many large-scale systems and applications [78, 38, 10, 12, 64]. For example, TIBCO RV [10] has been used extensively for market data feed dissemination and Google’s GooPS [78] constitutes the distributed message exchange for web-based applications operating worldwide.

The content-based pub/sub model, on the other hand, supports a more flexible subscription language allowing subscribers to specify their interests using fine-grained predicate-based filtering expressions, called subscriptions. In this case, matching involves examining whether the data encoded in a publication message evaluates a subscription expressions to true. In general, the high level of
expressiveness in a predicate-based language improves the selectivity of subscriptions and enables fine-grained publication filtering. However, content-based pub/sub exhibits more complexity in all aspects of system design, implementation, and evaluation. Examples of systems that fall under the content-based category are Padres [16], Gryphon [21], SIENA [26], JEDI [40], LeSubscribe [72], Hermes [74], and Elvin [82].

1.3 Problem Statement

An overlay network is a logical network that is built on top of a physical one (e.g., the Internet), by maintaining a set of links for each node in the network. Overlay design for distributed systems is in line with the middleware philosophy and provides a well-designed architecture. A high-quality overlay supports decompositions of functionalities among different layers in the abstracted network, and thus reduces the complexity of system design and implementation. Overlay network, if well designed and implemented, significantly improves performance and scalability both theoretically and empirically for many distributed systems, e.g., DHTs [63], peer-to-peer [14], survival networks [58], wireless networks [41], etc.

Overlay design for pub/sub systems is an important problem. The pub/sub overlay directly impacts many crucial aspects of the pub/sub system, including the design and implementation of system (e.g., matching engine and routing scheme) and system performance and scalability (e.g., message routing cost and latency). Unfortunately, overlay design methods for existing distributed systems do not directly apply to pub/sub systems. The key challenge is that the properties and requirements of pub/sub overlay are fundamentally different from traditional overlays for object locating or content streaming. A distributed pub/sub system often connects nodes (e.g., publishers, subscribers, or brokers) as an overlay network to build message matching and event dissemination mechanisms. We want to systematically construct a pub/sub overlay which efficiently supports distributed pub/sub communication and services.

In this thesis, we mainly design overlay networks for topic-based pub/sub systems. We concentrate on the following desirable properties for the overlay:

A. **Topic-connected overlay (TCO):** for each topic, the sub-overlay graph induced by nodes
interested in the topic is connected (see the formal definition of TCO in §1.4).

B. Low fan-out: both maximum and average degrees are small. It is imperative for a pub/sub overlay to have low fan-out because it costs a lot of resources to maintain adjacent links for a high-degree node (i.e., monitor the links and the neighbors [35, 69]). Furthermore, for a typical pub/sub system, each link would have to accommodate a number of protocols, service components, message queues, etc. While overlay designs for different applications might be principally different, they all strive to maintain bounded node degrees, e.g., DHTs [63], peer-to-peer streaming [14], wireless networks [41], or survivable network design [58].

C. Efficiency to construct and maintain the overlay, e.g., time and space complexity of the overlay design algorithms.

D. Fault tolerance: the overlay tolerates a certain number of node failures, and thus the pub/sub service running on top of it is still functioning without interrupting.

E. Low diameters: overlay diameters impact many performance factors for efficient routing in pub/sub, e.g., message latency.

F. Adaptability to churn: it is important to restore the overlay upon node joins and leaves as soon as possible, especially since pub/sub is often running in highly dynamic environment.

1.4 Notation

We now present notation relating to TCO, essential for the problem formulation, the algorithm design and analyses, and evaluations in this thesis.

Let \( I(V, T, Int) \) represent an input instance that represents a topic-based pub/sub, where \( V \) is the set of nodes, \( T \) is the set of topics, and \( Int \) is the interest function such that \( Int : V \times T \rightarrow \{true, false\} \). Since the domain of the interest function is a Cartesian product, we also refer to this function as an interest matrix. Given an interest function \( Int \), we say that a node \( v \) is interested in some topic \( t \) if and only if \( Int(v, t) = true \). We also say that node \( v \) subscribes to topic \( t \).

We denote a topic-based pub/sub overlay network (TPSO) as \( TPSO(V, T, Int, E) \). In particular, a \( TPSO(V, T, Int, E) \) can be illustrated as an undirected graph \( G = (V, E) \) over the node set \( V \) with the edge set \( E \subseteq V \times V \). Given \( TPSO(V, T, Int, E) \), the sub-overlay induced by \( t \in T \) is
a subgraph $G^{(t)} = (V^{(t)}, E^{(t)})$ such that $V^{(t)} = \{v \in V | \text{Int}(v, t)\}$ and $E^{(t)} = \{(v, w) \in E | v \in V^{(t)} \land w \in V^{(t)}\}$. A topic-connected component (TC-component) on topic $t \in T$, is a maximal connected subgraph in $G^{(t)}$. A TPSO is called topic-connected if for each topic $t \in T$, $G^{(t)}$ has at most one TC-component. We denote the topic-connected overlay (TCO) as $\text{TCO}(V, T, \text{Int}, E)$.

![Figure 1.1](image)

Figure 1.1: (1) An overlay $G$. (2) Subgraph $G^{(a)}$ is topic-connected. (3) Subgraph $G^{(b)}$ is not topic-connected.

Aiming to achieve topic-connectivity while optimizing node degrees has resulted in the formulation of various problems: MinAvg-TCO for average degree [35], MinMax-TCO for maximum degree [70], and Low-TCO for both average degree and maximum degree simultaneously [69]. Formal definitions are as follows:

**Problem 1.** MinAvg-TCO($V, T, \text{Int}$): Given a set of nodes $V$, a set of topics $T$, and the interest function $\text{Int}$, construct a TCO which has the least possible total number of edges (i.e., the least possible average node degree).

**Problem 2.** MinMax-TCO($V, T, \text{Int}$): Given a set of nodes $V$, a set of topics $T$, and the interest function $\text{Int}$, construct a TCO with the smallest possible maximum node degree.

**Problem 3.** Low-TCO($V, T, \text{Int}$): Given a set of nodes $V$, a set of topics $T$, and the interest function $\text{Int}$, construct a TCO with both low average and low maximum node degree.
CHAPTER 1. INTRODUCTION

1.5 Existing Solutions and Limitations

1.5.1 TCO Design for Topic-Based Pub/Sub

Overlay network design for distributed pub/sub systems has mainly been concentrating on topic-based pub/sub, both in industry \([78, 38]\) and in academia \([35, 69, 84, 36, 76]\). Gregory Chockler \textit{et al.} define the notion of a \textit{topic-connected overlay} (TCO), which organizes all nodes that subscribe to the same topic into a connected dissemination sub-overlay \([35]\). TCOs support efficient, simple, and secure message routing for topic-based pub/sub. First, a TCO ensures more efficient routing protocols – nodes that not interested in a topic never need to contribute to disseminating information on that topic. Publication routing atop TCOs saves bandwidth and computational resources otherwise wasted on forwarding messages of no interest to the node. Second, TCO results in a simpler matching engine design and smaller forwarding tables. With TCOs it suffices for each node to keep the topics it subscribes to, without the need to maintain the interests of other nodes and to serve as forwarding node. From a security perspective, TCOs are desirable when messages are to be shared across a network among a set of trusted users without leaving this set \([55]\).

The concept of TCO is applicable to both P2P solutions for pub/sub in which the clients form the TCO and broker-based solutions in which the brokers form the TCO. TCO does not differentiate between publishers and subscribers. This abstraction simplifies the presentation for a theoretical and algorithmic treatment of the problem, while fully preserving its practical character.

We can classify existing works into two categories (see Table 1.1): (a) centralized algorithms that statically construct a provably low-degree TCO from scratch and (b) decentralized protocols that strive to dynamically maintain low degrees (and in many cases, topic-connected) in a best-effort fashion. Unfortunately, the former are known to have high runtime cost, which makes them unsuitable as a dynamic solution. Meanwhile, the latter produce significantly higher node degrees compared to the former – e.g., the node degrees produced by PolderCast \([84]\) grow almost linearly with subscription size under typical pub/sub workloads.
# Chapter 1. Introduction

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Churn Handling</th>
<th>Runtime</th>
<th>Avg Degree</th>
<th>Max Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized Algo [69]</td>
<td>Global</td>
<td>✗</td>
<td>Slow</td>
<td>$O(\rho \log</td>
</tr>
<tr>
<td>GM [35]</td>
<td>Global</td>
<td>✗</td>
<td>Slow</td>
<td>$O(</td>
</tr>
<tr>
<td>MinMax-ODA [69]</td>
<td>Global</td>
<td>✗</td>
<td>Slow</td>
<td>$\Theta(</td>
</tr>
<tr>
<td>Decentralized Protocols [36, 76, 84, 45]</td>
<td>Global/Local</td>
<td>✓</td>
<td>Fast</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

## 1.5.2 Overlay Design for Content-Based Pub/Sub

Generally speaking, pub/sub systems are categorized as either topic-based or content-based pub/sub (see §1.2). Topic-based pub/sub systems (e.g., [78, 38, 12, 64]) define a set of topics: each publisher associates its publications with a subset of topics and subscribers register their interests among a subset of topics. In content-based pub/sub systems (e.g., [26, 16, 21, 40, 72, 74, 82]), publishers structure publications as sets of attribute-value tuples and subscribers register their interests through Boolean predicates over these tuples. For example, in a stock quote notification system, a publication could be structured as follows: [Stock = ‘Apple’, Price = 504, Date = ‘Oct-01-2013’], which could be selected by predicates such as: [Stock = ‘Apple’ $\land$ Price $\geq$ 500].

A distributed pub/sub system, either topic-based or content-based, often organizes nodes (e.g., brokers, servers or routers) in a federated or peer-to-peer manner as an overlay at the application or network layer. Once an overlay network is defined, the pub/sub system runs routing protocols that establish appropriate message dissemination paths and deliver messages to all interested nodes. The routing protocol establishes the forwarding function that determines the set of next-hop destinations for a message reaching a node. The overlay structure and routing protocol are closely related – both impact the performance and scalability of the pub/sub system.

In the literature, overlay design algorithms for distributed pub/sub systems have mainly been concentrating on topic-based pub/sub, both in industry [78, 38] and in academia [35, 69, 84, 36, 76]. Gregory Chockler et al. define the notion of a topic-connected overlay (TCO), which organizes all nodes that subscribe to the same topic into a connected dissemination sub-overlay [35]. TCOs
support simple and efficient message routing for topic-based pub/sub. With TCOs it suffices for each node to maintain the topics it subscribes to, without needing to serve as forwarding node for the interests of other nodes.

Content-based pub/sub systems tend to place more emphasis on routing protocols [26, 16, 40, 72, 74, 82, 21]. There exist a few approaches that construct overlays for content-based pub/sub [96, 91, 86]. However, these approaches do not take full advantage of the sophistication of content-based pub/sub and may encounter scalability issues when the system grows (see §2.2 for more details). Most content-based pub/sub systems solely rely on routing protocols supported by sophisticated matching engines that rely on large routing tables at each node [16, 21, 74]. In a typical pub/sub routing protocol (e.g. [26, 16]), each node \( v \) maintains a content-based forwarding table, i.e., a map between \( v \)'s neighbors and predicates. Each predicate \( p_n \) corresponds to the union of the subscriptions of all downstream nodes reachable through node \( n \), one of \( v \)'s neighbors. As a result, to support content-based pub/sub routing, many nodes have to maintain a global view of all subscriptions on all nodes in the system. Consequently, many content-based pub/sub systems suffer from large forwarding tables, excessively high matching complexity, need for selective message flooding, expensive routing computations, and so on.

A well-constructed overlay network could potentially simplify the complexity of content-based pub/sub routing protocols and thus improve the efficiency of message dissemination. However, overlay design poses unique challenges for content-based pub/sub. Techniques for topic-based pub/sub overlay designs are not readily applicable to content-based pub/sub. For example, as opposed to the TCO model for topic-based pub/sub, it is difficult to define a “content-connected overlay” for content-based pub/sub, because there is no explicit boundary in the content-based message space. Advocates of TCO might argue that they can follow a TCO-based approach and define content-connectivity as a graph property such that all nodes interested in the same \textit{content} are connected in a single dissemination sub-overlay. Unfortunately, content-based subscriptions can be highly diverse, and different events may satisfy the interests of widely varying groups of subscribers. If we regard each combination of attribute values or ranges as a single dynamic logical topic, the number of potential logical topics can grow exponentially (i.e., \( 2^{|V|} \) where \( |V| \) is the number of nodes) in the worst-case [20, 23]. Our experiments in §8.5 also show that TCO-based algorithms often end up
CHAPTER 1. INTRODUCTION

with an almost full-mesh overlay for content-based pub/sub.

Low fan-out is another challenging and imperative requirement for the pub/sub overlay. While overlay designs for different applications might be principally different, they all strive to maintain bounded node degrees, e.g., DHTs [63], peer-to-peer streaming [14], wireless networks [41], or survivable network design [58]. First, the node degree represents limited resources – it costs a lot to maintain adjacent links for a high-degree node (i.e., monitor the links and the neighbors [35, 69]). For a typical pub/sub system, each link would also have to accommodate a number of protocols, service components, message queues, etc. Second, the node degree directly influences the size of the pub/sub routing tables, the complexity of matching, and the efficiency of message delivery.

1.6 Contributions

We summarize the main contributions of this thesis as follows:

1. We devise a general indexing data structure that provides a significantly faster implementation, with $O(|V|^2|T|)$ running time, for different state-of-the-art algorithms [33]. The generality of the indexing data structure is due to the fact that it enables edge lookup by both node degree and edge contribution, a central metric in all existing algorithms. When tested on typical pub/sub workloads, the speedup observed was by a factor of over 1 000, thereby rendering the algorithms more suitable for practical use. For example, under a typically Zipf distributed pub/sub workload, with 1 000 nodes and 100 topics, our new implementation completes in 3.823 seconds, while the previous alternative takes over 555 minutes.

2. We derive a number divide-and-conquer algorithms for the MinAvg-TCO problem [32]. Both theoretical analysis and experimental evaluations demonstrate that our divide-and-conquer algorithms seek a balance between time efficiency and the number of links required: our algorithms cost a fraction (up to 1.67%) of the running time cost of their greedy alternatives, which comes at the expense of an empirically insignificant increase in the average node degree.

3. We develop a novel algorithm for the MinMax-TCO problem [30]. As compared to previously known algorithms, our proposed algorithm produces a TCO with marginally higher degrees. At the same time, it has drastically reduced runtime cost, which is corroborated by both theoretical
analysis and empirical evaluation. The latter shows a speedup by a factor of more than 25 on average for typical pub/sub workloads.

4. We incorporate fault tolerance in designing reliable and scalable overlay networks to support topic-based pub/sub communication [34]. We propose the MinAvg-$k$TCO problem parameterized by $k$: use the minimum number of edges to create a $k$-topic-connected overlay ($k$TCO) for pub/sub systems, i.e., for each topic the sub-overlay induced by nodes interested in the topic is $k$-connected. We prove the NP-completeness of MinAvg-$k$TCO and show a lower-bound for the hardness of its approximation. With regard to MinAvg-$2$TCO, we present $GM^2$, the first polynomial time algorithm with an approximation ratio. With regards to MinAvg-$k$TCO, where $k \geq 2$, we propose a simple and efficient heuristic, namely HararyPT, that aligns nodes across different sub-overlays.

5. We propose, ElastO [31], a distributed system for constructing and maintaining scalable churn-resistant overlay networks for topic-based pub/sub systems. ElastO is designed to dynamically tread the balance among several key dimensions: (a) topic-connectivity, (b) low maximum and average node fan-outs, (c) high efficiency to maintain the overlay in presence of churn, (d) balanced computation and communication overhead across all the nodes.

6. We embark on the overlay design for content-based pub/sub [29]. We formalize a family of problems that attempts to capture the key trade-offs in content-based pub/sub overlay design. We propose a simple and efficient heuristic algorithm, which shows scalability in various typical pub/sub workloads. This line of work serves as a stepping stone for new research directions on overlay design for content-based pub/sub.

1.7 Organization

The organization of this thesis is as follows: Chapter 2 places this thesis in the context of related approaches. In Chapter 3, we devise the Gen-ODA algorithm, which provides a significantly faster implementation, with $O(|V|^2|T|)$ running time, for different state-of-the-art TCO construction algorithms. In Chapter 4, we present divide-and-conquer algorithms for the MinAvg-TCO problem that significantly improve time efficiency. In Chapter 5, we develop the DCBR-M algorithm for MinMax-TCO. The algorithm scales the construction of low fan-out TCOs. In Chapter 6, we pro-
pose the MinAvg-$k$TCO problem that incorporates fault tolerance for node failures in designing reliable and scalable topic-based pub/sub overlays. In Chapter 7, we design, ElastO, a hybrid system for constructing and maintaining scalable overlay networks for topic-based pub/sub in presence of churn. In Chapter 8, we extend our work from topic-based pub/sub overlay to content-based pub/sub. Chapter 9 reaches a conclusion and discusses future work.
Chapter 2

Related Work

2.1 Overlay Design for Topic-Based Pub/Sub

In order to improve the performance and scalability of distributed topic-based pub/sub, two directions have crystallized in the literature: (1) the design of routing protocols so that publications and subscriptions are sent in a most efficient way across the overlay network (see [87, 27, 60, 24]) and (2) the construction of the overlay topology such that network traffic is minimized (e.g., [35, 70, 69, 49, 19, 65, 45]). Much attention has focused on the first direction with some recent work addressing the second direction. This thesis focuses on the second direction.

Topic-connectivity is a required property in approaches such as [18, 36, 76] and is an implicit, not directly specified, requirement in approaches such as [27, 17, 19, 28, 45], which all aim to diminish the number of unrelated intermediate overlay hops in one way or another.

Chockler et al. [36] showed empirically that on many practical workloads exhibiting well-correlated subscription patterns, and a simple distributed heuristic could be effective for constructing topic-connected overlays with a small average node degree. [36] emphasized on empirical results, but it serves as a good base for further exploration about theoretical analysis and algorithm design to construct TCOs. Baldoni et al. [19] proposed a self-organizing algorithm to connect brokers matching similar events, which aimed to improve the overall system performance by reducing the overlay hops for event routing. The authors pointed out that an overlay should be constructed in a way such
that brokers sharing interests should be closer to each other.

On the one hand, aiming to achieve TCO while minimizing node degrees has been explored algorithmically [35, 70, 69]. Different optimization goals lead to a number of polynomial-time algorithms. Those algorithms have proven approximation bounds in the worst-case scenarios and can serve as comparison baselines for developing other approaches. The theoretical formulation of the MinAvg-TCO problem originated in [35]. The problem was defined as the construction of a topic-connected overlay network with minimal edges. Chockler et al. [35] proved the hardness of MinAvg-TCO and presented a greedy algorithm, named GM (Greedy Merge) algorithm. The GM algorithm outputs a topic-connected overlay whose average node degree is within a logarithmic bound compared to the optimal solution. Onus and Richa [70] pointed out that GM might produce an overlay with very unbalanced node degrees by effectively creating hotspot nodes with a very high degree. They defined the MinMax-TCO problem which aimed to minimize the maximum degree of the overlay network. More recently, Onus and Richa [69] defined the Low-TCO problem of constructing a topic-connected overlay while minimizing both the maximum and average node degrees. The GM and MinMax-ODA algorithms each focus on minimizing one single node degree metric, either average or maximum node degree. Each algorithm was shown to perform poorly with respect to the complementary metric. Onus and Richa [69] introduced the Low-TCO problem for minimizing both average and maximum node degrees in a topic-connected pub/sub overlay design at the same time. The authors designed the Low Degree Overlay Design Algorithm (LowODA), which achieves sub-linear approximations for both metrics [69]. Unfortunately, all the above state-of-the-art algorithms are static by design: (1) requiring global knowledge, (2) assuming centralized operation, and (3) constructing the overlay from scratch only. Because of these innate static properties the algorithms do not lend themselves to dynamic environments. In particular, the state-of-the-art algorithms suffer from the high runtime complexity – both MinMax-ODA and LowODA have the high time complexity of $O(|V|^4|T|)$, where $|V|$ is the number of nodes and $|T|$ is the number of topics.

On the other hand, systems like [36, 76, 84, 65, 45] build the TCO in a decentralized manner. These systems implement non-coordinated decentralized overlay construction protocols such that each node decides upon its own neighbors. The protocols are fast and can operate with only local
and partial knowledge. However, these heuristics do not provide any theoretical guarantees for the node degrees in these systems. In practice, the node degrees are usually multiple times higher than the bounds provided by the static baselines \cite{35,69}.

### 2.1.1 GM algorithm for MinAvg-TCO

Chockler et al. \cite{35} proved the decision problem for MinAvg-TCO to be NP-complete and proposed a greedy algorithm, namely the Greedy Merge (GM) algorithm, that achieves a logarithmic approximation for the problem. The algorithm is centralized and requires the complete knowledge of $V$ and $Int$. The time complexity for GM is $O(|V|^2|T|)$. The approach is only capable of building an overlay from scratch, whereas in practice due to network partitioning, for instance, the combination of sub-overlays is an important concern.

Alg. 1 specifies the GM algorithm, developed by Chockler et al. \cite{35}. The GM algorithm starts with an empty set of links and iteratively adds carefully selected edges one by one until topic-connectivity is attained. At each iteration, the algorithm greedily selects an edge whose addition to the overlay would maximally reduce the total number of TC-components for all the topics.

In Chapter 4, we use techniques from GM to devise an algorithm for the overlay join problem. We further derive a more efficient solution for the MinAvg-TCO problem in §4.4.1. GM algorithm also serves as baseline in our experimentation.

---

**Alg. 1** Greedy Merge (GM) for MinAvg-TCO

\textbf{GreedyMerge}($I(V, T, Int)$)

\textbf{Input:} $I(V, T, Int)$

\textbf{Output:} A topic-connected overlay $TCO(V, T, Int, E_{GM})$

1: \(E_{pot} \leftarrow \emptyset\)

2: \textbf{for all} \(e = (v, w)\) \textbf{s.t.} \((w, v) \not\in E_{pot}\) \textbf{do}

3: \textbf{add} \(e\) \textbf{to} \(E_{pot}\)

4: \(E_{GM} \leftarrow \text{buildMAEdges}(V, T, Int, \emptyset, E_{pot})\)

5: \textbf{return} $TCO(V, T, Int, E_{GM})$
Algorithm 2 Overlay construction for the GM algorithm

\begin{algorithm}
\textbf{constructOverlayEdges}(V, T, Int, E_{cur}, E_{pot})
\end{algorithm}

\textbf{Input:} V, T, Int, E_{cur}, E_{pot}

// E_{cur}: Set of current edges that exist in the overlay
// E_{pot}: Set of potential edges that can be added

\textbf{Output:} Edge set $E_{new}$ that combined with $E_{cur}$, forms a TCO

1: $E_{new} \leftarrow \emptyset$
2: \textbf{for all} $e = (v, w) \in E_{pot}$ \textbf{do}
3: $\text{contrib}(e) \leftarrow |\{t \in T | \text{Int}(v, t) \land \text{Int}(w, t) \land v, w \text{ belong to different connected components for } t \text{ in } G(V, E_{cur})\}|$
4: \textbf{while} $G(V, E_{new} \cup E_{cur})$ is not topic-connected \textbf{do}
5: $e \leftarrow \text{find edge } e \text{ s.t. } \text{contrib}(e) \text{ is maximum}$
6: $E_{new} \leftarrow E_{new} \cup \{e\}$
7: $E_{pot} \leftarrow E_{pot} - \{e\}$
8: \textbf{for all} $e = (v, w) \in E_{pot}$ \textbf{do}
9: $\text{contrib}(e) \leftarrow \text{update the contribution of a potential edge } e \text{ as the reduction}$
\hspace{1cm} on the number of topic-connected components which would result from the
\hspace{1cm} addition of $e$ to $G(V, E_{new} \cup E_{cur})$
10: \textbf{return} $E_{new}$

\subsection{2.1.2 MinMax-ODA algorithm for MinMax-TCO}

Onus and Richa [70] introduced the MinMax-TCO problem for minimizing the maximum degree in a TCO (see the formal definition in §1.4). MinMax-TCO was proven to be NP-complete and can not be approximated by in polynomial time within a constant factor unless P=NP [70]. Onus \textit{et al.} proposed the MinMax-ODA algorithm, which always delivers a TCO that has a maximum node degree within at most $\log(|V||T|)$ times the minimum possible maximum node degree for any TCO.
Alg. 3 MinMax-ODA for the MinMax-TCO Problem

MinMax-ODA(I(V, T, Int))

Input: I(V, T, Int)

Output: A topic-connected overlay TCO(V, T, Int, E_{MMODA})

1: \( E_{pot} \leftarrow \emptyset \)
2: for all \( e = (v, w) \) s.t. \((w, v) \notin E_{pot} \) do
3: \( \text{add } e \text{ to } E_{pot} \)
4: \( E_{new} \leftarrow \text{buildMMEdges}(V, T, Int, \emptyset, E_{pot}) \)
5: return TCO(V, T, Int, E_{new})

Alg. 4 Overlay Construction for MinMax-ODA

buildMMEdges(V, T, Int, E_{cur}, E_{pot})

Input: V, T, Int, E_{cur}, E_{pot}

// \( E_{cur} \): Set of current edges that exist in the overlay
// \( E_{pot} \): Set of potential edges that can be added

Output: Edge set \( E_{new} \) that combined with \( E_{cur} \), forms a TCO

1: \( E_{new} \leftarrow \emptyset \)
2: for all \( e = (v, w) \in E_{pot} \) do
3: \( \text{contrib}(e) \leftarrow \left| \{t \in T | \text{Int}(v, t) \land \text{Int}(w, t) \land v, w \text{ belong to different connected components for } t \text{ in } G(V, E_{cur}) \} \right| \)
4: while \( G(V, E_{new} \cup E_{cur}) \) is not topic-connected do
5: \( e \leftarrow \text{find edge } e \text{ s.t. } \text{contrib}(e) \text{ is maximum and } e \text{ increases the maximum degree of } G(V, E_{new} \cup E_{cur}) \) minimally
6: \( E_{new} \leftarrow E_{new} \cup \{e\} \)
7: \( E_{pot} \leftarrow E_{pot} - \{e\} \)
8: for all \( e = (v, w) \in E_{pot} \) do
9: \( \text{contrib}(e) \leftarrow \text{update the contribution of a potential edge } e \text{ as the reduction on the number of topic-connected components which would result from the addition of } e \text{ to } G(V, E_{new} \cup E_{cur}) \)
10: return \( E_{new} \)

Alg. 3 specifies MinMax-ODA, which operates in a greedy manner as follows: It starts with an empty set of edges and iteratively adds carefully selected edges one by one until topic-connectivity is attained. The edge selection criterion is as follows: If there exist edges whose addition to the
overlay does not increase the maximum node degree, the algorithms picks an edge with the largest
correlation from the set of all such edges. Otherwise, it selects an edge with the largest correlation among
all edges. The correlation of an edge \( e \), denoted as \( \text{cor}(e) \), is defined as the number
of TC-components reduced by adding the edge to the current overlay.

The following results about \text{MinMax-ODA} were proven in an elegant fashion in \cite{70}:

**Lemma 1 (MinMax-ODA Approximation Ratio & Running Time).** The TCO output by Algorithm
3 has a maximum node degree within a factor of \( \log(|V||T|) \) of the maximum node degree of
the optimal solution for \text{MinMax-TCO}(V, T, Int). Alg. 3 has a running time of \( O(|E_{pot}|^2|T|) =
O(|V|^4|T|) \).

In Chapter 3, we provide a faster implementation for the \text{MinMax-ODA} algorithm such that the
running time is improved from \( |V|^4|T| \) to \( |V|^2|T| \). In Chapter 5, we take \text{LowODA} as a building
block for devising divide-and-conquer algorithms for the \text{MinMax-TCO} problem. We also use \text{MinMax-ODA}
as a comparison baseline for evaluation.

### 2.1.3 \text{LowODA} algorithm for \text{Low-TCO}

Onus and Richa \cite{69} introduced the \text{Low-TCO} problem that aims to optimize both the maximum
node degree and the average node degree in a TCO – the formal definition is in \S 1.4. Onus \textit{et}
al. proposed the \text{LowODA} algorithm. As specified in Alg. 5, \text{LowODA} starts with the overlay
\( G(V, E_{new}) \) where \( E_{new} = \emptyset \), and the algorithm progresses by adding edges to \( E_{new} \) iteration by
iteration until the resulting overlay is topic-connected. At each iteration, \text{LowODA} uses a parameter
\( \rho \) to trade off the balance between maximum and average node degrees. \text{LowODA} makes a weighed
selection between the edge \( e_1 \) chosen by the GM and the edge \( e_2 \) selected by the \text{MinMax-ODA}:
If \( \text{cor}(e_1) \) is greater than \( \rho \cdot \text{cor}(e_2) \), \( e_1 \) is added; otherwise \( e_2 \) is added.

\text{LowODA} is capable of producing a TCO that achieves sub-linear approximation ratios for both
the maximum node degree and the average node degree. Formally speaking, \cite{69} offered the proof
for the following results:

**Lemma 2.** Alg. 5 has the following properties:

(a) the running time is \( O(|V|^4|T|) \).
(b) the output overlay is topic-connected,

(c) the maximum node degree is within a factor of \(O\left(\frac{|V|}{p} \log |V||T|\right)\) from the minimum possible maximum node degree for any TCO on the given input,

(d) the average node degree is within a factor of \(O(p \log |V||T|)\) from the minimum possible average node degree for any TCO on the given input.

Alg. 5 LowODA for Low-TCO Problem

**LowODA**\((V, T; \text{Int}, \rho)\)

**Input:** \((V, T, \text{Int})\), \(\rho\)

**Output:** A topic-connected overlay \(\text{TCO}(V, T; \text{Int}, E_{\text{LowODA}})\)

1: \(E_{\text{new}}, E_{\text{pot}} \leftarrow \emptyset\)
2: for all \(e = (v, w)\) s.t. \((w, v) \notin E_{\text{pot}}\) where \(v, w \in V\) do
3: add \(e\) to \(E_{\text{pot}}\)
4: for all \(e = (v, w) \in E_{\text{pot}}\) do
5: \(\text{contrib}(e) \leftarrow \{|t \in T| \text{Int}(v, t) \land \text{Int}(w, t) \land v, w \text{ belong to different connected components for } t \text{ in } G(V, \emptyset)|\}
6: while \(G(V, E_{\text{new}})\) is not topic-connected do
7: \(e_1 \leftarrow \text{find edge } e_1 \text{ s.t. } \text{contrib}(e_1) \text{ is maximum}\)
8: \(e_2 \leftarrow \text{find edge } e_2 \text{ s.t. } \text{contrib}(e_2) \text{ is maximum and } e_2 \text{ increases the maximum degree of } G(V, E_{\text{new}}) \text{ minimally}\)
9: if \(\text{contrib}(e_1) \geq \text{contrib}(e_2) \times \rho\) then
10: \(e \leftarrow e_1\)
11: else
12: \(e \leftarrow e_2\)
13: \(E_{\text{new}} \leftarrow E_{\text{new}} \cup \{e\}\)
14: \(E_{\text{pot}} \leftarrow E_{\text{pot}} \setminus \{e\}\)
15: for all \(e = (v, w) \in E_{\text{pot}}\) do
16: \(\text{contrib}(e) \leftarrow \text{update the contribution of a potential edge } e \text{ as the reduction on the number of topic-connected components which would result from the addition of } e \text{ to } G(V, E_{\text{new}})\)
17: return \(\text{TCO}(V, T; \text{Int}, E_{\text{new}})\)

In Chapter 3, we design an indexing data structure to improve the running time of the LowODA algorithm, and the time complexity of our LowODA implementation is reduced from \(|V|^4|T|\) to
In Chapter 7, we employ LowODA as a building block for developing more advanced protocols for maintaining TCO under churn. We also use LowODA as a comparison baseline to evaluate our algorithms and protocols.

### 2.1.4 Decentralized approaches

In practice, many pub/sub systems adopt the Overlay-Per-Topic approach, which builds a separate sub-overlay for each topic independently [35, 91]. These systems only need to disseminate publications for a certain topic among the subscribed nodes, thus eliminating the traffic overhead. However, each node should join as many sub-overlays as the number of topics it subscribes to. Thus, the node degree and overlay maintenance overhead grow linearly with the size of node subscriptions. This is not scalable for Internet-scale applications where users may subscribe to a large number of topics. Many systems strive to migrate this scalability issue.

**SpiderCast** exploits the similarities between different nodes in a distributed manner. **SpiderCast** observes that nodes in the network are highly correlated, which is aligned with the theory about small-world networks [54]. Provided subscription correlation among nodes, a single edge can contribute to more than one sub-overlays for many topics. **SpiderCast** employs two local heuristics for neighbor selection: greedy and random coverage. Each node utilizes its partial views to minimize the average node degree while achieving TCO (with high probability).

**StAN** [65] aligns multiple independent sub-overlays in order to promote link sharing. Despite being managed independently and in a decentralized fashion, the sub-overlays converge to share a large number of links due to the subscription correlation.

**PolderCast** [84] maintains a ring structure for each pub/sub topic using gossiping techniques. Over these rings, PolderCast combines deterministic dissemination with probabilistic dissemination, which follows a limited number of random shortcuts.

Although these systems outperform the naive Overlay-Per-Topic approaches, they do not guarantee any theoretical bounds on the output overlays, and the node degrees of these systems are usually a number of times higher than the output of static TCO algorithms [35, 69, 33, 30, 32].

Apart from Overlay-Per-Topic approaches, some decentralized systems attempt to cluster nodes with similar subscription interests. **Vitis** [76] enables rendezvous routing on unstructured overlay.
Vitis assigns a dedicated rendezvous node for each topic and builds a sub-cluster for each topic. Vitis relies on rendezvous nodes for the communication among sub-clusters. Magnet [45] is also a rendezvous-based pub/sub systems. Magnet dynamically clusters nodes with similar subscriptions into dissemination structures based on a skewed DHT. However, these systems have several limitations: (1) they do not ensure topic-connectivity; and (2) the reliance on central nodes conceptually limits the potential for high hit-ratio under churn.

In contrast, Chapter 7 adopts a principally different and less decentralized approach that resides between the static algorithms and decentralized protocols. In our system (i.e., ElastO), only a subset of nodes, which we call shadow set, are involved in the automatic overlay recovery when churn occurs. The conceptual idea of shadow set also appears in [59], which, however, is not designed for pub/sub communication and is nontrivial to address the new topological properties required by pub/sub. We also tailor recent peer sampling services [50, 51] for our own purposes and rely on existing failure detectors [42] as a building block.

### 2.2 Overlay Design for Content-Based Pub/Sub

Research on distributed pub/sub systems has been focusing on the design of efficient routing protocols to disseminate messages from numerous publishers to large numbers of subscribers. The premise of these approaches are based on sophisticated matchine schemes, large forwarding tables, especially for content-based pub/sub [26, 16, 21, 40, 72, 74, 82]. A number of optimization techniques have been developed for content-based pub/sub routing, such as advertisement forwarding [26, 16], assigning rendezvous points [22], constructing matching trees [21], etc. However, in all these approaches, some (or all) nodes still need to collect large volumes of subscriptions from across the nodes in the system.

More recent research aims at constructing the underlying overlays for pub/sub. In particular, overlay design for topic-based pub/sub results in less complicated routing protocols and more efficient message dissemination [35, 70, 69, 49, 19, 65, 45].

A few approaches construct overlays for content-based pub/sub by converting “content” into “topics” [96, 91, 75, 86]. Hyper [96] and Sub-2-Sub [91] cluster the content-based message space
into a number of subspaces (or virtual groups), and then they treat each subspace like a topic in topic-based pub/sub. However, it is not always straightforward and effective to transform content-based pub/sub into topic-based pub/sub, especially when publications have numerous attributes and subscriptions are complex. How to scale these approaches remains an open problem.

Hyper [96] dynamically identifies a number of virtual groups in the content space. Hyper partitions the content space into a regular grid, and forms the virtual groups by combining a set of cells that follow the same “features” based on common subscriptions. However, Hyper only discusses high-level heuristics, but does not provide concrete methods to handle key challenges, such as how to identify virtual groups and how to determine the size of the cell in the grid.

Sub-2-Sub [91] clusters the event space into multiple sub-spaces. A subspace has all the nodes that subscribe to this subspace and does not contain any node that is not interested in this subspace. In other words, a subspace has all the nodes that subscribe to this subspace and does not contain any node that is not interested in this subspace. Sub-2-Sub builds a ring over each subspace for disseminating the events inside that subspace. However, constructing sub-spaces is difficult when the subscriptions have a large number of attributes, which may result in scalability issues. For example, Sub-2-Sub was simply evaluated on pub/sub workloads with three attributes.

To evaluate the efficiency and performance of pub/sub systems, the pub/sub literature often uses recall and precision without formally defining these notions. Many approaches implicitly strive to reach 100% for both metrics. High hit ratio is a crucial requirement in [27, 84, 75], which reflects the definition of recall. Eliminating the number of pure forwarding nodes (e.g., aimed for in [35, 36]) is a technique that improves precision.

### 2.3 Peer-to-Peer Overlay Networks

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. We say that they form a P2P network of nodes. There are numerous applications of P2P networks. The most commonly known is content distribution, such as content delivery [4, 83], communication networks [8], networking [2], and search [11].
A P2P network is usually subjected to high dynamism: nodes can join or leave the network continuously and concurrently. Also network capacities change due to congestion, link failures, etc. We call this phenomenon churn, and any distributed systems (especially P2P), therefore, must handle churn in order to provide a reasonable quality of service. We focus on churn handling for pub/sub overlay in Chapter 7.

P2P systems often implement an abstract overlay network, built at the application layer, on top of the native or physical network topology. We use such overlays for indexing and peer discovery and make the P2P system independent from the physical network topology.

The P2P overlay network consists of all the participating peers as network nodes. There are links between any two nodes that know each other: i.e., if a participating peer knows the location of another peer in the P2P network, then there is a directed edge from the former node to the latter in the overlay network. Based on how the nodes in the overlay network are linked to each other, we can classify the P2P networks as structured or unstructured.

P2P Overlay network infrastructures are an effective way for realizing a large-scale pub/sub communication. For example, Bayeux [98] and Scribe [27] are designed for topic-based pub/sub, and Hermes [74], Rebeca [89] and [68, 88] target at content-based pub/sub. We develop a P2P-like system for maintaining pub/sub overlay network in Chapter 7.

### 2.4 Distributed Hash Tables

Distributed hash tables (DHTs) are a class of structured and decentralized P2P systems that provide a lookup service similar to a hash table: DHT stores (key, value) pairs, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. In 2001, four systems – CAN [77], Chord [85], Pastry [80], and Tapestry [97] – ignited DHTs as a popular research topic, and this area of research remains active. Outside academia, DHT technology has been adopted as a component of BitTorrent [1] and in the Coral Content Distribution Network [9].
Most DHTs use some variant of consistent hashing to map keys to nodes. Consistent hashing has the essential property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected. In contrast, a traditional hash table usually needs to re-map nearly the entire key space when adding or removing one bucket. Since any change in ownership typically corresponds to bandwidth-intensive movement of objects stored in the DHT from one node to another, minimizing such reorganization is required to efficiently support high rates of churn (node arrival and failure).

In DHT, each node maintains a set of links to other nodes (its neighbors or routing table). Together, these links form the overlay network. A node picks its neighbors according to a certain structure, called the network’s topology. All DHT topologies share some variant of the most essential property: for any key $k$, each node either has a node ID that owns $k$ or has a link to a node whose node ID is closer to $k$, in terms of the key space distance the DHT defines. It is then easy to route a message to the owner of any key $k$ using the following greedy algorithm (that is not necessarily globally optimal): at each step, forward the message to the neighbor whose ID is closest to $k$. When there is no such neighbor, then we must have arrived at the closest node, which is the owner of $k$. This style of routing is sometimes called key-based routing.

DHTs form an infrastructure that can be used to build more complex services, such as anycast, cooperative Web caching, distributed file systems, domain name services, instant messaging, multicast, and also peer-to-peer file sharing and content distribution systems. We use DHTs (e.g., consistent hashing and key-based routing) for efficient churn handling in pub/sub overlay maintenance in Chapter 7.
Chapter 3

Generalized Algorithms for Pub/Sub

TCO Construction

3.1 Introduction

Several centralized algorithms have been proposed for constructing topic-connected overlays with
the average node degree or the maximum node degree provably close to the optimal ones [35, 70,
69, 30, 32]. These state-of-the-art algorithms target overlay construction in a managed large cluster
of up to thousands of servers where full mesh solutions exhibit scalability problems [78, 38]. Such
clusters are characterized by a large degree of control and relatively low churn rates (in the order of
one change every hour, depending on the size of the cluster [5]), which makes centralized overlay
construction a viable solution. Besides, these algorithms serve as stepping stones and comparison
baselines for dynamic environments and decentralized overlay construction protocols.

However, the algorithms in [70, 69, 32] have the prohibitively expensive runtime cost of $O(|V|^4|T|)$
where $|V|$ is the number of nodes and $|T|$ is the number of topics. This fundamental drawback makes
the algorithms non-suitable for the managed cluster environment because it takes tens of minutes or
hours to compute an overlay for a realistic scale on a high-end machine. The runtime cost also limits
the applicability of the algorithms as a comparison baseline.
The main contribution of this chapter is that we generalize the above algorithms and come up with a new indexing data structure that supports a significantly faster implementation, with $O(|V|^2|T|)$ time efficiency. Specifically, all algorithms follow the same pattern: they iteratively add edges until the resulting overlay satisfies topic-connectivity. The data structure that we propose exhibits the following properties: (a) its initialization complexity is $O(|V|^2|T|)$, (b) the cumulative complexity of selecting an edge at all iterations is $O(|V|^2|T|)$, and (c) the amortized complexity of updating the data structure over all iterations is also $O(|V|^2|T|)$. The generality of the indexing data structure is due to the fact that it allows edge lookup by both node degree and the edge contribution, a central metric in the above algorithms.

To complement the theoretical analysis, we conduct comprehensive experiments under a variety of characteristic pub/sub workloads. Our experiments show that on average, for a typical pub/sub scale and interest distribution, our generalized algorithm with its efficient implementation builds the same overlay as previously known state-of-the-art algorithms in less than 0.37% of the running time. For example, under the Zipf distributed pub/sub workload, with 1000 nodes and 100 topics, our new implementation completes in 3.823 seconds, while the previous alternative takes over 555 minutes.

### 3.2 Generalized Overlay Design Algorithm

In this section, we introduce Generalized Overlay Design Algorithm (Gen-ODA) as specified in Alg. 6. It captures the similarities embedded in the GM, MinMax-ODA, and LowODA algorithms and offers an easy-to-specialize pattern for studying families of algorithms for solving TCO design problems. We illustrate some of the specializations of this pattern in this chapter.

Gen-ODA starts with the overlay $G(V, E_{new})$ where $E_{new} = \emptyset$ so that there are $|\{v: \text{Int}(v, t)\}|$ singleton TC-components for each topic $t \in T$, i.e., there are $\sum_{t \in T} |\{v: \text{Int}(v, t)\}|$ separate TC-components in total. The algorithm progresses by adding edges to $E_{new}$, thus merging TC-components until $G(V, E_{new})$ contains at most one TC-component for each $t \in T$, i.e., the resulting overlay is topic-connected.

At each step, an edge $e$ is selected from the potential edge set $E_{pot}$ by **findEdge**() in Line 6 of Alg. 6. Specific algorithms for different TCO problems have their own rules for edge selection, i.e.,
findEdge() is a virtual function that needs to be overwritten with an implementation of a concrete criterion, which governs edge selection. We next illustrate these rules for the above listed algorithms. The rules are based on a combination of two criteria: node degree and edge contribution, which is defined as reduction in the number of TC-components caused by the addition of the edge to the current overlay. The edge contribution for an edge $e$ is denoted as $\text{contrib}(e)$.

Alg. 6 Generalized Overlay Design Algorithm

$\text{Gen-ODA}(V, T, \text{Int})$

**Input:** $V, T, \text{Int}$

**Output:** A topic-connected overlay $\text{TCO}(V, T, \text{Int}, E)$

1: $E_{\text{new}}, E_{\text{pot}} \leftarrow \emptyset$
2: for all $e=(v, w)$ s.t. $(w, v) \notin E_{\text{pot}}$ where $v, w \in V$ do
3: add $e$ to $E_{\text{pot}}$
4: initDataStructures()
5: while $G(V, E_{\text{new}})$ is not topic-connected do
6: $e \leftarrow \text{findEdge}()$
7: $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{e\}$
8: $E_{\text{pot}} \leftarrow E_{\text{pot}} - \{e\}$
9: updateDataStructures($e$)
10: return $\text{TCO}(V, T, \text{Int}, E_{\text{new}})$

1. Chockler et al. [35] use the GM-rule for edge selection with regard to MinAvg-TCO: GM greedily selects an edge with the highest contribution (regardless of the node degree). An optimized implementation of GM has the runtime of $O(|V|^2|T|)$. GM achieves a logarithmic approximation ratio for the average node degree; however, GM only provides an approximation ratio of $\Theta(|V|)$ for the maximum node degree [70].

2. Onus et al. [70] use the MinMax-ODA-rule for edge selection with regard to MinMax-TCO: MinMax-ODA also selects the edge with the highest contribution, but only among the edges that would minimally increase the maximum node degree. MinMax-ODA always produces a TCO that has a maximum node degree within at most $\log(|V||T|)$ times the optimal maximum node degree. However, MinMax-ODA only attains an approximation ratio of $\Theta(|V|)$ for the average node degree [69].
3. Onus et al. [69] propose the LowODA-rule for solving the Low-TCO problem: LowODA uses a parameter $k$ to trade off the balance between average and maximum node degrees. The algorithm makes a weighed selection between the edge $e_1$ chosen by the GM-rule and the edge $e_2$ selected by the MinMax-ODA-rule: If $\text{contrib}(e_1)$ is greater than $k \cdot \text{contrib}(e_2)$, $e_1$ is added; otherwise $e_2$ is added. LowODA achieves sub-linear approximation ratios on both average and maximum node degrees.

Both MinMax-ODA and LowODA find an edge in $O(|V|^2)$ time by scanning all potential edges in a brute force manner, which leads to the time complexity of $O(|V|^4|T|)$ [70, 69]. This runtime cost is the main impediment for deploying the algorithms in a relatively static cluster environment where the large degree of control makes a centralized overlay construction feasible. Furthermore, it limits the scale of validation for MinMax-ODA and LowODA which in turn diminishes the potential for using these algorithms as the building blocks (e.g., in the design of divide and conquer algorithms [32]) and the comparison baselines for distributed alternatives.

3.3 Fast Implementation of TCO Construction Algorithms

This section offers an efficient implementation for our proposed Gen-ODA algorithm pattern and its various instantiations. With Alg. 6 as the common pattern, functions initDataStructures() and updateDataStructures() are shared by different instantiations of Gen-ODA, while findEdge() is specialized for different edge selection rules. The fast implementation is based on the new indexing structure that we introduce in this work. A simpler structure was used in [35], which only provided indexing by the edge contribution. In contrast, the structure we propose in this work allows for indexing both by the edge contribution and node degree. In particular, the use of this structure allows us to implement a faster version of MinMax-ODA and LowODA running in $O(|V|^2|T|)$ time. By using this faster version, we can accelerate the efficiency of divide-and-conquer algorithms proposed in [32].

We first present the central data structures and elementary functions utilized in our fast implement-
different edge selection rules under the umbrella of this common algorithm pattern. We prove results about the runtime complexity for each of these elements, which allows us to derive the total complexity of $O(|V|^2|T|)$. Table 5.1 summarizes the overlay construction problems and algorithms, which will be discussed in this section.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinAvg-TCO</td>
<td>Minimum Average Degree TCO Problem</td>
<td>$O(</td>
</tr>
<tr>
<td>GM</td>
<td>Greedy Merge algorithm [35], $O(</td>
<td>V</td>
</tr>
<tr>
<td>F-MinAvg-ODA</td>
<td>Fast implementation for GM, $O(</td>
<td>V</td>
</tr>
<tr>
<td>MinMax-TCO</td>
<td>Minimum Maximum Degree TCO Problem</td>
<td></td>
</tr>
<tr>
<td>MinMax-ODA</td>
<td>Minimum Maximum Degree Overlay Design Algorithm [70], $O(</td>
<td>V</td>
</tr>
<tr>
<td>F-MinMax-ODA</td>
<td>Fast MinMax-ODA, $O(</td>
<td>V</td>
</tr>
<tr>
<td>Low-TCO</td>
<td>Low Avg and Max Degree TCO Problem</td>
<td></td>
</tr>
<tr>
<td>LowODA</td>
<td>Low Degree Overlay Design Algorithm [69], $O(</td>
<td>V</td>
</tr>
<tr>
<td>F-Low-ODA</td>
<td>Fast LowODA, $O(</td>
<td>V</td>
</tr>
</tbody>
</table>

*: ALG stands for any of the discussed algorithms.

### 3.3.1 Indexing Data Structure for Fast Implementation

We introduce an indexing data structure, EdgeContrib, as the underlying bedrock for our fast implementation of Gen-ODA. We opt to present EdgeContrib in Class 7 using object-oriented design principles, because: (1) it provides a standard interface that can be reused efficiently to develop key functions of Gen-ODA; and (2) the grouping of data and procedures facilitates reasoning about the algorithms and time complexity.

EdgeContrib defines an internal class EdgeStruct, which encapsulates an edge and meta-information about it, such as its contribution. Besides, EdgeStruct contains pointer fields prev and next to allow inclusion into a doubly-linked list. EdgeContrib contains two additional member attributes: edgeArray and edgeMap. As illustrated in Fig. 3.1(a), member edgeArray is a 2-dimensional array of size $|T| \times |V|$, which is designed for quick search for the “best” edge at each iteration of Alg. 6. All potential edges (encapsulated in EdgeStruct objects) are put into this
2-dimensional array according to the edge contribution and bigger node degree of the edge. Each entry \( \text{edgeArray}[c][d] \) contains a doubly-linked list of \( \text{EDGESTRUCT} \) objects corresponding to different edges with contribution \( c \) and higher node degree \( d \). More specifically, if \( \text{EDGESTRUCT}(e(v, w)) \in \text{edgeArray}[c][d] \), then: \( c = \text{contrib}(e) \), i.e. adding \( e \) to the overlay at the current iteration will reduce the number of \( TC\)-components of \( G(V, E_{\text{new}}) \) by \( c \), \( 1 \leq c \leq |T| \), and \( d = \max\{\text{deg}(v), \text{deg}(w)\} \), where \( \text{deg}(v) \) is the degree of node \( v \) in \( G(V, E_{\text{new}}) \). Member \( \text{edgeMap} \) is a hashtable such that given an edge \( e \), \( \text{edgeMap} \) allows for an efficient lookup of the corresponding \( \text{EDGESTRUCT}(e) \). In a well-dimensioned hashtable, arbitrary insertions, lookups and deletions have a constant average time cost per operation.

![Table and Diagram](image)

Figure 3.1: (a) \( \text{EdgeContrib, edgeArray} \) (b) \( TCC\) Nodes

While the implementation of individual functions in \( \text{EdgeContrib} \) is rather straightforward, it is important to observe that each function has a per-invocation runtime cost of \( O(1) \). Edge addition or deletion takes constant time thanks to the use of a doubly-linked list. Edge lookup takes \( O(1) \) due to using the \( \text{edgeMap} \) hashtable. This property of the constant per-invocation cost is essential for the time efficiency of updating all \( \text{EDGESTRUCTs} \) in \( \text{edgeArray} \) after adding each edge to the overlay, as we further elaborate upon in Lemma 5.
Class 7 EdgeContrib Interface and Implementation

// Definition of EdgeStruct - data structure for EdgeContrib entries

EdgeStruct: an encapsulation of an edge and its corresponding information. It is implemented as an element in a doubly-linked list, so that inserting and deleting an edge can be performed in constant time.

- \( e(v, w) \): the edge
- \( \text{contrib} \): the edge contribution of \( e(v, w) \), i.e., \( \text{contrib}(e) \)
- \( \text{prev} \): pointer to its predecessor in the linked list
- \( \text{next} \): pointer to its successor in the linked list
- \( \text{degree} \): \( \max\{\deg(v), \deg(w)\} \) where \( \deg(v) \) is the degree of node \( v \) in \( G(V, E_{\text{new}}) \)

// Member attributes and auxiliary variables for EdgeContrib

▷ edgeArray: a 2-dimensional array with \(|T| \times |V|\) entries, each representing a set of edges (and their corresponding information) chosen from \( V \times V \). An edge \( e(v, w) \) is wrapped in an EdgeStruct object (see the data structure definition above), denoted as \( \text{EdgeStruct}(e) \), when storing in an entry of edgeArray. If \( \text{EdgeStruct}(e) \in \text{edgeArray}[c][d] \), then: (1) \( e \in E_{\text{pot}} \); (2) \( c = \text{EdgeStruct}(e).\text{contrib} \); (3) \( d = \text{EdgeStruct}(e).\text{degree} \).

▷ edgeMap: A hashtable that maps an edge \( e \) (as a key) to its associated EdgeStruct(e) (as a value) in edgeArray.

// Functions for EdgeContrib

▷ initEntry(c, d)
1: \( \text{edgeArray}[c][d] \leftarrow \emptyset \)

▷ insertEdge(e(v, w), c, d)
1: construct \( \text{EdgeStruct}(e) \) s.t. \( \text{contrib} = c \) and \( \text{degree} = d \)
2: put key-value pair \( (e, \text{EdgeStruct}(e)) \) into edgeMap
3: insert \( \text{EdgeStruct}(e) \) into \( \text{edgeArray}[c][d] \)

▷ deleteEdge(e(v, w), c, d)
1: \( \text{EdgeStruct}(e) \) from edgeMap using \( e \) as the key
2: delete key-value pair \( (e, \text{EdgeStruct}(e)) \) from edgeMap
3: delete \( \text{EdgeStruct}(e) \) from \( \text{edgeArray}[c][d] \)

▷ getOneEdge(c, d)
1: return the first edge from \( \text{edgeArray}[c][d] \)

▷ getContrib(e(v, w))
1: get \( \text{EdgeStruct}(e) \) from edgeMap by key \( e \)
2: return \( \text{EdgeStruct}(e).\text{contrib} \)

▷ getDegree(e(v, w))
1: get \( \text{EdgeStruct}(e) \) from edgeMap by key \( e \)
2: return \( \text{EdgeStruct}(e).\text{degree} \)

▷ entrySize(c, d)
1: return \( |\text{edgeArray}[c][d]| \)
3.3.2 Common Implementation Pattern

**Alg. 8 Global Variables**

- **EdgeContrib**: an indexing data structure designed for quick search for the best candidate edge using various edge selection rules. See Class 7.
- **TCC-Nodes**: a 2-dimensional array of size $|V| \times |T|$ in which each element $TCC-Nodes[v][t]$ is a subset of $V$ s.t. for each $w \in TCC-Nodes[v][t]$, (1) $Int(w, t) = true$, and (2) both $w$ and $v$ belong to the same $TC$-component for $t$.
- **Enew**: set of edges in the overlay built so far.
- **Epot**: set of potential edges that can be added.
- **nodeDegree**: an array with length $|V|$ s.t. $nodeDegree[v]$ is the degree of node $v$ in $G(V, E_{new})$.
- **mazContrib**: the highest edge contribution in $E_{pot}$.
- **mazDegree**: the maximum node degree in $G(V, E_{new})$.
- **curContrib**: contribution of the currently selected edge.
- **curDegree**: the higher node degree of the currently selected edge.

We have showed the outline of Gen-ODA in Alg. 6. A more detailed description with actual data structures for Gen-ODA is presented in the following algorithms: definitions of data structures (Alg. 8), initialization of data structures (Alg. 9) and the update of data structures after each edge addition (Alg. 10). GM [35], MinMax-ODA [70] and LowODA [69] all fit into the framework of the Gen-ODA, and the only difference is that they use different criteria to select an edge at each iteration (Line 6 of Alg. 6).

Our implementation of Gen-ODA uses several global variables defined in Alg. 8. Among these data structures, **EdgeContrib** and **TCC-Nodes** play the most important roles (see Fig. 3.1). **EdgeContrib** is an indexing data structure designed to organize all potential edges (see Class 7). **TCC-Nodes** is a 2-dimensional array of size $|V| \times |T|$ which keeps track of the $TC$-components in the current overlay $G(V, E_{new})$: $TCC-Nodes[v][t]$ holds the set of nodes belonging to the same $TC$-component for $t$ as $v$. To support all these variables for Gen-ODA, a polynomial space is sufficient.

**Lemma 3.** Alg. 6 takes $O(|V|^2|T|)$ space.

The initialization of these data structures (Alg. 9) takes place at the very beginning of the Gen-ODA algorithm. Gen-ODA starts with the overlay $G(V, \emptyset)$, and Alg. 9 initializes all global variables defined in Alg. 8 accordingly. Lemma 4 shows the time complexity of the initialization.
Alg. 9 Data Structure Initialization

\begin{algorithm}
\caption{initDataStructures()}
\begin{algorithmic}
\STATE 1: \textbf{for all} $v \in V$ \textbf{do}
\STATE 2: \hspace{1em} $\text{nodeDegree}[v] \leftarrow 0$
\STATE 3: \textbf{for all} $v \in V$ \textbf{and} $t \in T$ \textbf{such that} $\text{Int}(v, t)$ \textbf{do}
\STATE 4: \hspace{1em} $\text{TCC-Nodes}[v][t] \leftarrow \{v\}$
\STATE 5: \textbf{for} $c \leftarrow |T|$ \textbf{down to} 1 \textbf{do}
\STATE 6: \hspace{1em} \textbf{for} $d \leftarrow 0$ \textbf{to} $|V| - 1$ \textbf{do}
\STATE 7: \hspace{2em} $\text{EdgeContrib}.\text{initEntry}(c, d)$
\STATE 8: \textbf{for all} $e = (v, w) \in \text{E}_{\text{pot}}$ \textbf{do}
\STATE 9: \hspace{2em} $c \leftarrow |\{t \in T \mid \text{Int}(v, t) \land \text{Int}(w, t)\}|$
\STATE 10: \hspace{2em} \textbf{if} $c > 0$ \textbf{then}
\STATE 11: \hspace{3em} $\text{EdgeContrib}.\text{insertEdge}(e, c, 0)$
\STATE 12: \hspace{2em} $\text{maxContrib} \leftarrow \max\{e \mid \exists d \text{ s.t. } \text{EdgeContrib}.\text{entrySize}(c, d) > 0\}$
\STATE 13: \hspace{2em} $\text{curContrib} \leftarrow \text{maxContrib}$
\STATE 14: \hspace{2em} $\text{curDegree} \leftarrow 0$, $\text{maxDegree} \leftarrow 0$
\end{algorithmic}
\end{algorithm}

Lemma 4. The running time of Alg. 9 is $O(|V|^2|T|)$.

\textit{Proof.} The cost of Gen-ODA’s initialization is dominated by the calculation of edge contribution for all potential edges $E_{\text{pot}}$ in Lines 8-11 of Alg. 9. If the interest of each node is stored as a list of topics, then the complexity of this computation for $E_{\text{pot}}$ will be $O(\sum_{e=(v, w) \in E_{\text{pot}}} |\{t \in T \mid \text{Int}(v, t) \land \text{Int}(w, t)\}|) = O(|V|^2|T|)$. \hfill $\square$

After adding $e$ to the overlay and removing it from the potential set (Line 7-8 in Alg. 6), we ought to re-arrange $\text{EdgeContrib}$ and $\text{TCC-Nodes}$ dynamically to reflect the new edge contributions, $\text{TC-components}$, and node degrees (Line 9 in Alg. 6). This is performed by Alg. 10.

As shown in Alg. 10, the update has four parts: (1) Lines 1-2 update $\text{curContrib}$ and $\text{curDegree}$ using the currently selected edge; (2) Lines 4-14 update edge contributions for $\text{EdgeStructs}$ stored in $\text{EdgeContrib}$ and $\text{TC-components}$ recorded in $\text{TCC-Nodes}$; (3) Lines 17-23 update the array entries in $\text{EdgeContrib}$ according to node degrees of $G(V, E_{\text{new}})$; (4) Lines 24-28 update the global variable $\text{maxContrib}$.
Part (1) and Part (4) deal with basic data types, and are relatively straightforward. Parts (2) and Part (3) are responsible for handling complex data structures.

In Part (2), Lines 6-11 update the contribution of each edge affected by the addition of \( e(v, w) \) to the overlay. An edge is affected if its endpoints belong to different TC-components prior to the addition but those components are merged as a result of the addition. Once edge \( e(v, w) \) is added to the overlay, two TC-components are merged into a single one \( \text{new}_\text{tcc}\_\text{nodes} = \text{TCC-Nodes}[v][t] \cup \text{TCC-Nodes}[w][t] \) (Lines 12). Accordingly, for each node \( u \in \text{new}_\text{tcc}\_\text{nodes} \), \( \text{TCC-Nodes}[u][t] \) is updated (Lines 13-14).

In Part (3), Alg. 10 handles the node degree update. Lines 15-16 update global variables \( \text{nodeDegree} \) and \( \text{maxDegree} \) following the addition of a new edge \( e(v, w) \). Lines 17-23 exam-
ine all potential edges incident on either \( v \) or \( w \) and update the corresponding node degrees as the dimension in \( \text{EdgeContrib.edgeArray} \). For each edge \( e'(v', w') \), Line 18 retrieves the old degree as the index in \( \text{EdgeContrib.edgeArray} \), and Line 19 computes the new degree in \( G(V, E_{\text{new}}) \); Lines 20-23 update the indexing structure if \( d_{\text{old}} < d_{\text{new}} \).

Lemma 5 shows the cumulative running time of updates performed by Alg. 10 for all edges added to the TCO.

**Lemma 5.** The cumulative running time of all invocations of Alg. 10 during the entire execution of Alg. 6 is \( O(|V|^2|T|) \).

**Proof.** The runtime cost of updates in Alg. 10, invoked after adding an edge, is dominated by Part (2) and Part (3).

When updating a contribution or a degree for an edge, we can efficiently locate its corresponding \( \text{EDGESTRUCT} \) entry in \( \text{EdgeContrib.edgeArray} \) in amortized \( O(1) \) time with the assistance of \( \text{EdgeContrib.edgeMap} \) (see Class 7 in §3.3.1). The update of each individual \( \text{EDGESTRUCT} \) in \( \text{EdgeContrib} \) can be performed in \( O(1) \) by constant operations of \( \text{EdgeContrib.deleteEdge()} \) and \( \text{EdgeContrib.insertEdge()} \).

In Part (2), in order to calculate the total count of individual edge updates at all iterations, it is sufficient to notice that every update decrements the contribution of the edge by one (Lines 6-11). Alg. 6 starts when the total contribution of all edges is \( O(\sum_{e=(v,w)\in E_{\text{pot}}} |\{t \in T | \text{Int}(v, t) \land \text{Int}(w, t)\}|) = O(|V|^2|T|) \), and terminates when the contribution of all the edges is reduced to zero. Therefore, the cumulative cost of updating all edge contributions is \( O(|V|^2|T|) \).

In Part (3), the update of node degrees when adding an edge is bounded by \( O(|V|) \), and the size of the output edge set is at most \( O(\min\{|V||T|, |V|^2\}) \), so the overall cost of updating node degrees is \( O(\min\{|V|^2|T|, |V|^3\}) \).

In summary, the cumulative runtime cost of updates for all edges added to the overlay is \( O(|V|^2|T|) \). \( \square \)

Having presented an efficient implementation of \texttt{initDataStructures()} and \texttt{updateDataStructures()} for Alg. 6, we now focus on the concrete realizations of \texttt{findEdge()} for different TCO construction criteria in §3.3.3, §3.3.4, and §3.3.5. At each iteration of Gen-ODA, \texttt{findEdge()} (Line 6
in Alg. 6) finds an edge \( e \), whose addition would merge at least two different \( TC \)-components (for at least one topic), thus reducing the total number of \( TC \)-components by at least one. While naive search for the next “best” edge takes \( O(|V|^2) \) time, the implementation presented here improves the time complexity by employing the auxiliary indexing data structure \( EdgeContrib \). This data structure facilitates finding the ‘best’ edge at each iteration taking both edge contribution and node degree into account because the algorithm can traverse \( EdgeContrib.edgeArray[c][d] \) in the order of decreasing contribution \( c \) and increasing degree \( d \) and pick an edge from the first non-empty entry.

### 3.3.3 Finding Edge Algorithm for MinAvg-TCO

**Alg. 11 Find a MinAvg Edge**

```latex
findMinAvgEdge()

Output: an edge \( e \) to be added to \( E_{new} \)

1: for \( curDegree \) to \( maxDegree \) do
2: \hspace{1em} if \( EdgeContrib.entrySize(maxContrib, degree) > 0 \) then
3: \hspace{2em} \( e \leftarrow EdgeContrib.getOneEdge(maxContrib, degree) \)
4: \hspace{1em} return \( e \)
```

Gen-ODA together with Alg. 11, referred to as F-MinAvg-ODA (Fast MinAvg Overlay Design Algorithm), builds the same overlay as GM [35]. Alg. 11 implements the **GM-rule**: it always chooses the edge with the highest contribution toward topic-connectivity regardless of node degrees.

Lemma 6 shows that F-MinAvg-ODA achieves the same time efficiency as GM. The formal proof for Lemma 6 is omitted here, since it basically is a simplification of the time efficiency proof for F-MinMax-ODA, which we present in § 3.3.4.

**Lemma 6.** The cumulative running time of all invocations of Alg. 11 during the entire execution of Alg. 6 is \( O(|V|^2|T|) \).

### 3.3.4 Finding Edge Algorithm for MinMax-TCO

The implementation of MinMax-ODA proposed in [70] yields the time complexity of \( O(|V|^4|T|) \). Gen-ODA with the **MinMax-ODA-rule** implemented in Alg. 12 provides an efficient realiza-
tion of MinMax-ODA, with an improved running time of $O(|V|^2 |T|)$. We refer to this combined algorithm as Fast MinMax-ODA, F-MinMax-ODA in short.

Alg. 12 Find a MinMax Edge

```
findMinMaxEdge()
Output: an edge $e$ to be added to $E_{new}$
1: $e \leftarrow \text{NIL}, \text{contrib} \leftarrow \text{curContrib}$
2: while $e = \text{NIL} \land \text{contrib} > 0$ do
3: $\text{initDegree} \leftarrow 0$
4: if $\text{contrib} = \text{curContrib}$ then
5: $\text{initDegree} \leftarrow \text{curDegree}$
6: for degree $\leftarrow \text{initDegree}$ to $\text{maxDegree} - 1$ do
7: if EdgeContrib.entrySize($\text{contrib}, \text{degree}$) > 0 then
8: $e \leftarrow \text{EdgeContrib.getOneEdge}(\text{contrib}, \text{degree})$
9: break from for loop in Line 6
10: $\text{contrib} \leftarrow \text{contrib} - 1$
11: if $e = \text{NIL}$ then
12: $e \leftarrow \text{EdgeContrib.getOneEdge}(\text{maxContrib}, \text{maxDegree})$
13: return $e$
```

In order to explain Alg. 12, we observe that MinMax-ODA (and consequently F-MinMax-ODA) adds new edges in phases. At the start of each phase, MinMax-ODA selects a new edge that increases the maximum degree of the overlay by one. Then, the algorithm proceeds with adding edges without raising the maximum degree until the addition of any extra edge would cause a new increase, at which point the phase ends. The number of such phases is limited by the highest possible overlay degree, i.e., $O(|V|)$.

When invoked by Alg. 6 at each iteration, Alg. 12 scans the entries corresponding to non-maximum degree ($< \text{maxDegree}$) in $\text{edgeArray}$ of $\text{EdgeContrib}$ in the order of increasing degree and decreasing contribution. If a non-empty entry is found, an arbitrary edge from the entry edge list is selected. Otherwise, an edge from the entry with the maximum contribution and maximum degree is selected, which leads to the increase in the overlay degree and signifies a start of a new phase.
The crucial element for the efficiency of the implementation is that rather than scanning the entire `edgeArray` of `EdgeContrib` at each invocation, Alg. 12 continues the scan from the last selected entry. First, it does not affect the correctness of the scan: while after an edge addition, Alg. 10 reshuffles potential edges across `edgeArray`, it only moves the edges in the order of decreasing `contrib` (Lines 7-9) or increasing `degree` (Lines 22-23). Since Alg. 12 scans the entries in precisely the same order, it cannot miss a potential edge.

Secondly, continuing the scan from the last selected entry upon each Alg. 12 invocation within a single phase implies that the number of entries scanned at each phase is limited by the sum of two factors: the total number of entries in `edgeArray` of `EdgeContrib` (which is equal to $|V| \cdot |T|$) plus the number of entries scanned multiple times, i.e., the number of Alg. 12 invocations, which is equal to the number of edges selected at this phase (which is limited by $\frac{|V|}{2}$) [70]). Therefore, the number of entries scanned during the entire execution of Alg. 6 (i.e., at all $O(|V|)$ phases) is $O(|V| \cdot (|V||T| + \frac{|V|}{2})) = O(|V|^2|T|)$. This underlines the proof of Lemma 7; please see a formal and complete proof below.

**Lemma 7.** The cumulative running time for all invocations of Alg. 12 during the entire execution of Alg. 6 is $O(|V|^2|T|)$.

**Proof.** We denote by $TCO_{FMM}(V, T, Int, E_{FMM})$ the overlay network produced by F-MinMax-ODA, and by $D_{FMM}$ the maximum node degree in $TCO_{FMM}$. As shown in § 3.3.4, F-MinMax-ODA works in phases. We denote $M_i$ as the edge set added to $E_{FMM}$ by F-MinMax-ODA at phase $i$. At the start of the $i$-th phase, F-MinMax-ODA selects an edge that increases $maxDegree$ from $(i - 1)$ to $i$, then the algorithm proceeds by adding edges that do not raise $maxDegree$. Phase $i$ ends and phase $i + 1$ starts when the addition of any edge would increase $maxDegree$ from $i$ to $i + 1$.

At each invocation by Line 6 of Alg. 6, Alg. 12 finds an edge for F-MinMax-ODA. The algorithm first searches for a non-empty entry in `EdgeContrib.edgeArray`, and then picks an edge from that entry. We consider all invocations of Alg. 12 for each phase. Using amortized analysis we show that the cumulative runtime cost is bounded by $O(|V||T|)$.

More specifically, the cumulative runtime cost of all invocations of Alg. 12 at the $i$-th phase is dominated by two components:
(1) The cumulative cost of searching for a non-empty entry in `edgeArray` of `EdgeContrib`: this is determined by the number of entry probing operations in Line 7 of Alg. 12. In fact, although a single invocation of Alg. 12 can probe multiple entries, the number of probing operations for all edges in $M_i$ is bounded by the sum of two factors: 1) the total number of times the probed entry turns out to be empty so that Line 7 of Alg. 12 return `false`, and 2) the total number of times Line 7 of Alg. 12 return `true`. Factor 1) is bounded by the total number of entries in `EdgeContrib.edgeArray`, i.e., $|V| \cdot |T|$. This is because: at each iteration within the $i$-th phase of adding $M_i$, we always start probing from the last selected entry. The pointer for array entry probing only moves in one direction: in the order of increasing `degree` or decreasing `contrib`. The probing never moves backwards, so entries that have been checked as empty would not be visited again in this phase. Factor 2) is bounded by the number of edges selected in this phase, which is not higher than $\frac{|V|}{2}$ [70]. In summary, the cumulative running time of searching for a non-empty entry in the $i$-th phase is $O(|V||T|)$.

(2) The cumulative cost of picking an edge from the identified non-empty entry: this is determined by the number edge selection operations in Line 8 of Alg. 12. It takes $O(1)$ to obtain an edge by calling `EdgeContrib.getOneEdge()`. Totally there are at most $\frac{|V|}{2}$ edges to be added in each phase [70]. Thus, the cumulative running time of edge picking in the $i$-th phase is $O(|V|)$.

To sum up, there are at most $O(|V|)$ phases, so the cumulative cost of all invocations of Alg. 12 during one entire execution of Alg. 6 is $O(|V|) \cdot (O(|V||T|) + O(|V|)) = O(|V|^2|T|)$.

### 3.3.5 Finding Edge Algorithm for Low-TCO

A naive implementation of LowODA yields the time complexity of $O(|V|^4|T|)$ (see Lemma 3 in [69]). The Gen-ODA implementing the LowODA-rule is described in Alg. 13, which we refer to as F-Low-ODA (Fast LowODA), produces the same overlay with the improved running time of $O(|V|^2|T|)$. Combined, Lemma 6 and Lemma 7 allow us to establish Lemma 8.

**Lemma 8.** The cumulative running time for all invocations of Alg. 13 during the entire execution of Alg. 6 is $O(|V|^2|T|)$.
Algorithm 13: Find a Low Edge

findLowEdge(k)

Input: \( k \): parameter to balance edge selection rules
Output: an edge \( e \) to be added to \( E_{\text{new}} \)

1: \( e_1 \leftarrow \text{findMinAvgEdge}() \)
   \( \text{contrib}_1 \leftarrow \text{EdgeContrib}.\text{getContrib}(e_1) \)
2: \( e_2 \leftarrow \text{findMinMaxEdge}() \)
   \( \text{contrib}_2 \leftarrow \text{EdgeContrib}.\text{getContrib}(e_2) \)
3: if \( \text{contrib}_1 \geq \text{contrib}_2 \times k \) then
4: return \( e_1 \)
5: else
6: return \( e_2 \)

3.3.6 Running Time of Gen-ODA

To summarize all complexity analyses based on Lemmas 4, 5, 6, 7 and 8, the following lemma establishes the time efficiency of our implementation for F-MinAvg-ODA, F-MinMax-ODA and F-Low-ODA.

Lemma 9. The running time of Alg. 6 with function \textbf{findEdge()} instantiated as either Alg. 11, Alg. 12 or Alg. 13 is \( O(|V|^2|T|) \).

3.4 Evaluation

We implement all algorithms in Table 5.1 in Java and evaluate the running time of different algorithms, i.e., F-MinMax-ODA (vs. MinMax-ODA) and F-Low-ODA (vs. LowODA). We denote by \( T_v \) the topic set which node \( v \) subscribes to, and by \( |T_v| \) the subscription size of node \( v \). In these experiments, we use the following value ranges as input: \( |V| \in [100, 1000] \), \( |T| \in [100, 1000] \), and \( |T_v| \in [10, 100] \), where the subscription size is fixed for each node in the input. Each topic \( t_i \in T \) is associated with probability \( q_i \). \( \sum q_i = 1 \), so that each node subscribes to \( t_i \) with a probability \( q_i \). The value of \( q_i \) is distributed according to either a uniform, a Zipf (with \( \alpha=2.0 \)), or an exponential distribution. According to [36], these distributions are representative of actual workloads used in
industrial pub/sub systems today. Liu et al. [64] show that the Zipf distribution faithfully describes the feed popularity distribution in RSS feeds (a pub/sub-like application scenario). The exponential distribution is used by stock-market monitoring engines for the study of stock popularity in the New York Stock Exchange [90].

3.4.1 F-MinMax-ODA Algorithm for MinMax-TCO

We consider F-MinMax-ODA’s performance compared to MinMax-ODA with respect to different input parameters. Both F-MinMax-ODA and MinMax-ODA algorithms use the MinMax-ODA-rule for edge selection but are based on different implementations. Since the TCOs they compute are the same, we only show their running time ratios here.

![Figure 3.2: F-MinMax-ODA vs. MinMax-ODA](image)

Fig. 3.2(a) depicts the comparison between F-MinMax-ODA and MinMax-ODA as the number of nodes increases when $|T| = 100$. As the figure shows, F-MinMax-ODA runs considerably faster. Under uniform distribution, $T_{\text{FMM}}$ is on average 0.858% of $T_{\text{MM}}$; under Zipf distribution, $T_{\text{FMM}}$ is on average 1.17% of $T_{\text{MM}}$. Additionally, the F-MinMax-ODA algorithm gains more speedup with the increase in the number of nodes compared to MinMax-ODA: when $|V|=1000$, $T_{\text{FMM}} = 0.0158\% \cdot T_{\text{MM}}$ for the uniform distribution and $T_{\text{FMM}} = 0.0115\% \cdot T_{\text{MM}}$ for the Zipf distribution. The gap in the running time between our algorithms and existing ones is so significant that instead of showing the absolute values on the same scale we opt to present the ratio. For example, under the Zipf distribution, with 1000 nodes and 100 topics, F-MinMax-ODA completes in 3.823 seconds, while MinMax-ODA takes over 555 minutes. This shows that F-MinMax-ODA provides an adequate solution for the above target settings while MinMax-ODA does not.
Fig. 3.2(b) depicts how F-MinMax-ODA and MinMax-ODA perform when the number of topics varies. The running time ratio of F-MinMax-ODA to MinMax-ODA increases as the number of topics increases from 100 to 1000. In order to explain this effect, we observe that the running time of scanning the indexing structure in F-MinMax-ODA is proportional to the maximum edge contribution while the running time of MinMax-ODA is independent of edge contributions. Increasing the number of topics leads to reduced correlation, i.e., the probability of having two nodes interested in the same topic drops as the number of topics increases, and with reduced correlation the edge contribution tends to be lower. This reduction in correlation is more pronounced for the uniform distribution of interests compared to skewed ones, such as Zipf. Yet, the increase in the running time ratio is not very significant: on average, F-MinMax-ODA is less than 0.236% of MinMax-ODA under the uniform distribution, and less than 0.019% under the Zipf distribution.

Fig. 3.2(c) depicts the impacts of the subscription size on F-MinMax-ODA and MinMax-ODA. We set $|T| = 200$, and $|T_v|$ varies from 10 to 100. As shown in the figure, the ratio of $T_{FMM}$ to $T_{MM}$ decreases with the increase of $|T_v|$, and the ratio becomes relatively stable around 0.02% when $|T_v| > 50$.

### 3.4.2 F-Low-ODA Algorithm for Low-TCO

We now explore the impact of different input variables on the performance of the F-Low-ODA and LowODA algorithms. Both apply the LowODA-rule for edge selection, so for the evaluation, we only consider their implementation efficiency.

Fig. 3.3(a) depicts the comparison between these two algorithms as the number of nodes increases where $|T| = 100$. As the figure shows, F-Low-ODA runs significantly faster. Under the uniform distribution, $T_{FLOW}$ is on average 1.2% of $T_{LOW}$. Under the Zipf distribution, $T_{FLOW}$ is on average 0.6% of $T_{LOW}$. Additionally, F-Low-ODA gains more speedup with the increase in the number of nodes compared to LowODA: when $|V| = 1000$, $T_{FLOW} = 0.15 \cdot T_{LOW}$ for the uniform distribution and $T_{FMM} = 0.11 \cdot T_{MM}$ for the Zipf distribution.

Fig. 3.3(b) depicts the performance of F-Low-ODA and LowODA when we vary the number of topics. The ratio of $T_{FLOW}$ to $T_{LOW}$ increases as the number of topics increases from 100 to 1000, yet this effect is insignificant: on average, F-Low-ODA takes less than 0.172% of LowODA’s run-
Figure 3.3: F-Low-ODA vs. LowODA

ning time under the uniform distribution and less than 0.020% under the Zipf distribution. Further, F-Low-ODA has more speedup on the time efficiency for skewed distributions as the number of topics increases. The reason is that increasing the number of topics leads to less correlation, and under skewed distribution, the correlation among nodes drops relatively slower compared to that under the uniform distribution.

Fig. 3.3(c) depicts the effects of the subscription size on F-Low-ODA and LowODA. We set $|T|=200$ and $|T_v| \in [10, 100]$. As shown in the figure, the running time ratio $T_{\text{LOW}} / T_{\text{LOW}}$ decreases with the increase of $|T_v|$. The ratio becomes stable around 0.02% as $|T_v| > 50$.

### 3.5 Conclusions

In this chapter, we develop the Gen-ODA framework that covers existing greedy algorithms with different edge selection rules for different optimization criteria. By using the indexing data structures that we have devised, a number of known algorithms gain a significant running time speedup, i.e., the time complexity of MinMax-ODA and LowODA is improved from $O(|V|^4|T|)$ to $O(|V|^2|T|)$.

We have evaluated the algorithms through a comprehensive experimental analysis, which demonstrates their performance and scalability under various practical pub/sub workloads. Our proposed Gen-ODA is well suited to different TCO construction problems: its efficient implementation accelerates the time efficiency by a factor of more than 1000, and it gains more impact in the running time when the workloads scale up.
Chapter 4

Divide and Conquer Algorithms for Pub/Sub TCO Design

4.1 Introduction

We propose a number of novel techniques for the overlay construction problem. First, we formulate the TCO join problem and establish its complexity properties before proposing algorithms for this problem. Given two or more TCOs, we aim to construct a single TCO that includes all nodes of the sub-overlays and respects all interest relations. We develop and analyze approximation algorithms for this problem and empirically evaluate their performance. Based on our approach, we develop divide-and-conquer algorithms, which basically divide the entire network into smaller partitions, conquer each partition separately by building sub-overlays locally, and then combine the separate sub-overlays into a single, global TCO. We show and analyze several key advantages of this divide-and-conquer approach for the original TCO construction problem. We present a numerical method for selecting the number of partitions, which is a key parameter for the performance of divide-and-conquer algorithms. We also compare a number different partitioning techniques in the context of the divide-and-conquer algorithms, including random partitioning, node clustering, and a novel bulk-lightweight partitioning method.
Both theoretical analysis and experimental evaluations show that it takes divide-and-conquer algorithms significantly less time to construct an overlay from scratch compared to previous algorithms, at the expense of an insignificant increase in the node degree of the resulting overlay. In our experiments based on typical pub/sub workloads, we show that the runtime cost of the proposed algorithms is at most $1.67\%$ of that of the existing state-of-the-art approaches while the average node degree is only 2.12 larger. In addition, divide-and-conquer approach only requires partial information about nodes, topics, and interests.

### 4.2 Topic-connected Overlay Join Problem

In this section we introduce the TCO join problem and discuss similarities and differences with MinAvg-TCO.

The joining of existing TCOs over the same set of topics but disjoint sets of nodes can be achieved by the known approaches for MinAvg-TCO. That is given sets of nodes and interest functions, compute the union of the sets of nodes and construct a common interest matrix as the union of individual interest functions (the union is well defined as the functions’ domains are disjoint), and apply the algorithms for MinAvg-TCO to rebuild the overlay from scratch. However, in practice, it can often be better to preserve existing edges and only incrementally add edges as needed to achieve topic-connectivity in the overlay combined from existing overlays. This is because establishing a new edge is a relatively costly operation and incremental computation of additional edges is more efficient than the re-computation of the entire overlay. Also, to re-build a new overlay from scratch, all existing connections in the existing overlays would have to be torn down, the new routing tables would have to be distributed, and the new connections would have to be established; all adding to the cost of reacting to a network partition or to incrementally evolving an overlay. We give the formal definitions of the TCO join problem: MinAvg-TCO-Join for the optimization version and Avg-TCO-Join for the decision version.

**Problem 4.** MinAvg-TCO-Join($V, T, \text{Int}, p$) Given $p$ node-disjoint TCOs, which we denote as $TCO_d(V_d, T, \text{Int}_d, E_d), \ d=1,...,p$, construct a TCO($V, T, \text{Int}, E$) with the least possible total number of edges where $V = \bigcup_{d=1}^{p} V_d$, $\text{Int} = \bigcup_{d=1}^{p} \text{Int}_d$, and $\bigcup_{d=1}^{p} E_d \subseteq E$. 
Problem 5. **Avg-TCO-Join**\((V,T,\text{Int},p,m)\) Given an integer \(k\) and \(p\) node-disjoint TCOs, i.e., \(\text{TCO}_d(V_d,T,\text{Int}_d,E_d),\ d=1,\ldots,p\), determine if there exists a TCO \((V,T,\text{Int},E)\) such that \(|E|=m\) where \(V=\bigcup_{d=1}^{p}V_d,\ \text{Int}=\bigcup_{d=1}^{p}\text{Int}_d,\) and \(\bigcup_{d=1}^{p}E_d \subseteq E\).

MinAvg-TCO is a special case of MinAvg-TCO-Join when considering that each joining overlay contains only one node. Hence, all impossibility results about MinAvg-TCO (NP-completeness for the decision problem and impossibility of linear approximation unless P=NP) apply to Avg-TCO-Join and MinAvg-TCO-Join as well. For the practical concerns raised above, the previously known greedy algorithms are not directly applicable to solve MinAvg-TCO-Join.

Also, based on the considerations from above, given a number of TCOs, it is often more beneficial to solve MinAvg-TCO-Join incrementally rather than solving MinAvg-TCO from scratch. There is, however, a drawback, as the total number of edges resulting from incremental addition of edges can become much larger than the total number of edges resulting from overlay construction from scratch, as illustrated in the following example.

Assume a \(n\)-node set \(V = \{v_1,v_2,\ldots,v_n\}\) and a \(n^2\)-topic set \(T = \{t_{ij}|i,j=1,2,\ldots,n\}\). \(T\) is divided into \(n\) groups \(T_i = \{t_{ij}|j=1,2,\ldots,n\} = \{t_{i1},t_{i2},\ldots,t_{in}\}, i=1,2,\ldots,n.\) Topics in which node \(v_i\) is interested, denoted by \(T_{v_i}\), comprise two parts: (1) all topics in \(T_i\) and (2) the \(i\)-th topic \(t_{ji}\) from another topic group \(T_j(\neq T_i)\). Therefore,

\[
T_{v_i} = \{t|t = t_{ij} \lor t = t_{ji}, j = 1,2,\ldots,n\}, i = 1,2,\ldots,n
\]

For an arbitrary \(i\), the part of the interest matrix restricted to \(V \times T_i\) looks as follows.

\[
\begin{pmatrix}
  t_{i1} & t_{i2} & \ldots & t_{ii} & \ldots & t_{in} \\
  v_1 & 1 & 0 & \ldots & 0 & \ldots & 0 \\
  v_2 & 0 & 1 & \ldots & 0 & \ldots & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
  v_i & 1 & 1 & \ldots & 1 & 1 & 1 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
  v_n & 0 & 0 & \ldots & 0 & \ldots & 1
\end{pmatrix}
\]

As an input instance to the MinAvg-TCO problem, a \(n \times n\) complete overlay \(\text{TCO}_0\) is generated. Each \(v_i\) has to link to all other nodes to achieve topic-connectivity for topics in \(T_j\).
Assume a new node $v_{\text{all}}$, which is interested in all topics in the entire topic set $T$, requests to join. If the new topic-connected overlay $TCO_1$ is constructed incrementally by adding edges to $TCO_0$, then the number of edges in $TCO_1$ is still $\Theta(n^2)$, as illustrated in Fig. 4.1(b). However, if we delete all existing edges and re-construct the TCO from scratch, it is sufficient to simply connect $v_{\text{all}}$ to all others $\{v_1, v_2, \ldots, v_n\}$, which results in a $TCO_2$ with $\Theta(n)$ edges (cf. Fig. 4.1(c)).

This example shows that the number of edges resulting from incremental addition can be $\Theta(n)$ times larger than the number of edges when building an overlay from scratch. However, our experimental findings in §6.6 show that for typical pub/sub workloads, this situation virtually never occurs in practice.

In the next section, we introduce several algorithms for the MinAvg-TCO-Join problem. Our algorithms are based on determining cross-TCO edges that are added incrementally.

### 4.3 Algorithms for MinAvg-TCO-Join Problem

Given the GM algorithm for MinAvg-TCO, we first devise a Naive Merge (NM) algorithm for MinAvg-TCO-Join, as specified in Alg. 14. This simple algorithm is also based on a greedy heuristic giving rise to similar properties as for GM. We include the design and analysis of this algorithm below to illustrate its weaknesses that motivate the need for a more sophisticated algorithm, presented thereafter.
Alg. 14 Naive Merge for MinAvg-TCO-Join

**NaiveMergeAlgorithm**(\(L_{TCO}\))

**Input:** A list \(L_{TCO}\) of \(p\) node-disjoint topic-connected overlays: \(TCO_d(V_d, T, Int_d, E_d)\), \(d=1,\ldots,p\)

**Output:** A topic-connected overlay \(TCO(V, T, Int, E_{NM})\) where

\[
V = \bigcup_{d=1}^{p} V_d, \quad Int = \bigcup_{d=1}^{p} Int_d, \quad \bigcup_{d=1}^{p} E_d \subseteq E_{NM}
\]

1: \(V \leftarrow \bigcup_{d=1}^{p} V_d\)
2: \(Int \leftarrow \bigcup_{d=1}^{p} Int_d\)
3: \(E_{inNM} \leftarrow \bigcup_{d=1}^{p} E_d\)
4: \(nodesInterests \leftarrow \emptyset\)
5: for \(d = 1\) to \(p\) do
6:   add \((V_d, Int_d)\) to \(nodesInterests\)
7: \(E_{outNM} \leftarrow \text{greedyConnect}(V, T, Int, nodesInterests)\)
8: \(E_{NM} \leftarrow E_{inNM} \cup E_{outNM}\)
9: return \(TCO(V, T, Int, E_{NM})\)

---

Alg. 15 Compute cross-TCO links

**greedyConnect**(\(V, T, Int, nodesInterests\))

**Input:** \(V, T, Int, nodesInterests\)

// \(nodesInterests\): A list of \(p (V_d, Int_d)\) pairs

**Output:** A set \(E_{out}\) of cross-TCO edges that along with inner overlay edges form a TCO

1: for \(d = 1\) to \(p\) do
2:   for all \(t \in T\) do
3:     for all \(v \in V_d\) do
4:       \(TCC-Nodes[v][t] \leftarrow \emptyset\)
5:     \(intNodes \leftarrow \{v \in V_d | Int_d(v, t)\}\)
6:     for all \(v \in intNodes\) do
7:       \(TCC-Nodes[v][t] \leftarrow intNodes\)
8:   for all \(e = (v, w)\) such that \(v \in V_i, w \in V_j, i \neq j\) do
9:     \(weight \leftarrow |\{t \in T : Int_i(v, t) \land Int_j(w, t)\}|\)
10:    if \(weight > 0\) then
11:      add \(e\) to \(LinkContrib[weight]\)
12: return \(\text{constructOverlayEdges}(V, T, Int, TCC-Nodes, LinkContrib)\)
The intuition behind our algorithm is that we need to determine a set of cross-TCO links, that in conjunction with the already existing links internal to the TCOs, produce a combined TCO. Below we refer to inner links as $E_{inNM}$ and outer links as $E_{outNM}$. $E_{inNM}$ is easily obtained by the union of all edges within TCOs. The algorithm starts with an empty $E_{outNM}$ and iteratively adds edges one by one until topic-connectivity is attained. At each iteration, the algorithm selects a cross-TCO edge whose addition to the overlay would maximally reduce the total number of TC-components.

Alg. 14 follows the design of the $GM$ algorithm given in §2.1.1. The correctness, approximation ratio, and running time properties below can be established by adapting the respective proofs for the $GM$ algorithm.

**Lemma 10.** (Correctness) Alg. 14 is correct: it yields a topic-connected overlay for the input instance $(V, T, Int)$.

Line 7 in Alg. 14 computes the set $E_{outNM}$ of all cross-TCO edges. Let $E'_{out}$ be an optimal edge set of cross-TCO edges and $T_d$ be the set of topics covered by overlay nodes in $V_d$, i.e., $T_d = |\{t \in T | \exists v \in V_d \text{ s.t. } Int_d(v, t)\}|$. Note, $T_d$ is also the number of TC-components for $TCO_d$ and thus $\sum_{d=1}^p T_d$ is the total number of TC-components for the input of all TCOs. With proof techniques similar to Lemma 6.5 from [35], Lemma 11 is established.

**Lemma 11.** (Approximation Ratio) Alg. 14 has a log approximation ratio $\frac{|E_{outNM}|}{|E'_{out}|} = O(\log(\sum_{d=1}^p T_d))$.

**Lemma 12.** (Running Time) The running time of Alg. 14 is $T_{NM} = O(|T| \cdot \sum_{i=1}^p \sum_{j \geq 1} |V_i||V_j|)$.

Alg. 14 also retains the undesirable properties of $GM$. The running time as given by Lemma 12 is not insignificant. Moreover, the algorithm cannot be easily decentralized, and the overlay computation requires the complete knowledge of $(V, T, Int)$. These shortcomings motivate the development of a new algorithm.

Our new algorithm to MinAvg-TCO-Join is based on the notion of star sets defined and illustrated next.

**Definition 1.** (Star set) Given an instance $I(V, T, Int)$, a star set is a subset $S \subseteq V$ such that

$$\cup_{s \in S} \{t \in T | Int(r, t)\} = \cup_{v \in V} \{t \in T | Int(v, t)\}.$$
For any \( s \in S \), we say node \( s \) is a star node (or a star) with regard to star set \( S \).

As illustrated in Fig. 4.2, a star set is a subset of overlay nodes that represents the interests of all the nodes in the overlay. The complete node set \( V \) is always a star set, but there probably exist many other star sets with much fewer stars. Star nodes can function as bridges for the purpose of determining cross-TCO connections. It is easy to see that it is possible to attain full topic-connectivity only by using cross-TCO links among star sets of different sub-TCOs. Suppose we have a number of TCOs such that each TCO includes nodes interested in a topic \( t \in T \). Then, each of the star sets will include a node interested in \( t \). By connecting nodes from different star sets we can achieve topic-connectivity for \( t \).

![Diagram](https://example.com/diagram.png)

**Figure 4.2:** (a) A TCO. (b) \( \{v_3, v_5\} \) is a star set which covers all topics \( \{a, b, c, d\} \). (c) \( \{v_2, v_3, v_4\} \) is not a star set; it only covers \( \{a, b, d\} \).

Since the minimum star sets tend to be substantially smaller than the entire node set, considering only star nodes as candidates for cross-TCO links gives rise to a number of advantages. One, the running time of the overlay construction algorithms considered in this chapter is roughly proportional to the square of the number of nodes, therefore our algorithm that only considers star nodes runs much faster. Two, we can compute star sets of different TCOs in parallel in a fully decentralized fashion. Three, calculation of cross-TCO links no longer requires the complete knowledge of \( V \) and \( Int \), and we just need a partial view of star nodes and their interests.

We still need to consider, how to efficiently determine a minimum star set given \((V, T, Int)\). The problem of computing a minimum star set turns out to be precisely equivalent (through a linear reduction) to the classic set cover problem, which is NP-complete but has a logarithmic approximation [39]. Note that the size of the star set only affects the performance of our algorithm rather than its correctness. Alg. 17 provides a standard greedy implementation that attains a provable logarith-
mic approximation. The algorithm starts with an empty star set and continues adding nodes to the star set one by one until all topics of interest are covered. At each iteration, the algorithm selects a node that is interested in the largest number of uncovered topics.

Alg. 16 presents our Star Merge (SM) algorithm. The algorithm operates in two phases. First, it determines a star set for each sub-TCO. Note that the star set for $T_{CO_d}$ does not need to cover all of $T_d$. It suffices to cover $T_{out,d} = T_d \cap (\bigcup_{i \neq d} T_i)$. Second, the algorithm connects all the nodes in the star sets into a TCO in a greedy manner by always selecting an edge with maximum contribution.

Below, we establish correctness, approximation ratio and running time properties for SM.

**Lemma 13.** (Correctness) Alg. 16 is correct: it yields a TCO for the input instance $(V, T, Int)$.

Let $E_{out}'$ and $T_d$ follow the same definitions as for Lemma 11. Let $E_{outSM}$ be the set of edges computed by Line 10 of Alg. 16. This edge set connects star nodes in $R_d \ (d = 1, \ldots, p)$ across all TCOs. Let $E_{outSM}'$ be the minimal edge set to connect $R_d \ (d = 1, \ldots, p)$. Then the following properties for Alg. 16 hold.

**Lemma 14.** (Approximation Ratio) The output of Alg. 16 has a log approximation ratio as compared to the optimal solution of just connecting stars: $\frac{|E_{outSM}|}{|E_{outSM}'|} = O(\log(\sum_{d=1}^{p} T_d))$.

Let $S_d^* \subseteq V_d$ be the star set of minimum size for $(V_d, T_{out,d}, Int_d)$. Then, $\forall d = 1, \ldots, p$ : $|S_d| = O(\log(|T_{out,d}|)|S_d^*|)$, and the following result holds:

**Lemma 15.** (Approximation Ratio) The output of Alg. 16 has the following approximation ratio as compared to the optimal solution: $\frac{|E_{outSM}|}{|E_{out}'|} = O(\log(\sum_{d=1}^{p} T_d) \cdot (\log |T|)^2 \cdot \sum_{d=1}^{p} |S_d^*|)$.

**Lemma 16.** (Running Time) The running time of Alg. 16 is $T_{SM} = O(|T| \cdot (\sum_{d=1}^{p} |V_d| \log |V_d| + (\log |T|)^2 \sum_{i=1}^{p} \sum_{j>i} |S_d^*||S_j^*|))$.

With Lemma 12 and Lemma 16, we can see that our SM algorithm improves the running time significantly because for most cases $|S_d^*| \ll |V_d|$ holds. We also show this experimentally.

The same idea to reduce the number of nodes by choosing from star nodes can be applied to solve MinAvg-TCO, which we discuss in the next section.
Alg. 16 Star Merge algorithm for MinAvg-TCO-Join

StarMergeAlgorithm($L_{TCO}$)

Input: A list $L_{TCO}$ of $p$ node-disjoint topic-connected overlays: $TCO_d(V_d, T, Int_d, E_d)$, $d=1, \ldots, p$

Output: A topic-connected overlay $TCO(V, T, Int, E_{SM})$ where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, $\bigcup_{d=1}^{p} E_d \subseteq E_{SM}$

1: $V \leftarrow \bigcup_{d=1}^{p} V_d$
2: $Int \leftarrow \bigcup_{d=1}^{p} Int_d$
3: $T_d \leftarrow \{ t \in T | \exists v \in V_d \text{ s.t. } Int(v, t) \}$, $d=1, \ldots, p$
4: $E_{inSM} \leftarrow \bigcup_{d=1}^{p} E_d$
5: $\text{nodesInterests} \leftarrow \emptyset$
6: for $d = 1$ to $p$ do
7: $T_{out,d} \leftarrow T_d \cap \bigcup_{d \neq d} T_i$
8: $S_d \leftarrow \text{getStarSetFromNodes}(V_d, T_{out,d}, Int_d)$
9: add $(S_d, Int_d|S_d)$ to $\text{nodesInterests}$
10: $E_{outSM} \leftarrow \text{greedyConnect}(V, T, Int, \text{nodesInterests})$
11: $E_{SM} \leftarrow E_{inSM} \cup E_{outSM}$
12: return $TCO(V, T, Int, E_{SM})$

Alg. 17 Determine star set for a TCO

getStarSetFromNodes($V, T, Int$)

Input: $V, T, Int$

Output: $S$: A star set wrt. $(V, T, Int)$

1: Start with $S \leftarrow \emptyset$ and $\text{Rest} \leftarrow \{ t \in T | \exists v \in V \text{ s.t. } Int(v, t) \}$
2: while $\text{Rest} \neq \emptyset$ do
3: $s \leftarrow \arg\max_{v \in V - S} | \{ t | t \in \text{Rest} \land Int(v, t) \} |$
4: $S \leftarrow S \cup \{ s \}$
5: $\text{Rest} \leftarrow \text{Rest} - \{ t | \text{Int}(s, t) \}$
6: Return $S$
4.4 Divide-and-Conquer for MinAvg-TCO

In this section we show how our SM algorithm for MinAvg-TCO-Join can be extended to solve the MinAvg-TCO problem in a divide-and-conquer manner. We also show analytically that the resulting algorithm leads to a significantly improved running time cost as compared to the GM algorithm. In §6.6, we quantify these improvements empirically. We first develop the divide-and-conquer algorithm in §4.4.1 and then present its main analytical properties in §4.4.2.

4.4.1 Divide-and-Conquer Algorithm

According to Chockler et al. [35], the number of nodes is a dominant factor for the running time of the GM algorithm (i.e., \(O(|V|^2|T|)\)) and a serious drawback to the algorithm’s performance and scalability. The running time could be improved by reducing the size of the node set when executing the GM algorithm. This suggests a divide-and-conquer strategy for the MinAvg-TCO problem: (1) divide the input MinAvg-TCO instance into several sub-instances with smaller node sets, (2) conquer the sub-MinAvg-TCO instances independently and build sub-TCOs and then (3) combine these sub-TCOs into one TCO as an output for the original instance.

\[
\text{Alg. 18} \quad \text{Divide-and-Conquer algorithm for MinAvg-TCO}
\]

\[
\text{DivideAndConquerForTCO}(V, T, \text{Int})
\]

\[
\text{Input: } V, T, \text{Int}
\]

\[
\text{Output: A topic-connected overlay } TCO(V, T, \text{Int}, E_{DC})
\]

1: \(E_{DC} \leftarrow \emptyset\)

2: \(p \leftarrow \text{chooseP}(V, T, \text{Int})\)

3: \(L_{TCO} \leftarrow \emptyset\)

4: Randomly divide \(V\) into \(p\) partitions \(V_d, d=1, 2, ..., p\)

5: for \(d = 1\) to \(p\) do

6: \(\text{Int}_d \leftarrow \text{Int}|_{V_d}\)

7: \(TCO_d(V_d, T, \text{Int}_d, E_d) \leftarrow \text{GreedyMerge}(V_d, T, \text{Int}_d)\)

8: add \(TCO_d\) to \(L_{TCO}\)

9: \(TCO(V, T, \text{Int}, E_{DC}) \leftarrow \text{StarMergeAlgorithm}(L_{TCO})\)

10: return \(TCO(V, T, \text{Int}, E_{DC})\)
The Divide-and-Conquer (DC) algorithm that follows this design is presented in Alg. 18. The existing GM algorithm can be employed to conquer the sub-MinAvg-TCO instances by determining inner edges used for the construction of the sub-overlays. The SM algorithm can tackle the combine phase by adding cross-TCO edges (i.e., outer edges) in a greedy manner. To divide nodes the algorithm uses a random partitioning. The number of partitions \( p \) is obtained by function \( \text{chooseP}() \) in Line 2 of Alg. 18, and we will give a detailed algorithm and analysis in §4.5. Each partition contains \( k = |V_d| = \frac{|V|}{p} \) nodes, where \( d = 1, \ldots, p \).

Basically there are two methods to divide the nodes: (1) node clustering and (2) random partitioning. Node clustering is partitioning the original node set into groups so that nodes with similar interests are placed in the same group while nodes with diverging interests belong to different groups. Random partitioning assigns each node in the given node set to one of the partitions based on a uniformly random distribution.

Node clustering seems appealing because well-clustered nodes with strongly correlated interests would result in GM producing lower average node degrees. The problem with this approach is its relative inefficiency. Clustering algorithms tend to exhibit high runtime cost. Additionally, they require the computation of a “distance” metric among nodes. In our case this translates to calculating pairwise correlations among node interests with significant runtime cost implications. It is challenging to fit node clustering into the DC algorithm so that the latter is still superior to the GM algorithm in terms of runtime cost. Furthermore, it is difficult to devise an effective decentralized algorithm for node clustering that would not require complete knowledge of \( V \) and \( \text{Int} \). Besides, node clustering by interests may yield clusters that vary in size depending on the clustering algorithm used.

We choose random partitioning for our DC algorithm because it is extremely fast, more robust than node clustering, can be done in a decentralized manner. Also, it is not difficult to obtain equal-sized partitions by random partitioning, which is optimal with respect to both average node degree and running time. Furthermore, the construction of inner edges for each sub-overlay only requires knowledge of node interests within the sub-overlay. Hence, random partitioning can be oblivious to the composition of nodes and their interests. The shortcoming of using random partitioning for the DC algorithm is the potentially higher average node degree because random partitioning may place nodes with diverging interests into the same partition thereby reducing the amount of correlation that
is present in the original node set. In §6.6, we validate the effect of the increase in the average node degree empirically and show that this effect is insignificant. Also we compare DC with random partitioning against clustering in various settings.

In order to diminish the risk of sub-optimality due to random partitioning and guarantee the quality of the output TCO, in §4.6 we design a more refined partitioning method in which the nodes that subscribe to large number of topics are replicated in each partition.

### 4.4.2 DC Algorithm Analysis

We prove correctness, approximation ratio and running time properties for the DC algorithm.

**Lemma 17.** (Correctness) Alg. 18 is correct: it yields a TCO for the input instance \((V, T, \text{Int})\).

Given an MinAvg-TCO instance of \(I(V, T, \text{Int})\), suppose \(TCO_{OPT}(V, T, \text{Int}, E_{OPT})\) is an optimal solution for it. As defined before, \(p\) denotes the number of partitions provided for Alg. 18. There are two types of edges that form the TCO produced by Alg. 18: (1) \(E_{\text{inDC}}\), the inner edges which are computed for each sub-overlay using the \(GM\) algorithm in Lines 5-8; (2) \(E_{\text{outDC}}\), the outer edges to connect star nodes across different sub-TCOs using the \(SM\) algorithm in Line 9. The following lemma is based on the assumption that the optimal overlay for \((V', T, \text{Int}|_{V'})\) where \(V' \subset V\) has fewer edges than the optimal overlay \(TCO'\) for \((V, T, \text{Int})\).

**Lemma 18.** (Approximation Ratio) Alg. 18 produces an overlay with a total number of edges that has an approximation ratio of \(O(p \cdot \log(|V||T|))\) as compared to the optimal solution for MinAvg-TCO: \(\frac{|E_{DC}|}{|E_{OPT}|} = O(p \cdot \log(|V||T|))\)

Next, we look at the running time of the DC algorithm. \(T_{\text{inDC}}\) denotes the running time to build \(E_{\text{inDC}}\) in the loop in Lines 5-8. \(T_{\text{outDC}}\) denotes the running time to build \(E_{\text{outDC}}\) in Line 9 and let \(T_{DC}\) be the total running time cost of Alg. 18.

**Lemma 19.** (Running Time) The running time of Alg. 18 is:

\[
T_{DC} = O(|T| \cdot \left(\frac{1}{p} |V|^2 + (\log |T|)^2 \sum_{i=1}^{p} \sum_{j>i} |S_i^*||S_j^*|\right)),
\]

\[
T_{\text{inDC}} = O(|T| \cdot \sum_{d=1}^{p} |V_d|^2),
\]

\[
T_{\text{outDC}} = O(|T| \cdot (\log |T|)^2 \cdot \sum_{i=1}^{p} \sum_{j>i} |S_i^*||S_j^*|).
\]
Let us denote the average size of a star set for a sub-TCO as $\tilde{k}$.

$$\tilde{k} = |S_d| = \frac{\sum_{d=1}^{p} |S_d|}{p}$$

Lemma 20 is given to simplify the analysis of the running time cost of the DC algorithm presented in §4.5. In §6.6, we evaluate $\tilde{k}$’s statistical properties.

**Lemma 20.** (Running Time) The running time of Alg. 18 is

$$T_{DC} = O(|T| \cdot (\frac{1}{p} |V|^2 + p^2 \tilde{k}^2))$$

### 4.4.3 Decentralizing the DC algorithm

Note that the DC algorithm as presented above is fully centralized. It is possible to decentralize it in the following way: (1) nodes autonomously organize themselves into random partitions, (2) different partitions construct inner edges in parallel, i.e., the nodes within each partition exchange their interests and run the GM algorithm, (3) different partitions compute star sets in parallel, (4) members of all star sets exchange their interests and compute outer edges. This decentralized implementation has several important advantages: the parallel calculation reduces the total running time and distributes the computational load. Furthermore, decentralization eliminates the need for a central entity that must have full knowledge of all nodes and their interests, as we further elaborate on in §4.5. Note, the original GM algorithm does not lend itself to such decentralization.

### 4.5 Selecting the Size of Partitions for DC

Our DC algorithm is parameterized with the number of partitions $p$. This parameter impacts all important characteristics of the algorithm’s performance. When the number of partitions is 1, the performance is dominated by the invocation of the GM algorithm at the conquer phase. In this case, average node degree, running time, and other performance characteristics are identical to the performance of the GM algorithm when applied to MinAvg-TCO. When the number of partitions is close to $|V|$ at the other end of the spectrum, the performance is dominated by the invocation of the
SM algorithm at the combine phase. As we observed in §4.3, the performance of the SM algorithm in this case is once again identical to that of the GM algorithm when applied to MinAvg-TCO. Thus, it is the intermediate values of $p$ that represent the interesting balance between the conquer and combine phases.

With respect to the average node degree, it is desirable to maintain $p$ reasonably small because $p$ bounds the value of the approximation ratio, as shown in Lemma 18. Further evaluation will be presented in §6.6.

With respect to the running time, the analysis in §4.4.1 establishes that the contribution of both partitioning the nodes and computing the star sets is of a small order of magnitude in comparison to the time needed for computing inner edges at the conquer phase and outer edges at the combine phase. The time needed for computing inner edges is bounded by $\frac{1}{p} |V|^2 |T|$ (under the assumption that inner edges for different sub-overlays are computed sequentially rather than in parallel) while the time required for computing outer edges is proportional to $(\tilde{k}/k)^2 |V|^2 |T|$. Approximately, the combined sum of these two values is minimum when $1/p + (\tilde{k}/k)^2$ is minimum, which occurs when

$$\frac{d}{dp} \left( \frac{1}{p} + (\tilde{k}/k)^2 \right) = 0.$$

If, on the other hand, inner edges for different sub-overlays are computed in parallel in a decentralized fashion as discussed in §5.2.4, then this computation requires time proportional to $1/p^2 |V|^2 |T|$. Then, the total running time is minimum if $1/p^2 + (\tilde{k}/k)^2$ is minimum, or

$$\frac{d}{dp} \left( \frac{1}{p^2} + (\tilde{k}/k)^2 \right) = 0.$$

A practical application of this running time analysis is complicated by the fact that it is difficult to assess $\tilde{k}$ analytically. However, it suggests an adaptive way for selecting $p$. Since partitioning the nodes and computing the star sets is relatively cheap, we can try partitioning for different $p$ values. Each time, we can only compute the star sets and thus obtain $\tilde{k}$ without running the expensive calculation of inner and outer edges. Then, we can use fast numerical methods to approximately determine the minimum of $1/p + (\tilde{k}/k)^2$ or $1/p^2 + (\tilde{k}/k)^2$.

Another important performance characteristic of the algorithm is the potential neighbor set. We define the potential neighbor set for a node $v$ as the set of nodes that are considered candidates for becoming a neighbor of $v$ when executing DC. We denote by $pn-size(v)$ the size of $v$’s potential neighbor set, and we also call it potential neighbor size. Note that $pn-size(v)$ is the maximal fan-out that an overlay design algorithm may yield for a single node. According to the DC algorithm, the
potential neighbor set for \( v \) consists of two subsets: (1) nodes in the same partition as \( s \): \( \{ u \mid u \neq v \in V_d \wedge v \in V_d \} \); (2) all other star set nodes: \( \{ s \mid s \in S_i(\neq S_d) \wedge v \in S_d \} \) (if \( v \) belongs to the star set \( S_d \)). Consequently, there are two classes of nodes with respect to the potential neighbor set:

\[
pn-size(v) = \begin{cases} 
(k - 1), & \text{if } v \text{ is not a star node} \\
(k - 1) + (p - 1)\hat{k}, & \text{if } v \text{ is a star node}
\end{cases}
\]

The significance of this characteristic is manyfold. First of all, the cardinality of the potential neighbor set directly impacts the running time for computing inner edges and outer edges of DC because it implies how many nodes are processed when choosing a link to add (see empirical results in §6.6). Secondly, the maximum node degree has an importance of its own. While the GM algorithm strives to minimize the average node degree, it may be severely suboptimal with respect to the degree of individual nodes by producing star-like overlays as it has been observed in [70]. Therefore, minimizing the potential neighbor size has a desirable effect from this point of view. Thirdly, the potential neighbor size has an additional important meaning for the decentralized implementation of the DC algorithm discussed in § 5.2.4. For a node \( v \), \( pn-size(v) \) is precisely equal to the number of nodes whose interests \( v \) may need to learn about in the course of an execution. This is important because gathering nodes’ interests in a scalable and robust decentralized manner is a problem in its own right.

We extend the definitions of potential neighbor size to apply to the entire node set:

\[
pn-size(V) = \max_{v \in V} pn-size(v) = (k - 1) + (p - 1)\hat{k}
\]

If we consider \( pn-size(V) \) as a function of \( p \), it becomes minimum when \( k + (p - 1)\hat{k} \) is minimum. Similar to the analysis of the running time, further development of this derivation requires to assess \( \hat{k} \) analytically. It is also possible, however, to select \( p \) adaptively so as to minimize the potential neighbor size by partitioning the nodes and computing the star sets and using efficient numerical methods to approximately determine the minimum.
**Alg. 19** Choose The Number of Partitions by *pn-size*

`choosePbyPNSizeEstimation(V, T, Int)`

**Input:** `V`, `T`, `Int`

**Output:** `p`: the number of partitions, `2 ≤ p ≤ |V|`

1. `q ← 2`
2. `pn-size_new ← estimatePNSize(q)`
3. **repeat**
   4. `pn-size_old ← pn-size_new`
   5. `q ← q ∙ 2`
   6. `pn-size_new ← estimatePNSize(q)`
4. **until** `pn-size_new ≥ pn-size_old`
   5. `low ← q/2`, `high ← q`
   6. `pn-size_min ← pn-size_old`
   7. `p ← 0`, `pn-size_est ← 0`
8. **while** `low < high` **do**
   9. `p ← (low + high)/2`
10. `pn-size_est ← estimatePNSize(p)`
11. **if** `pn-size_est < pn-size_min` **then**
   12. `low ← p`
13. `pn-size_min ← pn-size_est`
   14. **else**
     15. `high ← p`
16. **return** `p`
17. **end while**
18. **return** `p` estimatePNSize()

**Input:** `p`: the number of partitions

**Output:** `pn-size_est`: estimate of *pn-size* as a function of `p`

1. `pn-size_est ← |V|/p`
2. Randomly divide `V` into `p` partitions `V_d`, `d=1, 2, ..., p`
3. **for** `d = 1` **to** `p` **do**
4. `S_d ← getStarSetFromNodes(V_d, T_d, Int_d)`
5. `pn-size_est ← pn-size_est + |S_d|`
6. **return** `pn-size_est`

---

To this end, we consider *pn-size* as a function of `p`, and search for the value of `p` that approximately yields the minimum of *pn-size*. As shown in `estimatePNSize()` in Alg. 19, given `p`, we can estimate *pn-size* as a function of `p` by performing random partitioning of `V`, computing the star sets for each partition, and then obtaining *pn-size* for this particular collection of partitions. This method produces reasonable results because the size of star sets (and therefore *pn-size*) is quite stable across different trials of random partitioning (see §4.7.2). Besides, `estimatePNSize()` can be implemented efficiently, and the time complexity is negligible as long as this function is only invoked a small number of times. Using brute force it is possible to estimate *pn-size* for all values of `p` in the range of `2 ≤ p ≤ |V|` and compute the minimum, which requires Θ(|V|) invocation-
s. However, the shape of dependency of \( pn\text{-size} \) on \( p \) allows us to find \( p \) more efficiently. §4.7.2 shows that \( pn\text{-size} \) as a function of \( p \) roughly exhibits a convex profile: the graph of the function lies below the line segment joining two endpoints of the graph, \( p = 2 \) and \( p = |V| \). As the number of partitions increases starting from \( p = 2 \), \( pn\text{-size} \) would first decrease and then increase, and the point of transition is close to the leftmost endpoint \( p = 2 \). With this in mind, we use one-sided binary search to find the value of \( p \) that approximately yields the minimum of \( pn\text{-size} \), as specified in Alg. 19. In Lines 3-7 we estimate \( pn\text{-size} \) as a function of \( p \) (by \( \text{estimatePNSize}(q) \)) repeatedly at larger intervals (\( q = 2, q = 4, q = 8, q = 16, \ldots \)) until we find a \( q \) such that the value of \( \text{estimatePNSize}(q) \) stops decreasing. We obtain a window containing the target, i.e., \([\frac{p}{2}, q]\), and we proceed with binary search (Lines 11-18). Alg. 19 finds the target \( p \) using at most \( 2 \log p \) invocations of \( \text{estimatePNSize}() \), and it is asymptotically and practically negligible compared to the runtime cost of computing overlay edges in DC.

### 4.6 Divide-and-Conquer with Bulk Clones

As both the example in Fig. 4.1 and the experimental evaluation in §6.6 indicate, DC may produce a higher node degree compared to GM because of partitioning. The example in Fig. 4.1 further suggests that the average node degree of an TCO is highly sensitive to the placement of nodes that subscribe to a lot of topics (i.e., bulk subscribers). The impact of this placement is exacerbated by the fact that representative pub/sub workloads from real-world applications follow the “Pareto 80–20” rule [36]: Most nodes subscribe to a relatively small number of topics. This makes the few bulk subscribers a desirable addition to each partition: A partition without bulk subscribers would require significantly more links in order to attain topic-connectivity.

In view of this observation, we design an algorithm that applies random partitioning only to lightweight subscribers, while replicating bulk subscribers in every partition. More precisely, we identify the star set for the lightweight subscribers, perform the combine phase across the union of the star set and the bulk subscribers, and then conquer each partition separately. The purpose of this order is to ensure that the same links between bulk subscribers are reused in each partition. The new algorithm attains an improved worst-case bound on the average node degree compared to DC.
Alg. 20 Divide-and-Conquer with Bulk Clones

DCBulkClone($V, T, \text{Int}$)

Input: $V, T, \text{Int}$

Output: A topic-connected overlay $\text{TCO}(V, T, \text{Int}, E_{\text{DCBC}})$

1. $\eta \leftarrow \text{chooseEta}(V, T, \text{Int})$
2. $B \leftarrow \{v \in V : |T(v)| > \eta\}$
3. $L \leftarrow V - B$
4. $T_{\text{restricted}} \leftarrow T - \bigcup_{v \in B} T(v)$
5. $S \leftarrow \text{getStarSetFromNodes}(L, T_{\text{restricted}}, \text{Int}|L)$
6. $B \leftarrow B \cup S$
7. $L \leftarrow L - S$
8. $\text{TCO}_B(B, T, \text{Int}|B, E_B) \leftarrow \text{GreedyMerge}(B, T, \text{Int}|B)$
9. $p \leftarrow \text{chooseP}(V, T, \text{Int})$
10. Randomly divide $L$ into $p$ partitions $L_d, d=1, 2, ..., p$
11. for $d = 1$ to $p$ do
12. $BL_d \leftarrow B \cup L_d$
13. $\text{Int}_d \leftarrow \text{Int}|BL_d$
14. $\text{nodesInterests}_d \leftarrow \emptyset$
15. add $(B, \text{Int}|B)$ to $\text{nodesInterests}_d$
16. for $v \in L_d$ do
17. add $(v, \text{Int}|v)$ to $\text{nodesInterests}_d$
18. $E_d \leftarrow \text{greedyConnect}(BL_d, T, \text{Int}_d, \text{nodesInterests}_d)$
19. $E_{\text{DCBC}} \leftarrow E_B \cup (\bigcup_{d=1}^p E_d)$
20. return $\text{TCO}(V, T, \text{Int}, E_{\text{DCBC}})$

Alg. 20 specifies the Divide-and-Conquer with Bulk Clones algorithm (DCBC). We denote by $T(v)$ the topic set which node $v$ subscribes to, and by $|T(v)|$ the subscription size of node $v$. In order to tune the selection of bulk subscribers, we introduce an additional parameter called bulk subscription threshold $\eta$, $0 < \eta \leq |T|$. Parameter $\eta$ determines the subscription size of bulk subscribers $B$: $B \supseteq \{v : |T(v)| > \eta\}$ (Line 2) and Line 6 further extends $B$ to make sure that it covers all topics in $T$. We first construct an overlay $\text{TCO}_B$ for bulk subscribers in $B$ (Line 8), and just impose random partitioning on lightweight subscribers $L$ (Line 10). To conquer each lightweight partition, we replicate the bulk subscribers and its associated $\text{TCO}_B$ in each partition, and then build the TCO.
on a partially constructed overlay by invoking \texttt{greedyConnect()} (Line 11-18).

Before considering how to choose key parameters for the algorithm (i.e., the implementations for \texttt{chooseEta()} and \texttt{chooseP()}), we first establish correctness, approximation ratio and running time cost of the DCBC algorithm.

**Lemma 21.** (Correctness) Alg. 20 is correct: It yields a TCO for the input instance \((V, T, \text{Int})\).

Following the notations for the DC algorithm, for the \(d\)-th partition with node set \(BL_d \subseteq V\), we denote \(E_{d,OPT}\) as the minimum edge set to satisfy topic-connectivity for the instance \((BL_d, T, \text{Int}|_{\text{BL}_d})\) and \(E_{OPT}|_{BL_d}(\subseteq E_{OPT})\) as the edge subset induced by \(BL_d\). The DCBC algorithm does not construct edges across different lightweight partitions because lightweight nodes always connect to bulk nodes to attain topic-connectivity. This might introduce additional sub-optimality. Fortunately, this sub-optimality is not so profound due to the small subscription size of lightweight nodes.

**Lemma 22.** (Approximation Ratio) Alg. 20 produces an overlay with a total number of edges that has an approximation ratio of \(O(\eta \cdot \log(|V||T|))\) as compared to the optimal solution for MinAvg-TCO: \(\frac{|E_{DCBC}|}{|E_{OPT}|} = O(\eta \cdot \log(|V||T|))\)

The approximation ratio of DCBC given in Lemma 22 is improved as compared to that of DC given in Lemma 18. More specifically, \(\eta\) is relatively small and can be considered as a constant factor in typical pub/sub workloads, while \(p\) is a higher order variable parameter in DC.

If we define \(\beta = \frac{|B|}{|T|}\), then Lemma 23 establishes the running time for the DCBC algorithm.

**Lemma 23.** (Running Time) The running time of Alg. 20 is

\[
T_{DCBC} = O\left(\frac{(\beta(p - 1) + 1)^2}{p}|V|^2|T|\right)
\]

The selection of the bulk subscriber threshold \(\eta\) captures the tradeoff between node degree and running time. On the one hand, small threshold values result in treating the majority of nodes as bulk subscribers, which favors node degree over running time. As shown in Lemma 22, threshold \(\eta\) bounds the approximation ratio on the average node degree (regardless of the number of partitions \(p\)). When \(\eta\) is close to 0, all nodes are treated as bulk subscribers and the DCBC algorithm turns out to be equivalent to GM. On the other hand, large threshold values are favorable for the running
time at the expense of increased node degree. Keeping the threshold value low is important for reducing the risk of poor partitioning in the DCBC algorithm, and thus providing a better bound on the average node degree of the output TCO. Fortunately, even relatively small threshold values result in small bulk subscriber sets for typical pub/sub workloads that follow the “Pareto 80-20” rule (see §4.6). Therefore, in our implementation of chooseEta() in Line 1 of Alg. 20, we choose an \( \eta \) that causes \( \leq 20\% \) of all nodes to be considered as bulk subscribers. Since \( \beta \leq 0.2 \) in this case, it leads to a significantly improved running time of DCBC compared to that of GM according to Lemma 23.

In order to implement the selection of the number of partitions in chooseP(), we use Alg. 19 that estimates \( pn\text{-size} \), just as we do in the context of DC. The considerations discussed in §4.5 still apply so that Alg. 19 provides a reasonable value of \( p \) for DCBC (see §4.7.3).

### 4.7 Evaluation

We implemented the SM, DC and DCBC algorithms, and other auxiliary algorithms in Java. We denote by \( TCO_{ALG} \) the TCO produced by ALG, by \( d_{ALG} \) the average node degree in \( TCO_{ALG} \), and by \( T_{ALG} \) the running time of ALG, where ALG stands for any of the discussed algorithms. These algorithms are compared to the GM algorithm under varying experimental conditions, such as average node degree and running time. GM is used as a baseline because it produces the lowest average node degree of all known polynomial algorithms.

In these experiments, we use the following value ranges as input: \( |V| \in [1000, 8000] \), \( |T| \in [100, 2000] \). We define the average node subscription size, minimum subscription size, and the maximum subscription size as follows: \( |T_v| = \frac{\sum_{v \in V} |T_v|}{|V|} \), \( |T_v|_{min} = \min_{v \in V} \{|T_v|\} \), \( |T_v|_{max} = \max_{v \in V} \{|T_v|\} \). We used \( |V| = 1000 \), \( |T| = 100 \), and \( |T_v| = 50 \) (with \( |T_v|_{min} = 10 \), \( |T_v|_{max} = 90 \)) for most of the experiments unless specified otherwise. Each topic \( t_i \in T \) is associated with probability \( q_i \), \( \sum_{i} q_i = 1 \), so that each node subscribes to \( t_i \) with a probability \( q_i \). The value of \( q_i \) is distributed according to either a uniform, a Zipf (with \( \alpha = 2.0 \)), or an exponential distribution. According to [36], these distributions are representative of actual workloads used in industrial pub/sub systems today. For some evaluations we only present results under the uniform topic popularity
4.7.1 Star Merge Algorithm for MinAvg-TCO-Join

We evaluate the output and performance of two incremental TCO-join algorithms: Star Merge and Naive Merge. We compare these two with GM which builds the TCO from scratch.

We generate several sub-TCOs as follows: Each sub-overlay $TCO_d$ has $|V_d| = 100$ nodes ($d = 1, \ldots, p$, where $p$ is the number of sub-TCOs), and all sub-overlays share the same topic set $T$ with $|T| = 100$. These sub-overlays are constructed by GM and fed to the TCO-join algorithms as input.

Fig. 4.3 shows that under different distributions, SM and NM produce similar output overlays with a slightly higher number of edges compared to GM. However, the difference in the average node degree is insignificant ($\lesssim 2.5$).

Although SM and NM produce quite close average node degrees (the difference is $\lesssim 0.15$), SM runs considerably faster than NM. As it is shown in Fig. 4.4, under the uniform distribution, it takes SM less than 1% of $T_{GM}$ on average, while the cost for NM is as much as around 75% of $T_{GM}$. This could be explained by different pn-sizes between SM and NM. To illustrate the dependence, we normalize pn-size by the total number of nodes $|V|$. More precisely, we define the potential neighbor ratio as $pn\text{-ratio} = \frac{pn\text{-size}}{|V|}$. Note that $pn\text{-ratio}$ presented in our experiment plots refers to $pn\text{-ratio}(V)$, the potential neighbor ratio for the node set $V$. SM achieves much lower $pn\text{-ratio}$, which is $\lesssim 18.6\%$ in the worst case in our experiments under the uniform distribution.
NM’s \textit{pn-ratio} remains 100\%, the same as GM.

As the number of sub-TCOs that request to join increases, the average node degrees of the SM output remain steady while the running time ratio of SM to GM decreases considerably, which is caused by the decline in the \textit{pn-ratio}. Notably, it only takes 0.28\% of GM’s running time for SM to incrementally construct a TCO with 8 000 nodes. This further attests to SM’s scalability with respect to the number of nodes.

### 4.7.2 Random Partitioning for Divide-and-Conquer

First, we evaluate the effects of random partitioning for the DC algorithm. We run the algorithm 400 times for the same settings (namely, the default experimental settings discussed above, the uniform distribution) so that the only difference between different runs is due to the random node interest generation according to the given distribution parameters and due to random node partitioning. The statistics pertaining to the average node degree, running time ratio, and the average size of a star set are reported in Table 4.1. As the table illustrates, all the values are quite stable with negligible variance values across different experiments. Besides, the results validate our assumption that $\tilde{k} \ll k$. We conclude that when the number \( p \) of partitions is reasonable, random partitioning is an efficient and robust way to implement the divide phase of DC and results in small star sets that could be computed efficiently, which is vital to the performance of the DC algorithm.

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>variance</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_{DC} )</td>
<td>6.50</td>
<td>0.000578</td>
<td>6.42</td>
<td>6.56</td>
</tr>
<tr>
<td>( d_{GM} )</td>
<td>5.09</td>
<td>0.000175</td>
<td>5.07</td>
<td>5.13</td>
</tr>
<tr>
<td>( T_{DC}/T_{GM} )</td>
<td>0.0634</td>
<td>0.000143</td>
<td>0.0205</td>
<td>0.150</td>
</tr>
<tr>
<td>( \tilde{k} )</td>
<td>9.54</td>
<td>0.297</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Second, we study the impact of \( p \) on the output and performance of DC. Given input \( V, T \) and \( Int \), where \(|T| = 200\), the DC algorithm is executed with all possible values of \( p \) ranging from 1 to \(|V|\). We already know that DC exhibits identical behavior to GM at the two ends of \( p \)’s range where \( p = 1 \) and \( p = |V|\). Fig. 4.5 shows that as the number of partitions \( p \) grows from 1 to \(|V|\), \( d_{DC} \) first increases gracefully and then it starts to decrease until it becomes equal to \( d_{GM} \). Note that
Chapter 4. Divide and Conquer Algorithms for Pub/Sub TCO Design

$d_{DC}$ never moves far from the horizon line of $d_{GM}$. Under the uniform distribution, the difference in average node degree between DC and GM is less than 6 even when $d_{DC}$ reaches its peak. It is less than 3.5 when the running time of DC is minimum. On the other hand, DC’s running time (ratio to GM) first slides down sharply and then climbs up as $p$ increases. It touches the lowest point when $p$ is around 10, which is relatively close to the beginning of $p$’s spectrum. Aligned with our proposed Alg. 19 for choosing $p$, $T_{DC}$ (and consequently $T_{DC}/T_{GM}$) forms a similar U-shape as $pn$-ratio, and their lowest values are located close to each other. This further verifies our approach to choose $p$ by minimizing $pn$-size (which is equivalent to minimizing $pn$-ratio).

Fig. 4.6 looks inside the operation of the DC algorithm and shows how the “inner” (divide and conquer) phases and the “outer” (combine) phase contribute to the studied metrics for different values of $p$. As shown in Fig. 4.6, increasing $p$ leads to the increase of $E_{outDC}$ and $T_{outDC}$. However, we would like $E_{inDC}$ and $T_{inDC}$ to be the dominant terms, because we expect the total runtime cost to be as low as possible. Besides, in some pub/sub systems such as [76], inter-partition routing takes order of magnitude more time than inside-partition message distribution, so small $E_{outDC}$ is desired for efficient event dissemination. Thus, instead of having many small partitions, it is better to have fewer bigger and well-connected overlays. A well-selected $p$ should be relatively small and stand far away from the endpoint $p = |V|$, which is again in line with the value of $p$ output by Alg. 19.

Third, we compare two methods for the divide phase: random partitioning and clustering. We choose random partitioning for DC as discussed in §4.4. As a comparison alternative, we implement the $k$-means clustering algorithm with the following optimization techniques: we compute the Hamming distance between two nodes [92], i.e., $dist(v, w) = |\{t : \text{Int}(v, t) \neq \text{Int}(w, t)\}|$, using
random initialization and the elbow method to determine the number of clusters [53].

As Fig. 4.8 and Fig. 4.9 show, DC outperforms $k$-means in running time in all experiments. As shown in Fig. 4.8(a) and Fig. 4.9(a), as the number of nodes increases, $k$-means would produces TCOs with smaller average node degrees compared to the outputs of DC because grouping similar nodes would result in fewer inner edges. However, as the number of topics or the subscription size increases, $k$-means does not always outperform DC in terms of average node degree. Sometimes it even performs worse, which is quite noticeable in Fig. 4.9(b) under the Zipf topic popularity. This is because under skewed distribution, the placement of bulk nodes becomes increasingly important for the average node degree, and $k$-means clustering tends to isolate bulk nodes from lightweight nodes. Lightweight nodes need many more links to attain topic-connectivity without bulk nodes because the lack of bulk nodes results in a significant loss of correlation in one partition.

Overall, these experiments validate our choice of random partitioning for the divide-and-conquer algorithm design. We opt not to explore clustering techniques in greater depth, e.g., by using other distance metrics or applying more sophisticated clustering algorithms. There is not much potential for improvement due to using clustering in the divide phase, because divide-and-conquer with ran-
dom partitioning already achieves good performance both theoretically and empirically, which we show in §4.7.3.

### 4.7.3 DC and DCBC for MinAvg-TCO

We compare the performance of the DC and DCBC algorithms and demonstrate their scalability with respect to different input variables. Given any MinAvg-TCO instance, we determine the algorithm parameters based on the analysis discussed in Sec. 4.5 and Sec. 4.6: (1) we choose the number of partitions $p$ for both DC and DCBC by Alg. 19; (2) we choose the threshold $\eta$ for DCBC such that $\leq 20\%$ of all nodes to be treated as bulk subscribers.

**Comparison with Ring-Per-Topic**  We compare the average node degree obtained by DC, DCBC, GM and Ring-Per-Topic (RingPT). RingPT is an algorithm that mimics the common practice of building a separate overlay for each topic (usually a tree but we use a ring that has the same average node degree). According to RingPT, all the nodes interested in the same topic form a ring, and rings for different topics are merged into a single TCO. As Fig. 4.10 shows: the average node degrees output by DC and DCBC are quite close to those of GM, the average difference between GM and DCBC is 4.2, and DCBC is slightly better than DC (by 0.99 on average). At the same time, the average node degree of RingPT, which is roughly equal to twice the average subscription size, is around 10 times the average node degrees of DC and DCBC. This demonstrates the scalability of DC and DCBC when the number of nodes grows.
CHAPTER 4. DIVIDE AND CONQUER ALGORITHMS FOR PUB/SUB TCO DESIGN

Effect of $|V|$  
Fig. 4.7 depicts the comparison between DC and GM as the number of nodes increases under different distributions. The figure shows that as the total number of nodes increases, DC gains in speedup while keeping the average node degree steadily low. Under the uniform distribution, for example, DC takes on average 1.67% of GM’s running time to construct a high-quality TCO. The average node degree is around 10.02, which is less than 5.3 higher as compared to GM’s.

Fig. 4.11(a) compares the performance of DC, DCBC and GM with regard to the number of nodes under the uniform distribution. The figure shows that DC and DCBC output similar TCOs in terms of average node degree, which is slightly higher than $d_{GM}$. However, both DC and DCBC run considerably faster than GM. DCBC obtains a marginally better average node degree compared to DC, i.e., $d_{DCBC} \approx d_{DC} - 0.99$ on average, because DCBC uses a more refined partitioning mechanism. However, this gain in $d_{DCBC}$ has an extra runtime cost because of the additional calculation imposed by replicating the bulk nodes: under the uniform distribution, $T_{DCBC}$ is 164% · $T_{DC}$ and 2.6% · $T_{GM}$ on average.

Effect of $|T|$  
Fig. 4.11(b) depicts how DC and DCBC perform compared to GM as the number of topics scales up. As shown, under the uniform distribution, the average node degrees of all three algorithms increase with the number of topics. This is because increasing the number of topics leads to reduced correlation among the nodes. The gap in the average node degree between DC and GM becomes more noticeable as the number of topics increases: $d_{DC} - d_{GM} = 14.74$ when the number of topics reaches 2,000, which is an increase of over 69.0%. DCBC achieves a lower average node degree as compared to DC, and the difference against GM is insignificant, i.e., $d_{DCBC} - d_{GM} = 1.94$ when $|T| = 2,000$. This is in line with Lemma 22, which shows the improvement in the average
node degree produced by DCBC as compared to that of DC. Fig. 4.11(b). The results suggest that bulk nodes play a more important role in capturing the correlation among a node set when there is a large number of topics, and that the placement of bulk nodes has a more profound impact on the average node degree of the divide-and-conquer algorithms. In addition, both DCBC and DC run more efficiently than GM. DCBC exhibits a higher speedup as compared to DC when the number of topics increases. The time ratios of $T_{DC}/T_{GM}$ and $T_{DCBC}/T_{GM}$ increase at a low pace proportionally to the number of topics. Still $T_{DCBC} \leq 5.7\% \cdot T_{GM}$ and $T_{DCBC} \leq 10.5\% \cdot T_{GM}$ in the worst case when the number of topics is as high as 2000.

**Effects of subscription size** Fig. 4.11(c) depicts how the subscription size affects the DC and DCBC algorithms. We set $|V| = 1000$, $|T| = 400$, and $|\bar{T}(v)|$ varies from 50 to 150. The figure shows that under the uniform distribution, as the subscription size increases, both $d_{DC}$ and $d_{DCBC}$ decrease, and the gaps in the degree $(d_{DC} - d_{GM}$ and $d_{DCBC} - d_{GM})$ shrink. This can be explained by the increase in topic correlation within the node set when the subscription size grows. Like GM, both DC and DCBC also choose each link in a greedy manner (but from a smaller set), so that the selected edge is more likely to connect a higher number of TC-components since the nodes share more common interests.

Beside, the running time ratios of DC and DCBC compared to GM further improve with the increase in the average subscription size. Although more updates for an edge addition would be incurred, DC and DCBC significantly reduce the time for each update compared to GM, since each edge update in DC and DCBC only involves a small subset of all nodes.

Similarly to Fig. 4.11(a) and Fig. 4.11(b), Fig. 4.11(c) shows that DCBC treads the balance between the quality of the output and time efficiency, i.e., both $d_{DCBC}$ and $T_{DCBC}$ lie between those of DC and GM algorithms. DCBC marginally outperforms DC in terms of average node degrees, i.e., the difference $d_{DC} - d_{DCBC}$ is 0.97 on average. However, this benefit comes at the expense of additional runtime, i.e., DCBC takes 4.4% more time compared to DC.

In summary, Fig. 4.11 shows that (1) both DC and DCBC produce TCOs with low average node degree at significant speedup as compared to GM under a variety of typical pub/sub workloads; (2) DCBC provides a more fine-tuned mechanism to seek the balance between the average node
degree and the running time, and this benefit is especially important when the sub-optimality of DC becomes noticeable, e.g., when the number of topics scales up.

4.8 Conclusions

In this chapter, we introduce the MinAvg-TCO-Join optimization problem that attempts to add the minimum number of links to join a number of sub-TCOs into a single TCO. MinAvg-TCO-Join is NP-complete. We develop the efficient and scalable Star Merge (SM) algorithm that constructs the TCO with a logarithmic approximation ratio as compared to the optimal solution.

Our approach inspires by the divide-and-conquer strategy that gives rise to a number of new algorithms for the original MinAvg-TCO problem. Theoretical analysis shows that with a reasonable partitioning of the node set, our divide-and-conquer algorithms are able to construct a high-quality TCO significantly faster than earlier alternatives. We motivate novel techniques including random partitioning and bulk-lightweight partitioning that captures the trade-off between the runtime cost and the quality of the output TCO. We also provide a numerical algorithm to determine the number of partitions as the parameter for any input instance.

A comprehensive experimental analysis demonstrates the performance of the family of divide-and-conquer algorithms. It also validates important design decisions in the context of our algorithms for MinAvg-TCO, such as random partitioning, bulk-lightweight partitioning, the selection of the number of partitions, and the bulk subscriber threshold. Under realistic pub/sub workloads, divide-and-conquer algorithms successfully produce low average node degree TCOs with significantly lower running time than alternatives. Divide-and-conquer is more scalable than GM with regard to the number of nodes, the number of topics, and the subscription size.
5.1 Introduction

The main contribution of this chapter is the design of a MinMax-TCO solution that focuses on efficiency and alleviates the above limitations. In its core lies a divide-and-conquer approach to the problem: We partition the set of nodes into subsets, build a TCO for each subset, and combine all TCOs into a global overlay. The appeal of this scheme is in the substantially faster TCO construction for each subset of nodes that requires only partial knowledge about the nodes within the partition. Since the creation of TCOs for different partitions is independent, the process can be parallelized and decentralized. Yet, in order to apply this approach we need to overcome a number of obstacles.

The first challenge is comprised in the impact of partitioning on the quality of the solution. We show that the minimal overlay degree is very sensitive to partitioning and it may increase by up to a factor of $\Theta(|T|)$ in the worst case. Our solution is based on the study of workloads in existing pub/sub systems and the observation that in practical pub/sub deployments, only a relatively small number of nodes might be interested in a large number of topics [64]. We formalize this as an assumption and optimize our solution for this case. This assumption does not simplify the
MinMax-TCO problem: the number of bulk subscribers is still too large to make any brute force solution around the impossibility result effective. Furthermore, it does not reduce the running time of MinMax-ODA sufficiently for practical applications. Yet, it allows us to come up with a partitioning scheme for which the divide-and-conquer approach retains the logarithmic upper bound on the overlay degree provided by MinMax-ODA.

Next, we devise an algorithm for the combine phase. Our first solution is an adaptation of the MinMax-ODA algorithm along with the proof that the adapted algorithm preserves the approximation ratio. This solution serves as a baseline for analyzing performance, identifying bottlenecks and weaknesses, and devising more advanced algorithms. Unfortunately, it does not improve the running time of MinMax-ODA. Furthermore, it still requires global knowledge about the interests of each node.

To address this issue, we observe that not all nodes need to participate in the combine phase. In each partition, we can select a number of representative nodes so that their combined interest covers the interest of all nodes in the partition. We show that if the combine phase is only performed on the representative nodes (one representative set from each partition), then the resulting overlay will still be topic-connected. Running the combine phase only on representative nodes drastically improves the running time and eliminates the need for a central point of control that possesses complete knowledge about the system. At the same time, it may have a profound impact on the overlay degree unless we select representative nodes in a careful and controlled way. We show how to perform this selection so as to tread the balance between overlay degree and running time.

We evaluated our solution through a series of simulations on characteristic pub/sub workloads with up to 8,000 nodes and 1,000 topics. The results indicate that on average, our solution requires less than 4.0% of the running time of known state-of-the-art algorithms while yielding an insignificant increase in the maximum node degree of 2.0. While we did not analyze space complexity or measure the program footprint, the improvements in this respect are also noteworthy: For the same pub/sub workload distribution with 8,000 nodes and under the same environmental settings, our solution was taking less than a minute to construct the overlay whereas known state-of-the-art algorithms would experience memory-related problems (with 14GB of RAM allocated).
5.2 Divide-and-Conquer for MinMax-TCO

Taking into account the MinMax-ODA running time of $O(|V|^4|T|)$, the number of nodes is the most significant factor determining the performance and scalability of the solution for MinMax-TCO. In view of this, we devise a divide-and-conquer strategy to solve the problem: (1) divide the MinMax-TCO problem into several sub-overlay construction problems that are similar to the original overlay but with a smaller node set, (2) conquer the sub-MinMax-TCO problems independently and build sub-overlays into sub-TCOs, and then (3) combine these sub-TCOs to one TCO as a solution to the original problem.

In this section we present the design steps and key decisions of the divide-and-conquer approach for MinMax-TCO. We show analytically that the resulting algorithm leads to a significantly improved running time cost and reduced knowledge requirement as compared to the MinMax-ODA algorithm. In §6.6, we quantify these improvements empirically.

5.2.1 MinMax-ODA as a Building Block for Divide-and-Conquer

In this section, we analyze the MinMax-ODA algorithm in greater depth and derive several new results about its running time. We employ MinMax-ODA as a building block in our divide-and-conquer algorithms and it serves as the baseline for our experimentation.

First, we show that the maximum node degree of the overlay produced by MinMax-ODA is bounded by the maximum subscription size in the input.

**Lemma 24 (Bound on the maximum degree).** The maximum node degree of the TCO produced by MinMax-ODA is $O(\max_{v \in V} |T_v|)$.

In [70], Alg. 3 is the only entry point for Alg. 4. This means that $E_{\text{cur}} = \emptyset$ and $E_{\text{pot}} = V \times V$ upon the invocation of Alg. 4. When we adapt MinMax-ODA for the combine phase of the divide-and-conquer approach, we need to apply MinMax-ODA on a collection of TCOs already produced in the conquer phase. Therefore, we have to extend the analysis of MinMax-ODA for the case when $E_{\text{cur}}$ is non-empty and $E_{\text{pot}}$ is equal to $(V \times V) \setminus E_{\text{cur}}$. Let $E_{\text{new}}$ be the set of edges returned by Alg. 4. Denote the maximum degree of $G(V, E)$ by $D(V, E)$ and the maximum degree of the optimal solution for MinMax-TCO($V, T, Int$) by $D_{\text{OPT}}(V, T, Int)$. We have Lemma 25:
Lemma 25. If invoked on $V$, $T$, $\text{Int}$, $E_{\text{cur}}$, and $E_{\text{pot}}$ such that $E_{\text{cur}} \cup E_{\text{pot}} = V \times V$, Alg. 4 outputs $E_{\text{new}}$ such that (a) $G(V, E_{\text{cur}} \cup E_{\text{new}})$ is topic-connected and (b) the maximum node degree $D(V, E_{\text{cur}} \cup E_{\text{new}})$ is bounded by $O(D(V, E_{\text{cur}}) + D_{\text{OPT}}(V, T, \text{Int}) \cdot \log(|V||T|))$.

5.2.2 Divide and Conquer Phases of the Solution

There exist two principal methods to divide the nodes: (1) node clustering and (2) random partitioning. Node clustering is organizing the original node set into groups so that nodes with similar interests are placed in the same group while nodes with diverging interests belong to different groups. Random partitioning assigns each node in the given node set to one of the partitions based on a uniformly random distribution. Once the partitions are determined, the existing MinMax-ODA algorithm can be employed to conquer the sub-MinMax-TCO problems by determining inner edges used for the construction of the sub-overlays.

The idea of node clustering seems attractive because well-clustered nodes with strongly correlated interests would result in lower maximum node degrees in the sub-TCOs produced by MinMax-ODA. The problem with this approach is the high runtime cost of clustering algorithms taking into account the large number of nodes and varying subscription size. Additionally, they require the computation of a “distance” metric among nodes. In our case this translates to calculating pairwise correlations among node interests with significant run time cost implications. It is challenging to fit node clustering into the divide-and-conquer approach so that the latter is still superior to the MinMax-ODA algorithm in terms of running time cost. Furthermore, it is difficult to devise an effective decentralized algorithm for node clustering that would not require complete knowledge of $V$ and $\text{Int}$. Finally, node clustering by interests may yield clusters that vary in size depending on the clustering algorithm used. On the other hand, the divide-and-conquer approach performs optimally when partitions are equal-sized and there are no large clusters that stand out.

We choose random partitioning for the divide-and-conquer approach because it is extremely fast, more robust than node clustering, easier to tune, and it can be realized in a decentralized manner. Furthermore, the construction of inner edges for each overlay only requires knowledge of node interests within the overlay. Hence, random partitioning can be oblivious to the composition of nodes and their interests. The number of partitions $p$ is given as an input parameter and each sub-
overlay has \( k = |V_d| = \frac{|V|}{p} \) nodes, where \( d = 1, \ldots, p \). This equal-sized division is optimal with respect to the running time. We present the resulting algorithm in Alg. 21.

**Alg. 21** Naive algorithm for *divide* and *conquer* phases

**Input:** \( V, T, \text{Int}, p \)

\( \text{//} p: \) the number of partitions, \( 1 \leq p \leq |V| \).

**Output:** A list of TCOs \( \text{List}_{\text{TCO}} \), one TCO for each partition

1: \( \text{List}_{\text{TCO}} \leftarrow \emptyset \)

2: Randomly divide \( V \) into \( p \) partitions \( V_d \), \( d = 1, 2, \ldots, p \)

3: for \( d = 1 \) to \( p \) do

4: \( \text{Int}_d \leftarrow \text{Int}|V_d| \)

5: \( TCO_d(V_d, T, \text{Int}_d, E_d) \leftarrow \text{MinMax-ODA}(V_d, T, \text{Int}_d) \)

6: add \( TCO_d \) to \( \text{List}_{\text{TCO}} \)

7: return \( \text{List}_{\text{TCO}} \)

Unfortunately, random partitioning may place nodes with diverging interest into the same partition thereby reducing the amount of correlation that is present in the original node set. As Lemma 26 shows, this may have a profound effect on the maximum node degree. Consider the overlay \( G(V, E) \) for the *conquer* phase produced by this algorithm where \( \text{List}_{\text{TCO}}[d] = TCO_d(V_d, T, \text{Int}_d, E_d), E = \bigcup_{d=1}^{p} E_d \). Then,

**Lemma 26.** There is an instance \( I(V, T, \text{Int}) \) of MinMax-TCO on which the maximum degree \( D(V, E) \) of the overlay output by Alg. 21 is greater by a factor of \( \Theta(|T|) \) than the maximum node degree \( D_{OPT}(V, T, \text{Int}) \) of the optimal solution for \( I(V, T, \text{Int}) \).

**Proof.** Construct an instance \( I(V, T, \text{Int}) \) of MinMax-TCO as follows: Consider the topic set \( T = \{t_1, t_2, \ldots, t_m\} \) of size \( m = 2^h \). Node set \( V \) consists of \( h + 1 = \log m + 1 \) subsets, denoted as \( A_i, 0 \leq i \leq h; \) each node subset \( A_i \) contains \( 2^i \) nodes, i.e., \( A_i = \{v(i,1), \ldots, v(i,2^i)\} \). Each node \( v(i,j), 1 \leq j \leq 2^i \) subscribes to a topic set \( T_{(i,j)} \) of size \( \frac{m}{2^j} \), defined as follows:
Chapter 5. Scaling Construction of Low Fan-out Pub/Sub TCO

\[ V = \bigcup_{i=0}^{h} A_i \quad \quad \quad T = \{t_1, \ldots, t_m\} \]

\[ A_0 = \{v_{(0,1)}\} \quad \quad \quad T_{(0,1)} = \{t_1, \ldots, t_m\} \]

\[ A_1 = \{v_{(1,1)}, v_{(1,2)}\} \quad \quad \quad T_{(1,1)} = \{t_1, \ldots, t_{\frac{m}{2}}\}, T_{(1,2)} = \{t_{\frac{m}{2}+1}, \ldots, t_m\} \]

\[ \vdots \quad \quad \quad \vdots \]

\[ A_{i} = \{v_{(i,1)}, \ldots, v_{(i,2^i)}\} \quad \quad \quad T_{(i,1)} = \{t_{\frac{m}{2^i}+1}, \ldots, t_{\frac{m}{2^i}(2^i)}\}, 1 \leq j \leq 2^i \]

\[ \vdots \quad \quad \quad \vdots \]

\[ A_{h} = \{v_{(h,1)}, \ldots, v_{(h,m)}\} \quad \quad \quad T_{(h,1)} = \{t_1\}, \ldots, T_{(h,m)} = \{t_m\} \]

The optimal overlay \( TCO_{OPT} \) for \( I(V, T, Int) \) is shown in Fig. 5.1(a). Its maximum node degree is a constant: \( D_{OPT} = 3 \). Consider the output of Alg. 21. Due to the random partitioning in Line 2, there is a chance for generating a partition that consists of nodes in \( A_0 = \{v_{(0,1)}\} \) and \( A_{h} = \{v_{(h,1)}, \ldots, v_{(h,m)}\} \) (see Fig. 5.1(b)). To attain topic-connectivity for this partition, \( v_{(0,1)} \) has to be linked to all \( m \) nodes in \( A_{h} \), which makes the node degree of \( v_{(0,1)} \) to be \( m = |T| \).

![Figure 5.1](image-url)

Consequently, the maximum degree \( D(V, E) \) of the overlay output by Alg. 21 for \( I(V, T, Int) \) is greater by a factor of \( \Theta(\frac{m}{|T|}) = \Theta(|T|) \) than the maximum node degree \( D_{OPT}(V, T, Int) \) of the optimal solution for \( I(V, T, Int) \).

Essentially, Lemma 26 shows that if we use random partitioning for the divide-and-conquer approach, then the overlay degree for the conquer phase alone may exceed the overlay degree for
the complete optimal solution by a factor of $\Theta(|T|)$. Furthermore, our empirical validation indicates that not only for a manually constructed worst case but also for the typical pub/sub workloads, random partitioning causes significant increase in the maximum node degree. Fig. 5.2 illustrates this effect for the default workload defined and motivated in §6.6. It compares the maximum degree for the conquer phase produced by Alg. 21 with the total degree of the overlay produced by MinMax-ODA.

This effect of increased maximum degree occurs when a node subscribed to a large number of topics (i.e., a bulk subscriber) is placed into the same partition with nodes whose subscriptions are not correlated. Then, such nodes do not benefit from creating a link to each other and need to connect to the bulk subscriber. Our solution to this problem is based on the study of representative pub/sub workloads used in actual applications that are described and characterized in [36]. According to these characterizations, the “Pareto 80–20” rule works for pub/sub workloads: Most nodes subscribe to a relatively small number of topics. The phenomenon of increased maximum degree due to partitioning still exists in such workloads as the example of Lemma 26 indicates. Yet, this observation allows us to devise an effective solution: We provide an algorithm that applies random partitioning only to such lightweight subscribers, performs the conquer phase for lightweight partitions, and merges them with bulk subscribers at the combine phase.
**Alg. 22** Divide and conquer phases for lightweight nodes

\texttt{conquerLightweight}\(I(V; T; \text{Int}; p)\)

\textbf{Input:} \(I(V; T; \text{Int}; p)\)

// \(\eta\): the bulk node threshold;
// \(p\): the number of partitions for lightweight nodes.

\textbf{Output:} A list of TCOs \texttt{List\_TCO}, one TCO for each partition

1. \(B \leftarrow \{v \in V : |T_v| > \eta\}, \text{Int}_B \leftarrow \text{Int}|_B\)

2. \(\text{List\_TCO} \leftarrow \emptyset\)

3. Randomly divide \(L = V - B\) into \(p\) partitions \(L_d, d = 1, \ldots, p\)

4. for \(d = 1\) to \(p\) do

5. \(\text{Int}_d \leftarrow \text{Int}|_{V_d}\)

6. \(\text{TCO}_d(V_d; T; \text{Int}_d; E_d) \leftarrow \text{MinMax-ODA}(L_d; T; \text{Int}_d)\)

7. add \(\text{TCO}_d\) to \texttt{List\_TCO}

8. return \texttt{List\_TCO}

Formally, given an instance \(I(V; T; \text{Int})\) for MinMax-TCO, we introduce an additional parameter called bulk subscription threshold \(\eta, \eta \in (0, |T|]\). \(\eta\) determines the division of \(V\) into the set of bulk and lightweight subscribers \(B\) and \(L\), respectively: \(B = \{v : |T_v| > \eta\}\) and \(L = \{v : |T_v| \leq \eta\}\). Alg. 22 applies random partitioning to \(L\), creates a TCO for each of the partitions, and returns a list of these TCOs.

Lemma 24 can be directly applied to the overlay output by Alg. 22 to produce a bound on its maximum degree.

**Lemma 27** (Bound on the maximum degree for the conquer phase). The maximum node degree of an overlay \(G(V, E)\) produced by Alg. 22 is bounded by the bulk subscription threshold: \(D(V, E) = O(\eta)\).

We now analyze the running time of Alg. 22.

**Lemma 28** (Running time for the conquer phase). The running time cost of Alg. 22 is \(O\left(\frac{|L| \cdot |T|}{p^2}\right)\).

\textit{Proof.} The loop in Lines 4–7 of Alg. 22 uses MinMax-ODA to build a sub-TCO for each sub-overlay. Each sub-overlay has at most \(\frac{|L|}{p}\) nodes so that by Lemma 7, the running time for constructing each sub-TCO is \(O\left(\frac{|L| \cdot |T|}{p^2}\right)\). Thus, the running time for constructing all \(p\) overlays is \(O\left(\frac{|L| \cdot |T|}{p^2}\right)\). \(\square\)
We use Alg. 22 as a building block to complete the divide-and-conquer approach for MinMax-TCO (see § 5.2.3).

5.2.3 Combine Phase of the Solution

In the core of our design for the combine phase solution lies the observation that Alg. 4 can be used to merge the sub-overlays for different partitions and the set of bulk subscribers into a single TCO. To implement this idea, we devise Alg. 24 that applies Alg. 4 on a union of the sub-overlays produced at the conquer phase by Alg. 22. Alg. 23 called DCB-M, presents a complete divide-and-conquer solution for MinMax-TCO.

Alg. 23 Divide-and-Conquer with Bulk Nodes and Lightweight Nodes for MinMax

DCB-M(I(V, T, Int), \eta, p)

Input: \( I(V, T, Int) \), \( \eta \)

// \( \eta \): the bulk node threshold;
// \( p \): the number of partitions for lightweight nodes.

Output: A topic-connected overlay \( TCO(V, T, Int, E_{DCB}) \)

1: \( \text{List}_{TCO} \leftarrow \text{conquerLightweight}(I(V, T, Int), \eta, p) \)
2: \( TCO(V, T, Int, E_{DCB}) \leftarrow \text{combineB&L}(V, T, Int, \text{List}_{TCO}) \)
3: \( \text{return } TCO(V, T, Int, E_{DCB}) \)

Alg. 24 Combine B nodes and L nodes greedily

combineB&L(V, T, Int, \text{List}_{TCO})

Input: \( V, T, Int, \text{List}_{TCO} \)

\( *\) \( \text{List}_{TCO} \): a list of \( p \) node-disjoint TCOs for lightweight nodes: \( \text{List}_{TCO}[d] = TCO_d(L_d, T, Int_d, E_d), \)
\( d=1, \ldots, p, *\)

Output: A topic-connected overlay \( TCO(V, T, Int, E_{DCB}) \)

1: \( B \leftarrow V - \bigcup_{d=1}^{p} L_d \) // \( L_d \) is short for \( \text{List}_{TCO}[d].L_d \)
2: \( E_{inDCB} \leftarrow \bigcup_{d=1}^{p} E_d \) // \( E_d \) is short for \( \text{List}_{TCO}[d].E_d \)
3: \( E_{potDCB} \leftarrow \{e = (v, w)|(v \in B, w \in V) \land (w, v) \notin E_{potDCB}\} \)
4: \( E_{potDCB} \leftarrow E_{potDCB} \cup \{e = (v, w)|v \in L_i, w \in L_j, i < j\} \)
5: \( E_{outDCB} \leftarrow \text{buildMMEdges}(V, T, Int, E_{inDCB}, E_{potDCB}) \)
6: \( E_{DCB} \leftarrow E_{inDCB} \cup E_{outDCB} \)
7: \( \text{return } TCO(V, T, Int, E_{DCB}) \)
Lemma 29. (Correctness) Alg. 23 is correct: it yields a TCO.

Given an instance \(I(V, T, \text{Int})\) for MinMax-TCO, let \(TCO_{\text{DCB}}\) be the TCO produced by Alg. 23. Denote its maximum node degree as \(D_{\text{DCB}}\). There are two types of edges that form the \(TCO_{\text{DCB}}\): (1) \(E_{\text{inDCB}}\), the inner edges constructed by Alg. 22, \(E_{\text{inDCB}} = \bigcup_{d=1}^{p} E_{d}\) and (2) \(E_{\text{outDCB}}\), the outer edges conjoining bulk subscribers and lightweight node sub-TCOs, which are created in Line 5 of Alg. 24. The maximum node degree induced by \(E_{\text{inDCB}}\) and \(E_{\text{outDCB}}\) are denoted as \(D_{\text{inDCB}}\) and \(D_{\text{outDCB}}\), respectively. It holds that

\[
D_{\text{DCB}} \leq D_{\text{inDCB}} + D_{\text{outDCB}}. \tag{5.1}
\]

Equation 5.1 along with Lemma 27 and Lemma 25 allow us to establish an upper bound on the degree of the overlay produced by DCB-M.

Lemma 30 (Degree bound for DCB-M). The overlay network output by Alg. 23 has maximum node degree \(D_{\text{DCB}} = O(\eta + D_{\text{OPT}}(V, T, \text{Int}) \cdot \log(|V||T|))\).

Corollary 1 (Approximation ratio for DCB-M). If we regard the bulk node threshold as a constant factor or if the maximum degree \(E_{\text{inDCB}}\) of the sub-overlays constructed at the conquer phase is smaller than the maximum degree \(D_{\text{OPT}}(V, T, \text{Int})\) of the optimal overlay for MinMax-TCO\((V, T, \text{Int})\), then

\[
D_{\text{DCB}} = D_{\text{OPT}}(V, T, \text{Int}) \cdot O(\log(|V||T|)). \tag{5.2}
\]

If the conditions in Corollary 1 hold, then the DCB-M algorithm achieves the same logarithmic approximation ratio as the MinMax-ODA algorithm (Alg. 3).

Next, we consider the running time of the DCB-M algorithm. Let \(T_{\text{inDCB}}\) and \(T_{\text{outDCB}}\) denote the running time to build \(E_{\text{inDCB}}\) and \(E_{\text{outDCB}}\), respectively. Let \(T_{\text{DCB}}\) be the total running time cost of Alg. 23. Then, we obtain the running time of DCB-M with:

Lemma 31 (Running time for DCB-M). The running time of Alg. 23 is \(T_{\text{DCB}} = O(T_{\text{inDCB}} + T_{\text{outDCB}}) = O(|T| \cdot (|B||V| + |L|^2))^2\).
Proof. $T_{inDCB} = O\left(\frac{\|T\|}{p}\right)$ following Lemma 28.

$T_{outDCB}$ is determined by Alg. 24, whose running time is dominated by the invocation of Alg. 4 in Line 5. Based on Lemma 7, we have:

$$T_{outDCB} = O(|T| \cdot |E_{outDCB}|^2) = O(|T| \cdot (|B||V| + |L|^2)^2) \quad (5.3)$$

$$T_{DCB} = O(T_{inDCB} + T_{outDCB}) = O(T_{outDCB}) = O(|T| \cdot (|B||V| + |L|^2)^2) \quad (5.4)$$

\[\square\]

In summary, Alg. 23 has asymptotic performance very similar to that of Alg. 3, both with respect to the maximum degree and running time. As both the above analysis and experimental evaluations in § 6.6 indicate, it produces a marginally higher overlay degree at marginally better runtime cost. Furthermore, the combine phase still requires complete knowledge of $V$ and $Int$, which makes decentralization infeasible. These shortcomings motivate the development of an improved solution for the combine phase. Our improvement is based on the notion of Representative Set, which we explain next.

Given $I(V, T, Int)$ for MinMax-TCO and a topic $t$, $t \in T$, we denote the set of subscribers to $t$ by $subs(t)$, $subs(t) = \{v|v \in V \land Int(v, t)\}$. Then, the notion of a representative set (rep-set) is defined as follows:

**Definition 2 (Representative set).** Given $I(V, T, Int)$, a rep-set with the coverage factor $\lambda$, denoted as $R(\lambda)$ (or $R$), is a subset of $V$ such that

$$|subs(t)|_R \geq \min\{\lambda, |subs(t)|\}, \forall t \in T \quad (5.5)$$

A node $r \in R(\lambda)$ is referred to as a representative node (rep-node).

As illustrated in Figure 5.3, a rep-set is a subset of overlay nodes that represents the interests of all the nodes in the overlay. Each topic of interest is covered by at least $\lambda$ subscribers in $R$ unless
the total number of subscribers to this topic is smaller than $\lambda$. The complete node set $V$ is always a rep-set, but there might exist many other rep-sets with much fewer rep-nodes. In essence, these nodes can function as bridges for the purpose of determining cross-TCO connections. Observe that it is possible to attain full topic-connectivity only by using cross-TCO links among rep-nodes for different partitions. Suppose we have a number of TCOs, and each TCO is represented by a rep-set (of a smaller size). To achieve topic-connectivity for a topic $t \in T$, we can just connect nodes from different rep-sets which are interested in $t$.

![Diagram](image)

Figure 5.3: $R_1 = \{v_1, v_2, v_3, v_4\}$ and $R_2 = \{u_1, u_4, u_5, u_6\}$ are rep-sets with $\lambda = 2$ for TCO$_1$ and TCO$_2$ respectively; a complete TCO for all nodes is obtained by adding cross-TCO links between $R_1$ and $R_2$.

For typical pub/sub workloads and sufficiently large partitions, minimal rep-sets are several times smaller than the total number of nodes. This leads to significant benefits if we consider only rep-nodes as candidates for cross-TCO links. One, the running time of the overlay construction algorithms discussed in this chapter is roughly proportional to the number of nodes up to the fourth degree, therefore our algorithm that only considers rep-nodes runs much faster. Two, calculation of cross-TCO links no longer requires complete knowledge of $V$ and $\text{Int}$, and only a partial view of rep-nodes from rep-sets and their interests is needed. Three, rep-sets of different TCOs can be computed in parallel in a fully decentralized fashion.

At the same time, minimality of rep-sets also has an adverse effect on the maximum overlay degree due to reduced correlation across rep-sets for different TCOs. Revisit the instance $I(V, T, \text{Int})$ of MinMax-TCO described in the example of Lemma 26. Suppose $V$ is divided into two partitions: a partition of bulk subscribers that includes node subsets $A_i, 0 \leq i \leq h - 1$ and a partition of lightweight subscribers that includes node subset $A_h$. The minimal rep-set for the first partition contains a single node $v_{(0,1)}$ whereas the minimal rep-set for the second partition contains all nodes...
in $A_h$. If we merge the rep-sets at the combine phase, the degree of $v_{(0,1)}$ will be $|A_h| = \Theta(|T|)$.

On the other hand, if we merge the whole partitions, the optimal overlay will be the one depicted in Fig. 5.1(a) with constant maximum degree.

The difference arises due to the fact that in the former case, $v_{(0,1)}$ serves the focal point for all cross-overlay links while in the latter case, these links are evenly distributed across all the nodes of the first partition. We employ two techniques to prevent the above effect. First, we only use rep-sets for the partitions of lightweight nodes and not for bulk subscribers. This is because the degree of lightweight nodes is bounded by $O(\eta)$ as we later show in Lemma 33 so that the effect is not as significant for lightweight nodes compared to bulk subscribers. Second, we use a coverage factor greater than one to ensure that there are always multiple choices when connecting the nodes for any topic.

We still need to consider, how to efficiently determine a minimal rep-set given $V$, $T$, $\text{Int}$, and $\lambda$. The problem of computing a minimal rep-set set is equivalent (through a linear reduction) to a variation of the classic NP-complete Set Cover problem, in which each item has to be covered by at least $\lambda$ sets. Alg. 27 provides a greedy implementation that attains a provable logarithmic approximation [71]. The algorithm starts with an empty rep-set and continues adding nodes to the rep-set one by one until all topics of interest are $\lambda$-covered, i.e., covered by at least $\lambda$ nodes. At each iteration, the algorithm selects a node that is interested in the largest number of topics that are not yet $\lambda$-covered.

Alg. 26 presents the resulting implementation for combining sub-TCOs. The algorithm operates in two phases. First, it determines a rep-set for each sub-TCO. Note that the rep-set for $\text{TCO}_d$ does not need to cover all of $T_d$. It suffices to cover $T_{\text{out},d} = T_d \cap \left( \bigcup_{i \neq d} T_i \right)$. Second, the algorithm connects all the nodes in the rep-sets as well as bulk subscribers into a TCO in a greedy manner by using Alg. 4. Alg. 25 called DCBR-M, presents our complete solution for MinMax-TCO.

Below, we establish correctness, approximation ratio and running time properties for DCBR-M.

**Lemma 32 (Correctness).** Alg. 25 is correct: it yields a TCO.

Following the notations for DCB-M algorithm, we denote the TCO produced by Alg. 25 as $\text{TCO}_{\text{DCBR}}$ and its maximum node degree as $D_{\text{DCBR}}$. Observe that by operating on a reduced set of nodes at the combine phase, the invocation of Alg. 4 in line 12 of Alg. 26 solves an instance of
MinMax-TCO\((BR, T, \text{Int}|_{BR})\) where \(BR\) is a union of \(B\) and all rep-sets \(\bigcup_{d=1}^{p} R_d\). The fact that \(D_{\text{DCBR}} \leq D_{\text{inDCBR}} + D_{\text{outDCBR}}\) along with Lemma 27 and Lemma 25 allow us to establish an upper bound on the degree of the overlay produced by DCBR-M.

**Lemma 33** (Degree bound for DCBR-M). The overlay network \(\text{TCO}_{\text{DCBR}}\) output by Alg. 25 has maximum node degree:

\[
D_{\text{DCBR}} = O(\eta + D_{\text{OPT}}(BR, T, \text{Int}|_{BR}) \cdot \log(|BR|/|T|)).
\]

According to Lemma 33, if we choose a sufficiently large coverage factor \(\lambda\) so that

\[
D_{\text{OPT}}(BR, T, \text{Int}|_{BR}) \approx D_{\text{OPT}}(V, T, \text{Int})
\]

Then Alg. 25 will generate a TCO whose maximum node degree is asymptotically the same as that of the TCO output by Alg. 23.

Using the same reasoning as in Lemma 28, we can derive the running time cost of DCBR-M.

**Lemma 34** (Running time for DCBR-M). The running time of Alg. 25 is

\[
T_{\text{DCBR}} = O(|T| \cdot ((|B| + |R|)^4 + \frac{|L|^4}{p^3})).
\]

**Proof.**

\[
T_{\text{outDCBR}} = O(|T| \cdot |E_{\text{potDCBR}}|^2) = O(|T| \cdot (\frac{|B| \cdot (|B| + |R|) + |R|^2}{p^3})^2) = O(|T| \cdot ((|B| + |R|)^4))
\]

(5.6)

\[
T_{\text{DCBR}} = O(T_{\text{outDCBR}} + T_{\text{inDCBR}}) = O(|T| \cdot ((|B| + |R|)^4 + \frac{|L|^4}{p^3}))
\]

(5.7)

Lemma 34 shows that if representative sets are significantly smaller than partitions, the bulk
subscribers threshold is selected so that there are few bulk subscribers, and the number of partitions is sufficiently large, then Alg. 25 achieves significant speedup compared to Alg. 23. This is also corroborated by our experiments in § 6.6.

**Alg. 25** Divide-and-Conquer with Bulk Nodes and Lightweight Rep-nodes for MinMax-TCO

\[ \text{DCBR-M}(I(V, T, \text{Int}), \eta, p, \lambda) \]

**Input:** \( I(V, T, \text{Int}), \eta, p, \lambda \)

// \( \lambda \): the coverage factor.

**Output:** A topic-connected overlay \( TCO(V, T, \text{Int}, E_{\text{DCBR}}) \)

1. \( L_{TCO} \leftarrow \text{conquerLightweight}(I(V, T, \text{Int}), \eta, p) \)
2. \( TCO(V, T, \text{Int}, E_{\text{DCBR}}) \leftarrow \text{combineB&LReps}(V, T, \text{Int}, L_{TCO}, \lambda) \)
3. return \( TCO(V, T, \text{Int}, E_{\text{DCBR}}) \)

**Alg. 26** Combine B nodes and L rep-nodes greedily

\[ \text{combineB&LReps}(V, T, \text{Int}, \text{List}_{\text{TCO}}, \lambda) \]

**Input:** \( V, T, \text{Int}, \text{List}_{\text{TCO}}, \lambda \)

**Output:** A topic-connected overlay \( TCO(V, T, \text{Int}, E_{\text{DCBR}}) \)

1. \( B \leftarrow V - \bigcup_{d=1}^{p} L_d \) // \( L_d \) is short for \( \text{List}_{\text{TCO}}[d], L_d \)
2. \( E_{in_{\text{DCBR}}} \leftarrow \bigcup_{d=1}^{p} E_d \) // \( E_d \) is short for \( \text{List}_{\text{TCO}}[d], E_d \)
3. for \( d = 1 \) to \( p \) do
   4. \( T_d \leftarrow \bigcup_{v \in L_d} T_v \)
4. for \( d = 1 \) to \( p \) do
   5. \( T_{out,d} \leftarrow T_d \cap \bigcup_{i \neq d} T_i \)
7. \( R_d \leftarrow \text{getRepSetFromNodes}(L_d, T_{out,d}, \text{Int}, \lambda) \)
8. \( R \leftarrow \bigcup_{d=1}^{p} R_d \)
9. \( BR \leftarrow B \cup R \)
10. \( E_{pot_{\text{DCBR}}} \leftarrow \{e = (v, w) |(v \in B, w \in BR) \land (w, v) \not\in E_{pot_{\text{DCBR}}})\} \)
11. \( E_{pot_{\text{DCBR}}} \leftarrow E_{pot_{\text{DCBR}}} \cup \{e = (v, w) |v \in R_i, w \in R_j, i < j)\} \)
12. \( E_{out_{\text{DCBR}}} \leftarrow \text{buildMMEmes}(BR, T, \text{Int} | BR, E_{in_{\text{DCBR}}}, \text{BR}, E_{pot_{\text{DCBR}}} ) \)
13. \( E_{\text{DCBR}} \leftarrow E_{in_{\text{DCBR}}} \cup E_{out_{\text{DCBR}}} \)
14. return \( TCO(V, T, \text{Int}, E_{\text{DCBR}}) \)
Alg. 27 Determine a representative set for a partition

\texttt{getRepSetFromNodes}(V_d, T_{out,d}, Int, \lambda)

Input: \(V_d, T_{out,d}, Int, \lambda\)

Output: \(R_d\): A representative set for \(V_d\)

1: Start with \(T_{toCover} = T_{out,d}\) and \(R_d = \emptyset\)
2: \textbf{for all} \(t \in T_{toCover}\) \textbf{do}
3: \(N_{toCover}[t] = \lambda\)
4: \textbf{while} \(T_{toCover} \neq \emptyset\) \textbf{do}
5: \(r \leftarrow \arg\min_{v \in V_d - R_d} \left( \sum_{t \in \{t \in T_{toCover} \mid \text{Int}(v, t)\}} \right)\)
6: \(R_d \leftarrow R_d \cup \{r\}\)
7: \textbf{for all} \(t \in T_{toCover} \land \text{Int}(r, t)\) \textbf{do}
8: \(N_{toCover}[t] \leftarrow N_{toCover}[t] - 1\)
9: \textbf{if} \(N_{toCover}[t] = 0\) \textbf{then}
10: \(T_{toCover} \leftarrow T_{toCover} - \{t\}\)
11: Return \(R_d\)

5.2.4 Decentralizing the DCBR-M Algorithm

Note that DCBR-M as presented above is fully centralized. It is possible to decentralize it in the following way: (1) each lightweight node autonomously decides which random partition it belongs to and registers itself under the partition name (it is possible, e.g., to use a DHT for that purpose), (2) nodes from the same partition learn about each other and establish a communication channel, (3) different partitions construct sub-TCOs in parallel, i.e., the nodes within each partition exchange their interests and execute the MinMax-ODA algorithm, (4) different partitions compute rep-sets in parallel, (5) bulk subscribers and rep-nodes from different rep-sets communicate their interests and compute outer edges. Note that the original MinMax-ODA algorithm does not lend itself to such decentralization.

This decentralization scheme has several important benefits: reducing the total runtime cost, optimizing distributed resource utilization, and balancing the computational load. The time \(T_{\text{inDCBR}}\) for computing inner edges becomes \(O\left(\frac{|L|^4}{p^2}\right)\) and the total time \(T_{\text{DCBR}}\) becomes

\[
T_{\text{DCBR}} = O(|T| \cdot ((|B| + |R|)^4 + \frac{|L|^4}{p^2})).
\] (5.8)
Furthermore, decentralization eliminates the need for a central entity that must maintain a global knowledge of all nodes and their interests. To quantify this benefit, we introduce additional performance characteristic of the algorithm called potential neighbor set, which is the set of other nodes a node has to learn about in the course of the algorithm. This characteristic is important because gathering nodes’ interests in a scalable and robust decentralized manner is a problem in its own right. Additionally, the fan-out of node \( v \) in the overlay produced by any algorithm cannot exceed the size of the potential neighbor set of \( v \). Therefore, minimizing the potential neighbor set has an additional desirable effect from this point of view.

To formalize this argument, we define the potential neighbor ratio for a node \( v \), denoted as \( pn\text{-}ratio(v) \). Potential neighbor ratio is the size of potential neighbor set for \( v \) (including \( v \) itself) normalized by the total number of nodes \( |V| \). For any centralized algorithm, this ratio is equal to 1. For DCBR-M, the potential neighbor set for \( v \) consists of three subsets: (1) nodes in the same partition as \( v \): \( V_d \) such that \( v \in V_d \) (if \( v \) is a lightweight node); (2) bulk subscribers \( B \) (if \( v \) is a bulk subscriber itself or it belongs to some rep-set); and (3) all rep-nodes from other partitions: \( \{ u | u \in R_i \text{ s.t. } v \in B \lor v \in R_d \land R_i \neq R_d \} \) (if \( v \) is a bulk subscriber or it belongs to the rep-set \( R_d \)). Consequently, the potential neighbor ratio is always the biggest for lightweight nodes selected as rep-nodes. For such nodes, \( pn\text{-}ratio(v) = \frac{|L|}{|V|} \cdot p + \frac{|B|}{|V|} + \frac{|R|}{|V|} \cdot \frac{p-1}{p} \).

We extend the definition of a potential neighbor ratio to apply to the entire node set:

\[
pn\text{-}ratio(V) = \max_{v \in V} pn\text{-}ratio(v) = \frac{|L|}{|V|} \cdot p + \frac{|B|}{|V|} + \frac{|R|}{|V|} \cdot \frac{p-1}{p} \tag{5.9}
\]

Eq. 5.9 shows that DCBR-M has improved \( pn\text{-}ratio \) compared to any centralized algorithm (such as MinMax-ODA). This is further confirmed by our experiments in § 6.6.

### 5.3 Selecting Parameters

This section discusses how to choose optimal parameter values for our DCBR-M algorithm. Algorithm 25 is parametrized with (1) the bulk subscriber threshold \( \eta \), (2) the coverage factor \( \lambda \), and (3) the number of partitions for lightweight nodes \( p \). The choice of values for these parameters substan-
tially affects the algorithm’s behavior. It is therefore essential to identify a combination of them that leads to satisfactory performance. First, we pick reasonable values for \( \eta \) and \( \lambda \) for typical pub/sub workloads; then, we provide a numerical method to determine the optimal value of \( p \).

The selection of the bulk subscriber threshold \( \eta \) exhibits the tradeoff between maximal overlay degree and running time: Small threshold values cause all nodes to be treated as bulk subscribers thereby favoring the degree over running time and making the overall performance very similar to that of MinMax-ODA. On the other hand, large threshold values favor the running time and \( pn\text{-ratio} \) at the expense of increased overlay degree. Fortunately, even relatively small threshold values result in small bulk subscriber sets for typical pub/sub workloads that follow the “Pareto 80–20” rule, as discussed in §5.2.2. In our implementation, we sort the subscribers by subscription size and choose \( \eta \) that causes \( \leq 20\% \) of the nodes to be considered bulk subscribers.

The coverage factor selection exhibits a similar tradeoff: If we choose the coverage factor to be as large as the size \(|L|/p\) of the partitions, then the behavior of Alg. 25 becomes identical to that of Alg. 23. On the other hand, \( \lambda = 1 \) minimizes the size of rep-sets, \( pn\text{-ratio} \), and running time but leads to a severe impact on the node degree. According to our experiments in §6.6, an increase in \( \lambda \) beyond 3 only marginally improves the node degree, even for large partitions. The rep-sets for \( \lambda = 3 \) are significantly smaller for such large partitions than partitions themselves so that we choose 3 as the default value for \( \lambda \).

The tradeoff in the selection of the number \( p \) of partitions is more complex. When \( p \) is as large as \(|L|\), the performance is dominated by the invocations of the combineB&LReps() function in the combine phase. As \( p \) decreases, the effect of executing the MinMax-ODA algorithm at the conquer phase becomes more and more pronounced, both with respect to the degree and the running time. When we use a very small number of partitions, it starts to dominate the running time assuming \(|B|\) is relatively small. Therefore, we need to find intermediate values of \( p \) that minimize the running time and \( pn\text{-ratio} \).

The bound on the running time \( T_{DCBR} \) is established by Lemma 34 and Equation 5.9 for centralized and distributed implementations, respectively. Since the bound on \( T_{DCBR} \) depends on \(|R|\), which is difficult to assess analytically, we use an adaptive way for selecting \( p \). Since partitioning the nodes and computing the rep-sets is relatively cheap, we try partitioning for different \( p \) values. Each
We implemented all algorithms described in this chapter in Java and compared them under various experimental conditions. Table 5.1 summarizes the algorithms evaluated. We use MinMax-ODA as a baseline because it produces the lowest maximum node degree of all known algorithms that run in polynomial time. To be precise, we are using a faster implementation of the MinMax-ODA algorithm described in Chapter 3 both for baseline MinMax-ODA and as a building block for DCB-M and DCBR-M. This faster implementation produces exactly the same overlay as the original one in [70] at a lower runtime cost by manipulating data structures more efficiently. Our divide-and-conquer design is orthogonal to the data structures used in the algorithm: Our faster version still has prohibitively high running time without divide-and-conquer. In fact, the speedup of DCBR-M compared to MinMax-ODA would have been even more significant for the slower, original MinMax-ODA implementation. However, using a faster implementation allows us to run comparative experiments on a larger scale.

In the experiments, we use the following ranges for the input instances: $|V| \in [1000, 8000]$ and $|T| \in [100, 1000]$. We define the average node subscription size, minimum subscription size, and
We first evaluate the effects of random partitioning.

Figure 5.4: DCBR-M wrt. $p$

Figure 5.5: DCBR-M wrt. $\lambda$

Figure 5.6: DCBR-M wrt. dist.

the maximum subscription size as follows: $|T_v| = \sum_{i \in V} |T_i|/|V|$, $|T_v|_{min} = \min_{v \in V} \{|T_v|\}$, $|T_v|_{max} = \max_{v \in V} \{|T_v|\}$. We used $|V| = 4000$, $|T| = 200$, and $|T_v| = 50$ (with $|T_v|_{min} = 10$, $|T_v|_{max} = 90$) to generate the input workloads for most of the experiments unless specified otherwise. Each topic $t_i \in T$ is associated with probability $q_i$, $\sum_i q_i = 1$, so that each node subscribes to $t_i$ with a probability $q_i$. The value of $q_i$ is distributed according to either a uniform, a Zipf (with $\alpha = 2.0$), or an exponential distribution. According to [36], these distributions are representative of actual workloads used in industrial pub/sub systems today. The Zipf distribution is chosen because [64] shows it faithfully describes the feed popularity distribution in RSS. The exponential distribution is used by stock-market monitoring engines in [90] for the study of stock popularity in the New York Stock Exchange (NYSE). The $\eta$, $p$, and $\lambda$ parameters are selected as described in § 5.3.

### 5.4.1 Partitioning for Lightweight Nodes

**Random partitioning for lightweight nodes** We first evaluate the effects of random partitioning of lightweight nodes for the DCB-M and DCBR-M algorithms. We run the algorithm 400 times for the same settings (namely, the default experimental settings discussed above except $|V| = 1000$, under three different distributions) so that the only difference between different runs is due to the random node interest generation according to the given distribution parameters and due to random node partitioning. The statistics pertaining to maximum node degree, average node degree, running time ratio, and the ratio of nodes selected as rep-nodes are reported in Table 5.2. As the table il-
Table 5.2: Random Partitioning Under Exponential Distribution

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>min</th>
<th>max</th>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{DCBR}$</td>
<td>10.798</td>
<td>9</td>
<td>13</td>
<td>0.468</td>
</tr>
<tr>
<td>$D_{DCB}$</td>
<td>10.793</td>
<td>9</td>
<td>13</td>
<td>0.461</td>
</tr>
<tr>
<td>$D_{MMODA}$</td>
<td>8.4425</td>
<td>7</td>
<td>12</td>
<td>0.348</td>
</tr>
<tr>
<td>$d_{DCBR}$</td>
<td>4.499</td>
<td>4.394</td>
<td>4.586</td>
<td>0.00122</td>
</tr>
<tr>
<td>$d_{DCB}$</td>
<td>4.499</td>
<td>4.392</td>
<td>4.59</td>
<td>0.00124</td>
</tr>
<tr>
<td>$d_{MMODA}$</td>
<td>3.93</td>
<td>3.86</td>
<td>4.02</td>
<td>0.000747</td>
</tr>
<tr>
<td>$T_{DCBR}/T_{MMODA}$</td>
<td>0.130</td>
<td>0.111</td>
<td>0.161</td>
<td>0.0000678</td>
</tr>
<tr>
<td>$T_{DCB}/T_{MMODA}$</td>
<td>0.833</td>
<td>0.787</td>
<td>0.883</td>
<td>0.000225</td>
</tr>
<tr>
<td>$</td>
<td>R</td>
<td>/</td>
<td>L</td>
<td>$</td>
</tr>
</tbody>
</table>

I llustrates, under the exponential distribution, all the values are quite stable with negligible variance across different experiments; and the variance is even more insignificant under less skewed distributions. Besides, the results validate our assumption that $|R| \ll |L|$ (with $\lambda = 3$). We conclude that when the number $p$ of partitions is reasonable, random partitioning of lightweight nodes is an efficient and robust way to implement the divide phase of DCBR-M. Furthermore, it results in small rep-sets, which is vital to the performance of the DCBR-M algorithm.

Impact of $p$. Given $I(V, T, Int)$ where $|V| = 1,000$, we execute DCBR-M and DCB-M with all possible $p$ values ranging from 1 to $|L|$. Fig. 5.4 shows that under different values of $p$, $TCO_{DCBR}$ and $TCO_{DCB}$ have similar maximum and average node degrees, which are slightly higher than those of $TCO_{MMODA}$, DCBR-M runs substantially faster than DCB-M with the proper value of $p$.

We already know that DCBR-M and DCB-M exhibit identical behavior to MinMax-ODA when $p = |L|$. Fig. 5.4 shows that as the number of partitions $p$ grows from 1 to $|L|$, $D_{DCBR}(\approx D_{DCB})$ first increases gradually and then it starts to decrease until it becomes equal to $D_{MMODA}$. Note that the $D_{DCBR}$ never moves far from the horizon line of $D_{MMODA}$: It is always smaller than 6.5. It is less than 2.8 for the value of $p$ that minimizes $T_{DCBR}$.

While $T_{DCB}$ stays close to $T_{MMODA}$ for all values of $p$, $T_{DCBR}$ first slides down sharply and then climbs up as $p$ increases. The range of $p$ values for which this phenomenon occurs is small and relatively close to 0. (It touches the lowest point when $p$ is around 10.) Furthermore, the $T_{DCBR}/T_{MMODA}$ ratio follows the same trend as $pn$-ratio for DCBR-M, which is compatible with our discussion of choosing $p$ in § 5.3.
5.4.2 Impact of Coverage Factor

Here, we explored the impact of $\lambda$ on the output and performance of the DCBR-M algorithm. Given an input $I(V, T, \text{Int})$ where $|V| = 2000$ and $|T| \in [500, 1000]$, the DCBR-M algorithm is evaluated for four different values of the coverage factor ($\lambda = 1, 2, 3, 4$). As Fig. 5.5 shows, under uniform distribution, as $\lambda$ increases, $D_{\text{DCBR}}$ decreases. The differences in maximum node degrees also decrease with successive coverage factors, i.e., $D_{\text{DCBR}}|_{\lambda=k-1} - D_{\text{DCBR}}|_{\lambda=k} > D_{\text{DCBR}}|_{\lambda=k} - D_{\text{DCBR}}|_{\lambda=k+1}$. More specifically, compared to $D_{\text{DCBR}}|_{\lambda=2} - D_{\text{DCBR}}|_{\lambda=3} \approx 0.71$ on average, $D_{\text{DCBR}}|_{\lambda=1} - D_{\text{DCBR}}|_{\lambda=2}$ is noticeable, which could be as much as $\geq 3$ for most cases. When $\lambda \geq 3$, the difference is insignificant ($\leq 0.32$ on average). Meanwhile, under all coverage factors that were tested, DCBR-M runs remarkably faster as compared to MinMax-ODA ($T_{\text{DCBR}} \leq 3\%T_{\text{MMODA}}$ on average). Also, $T_{\text{DCBR}}$ slightly increases as $\lambda$ increases, because $pn$-ratio increases as a result.

However, differences among the running time cost for different coverage factors are insignificant:

$$\frac{T_{\text{DCBR}}|_{\lambda=4}}{T_{\text{MMODA}}} - \frac{T_{\text{DCBR}}|_{\lambda=1}}{T_{\text{MMODA}}} \leq 0.69\%$$

This experiment confirms the validity of choosing a relatively small integer as coverage factor in §5.3, both in terms of node degree and running time cost.

5.4.3 Effects Under Different Distributions

We now consider DCBR-M’s behavior under different input instances. We first provide an overview of the overall DCBR-M performance under different typical distributions, and then we analyze how DCBR-M is affected by various aspects of the input.

Fig. 5.6 depicts that under different distributions, DCBR-M produces high-quality TCOs in terms of maximum and average node degrees, which are slightly higher than $D_{\text{MMODA}}$ and $d_{\text{MMODA}}$, respectively. However, the differences are insignificant: $D_{\text{DCBR}} - D_{\text{MMODA}} \leq 2.0$, $d_{\text{DCBR}} - d_{\text{MMODA}} \leq 0.70$ on average. Although DCBR-M and MinMax-ODA produce quite close maximum and average node degrees, DCBR-M runs considerably faster than MinMax-ODA: $T_{\text{DCBR}} \leq 4.0\% \cdot T_{\text{MMODA}}$ on average. As the number of nodes increases, $D_{\text{DCBR}}$ and $d_{\text{MMODA}}$ remain steadily low while the running time ratio $T_{\text{DCBR}}/T_{\text{MMODA}}$ decreases considerably. This further attests to DCBR-M’s scalability with respect to the number of nodes in the network.
5.4.4 Impact of Number of Nodes

We now demonstrate DCBR-M’s scalability with respect to different input parameters. In the rest of the section, while we report on results and analysis for all distributions in the text, we only show figures for the uniform topic popularity distribution due to space limit.

Fig 5.7 depicts the comparison between DCBR-M, DCB-M and MinMax-ODA as the number of nodes increases where \(|T| = 100\). The figure shows that DCBR-M and DCB-M output similar TCOs with regard to maximum and average node degrees, but DCBR-M runs considerably faster. Under the uniform distribution, for example, \(T_{DCBR}\) is on average 4.79% of \(T_{MMODA}\) while \(T_{DCB}\) is as much as 75.7% of \(T_{MMODA}\). Additionally, DCBR-M gains more speedup with the increase in the number of nodes compared to the other algorithm.

5.4.5 Impact of Number of Topics

Fig. 5.8 depicts how DCBR-M and DCB-M perform compared to MinMax-ODA when we vary the number of topics. As the figure shows, under the uniform distribution, the maximum and average node degrees tend to be a bit more fluctuating under Zipf and exponential distributions compared to those under the uniform distribution. This could be explained by the slightly higher variance under skewed distributions, as presented in Table 5.2. Although skewed distributions are more sensitive to the variations in the input, the maximum and average node degrees always stay low, even in the worst cases.
node degrees of all three algorithms increase for a higher number of topics. This is because increasing the number of topics leads to reduced correlation among subscriptions. However, the increase is slow paced and the difference $D_{\text{DCBR}} - D_{\text{MMODA}}$ remains insignificant: 3.51 for the uniform, 4.46 for the Zipf, and 3.46 for the exponential distribution on average.

The running time ratio of DCBR-M to MinMax-ODA slightly increases as the number of topics increases, yet this effect is insignificant: $T_{\text{DCBR}}$ is less than 3.8% of $T_{\text{MMODA}}$ on average.

### 5.4.6 Impact of Average Subscription Size

Fig. 5.9 depicts how the node subscription size affects the DCBR-M and DCB-M algorithms. We set $|V| = 1000$, $|T| = 400$, and $|T_v|$ varies from 50 to 150.

The figure shows that under the uniform distribution, DCBR-M and DCB-M produces quite close TCOs in terms of both maximum and average node degrees. As the subscription size increases, $D_{\text{DCBR}}$ and $D_{\text{MMODA}}$ decrease, and the difference of $(D_{\text{DCBR}} - D_{\text{MMODA}})$ shrinks. $d_{\text{DCBR}}$ follows the same trend.

This decrease occurs because the growth of $|T_v|$ causes increased correlation across the subscriptions. Upon bigger correlation, an edge addition to the overlay reduces a higher number of topic-connected components on average because the nodes share more comment interests. Therefore, a smaller number of edge additions are required before the overlay becomes topic-connected.

The ratio of $T_{\text{DCBR}}$ to $T_{\text{MMODA}}$ also decreases with the increase of $|T_v|$. In both algorithms, an edge addition causes a higher number of updates to topic-connected components for bigger $|T_v|$ (Lines 8-9 in Alg. 4). Yet, this effect has less influence on $T_{\text{DCBR}}$ compared to $T_{\text{MMODA}}$ since each update in DCBR-M affects a smaller portion of edges. Unlike $T_{\text{DCBR}}$, however, $T_{\text{DCB}}$ does not gain significant speedup.

### 5.4.7 Comparison with Ring-Per-Topic

Finally, we compare the maximum and average node degrees produced by DCBR-M MinMax-ODA and RingPT. RingPT is an algorithm that mimics the common practice of building a separate overlay for each topic (usually a tree but we use a ring that has the same average node degree). According to RingPT, all the nodes interested in the same topic form a ring for that topic, after which rings for
different topics are merged into a single overlay. $|T|$ is set to 100 in this experiment. As Fig. 5.10 shows, $D_{DCBR}$ and $D_{MMODA}$ are quite close (the average difference is 1.22), but the maximum node degree of RingPT exceeds $D_{DCBR}$ by a factor of approximately 30. This demonstrates the general significance of overlay construction algorithms for pub/sub.

5.5 Conclusions

This chapter focuses on a number of design objectives for the MinMax-TCO problem that are central to creating a practical solution. We have designed the DCBR-M algorithm which is capable of constructing a low fan-out TCO, while being significantly more efficient than previously known solutions. Numerical techniques can be employed to effectively obtain a good combination of parameters which adapts to various inputs and guarantees the output and the performance of the algorithm. The algorithm is thoroughly examined via a comprehensive experimental analysis, which demonstrates the scalability of DCBR-M under different distributions as the number of nodes, the number of topics, and the subscription size increase.
Chapter 6

Constructing Fault-Tolerant Overlays for Topic-based Pub/Sub

6.1 Introduction

We improve the scalability and performance of TCO construction in Chapter 3, Chapter 4, and Chapter 5. Unfortunately, topic-connectivity per se does not address critical reliability requirements for the pub/sub overlay. In particular, there is no guarantee that topic-connectivity is preserved under even a single node crash. That is, all the desirable properties about TCOs are fragile and easily break in a dynamic environment. The root cause for this lies in the definition of TCO and TCO-related problems [35, 70, 69]. These definitions make an implicit assumption that the pub/sub overlay is reliable and robust, i.e., nodes and links in the network are fault-free.

In order to address this shortcoming, we propose a problem of constructing a $k$-topic-connected overlay ($k$TCO): topic-connectivity still holds as long as fewer than $k$ nodes fail simultaneously on the same topic (see Def. 3 in §6.2). The extension from TCO to $k$TCO captures the overlay’s resilience to churn by introducing a safety factor, $k$. This safety factor is important from an engineering perspective because pub/sub systems are dynamic in nature. Node churn may occur due to administrative maintenance or inevitable failures, such as hardware faults, misconfigurations, or software
bugs [95]. In practice, the set of active machines in a data center shows non-negligible variations over time [5]. Furthermore, the advent of new pub/sub applications, e.g., in sensor networks [62, 43] or mobile networks [66, 73], makes it increasingly important and challenging to enable the overlay’s reliability. In these scenarios, overlay nodes are not necessarily dedicated servers or brokers, and the pub/sub system is subjected to growing dynamism and additional resource constraints.

Advocates for TCO-structured pub/sub overlays might argue that $kTCO$ is not necessary. In principle, the $TCO$ can always be reconstructed in the presence of churn. However, this is impractical and wasteful since state-of-the-art algorithms suffer from a high computational complexity [35, 70, 69]. On the other hand, a few pub/sub systems (e.g., [36, 45, 84, 65]) have explored the problem of dynamically maintaining the $TCO$. Basically, these approaches constantly make incremental adjustments of the overlay in presence of churn. However, the overlays they produced are not as optimal in terms of the node degree as the centralized algorithms for $TCO$ construction, as corroborated by experimental studies, e.g., in [84]. Besides, approaches for incremental overlay maintenance can be applied to $kTCO$ as well to produce even more reliable solutions.

Furthermore, $kTCO$ can lead to better performance. First, $kTCO$ indicates that $k$ disjoint data paths exist from end to end for each topic (see Menger’s Theorem [93]). Thus, we can harvest network intelligence in the routing protocols on top of $kTCO$ by steering the traffic among multiple alternate paths in a more optimized and secure manner. Second, it is possible to reduce the diameter of the overlay, as we improve its connectivity [35]. With lower diameter, message delays are likely to be diminished because fewer hops are needed for message delivery.

Nevertheless, these merits of $kTCO$ come with a price – additional links are required. Intuitively, a sparse overlay is unlikely to be $kTCO$, while a dense overlay is sub-optimal with respect to node degree. However, it is also imperative for a pub/sub overlay network to have low node degrees. This is because it costs a lot of resources to maintain adjacent links for a high-degree node (i.e., monitor links and neighbors [35, 69]). For a typical pub/sub system, each link would also have to accommodate a number of protocols, service components, message queues, and so on. While overlay designs for different applications might be principally different, they all strive to maintain bounded node degrees, e.g., DHTs [63], peer-to-peer streaming [14], wireless networks [41], and survivable networks [58].
In this chapter, we formally study the fundamental trade-offs between attaining the $k\text{TCO}$ property while preserving low node degrees. Our main contributions are as follows:

- We propose the MinAvg-$k\text{TCO}$ problem of devising $k\text{TCO}$ with the minimum number of links (see Problem 6 in §6.2). Formally, we prove the NP-completeness of the MinAvg-$k\text{TCO}$ problem. We also show that MinAvg-$k\text{TCO}$ is difficult to approximate within a logarithmic ratio (§6.2).

- We design two algorithms for the MinAvg-$k\text{TCO}$ problem. First, with regards to the MinAvg-$2\text{TCO}$ problem, we present the first polynomial-time approximation algorithm, namely the GM2 algorithm in §6.3. We provide an approximation ratio for GM2, which almost meets the lower bound on the approximation ratio for the problem. Our proof of GM2’s approximation ratio exhibits novelties in several respects, including the concept of ear decomposition based on an edge sequence, the amortized analysis to measure the progress of the algorithm, the estimate of edge contribution towards $2\text{TCO}$, the charging argument against the optimal solution, and the mathematical analysis using number theory (see the detailed proof in §6.4). Second, with regards to the MinAvg-$k\text{TCO}$ problem, where $k \geq 2$, we propose a simple and efficient heuristic algorithm, namely HararyPT, that aligns nodes across different sub-overlays (see §6.5).

- In §6.6, we validate both GM2 and HararyPT with comprehensive experiments under a variety of characteristic pub/sub workloads of up to 1 000 nodes, 1 000 topics, and 100 subscriptions per node. GM2 requires an empirically small amount of additional edges to obtain a $2\text{TCO}$, whose average node degree is around 1.5 times that of the $1\text{TCO}$ produced by the algorithm with the best known approximation ratio. To achieve $k\text{TCO}$ ($k \geq 2$) for highly correlated pub/sub workloads, we show the practical effectiveness and scalability of the HararyPT algorithm with respect to the number of nodes, the number of topics, and the number of subscriptions per node.

### 6.2 Parameterized MinAvg-$k\text{TCO}$ Problem and its Complexity

The definition of a $k$-connected graph [93] can be directly applied to the sub-overlay induced by a topic $t \in T$. We call a $TCO(V, T, Int, E)$ $k$-connected for topic $t \in T$ if $G^{(t)} = (V^{(t)}, E^{(t)})$ is $k$-connected, i.e., $|V^{(t)}| > k$ and $G^{(t)} - X = (V^{(t)} - X, E^{(t)}\setminus \{e(v, w)| \text{either } v \in X \text{ or } w \in X\})$
is connected for every $X \subseteq V(t)$ with $|X| < k$.

We want to extend the definition of $k$-connectivity to a TPSO considering all topics in $T$. However, given a parameter $k$, $|V(t)|$ might be smaller than $k$ for some topic $t \in T$; in these cases, “$k$-connectivity” is not defined in classic graph theory, but we need to adopt a convention for TPSO. Intuitively, for a fixed $k$, a $k$-topic-connected overlay should have the property that the TPSO can still provide pub/sub service (for all topics) as long as fewer than $k$ nodes fail simultaneously on the same topic $t \in T$. If $|V(t)| < k$, the removal of $(k - 1)$ nodes on $t$ implies that none subscribes to $t$ any more, and thus the overlay no longer serves $t$. To ensure the pub/sub service continues with topic $t$ under other cases, we need to make sure $G(t)$ has no separate set, i.e., $G(t)$ is a complete graph. With this convention, we formally give Def. 3 and Problem 6.

**Definition 3.** A TCO$(V, T, \text{Int}, E)$ is k-topic-connected if for any $t \in T$, $G(t) = (V(t), E(t))$ is either (1) $k$-connected or (2) a clique if $|V(t)| \leq k$. We denote a k-topic-connected overlay by kTCO$(V, T, \text{Int}, E)$ (or kTCO).

**Problem 6.** The MinAvg-$k$TCO$(V, T, \text{Int})$ problem parameterized by an integer $k$ is defined as: Given a set of nodes $V$, a set of topics $T$, and the interest function Int, construct a kTCO that has the least possible total number of edges, i.e., the minimum average node degree.

For brevity, we often omit “parameterized by $k$” and just refer to the problem as MinAvg-kTCO.

The MinAvg-TCO problem is the base case of MinAvg-kTCO where $k = 1$. We have the Greedy Merge (GM) algorithm for MinAvg-TCO [35]. The GM algorithm starts with TPSO$(V, T, \text{Int}, E)$ where $E = \emptyset$ and proceeds by iteratively adding edges to $E$ until topic-connectivity is attained. At each iteration, GM greedily selects an edge $e$ with the highest GM-edge-contribution, which is defined as the number of TC-components reduced if an edge $e$ is added to the current overlay. The GM algorithm achieves a logarithmic approximation ratio, which is the lowest among all known polynomial-time algorithms. We use GM as the baseline for developing, analyzing, and evaluating new algorithms for the more generalized problem of MinAvg-kTCO.

We summarize the complexity analysis of the MinAvg-kTCO problem in Theorem 1. The proof is in Appx. A.1.
Theorem 1. For any given positive integer \( k \), the MinAvg-\( k \) TCO problem parameterized by \( k \) is NP-complete and cannot be approximated in polynomial time within a factor of \( O(\log |V|) \) unless \( P = NP \).

6.3 GM2 Algorithm to Build 2TCO

For the MinAvg-2TCO problem, we devise Greedy Merge for the 2TCO algorithm, GM2 for short. Although GM2 is structurally similar to the GM and other existing centralized algorithms that build TCO \([35, 70, 69]\), GM2 uses a principally different progress measure (see Line 5 of Alg. 28), which we will elaborate upon §6.4.

 Alg. 28 The GM2 algorithm for 2TCO

\[
\text{GM2}(V, T, \text{Int})
\]

**Input:** \( V, T, \text{Int} \)

**Output:** A 2-topic-connected overlay 2TCO\((V, T, \text{Int}, E_{\text{GM2}})\)

1: \( E_{\text{GM2}} \leftarrow \emptyset \)
2: \( E_{\text{pot}} \leftarrow V \times V \)
3: while TPSO\((V, T, \text{Int}, E_{\text{GM2}})\) is not 2-topic-connected do
4: for all \( e = (v, w) \in E_{\text{pot}} \) do
5: \hspace{1em} estimate\((e, E_{\text{GM2}}) \leftarrow \{|t \in T | \text{Int}(v, t) \land \text{Int}(w, t) \land \text{no TC-block in } G^{(t)} \text{ contains both } v \text{ and } w\}|\)
6: \hspace{1em} \( e \leftarrow \text{find } e \text{ s.t. } \text{estimate}(e, E_{\text{GM2}}) \text{ is maximum among } E_{\text{pot}} \)
7: \hspace{1em} \( E_{\text{GM2}} \leftarrow E_{\text{GM2}} \cup \{e\} \)
8: \hspace{1em} \( E_{\text{pot}} \leftarrow E_{\text{pot}} - \{e\} \)
9: return 2TCO\((V, T, \text{Int}, E_{\text{GM2}})\)

Given a TPSO\((V, T, \text{Int}, E)\), the 2-topic-connected component on topic \( t \in T \), is a maximal 2-connected subgraph induced on topic \( t \) (i.e., it is not contained in any larger 2-connected subgraph induced on \( t \)). We also call it topic-biconnected component or topic-connected block, TC-block for short. Thus, each TC-block on \( t \in T \) is either a maximal 2-topic-connected subgraph, a bridge (including its endpoints), or an isolated node in \( G^{(t)} \). Also, every such subgraph is a TC-block in
Due to their maximality property, different TC-blocks on \( t \in T \) overlap in at most one node in \( G(t) \). Hence, every edge \( e \in E(t) \) lies in a unique TC-block on \( t \) in \( G(t) \).

As specified in Alg. 28, \( \text{GM}^2 \) starts with the overlay \( TPSO(V, T, \text{Int}, E) \) where \( E = \emptyset \), so that there are \( |\{v | \text{Int}(v, t)\}| \) singleton TC-blocks for each \( t \in T \). The total number of TC-blocks at the start is

\[
B_{\text{start}} = \sum_{t \in T} |\{v | \text{Int}(v, t)\}| = O(|V||T|). \tag{6.1}
\]

The algorithm carefully adds an edge to \( E \) iteration by iteration until \( TPSO(V, T, \text{Int}, E) \) contains at most one TC-block for each \( t \in T \), i.e., 2-topic-connected, and the total number of TC-blocks at the end is reduced to

\[
B_{\text{end}} = |\{t \in T | \exists v \in V \text{ s.t. } \text{Int}(v, t) = \text{true}\}|. \tag{6.2}
\]

Lemma 35 summaries the correctness and running time of Alg. 28. We provide the proof in Appx. A.2.

**Lemma 35.** Alg. 28 outputs a 2TCO with time complexity \( O(|V|^4|T|) \).

### 6.4 Approximation Ratio of \( \text{GM}^2 \)

While both, \( \text{GM}^2 \) and \( \text{GM} \), are greedy algorithms employing similar heuristics, the analysis of \( \text{GM}^2 \) is much more complex as compared to that of \( \text{GM} \) [35]. The crux lies in the measure of progress each algorithm employs – namely, a quantity that strictly decreases (or increases) with every edge addition up to an absolute limit. The limit can be used to bound the number of edges produced by the algorithm. For example, \( \text{GM} \) defines the progress measure to construct 1TCO as the number of TC-components in the resulting overlay. The number of TC-components decreases every time \( \text{GM} \) adds an edge, and the number of TC-components is an integer-valued function on the current edge set. Based on the well-defined progress measure, the key is to establish a lower bound on the optimum – since the optimum solution must cover a complete round of progress (i.e., attaining full topic-connectivity), it needs at least a certain number of edges.
Unfortunately, the techniques for $\mathbb{G}M$ do not directly apply to the design of $\mathbb{G}M^2$. We need to overcome three major challenges: (1) find a meaningful measure of progress towards $2TCO$ (see §6.4.1), (2) estimate the progress as the algorithm proceeds (see §6.4.2), and (3) compare the output to the unknown optimum (see §6.4.3).

### 6.4.1 Progress Measure Towards the Construction of $2TCO$

Probably the most natural progress measure to construct $2TCO$ would be the number of $TC$-blocks. However, the number of $TC$-blocks does not always decrease when we add an edge at each iteration. Suppose some algorithm adds edges one by one as illustrated in Fig. 6.1, i.e., $e_i$ is added in the $i$-th iteration. The addition of edge $e_2$, $e_3$, or $e_4$ does not decrease the number of $TC$-blocks – the number of $TC$-blocks remains 4. Adding edge $e_5$ leads to a reduction from 4 to 1 in the number of $TC$-blocks, but not all of them should be accredited to $e_5$.

![Figure 6.1: An edge sequence forms $2TCO$ on $t$.](image)

The progress toward $2TCO$ is amortized over a sequence of edges added. We can still use the number of $TC$-blocks as a rough progress measure, but the task of comparing to the (unknown) optimum is more difficult. We will use a more subtle calculation (or estimation) of the progress measure that captures each edge contribution.

To present our progress measure we adopt the notation of sequences. Given an instance of the MinAvg-$2TCO$ problem $I(V, T, \text{Int})$, we look at an edge set $E \subseteq V \times V$. An edge sequence, $E$, is an ordered list of edges in $E$, denoted by $E = \langle e_1, e_2, \ldots, e_m \rangle$, where each edge $e_i (1 \leq i \leq m)$ is distinct. The length of an edge sequence $E$, denoted by $|E|$, is the number of ordered edges in $E$. So $|E| = |E|$. An edge sequence $F$ is a subsequence of $E$, if $F$ can be be derived from $E$ by deleting
some edges without changing the order of the remaining edges.

Suppose $E$ is the output edge set of some algorithm $A$ that adds edges iteratively one by one and produces a $2TCO$. The $A$-edge-sequence, $E = \langle e_1, e_2, ..., e_m \rangle$, indicates that $A$ adds $e_i$ in the $i$-th iteration. Given an edge $e \in E$, $ind_G(e)$, is the sequence index of $e$ in $E$, i.e., the iteration number of adding $e$ in algorithm $A$.

Consider $G^{(t)} = (V^{(t)}, E^{(t)})$ induced on topic $t \in T$, $E^{(t)}$ is a subsequence of $E$ that keeps the linear ordering of edge additions of $A$. $G^{(t)} = (V^{(t)}, E^{(t)})$ is 2-connected, which is equivalent to say that $G^{(t)}$ admits an ear decomposition [93]. Below, we adopt some additional concepts from graph theory and provide formal definitions for their use in our context, including the ear decomposition.

**Definition 4.** A path in a graph $G^{(t)} = (V^{(t)}, E^{(t)})$ is a nonempty set of edges $Y \subseteq E^{(t)}$ such that (1) edges in $Y$ can be linearly ordered as an edge sequence $Y = \langle (v_0, v_1), ..., (v_{n-1}, v_n) \rangle$ to connect a sequence of nodes $X = \{v_0, ..., v_n\}$, and (2) $v_0, ..., v_n$ are distinct and $v_1, ..., v_n$ are the terminal nodes of the path $Y$ and the other nodes $v_1, ..., v_{n-1}$ (which may not exist) are internal nodes. A closed path $Y$ with $v_0 = v_n$ is called a cycle, otherwise $Y$ is noncyclic.

**Definition 5.** Given $G^{(t)} = (V^{(t)}, E^{(t)})$ and a nonempty subset of edges $P^{(t)} \subseteq E^{(t)}$, let $W^{(t)} = \{v \in V^{(t)} | \exists e \in P^{(t)} \text{ s.t. } e \text{ is incident to } v \}$. An $P^{(t)}$-ear in $G^{(t)}$, denoted by $C^{(t)}$, is a noncyclic path in $G^{(t)}$ such that the two terminal nodes are in $W^{(t)}$ and the internal nodes are in $(V^{(t)} \setminus W^{(t)})$. The length of the ear $C^{(t)}$ is the number of edges in the path, which we denote as $|C^{(t)}|$. A trivial ear contains only one edge. We define the sequence index of $C^{(t)}$ with regards to $E$ as $ind_G(C^{(t)}) = \max\{ind_G(e) | e \in C^{(t)}\}$.

$G^{(t)}$ contains at least one cycle, otherwise it is not 2-connected. $A$ adds edges one by one according to $E$, so one cycle would at first be formed in $G^{(t)}$. This cycle has the minimum sequence index with regards to $E$. Moreover, given $E$, we can construct a corresponding ear decomposition for $G^{(t)}$.

**Definition 6.** The $E$-ear-decomposition on topic $t \in T$, denoted by $D^{(t)} = \left[ C_1^{(t)}, ..., C_2^{(t)} \right]$, is a partition of $G^{(t)} = (V^{(t)}, E^{(t)})$ into an ordered collection of edge-disjoint paths $C_1^{(t)}, ..., C_2^{(t)}$ s.t.
\[ S_1^{(t)} = C_1^{(t)} \] is the cycle in \( G^{(t)} \) with the minimum sequence index with regards to \( E \).

\( \triangleright \) For all \( 1 \leq j \leq z \), let \( S_j^{(t)} = C_1^{(t)} \cup \ldots \cup C_j^{(t)} \), then \( C_j^{(t)} \) is the \( S_{j-1}^{(t)} \)-ear with the shortest length among all \( S_{j-1}^{(t)} \)-ears that have the minimum sequence index with regards to \( E \). In other words, if \( C' \) is any other \( S_{j-1}^{(t)} \)-ear in \( G^{(t)} \), then either (1) \( \ind_E(C_j^{(t)}) < \ind_E(C') \) or (2) \( \ind_E(C_j^{(t)}) = \ind_E(C') \land |C_j^{(t)}| \leq |C'| \).

\( \triangleright \) \( S_z^{(t)} = \bigcup_{j=1}^{z} C_j^{(t)} = E^{(t)} \)

![Figure 6.2: Example of the E-ear-decomposition on topic t: (a) The initial cycle \( C_1^{(t)} \) is formed after adding \( e_6 \). (b) \( C_3^{(t)} \) is formed after adding \( e_9 \) and \( C_3^{(t)} \) is formed after adding \( e_{10} \). Note that \( C_3^{(t)} \) is trivial because it contains only one edge \( e_{10} \). (c) The addition of \( e_{14} \) forms a cycle \( \{e_{11}, e_{12}, e_{13}\} \), but this cycle does not make a \( S_3^{(t)} \)-ear as required in Def. 6. (d) \( C_4^{(t)} \) and \( C_5^{(t)} \) are formed after adding \( e_{14} \). The addition of edge \( e_{14} \) makes two \( S_3^{(t)} \)-ears: \( \{e_8, e_{13}, e_{14}\} \) and \( \{e_8, e_{11}, e_{12}, e_{14}\} \). By Def. 6, we set \( C_4^{(t)} \) by \( \{e_8, e_{13}, e_{14}\} \) because it has the shortest length, and consequently \( C_5^{(t)} \) is \( \{e_{11}, e_{12}\} \).](image)

Def. 6 serves as the basis to define the progress measure of algorithm \( A \). We consider the number of \( TC \)-blocks reduced by adding all edges in each ear. Given \( P^{(t)} \subseteq E^{(t)} \), we denote by \( B(V^{(t)}, P^{(t)}) \) the number of \( TC \)-blocks in the subgraph \( (V^{(t)}, P^{(t)}) \) of \( G^{(t)} = (V^{(t)}, E^{(t)}) \). As illustrated in the table on the right hand side of Fig. 6.2, the number of \( TC \)-blocks on topic \( t \) reduced by adding \( C_j^{(t)} \) is \(|C_j^{(t)}| - 1\), where \( 1 \leq j \leq z = 5 \). Formally, we generalize these observations as Claim 1, where we define \( S_0^{(t)} = C_0^{(t)} = \emptyset \). We prove it inductively in Appx. A.3.

**Claim 1.** The ear \( C_j^{(t)} \) reduces the number of \( TC \)-blocks on topic \( t \) in \( T \) in \( (V^{(t)}, S_{j-1}^{(t)}) \) by \(|C_j^{(t)}| - 1\), i.e.,

\[
B \left( V^{(t)}, S_{j-1}^{(t)} \right) - B \left( V^{(t)}, S_j^{(t)} \right) = |C_j^{(t)}| - 1, \forall C_j^{(t)} \text{ in } D^{(t)} = \left[ C_1^{(t)}, \ldots, C_z^{(t)} \right] . \quad (6.3)
\]

As each ear \( C_j^{(t)} \) is formed, edges in this ear account for the reduction in the number of \( TC \)-blocks. It is natural to distribute the reduction over all edges in the newly formed ear: Each edge \( e \in C_j^{(t)} \) contributes \( \frac{|C_j^{(t)}| - 1}{|C_j^{(t)}|} \) to the reduction of \( TC \)-blocks. The intuitive meaning of the ratio
\[
\frac{|C_j^{(t)}| - 1}{|C_j^{(t)}|} \text{ is the amortized contribution of } e \text{ toward } 2\text{-connectivity on topic } t. \text{ Furthermore, each edge in } E^{(t)} \text{ belongs to only one ear in the } E\text{-ear-decomposition on topic } t \in T. \text{ So, given } E, \text{ we can define the edge contribution of } e \in C_j^{(t)} \text{ on topic } t \text{ based on Def. 6 and Claim 1:}
\]

\[
\text{contrib}^{(t)}(e, E) = \frac{|C_j^{(t)}| - 1}{|C_j^{(t)}|}, \text{ where } e \in C_j^{(t)}. \tag{6.4}
\]

The overall edge contribution is defined as

\[
\text{contrib}(e, E) = \sum_i \text{contrib}^{(t)}(e, E). \tag{6.5}
\]

Furthermore, we define potential function \(\Phi(i, E)\) as the progress measure for \(A\) after adding the \(i\)-th edge of \(E\):

\[
\Phi(i, E) = B_{\text{start}} - \sum_{j=1}^{i} \text{contrib}(e_j, E), 0 \leq i \leq m. \tag{6.6}
\]

\(\Phi(0, E) = B_{\text{start}}, \Phi(i, E)\) monotonously decreases as \(i\) increases, and in the end \(\Phi(m, E) = B_{\text{end}}\) based on Claim 1.

### 6.4.2 Estimate of Edge Contribution

With the edge contribution and potential function defined in Eq. (6.4), (6.5) and (6.6), we could accurately tell the progress of algorithm \(A\) at each iteration – if we knew the output sequence of \(A\). Unfortunately, we do not have the output sequence until the algorithm returns, which makes the decision at each iteration of the algorithm more difficult. To circumvent this dilemma, we first find the bounds for the contribution of an edge currently considered for addition, with regard to all possible extensions of the edge sequence added up until the current iteration. Next, we use these bounds as a bookkeeping device to estimate each edge contribution.

According the \(i\)-th iteration of \(A\), we denote by \(P_i\) the set of edges added to the overlay and by \(P_i = (e_1, ..., e_i)\) the corresponding edge sequence. Given another edge sequence \(Q = (e_1', ..., e'|_Q)\) where \(e_j' \in (V \times V) \setminus P_i, P_i \circ Q\) means \(P\) concatenates with \(Q\), i.e., \(P_i \circ Q = (e_1, ..., e_i, e_1', ..., e'|_Q)\).
Let $\mathbb{R} = \mathbb{P}_i \circ \mathbb{Q}$, then $\mathbb{R}$ is an extension of $\mathbb{P}_i$. The extension set of $\mathbb{P}_i$ is

$$\mathcal{E}(\mathbb{P}_i) = \{ \mathbb{R} | \mathbb{R} \text{ is an extension of } \mathbb{P}_i \text{ and produces a } 2\text{TCO for } I(V, T, \text{Int}) \}.$$  \hspace{1cm} (6.7)

Note $\mathbb{E} \in \mathcal{E}(\mathbb{P}_i)$. Given some $\mathbb{R} \in \mathcal{E}(\mathbb{P}_i)$, we analyze the range of $\text{contrib}(e, \mathbb{R})$. Looking at topic $t \in T$, let $H^{(t)}_i = \left(V^{(t)}, P^{(t)}_i\right)$ be the current topic-induced subgraph on $t$ produced by $A$ after the $i$-th iteration, then $\text{contrib}^{(t)}(e, \mathbb{R})$ is the edge contribution of $e$ on topic $t \in T$ with regards to $\mathbb{R}$.

Let us consider an edge $e(v, w) \in (V^{(t)} \times V^{(t)}) \setminus P^{(t)}_i$:

- If there exists some TC-block on $t$ that contains both $v$ and $w$, then $e$ would form a trivial $P^{(t)}_i$-ear of length one. By Eq. (6.4), $\text{contrib}^{(t)}(e, \mathbb{R}) = 0$ (e.g., $e_{10}$ in Fig. 6.2).

- If no TC-block on $t$ contains both $v$ and $w$ (e.g., any edge except $e_{10}$ in Fig. 6.2), Eq. (6.4) implies that $\text{contrib}^{(t)}(e, \mathbb{R}) > 0$. By Def. 6, the length of the ear containing $e$ in the $\mathbb{R}^{(t)}$-ear-decomposition is at least 2, so $\text{contrib}^{(t)}(e, \mathbb{R}) \geq \frac{1}{2}$. Besides, $\text{contrib}^{(t)}(e, \mathbb{R})$ has an obvious upper bound of 1.

Thus,

$$\text{contrib}^{(t)}(e(v, w), \mathbb{R}) = \begin{cases} 
0, & \text{if some block in } \left(V^{(t)}, P^{(t)}_i\right) \text{ contains both } v \text{ and } w \\
\in \left[\frac{1}{2}, 1\right], & \text{otherwise}
\end{cases}, \mathbb{R} \in \mathcal{E}(\mathbb{P}_i).$$  \hspace{1cm} (6.8)

Claim 2 shows the contributions of an edge with regards to different sequences. The proof is in Appx. A.3.

**Claim 2.** Given $\mathbb{P}_i = \langle e_1, \ldots, e_i \rangle$, $\forall \mathbb{E}, \mathbb{R} \in \mathcal{E}(\mathbb{P}_i)$, $\text{contrib}(e_j, \mathbb{E}) \leq \text{contrib}(e_j, \mathbb{R}) \leq 2\text{contrib}(e_j, \mathbb{E})$, $1 \leq j \leq i$.

We now instantiate $A$ by $\text{GM}2$. Given the current edge set $P_i$, Line 5 of Alg. 28 defines the
estimate of $e$’s contribution on topic $t$:

$$\text{estimate}^{(t)}(e(v, w), P_i) = \begin{cases} 
0, & \text{if some block in } (V^{(t)}, P_i^{(t)}) \text{ contains both } v \text{ and } w \\
1, & \text{otherwise}
\end{cases} \quad (6.9)$$

The overall edge estimate is defined as

$$\text{estimate}(e, P_i) = \sum_t \text{estimate}^{(t)}(e, P_i). \quad (6.10)$$

Claim 3 gives us the bounds for an edge estimate in terms of the edge contribution. The proof is in Appx. A.3.

**Claim 3.** Given $P_i$ and a corresponding $\mathcal{P}_i$, $\forall R \in \mathcal{R}(\mathcal{P}_i)$, $\text{contrib}(e, \mathcal{R}) \leq \text{estimate}(e, P_i) \leq 2\text{contrib}(e, \mathcal{R})$.

### 6.4.3 Comparison Against the Unknown Optimum

We now complete our plan to compare the $2TCO$ produced by Alg. 28 to the optimal one.

**Lemma 36.** The approximation ratio of Alg. 28 is

$$O(U + \ln |V||T|),$$

where $U = \max\{|V^{(t)}|, t \in T\}$.

**Proof.** Given an instance $I(V, T, \text{Int})$, suppose $E$ is the output edge set of Alg. 28 and $\mathcal{E}$ is the GM12-edge-sequence, where $|E| = |\mathcal{E}| = m$. Let $E^*$ be the optimal solution where $|E^*| = m^*$.

Recall that $P_i$ is the edge set added to the overlay after the $i$-th iteration, and $\mathcal{P}_i = \langle e_1, \ldots, e_i \rangle$ is the corresponding edge sequence. Let $Q_i = E^* - P_i$ and $\mathcal{Q}_i$ be an edge sequence of $Q_i$ with an arbitrary order, i.e., $\mathcal{Q}_i = \langle e_1^*, \ldots, e_{|\mathcal{Q}_i|}^* \rangle$. Let $R_i = P_i \cup Q_i$ where $m_i = |R_i|$, and $\mathcal{P}_i \circ \mathcal{Q}_i = \langle e_1, \ldots, e_i, e_1^*, \ldots, e_{|\mathcal{Q}_i|}^* \rangle$, which means $\mathcal{P}_i$ concatenates $\mathcal{Q}_i$.

Since $E^* \subseteq R_i$, $\mathcal{R}_i$ would produce a $2TCO$, by Eq. (6.7),

$$\mathcal{R}_i \in \mathcal{R}(\mathcal{P}_i). \quad (6.11)$$
Adding $Q_i$ immediately after $P_i$ reduces the values of potential function from $\Phi(i, R_i)$ to $B_{end}$.

Since $|Q_i| \leq |E^*|$, there exists an edge $e' \in Q_i$ such that

$$\text{contrib}(e', R_i) \geq \frac{\Phi(i, R_i) - \Phi(m_i, R_i)}{|E^*|} = \frac{\Phi(i, R_i) - B_{end}}{m^*}. \quad (6.12)$$

Line 7 of Alg. 28 specifies the edge selection rule: Always choosing the edge with the highest estimate. At the $(i + 1)$-th iteration, Alg. 28 picks $e_{i+1}$ so that

$$\text{estimate}(e_{i+1}, P_i) \geq \text{estimate}(e', P_i) \quad \text{by greediness}$$

$$\geq \text{contrib}(e', R_i) \quad \text{by Claim 3}$$

$$\geq \frac{\Phi(i, R_i) - B_{end}}{m^*} \quad \text{by Eq. (6.12)} \quad (6.13)$$

According to Eq. (6.6), $\Phi(i, R_i) - 2\Phi(i, E) = -B_{start} + \sum_{j=1}^{i} (2\text{contrib}(e_j, E) - \text{contrib}(e_j, R_i))$.

With Claim 2 and Eq. (6.11), $\sum_{j=1}^{i} (2\text{contrib}(e_j, E) - \text{contrib}(e_j, R_i)) \geq 0$. So,

$$\Phi(i, R_i) \geq 2\Phi(i, E) - B_{start}. \quad (6.14)$$

Further,

$$\Phi(i, E) - \Phi(i + 1, E) = \text{contrib}(e_{i+1}, E) \geq 1/2 \cdot \text{estimate}(e_{i+1}, P_i) \quad \text{by Claim 3}$$

$$\geq \frac{\Phi(i, R_i) - B_{end}}{2m^*} \quad \text{by Eq. (6.13)}$$

$$\geq \frac{2\Phi(i, E) - (B_{start} + B_{end})}{2m^*} \quad \text{by Eq. (6.14)}$$

By Derivation A.3.1 in Appx. A.3,

$$\left(\Phi(i + 1, E) - \bar{B}\right) \leq \left(1 - 1/m^*\right) \left(\Phi(i, E) - \bar{B}\right), \quad \text{where } \bar{B} = \frac{B_{start} + B_{end}}{2} \quad (6.15)$$

Eq. (6.15) shows the progression of the potential function value within successive iterations in $GM^2$ as compared to the optimal solution. Based on Eq. (6.15), we derive the bound on the number of iterations of Alg. 28 (i.e., the number of edges in $E$) relative to $m^*$. We take $\left(\Phi(i, E) - \bar{B}\right)$ as
a function of $i$, and it decreases as $GM^2$ adds an edge at each iteration. Initially, $\Phi(0,E) - \bar{B} = \frac{B_{\text{start}} - B_{\text{end}}}{2} > 0$, and finally, $\Phi(m,E) - \bar{B} = -\frac{B_{\text{start}} - B_{\text{end}}}{2} < 0$. So at some iteration $\lambda_0$, the function turns from positive to negative. We have a sequence of the function values as:

$$\left\langle \left( \Phi(1,E) - \bar{B} \right), \ldots, \left( \Phi(\lambda_0,E) - \bar{B} \right), \left( \Phi(\lambda_0 + 1,E) - \bar{B} \right), \ldots, \left( \Phi(m,E) - \bar{B} \right) \right\rangle > 0$$

denote the number of such elements by $\lambda_0$. \hfill (6.16)

Further, by Derivation A.3.2 in Appx. A.3,

$$\lambda_0 \leq m^* \cdot O(\ln B_{\text{start}}), \hfill (6.17)$$

$$\lambda_1 \leq m^* \cdot O(U + \ln B_{\text{start}}), \text{ where } U = \max\{|V(t)|, t \in T\}. \hfill (6.18)$$

Therefore, $m = \lambda_0 + \lambda_1 = m^* \cdot O(U + \ln B_{\text{start}})$. \hfill $\square$

### 6.5 HararyPT Algorithm to Build $kTCO$

**Alg. 29 Harary-Per-Topic for $kTCO$**

HararyPT($I(V,T,\text{Int}), k$)

**Input:** $I(V,T,\text{Int}), k$

**Output:** $kTCO(V,T,\text{Int}, E_{\text{HararyPT}})$

1: $V \leftarrow$ get an arbitrary sequence for $V$
2: for all $t \in T$ do
3: $E(t) \leftarrow \text{buildHarary}(k, V(t))$
4: $E_{\text{HararyPT}} \leftarrow \bigcup_{t \in T} E(t)$
5: return $kTCO(V,T,\text{Int}, E_{\text{HararyPT}})$

For the MinAvg-$kTCO$ problem, we design the Harary-Per-Topic Algorithm (HararyPT) to build the $kTCO$, as specified in Alg. 29.

HararyPT stems from graph theory about vertex connectivity and Harary graphs. Function $\text{buildHarary}(k, V(t))$ (Line 3 of Alg. 29) represents the standard procedure to construct the $k$-
connected Harary graph for a given sequence of nodes \( \forall(t) \). \texttt{HararyPT} invokes \texttt{buildHarary()} for each topic \( t \in T \) (Lines 2-3). Since the Harary graph \( H_{k,n} \) is known to be \( k \)-connected with the minimum number of edges \( \lceil kn/2 \rceil \) [93], we can derive Lemma 37.

**Lemma 37.** Alg. 29 produces a \( kTCO \) with time complexity \( O \left( k \cdot \sum_{t \in T} |V(t)| \right) = O(k|V||T|) \).

The most straightforward approach is perhaps to build the sub-overlay (i.e., Harary graph) independently for each topic. One serious drawback is that the probability of any two nodes sharing the edge in more than one sub-overlay is small [65]. Thus, the output overlay has an unnecessarily high average node degree. We also evaluate this naive approach in §6.6. In order to promote edge sharing across different sub-overlays, we first obtain a node sequence for all the nodes in Line 1 of Alg. 29. The \texttt{HararyPT} algorithm adopts the same linear ordering for all Harary constructions across all topics. By sharing the determined node sequence, these Harary graphs are likely to converge a lot of edges, especially when the workloads are highly correlated. As a consequence, the output \( kTCO \) tends to have a low node degree. Although we do not have an approximation ratio for the \texttt{HararyPT} algorithm, we can assume that subscriptions are highly correlated in typical pub/sub workloads. More specifically, the study of representative pub/sub workloads used in actual applications observes the “Pareto 80-20” rule: Most nodes subscribe to a relatively small number of topics [36]. Besides, many pub/sub workloads are modelled by a power law distribution in both topic popularity and subscription size per node [64, 90]. Our experimental findings in §6.6 further demonstrate that the \texttt{HararyPT} algorithm significantly reduces the number of edges while offering a high degree of topic-connectivity for typical pub/sub workloads in practice.

### 6.6 Evaluation

We implemented \texttt{GM2}, \texttt{HararyPT}, and other auxiliary algorithms in Java. We use \texttt{GM} as a baseline, because it produces a \( 1TCO \) with the lowest average node degree among all known polynomial-time algorithms [35]. We also develop the \texttt{Cycle-Per-Topic} algorithm (\texttt{CyclePT}) that mimics the common practice of building a separate overlay for each topic independently (usually a tree but we use a cycle that has the same average node degree and achieves \( 2 \)-topic-connectivity). By \texttt{CyclePT}, all nodes interested in the same topic form a cycle, and cycles for different topics are
merged into a single 2TCO. Note that CyclePT is fundamentally different from HararyPT, because CyclePT does not exploit the correlations in the workload and often ends up with an overlay with unnecessarily high node degrees. We evaluated the HararyPT algorithm with different parameters, i.e., $k \in [2, 14]$. We only plot the representative results for $k = 2, 4, 6, 8, 10$, but we report additional results. We mainly compare the average node degrees in the output overlays produced by different algorithms. For a specific algorithm $A$, we denote by $d_A$ the average node degree produced by $A$. We denote by $T(v)$ the topic set which node $v$ subscribes to, and we call $|T(v)|$ the subscription size of node $v$.

Our inputs have the following ranges: $|V| \in [100, 1000]$, $|T| \in [100, 1000]$, and $|T(v)| \in [10, 100]$, where each node has a fixed the subscription size. Each topic $t \in T$ is associated with probability $p(t)$, $\sum_{t \in T} p(t) = 1$, and each node $v \in V$ subscribes to $t$ with a probability $p(t)$ until $|T(v)|$ is reached. The value of $p(t)$ is distributed according to either an exponential, a Zipfian (with $\alpha = 2.0$), or a uniform distribution, which we call Expo, Zipf, or Unif for short. According to [36], these distributions are representative of actual workloads used in industrial pub/sub systems today. Expo is used by stock-market monitoring engines for the study of stock popularity in the New York Stock Exchange [90], and Zipf faithfully describes the feed popularity distribution in RSS feeds [64].

### 6.6.1 Impact of Number of Nodes

Fig. 6.3 depicts the comparison among GM2, HararyPT, GM, and CyclePT with regards to the number of nodes under different distributions. We set $|T| = 200$, $|T(v)| = 30$, and $|V| \in [100, 1000]$. 

![Figure 6.3: GM2 vs. HararyPT vs. GM vs. CyclePT wrt. $|V|$](image)
We look at $\mathcal{G}M2$ in Fig. 6.3. First, $\bar{d}_{\mathcal{G}M2}$ and $\bar{d}_{\mathcal{G}M}$ are quite close under all conditions. More specifically, $\bar{d}_{\mathcal{G}M2}$ is smaller than $1.66 \cdot \bar{d}_{\mathcal{G}M}$ on average, across all three distributions. $\mathcal{G}M2$ is capable of constructing a $2TCO$ with a marginal increase in the average node degree as compared to $1TCO$ produced by $\mathcal{G}M$. Second, $\bar{d}_{\text{CyclePT}}$ is roughly equal to twice the subscription size, which is about 5-times higher than $\bar{d}_{\mathcal{G}M2}$ on average. Third, $\bar{d}_{\text{CyclePT}}$ tends to increase with the number of nodes, while both $\bar{d}_{\mathcal{G}M2}$ and $\bar{d}_{\mathcal{G}M}$ decrease as the number of nodes scales up. The decrease of $\bar{d}_{\mathcal{G}M2}$ and $\bar{d}_{\mathcal{G}M}$ lies in the fact that increasing the number of nodes leads to higher chances for both $\mathcal{G}M2$ and $\mathcal{G}M$ to find neighbors with more interest overlap, thus reducing overall number of neighbors needed. All these results demonstrate the scalability of $\mathcal{G}M2$ with regards to the number of nodes.

We look at $\text{HararyPT}$ in Fig. 6.3. First, $\mathcal{G}M2$ outperforms $\text{HararyPT}$ both theoretically and empirically for constructing $2TCOs$. However, $\text{HararyPT}$ allows the overlay to have more reliability commitments over $2TCO$ by setting $k > 2$. Second, $\bar{d}_{\text{HararyPT}}$ increases as we increase $k$ under all three distributions, and the $\text{HararyPT}$ algorithm tends to output better average node degrees in more skewed distributions. Third, $\text{HararyPT}$ significantly reduces unnecessary redundancy as compared to $\text{CyclePT}$. With fewer edges than $2TCOs$ produced by $\text{CyclePT}$, $\text{HararyPT}$ can achieve $12 TCO$ under Expo, $7 TCO$ under Zipf, and $5 TCO$ under Unif.

### 6.6.2 Impact of Number of Topics

Fig. 6.4 depicts how $\mathcal{G}M2$ and $\text{HararyPT}$ perform as compared to $\mathcal{G}M$ and $\text{CyclePT}$ when the number of topics varies under different topic popularities. We set $|V| = 800$, $|T(v)| = 30$, and $|T| \in [100, 1000]$.
Chapter 6. Constructing Fault-Tolerant Overlays for Topic-based Pub/Sub

Referring to the \( GM^2 \) algorithm in Fig. 6.4, it can be seen that the average node degrees of both \( GM^2 \) and \( GM \) increase with the number of topics. Note that increasing the number of topics leads to reduced correlation, i.e., the probability of having two nodes interested in the same topic drops as the number of topics increases, and with reduced correlation the edge contribution at each iteration of \( GM^2 \) (and \( GM \)) tends to be lower. This reduction in the correlation is more pronounced for Unif as compared to skewed distributions, like Expo or Zipf. Yet, the increase in \( d_{GM^2} \) is slow paced. In particular, \( d_{GM^2} \) is no more than \( 1.66 \cdot d_{GM} \) on average. Moreover, the gap between \( d_{CyclePT} \) and \( d_{GM^2} \) remains significant: \( d_{CyclePT} - d_{GM^2} \geq 43.1 \) on average, across all experiment instances under various distributions. Besides, \( GM^2 \) exhibits more advantages over \( CyclePT \) for more correlated workloads.

Referring to the HararyPT algorithm in Fig. 6.4, first, the average node degree of HararyPT increases with the number of topics due to the reduction in the correlation. This reduced correlation has an considerable effect under Unif: To achieve \( 10 \) TCO, \( d_{HararyPT} \) is 54.22 when \( |T| = 100 \) and 222.84 when \( |T| = 1000 \) under Unif. As the number of topics increases, the input instances (especially those under Unif) are deviating from our assumption about the high correlation embedded in pub/sub workloads. As a result, HararyPT tends to lose its advantages of aligning the nodes. Second, HararyPT always outputs a 2TCO with fewer edges than that produced by CyclePT.

6.6.3 Impact of Subscription Size

Fig. 6.5 depicts the impact of the subscription size on the \( GM^2 \), HararyPT, \( GM \), and \( CyclePT \) algorithms. We fix \( |V| = 800 \), \( |T| = 200 \), and \( |T(v)| \in [10, 100] \).

![Figure 6.5: \( GM^2 \) vs. HararyPT vs. \( GM \) vs. \( CyclePT \) wrt. the subscription size](image)
CHAPTER 6. CONSTRUCTING FAULT-TOLERANT OVERLAYS FOR TOPIC-BASED PUB/SUB 114

We focus on the GM2 algorithm in Fig. 6.5. First, under all three distributions, GM2 and GM produce quite close overlays in terms of the average node degrees. As the subscription size increases, both \( \bar{d}_{GM2} \) and \( \bar{d}_{GM} \) decrease, and the difference \( (\bar{d}_{GM2} - \bar{d}_{GM}) \) shrinks. This decrease occurs because the growth of subscription size causes increased correlation across the nodes. Upon bigger correlation, an edge addition to the overlay has on average a higher contribution toward 1TCO (or 2TCO) because the nodes share more comment interests. Therefore, a smaller number of edges are needed until the overlay becomes topic-(bi)connected. Second, the average node degree of CyclePT increases linearly with the subscription size, \( \bar{d}_{CyclePT} \) is roughly equal to twice the subscription size of the workload. The above two facts render GM2 increasingly important when the subscription size scales up.

We focus on the HararyPT algorithm in Fig. 6.5. First, the average node degree increases as the subscription size varies from 10 to 50 under Zipf (or from 10 to 20 under Unif), since each node has more topics to cover. However, when the subscription size exceeds some threshold (e.g., around 50 under Zipf and around 20 under Unif), the average node degrees start to decrease. We can explain this phenomenon by the trend that the correlation becomes increasingly dominant as the subscription size increases. Second, HararyPT (for all \( k \) values) outperforms CyclePT significantly. For example, when \(|T(v)| = 100\), HararyPT can produce 12 TCO with \( \bar{d}_{HararyPT} = 60.20 \) on average, across all distributions, whereas CyclePT outputs only 2TCO with \( \bar{d}_{CyclePT} = 177.23 \).

6.6.4 Topic Diameters of Overlay

![Graphs showing topic diameters under different distributions](image)

We also compare another important metric in the overlays produced by different algorithms, namely the topic diameter. Overlay diameters impact many performance factors for efficient routing.
in pub/sub, e.g., message latency. Given $2TCO(V,T,Int,E)$, the topic diameter for $t \in T$ is 
\[ \text{diam}^{(t)} = \text{diam}(G^{(t)}), \]
where $\text{diam}(G^{(t)})$ is the maximum shortest distance between any two nodes in $G^{(t)} = (V^{(t)}, E^{(t)})$. We denote the maximum and average topic diameter across all topics as $\text{Diam}$ and $\overline{\text{diam}}$, respectively. Fig. 6.6 shows that $GM2$ significantly outperforms $GM$ in terms of both $\text{Diam}$ and $\overline{\text{diam}}$: $\text{Diam}_{GM2}$ is $0.45 \cdot \text{Diam}_{GM}$, and $\overline{\text{diam}}_{GM2}$ is $0.51 \cdot \overline{\text{diam}}_{GM}$, on average across all three distributions. Besides, the gap grows as the input instances scale up from 100 nodes to 1000: as $|V| = 1000$, $\text{Diam}_{GM} - \text{Diam}_{GM2}$ is 16.75 under Expo, $\text{Diam}_{GM} - \text{Diam}_{GM2}$ is 10.5 under Zipf, and $\text{Diam}_{GM} - \text{Diam}_{GM2}$ is 17.5 under Unif, respectively.

### 6.7 Conclusions

We study a new family of optimization problems $\text{MinAvg-}k\text{TCO}$ that constructs reliable overlay networks for topic-based pub/sub. We present a polynomial-time overlay design algorithm, $GM2$, which approximates $\text{MinAvg-}2\text{TCO}$ within a proven bound. We provide a novel proof for the approximation ratio of $GM2$, which is almost tight since no logarithmic approximation polynomial-time algorithm can exist for the $\text{MinAvg-}2\text{TCO}$ problem unless $P=NP$. Furthermore, we design a heuristic algorithm for the $\text{MinAvg-}k\text{TCO}$ problem, namely the HararyPT algorithm, especially for highly correlated pub/sub workloads.

Our experimental results validate our formal analysis for the $GM2$ algorithm: The average node degree of the $2TCO$ produced by $GM2$ is about 1.65 times that of the $1TCO$ generated by the baseline algorithm, $GM$. We also show the practical scalability of HararyPT for representative pub/sub workloads in terms of the number of nodes, the number of topics, and the subscription size. In sum, our designed algorithms are capable of achieving more reliable topic-connectivity by compromising the average node degrees insignificantly.
Chapter 7

ElastO: Efficient Maintenance of Pub/Sub TCO Under Churn

7.1 Introduction

We present ElastO for maintaining high-quality message dissemination overlays for topic-based pub/sub in presence of churn. The system properties include:

I. **Topic-connected overlay (TCO):** informally, TCO means that all nodes interested in the same topic are organized in a connected dissemination sub-overlay [35]. Topic-connectivity ensures that nodes not interested in a topic never need to contribute to disseminating information on that topic. Pub/sub message routing atop such overlays saves bandwidth and computational resources otherwise wasted on forwarding messages of no interest to the node. TCO also results in simpler routing protocols and smaller forwarding tables (see Chapter 1).

II. **Low fan-out:** which ensures that both maximum and average degrees are small. It is imperative for a pub/sub overlay to have low fan-out because it costs a lot of resources to maintain adjacent links for a high-degree node (e.g., monitoring the links and the neighbors [35, 69]). Furthermore, for a typical pub/sub system, each link would accommodate a number of protocols, service components, message queues, etc. While overlay designs for different applications might be principally
different, they all strive to maintain bounded node degrees, e.g., DHTs [63] and survivable network design [58].

III. Churn resistance: it is important to restore TCO upon node departure as soon as possible by pre-computing the knowledge before the churn occurs. In particular, the existing algorithms for building a high-quality TCO from scratch are known to be computationally expensive. It is imperative to avoid carrying this runtime cost into a dynamic solution.

IV. Balanced load of computation and communication due to overlay maintenance across all nodes in the overlay. Both restoring topic-connectivity and monitoring the overlay during the normal operation incurs communication as well as computation overhead. It is important to spread this overhead fairly and evenly across the nodes in a distributed system.

To the best of our knowledge, none of the approaches in the state-of-the-art manage to satisfy all of the above mentioned properties at the same time. We can classify existing works into two categories (see Table 1.1 in Chapter 2): (a) centralized algorithms that statically construct a provably low-degree TCO from scratch and (b) decentralized protocols that strive to dynamically maintain low degrees (and in many cases, topic-connected) in a best-effort fashion. Unfortunately, the former are known to have high runtime cost, which makes them unsuitable as a dynamic solution. Meanwhile, the latter produce significantly higher node degrees compared to the former – e.g., the node degrees produced by PolderCast [84] grow almost linearly with subscription size under typical pub/sub workloads.

In contrast, we propose Elasto, a hybrid approach between centralized algorithms and decentralized protocols, which combines the strengths from both. We summarize the contributions in this work as follows:

A. The Elasto approach is the first system with complete architecture and protocol design (see §7.2) such that (1) TCO is guaranteed to be quickly restored as the most recent decentralized protocols, and (2) the node degrees stay provably low under incremental node churn.

B. We propose TCO repairing algorithms with shadow sets in §7.3. The main obstacle of dynamic maintenance is to handle the departure of a node central to TCO. In this situation, additional edges need to be created in order to mend the overlay and restore topic-connectivity. If the system considers the entire set of potential edges that can be added, its running time will be problematic. If
the system only considers a small subset of edges for addition, those edges may turn out suboptimal thereby negatively affecting the quality of the overlay. To overcome these challenges, ElastO calculates a shadow set of nodes upon each churn event, and then computes edges among the shadow set for restoring TCO.

C. In §7.4, we show how ElastO efficiently computes the shadow set upon churn. In particular, when node \( v' \) becomes unavailable, the shadow set is calculated as the union of \( v' \)'s neighbors and the backup set for \( v' \). ElastO continuously maintains a backup set for each node – we initialize the backup set when ElastO boots up and incrementally update it as needed. We propose efficient algorithms to locally compute small-size backup sets that support scalable and balanced performance.

D. We conduct comprehensive experiments in §7.5 under a variety of pub/sub workloads. We use both synthetically generated workloads and large-scale real-world workloads (i.e., Facebook [94] and Twitter [56]) with up to 10K nodes, 10K topics and 5K subscriptions per-node. We evaluate the properties of ElastO through a month-long period churn trace from Google cluster data [5].

7.2 The ElastO Approach

7.2.1 Scope of ElastO

We lay out the practical application requirements that guide our system design:

- The system is built from many inexpensive commodity components, and a small but non-negligible number of server and network components can fail at any time. The system must constantly monitor itself, detect, tolerate, and recover promptly from failures on a routine basis [44, 57].
- The system needs to add nodes from time to time. For example, previously failed nodes can rejoin the network after being fixed, or the system may require new nodes to accommodate the increasing load [57].
- The target environment is large-scale data center with moderate churn [57]. Most of the churn is simple, i.e., only one node joins, leaves, or fails at a time. Concurrent churn events involving multiple nodes occur infrequently. Second, the intervals between successive churn rounds are in the order of tens of minutes, depending on the size of the cluster [5].

Based on these observations, our design principles are:
1. The system allows any node to join or leave (gracefully or by crashing) at any time. We focus on node crashes in the failure model and do not consider link failures or Byzantine failures.

2. The system supports correctness for concurrent churn events involving up to $L$ nodes simultaneously ($L$ is a configuration parameter). Without loss of generality, concurrent churn results in a sequence of simple churn events produced by the failure detector. For each simply churn event only one node joins, leaves, or fails. The next churn event may occur before the handling of the previous event is completed. Even in this case, topic-connectivity is eventually restored in the overlay. However, such concurrent churn may result in a small number of redundant communication messages and a number of redundant links created in the overlay. This is justified by the application requirements listed above and furthermore, evaluated in §7.5.

3. The system requires careful design to handle churn events of node leaving. When a node $w$ departs from a TCO, the departure may render the overlay disconnected for many topics. We repair the overlay by creating additional links among a set of shadow nodes, which in this case is the union of $w$’s neighbors and a set of nodes specially designated as backup nodes for $w$. The system keeps a set of backups for each node. If the backup set is large, the repair becomes similar to centralized static algorithms. If the backup set is of small cardinality, the system leans more towards decentralized protocols. A principal contribution of our design is in maintaining the balance when it comes to the backup set size: we show that it is possible to keep the size moderate, so that the computation is cheap as in decentralized protocols, yet sufficient for the overlay quality to approach that produced by centralized algorithms. We discuss building the backup set and related trade-offs in §7.3.

4. The crux of our design is in the even distribution of (a) responsibility to repair TCO, (b) backup nodes, and (c) load of maintaining meta-information in a decentralized fashion.

7.2.2 Overview

ElastO resides between the pub/sub routing protocols layer and the network transport protocols layer. ElastO needs to have the following functionalities: (a) membership management, (b) failure detection, (c) TCO recovery, and (d) distributing the responsibilities for TCO recovery across the nodes in a balanced fashion.
## Alg. 30 Basic data structures maintained by each node $v \in V$

<table>
<thead>
<tr>
<th>Node: encapsulation of node descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>o id: node identifier</td>
</tr>
<tr>
<td>o topics: subscribed topics</td>
</tr>
<tr>
<td>o neighbIds: nodeIds of TCO neighbors</td>
</tr>
<tr>
<td>CHURNEvt: simple churn event with a node</td>
</tr>
<tr>
<td>o type: churn type, JOIN or LEAVE</td>
</tr>
<tr>
<td>o node: NODE object for the churn node</td>
</tr>
</tbody>
</table>

| Message: abstract class to encapsulate messages transmitted between two nodes |
| o source: NODE of message source |
| o mclass: message class. Each class extends abstract Message to a concrete subclass. |

Alg. 30 specifies basic data structures in ElastO.

1. **Node** defines the **node descriptor**, which includes the identities of nodes along with their metadata, including topic interests and TCO neighbors. We use **Node** as elementary entry for many local container variables maintained at each node, e.g., $v.N$, $v.D$, and $v.L$ in Alg. 32.

2. **ChurnEvt** represents the class of simple churn events with only one node. Each ChurnEvt object needs to instantiate the churn node descriptor in ChurnEvt.node and the churn type in ChurnEvt.type, e.g., JOIN or LEAVE.

3. **Message** is abstraction for messages transmitted between nodes.

Alg. 31 extends base Message to concrete subclasses for various purposes of the protocols.

### Alg. 31 Extended subclasses of Message

<table>
<thead>
<tr>
<th>$\triangleright$ RepairQ: request to repair TCO</th>
<th>$\triangleright$ NeighbQ: request about TCO neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>o chnevt: the associated ChurnEvt</td>
<td>o nodeId: the nodeId to request</td>
</tr>
<tr>
<td>$\triangleright$ RepairS: TCO repair response</td>
<td>$\triangleright$ NeighbS: neighbor update response</td>
</tr>
<tr>
<td>$\triangleright$ AddneiQ: request to add new neighbor</td>
<td></td>
</tr>
<tr>
<td>o newnei: new TCO neighbor</td>
<td>$\triangleright$ LvQ/LvS: local view request/response</td>
</tr>
<tr>
<td>$\triangleright$ AddneiS: new neighbor response</td>
<td></td>
</tr>
</tbody>
</table>

Alg. 32 presents key local variables at each node and the framework of ElastO protocols. Node $v \in V$ maintains a set of node descriptors, $v.N$, as the overlay neighbors. The $v.N$ over all $v \in V$ define the ElastO pub/sub overlay. We denote by $v.D$ the updated output list of the failure detector at node $v \in V$, and node $v$ can query the local failure detector by checking whether $w \in v.D$ at any time. Each node $v \in V$ also keeps a local view $v.L$ about other nodes in the network. The
view includes node descriptions for \( L \) successors and predecessors of \( v \) in a circular space, the list of neighbors (node descriptors) for predecessors and most importantly, the list of backup nodes.

We implement a peer sampling service to build local view for each node. The service uses a data exchange protocol similar to those in [50, 51]. Yet, the information included for each node in the view is much broader and our local view selection mechanism is unique (see §7.4), especially because we rely on the local view for repairing TCO (see §7.3).
One core design feature for ElastO is the scalability under incremental churn. This requires ElastO to evenly distribute churn handling overhead over all nodes in the network. We design a “primary-backup” strategy to balance these loads based on an identifier circle. ElastO randomly assigns each node an unique m-bit node identifier (nodeId), which represents its position in an identifier circle module $2^m$. The identifier circle places a linear ordering over all nodeIds in a circular space, and each ElastO node has predecessors and successors on the identifier circle. We can generate nodeIds by using consistent hashing on nodes’ public keys or IP addresses.

The identifier circle allows ElastO to delegate the responsibility of handling each simple churn event to a specific node, depending on the location of the churn node. Once a churn event occurs (e.g., node $v'$ joins or leaves), the immediate successor to the churn node $v'$, say node $v$, is typically deemed the churn coordinator, i.e., node $v$ is responsible for repairing the TCO upon this churn event at $v'$. Coordinator $v$ reacts to churn event by invoking TCO repairing algorithms based on its local view $v.L$. In Fig. 7.1, node $v_{10}$ failed, and its immediate successor $v_{11}$ captured this churn event and worked as the churn coordinator to recover TCO.

To fulfill these functionalities, ElastO has three modules (see Fig. 7.2):

1. **MEMBERSERV** maintains the identifier circle and keeps the local view updated continuously at each node. It is also responsible for detecting failures.
2. TcoUpgrader repairs the pub/sub TCO upon churn. It implements TCO design algorithms for the churn coordinator.

3. TcoCreator constructs TCO from scratch initially (or periodically once in a while). TcoCreator can produce a close-to-optimal TCO, but it admits centralized operations, requires the global knowledge, and imposes expensive computation. TcoCreator is invoked infrequently.

We introduce each architectural module in §7.2.4, §7.2.3, and §7.2.5, respectively. Then §7.3 discusses the overlay repairing algorithms in TcoUpgrader, and §7.4 explains the local view selection mechanism in MembershipService.

### 7.2.3 The Membership Service Module

To maintain the local view, MembershipService extends a gossip-based peer sampling service [50, 51]. Each node $v$ periodically exchanges its $v.L$ with another node $w$, chosen uniformly among the existing members in $v.L$. Node $v$, then, merges its current $v.L$ with $w.L$ as a fresh list of local view candidates. Next, $v$ chooses a number of members among the candidates according to some selection criteria and updates $v.L$. The same process takes place at node $w$ (and other nodes). Please see the algorithm and implementation in §7.4.

MembershipService also enables ElastO nodes to form the global identifier circle. For each node $v \in V$, $2 \cdot L$ leaf entries of $v.L$ are always dedicated for maintaining the immediate neighbors on the identifier circle. Each node $v \in V$ selects $L$ nodes with the closest nodeIds to its own, in the two directions, among the current $v.L$, as its predecessors and successors. Although initially the
predecessors and successors may not reflect the global circular ordering, peer sampling guarantees that the circle topology rapidly converges and is constantly maintained [51, 76].

Further, MEMBERSERV is responsible for failure detection. With the support of failure detector, each node can \textit{locally} determine if any other node in the system is up or down. Besides, failure detector is also used to avoid attempts to communicate with unreachable nodes during various operations. Our MEMBERSERV module can rely on any standard failure detector [42]. For the rest of this paper, we assume that the failure detector is available as a black box.

### 7.2.4 The Lightweight TCOUPGRADER Module

The TCOUPGRADER module is responsible for maintaining the pub/sub TCO neighborhood for each node. Upon churn, Elasto invokes \texttt{repairTcoOnChurn()} of the TCOUPGRADER module, as shown in Alg. 33. Alg. 34 presents auxiliary functions for Alg. 33.

In \texttt{repairTcoOnChurn()}, node $v$ first checks whether it is the coordinator for $chnev$. If it is the churn coordinator, node $v$ prepares all the knowledge required to handle this churn, and then repairs the overlay until TCO is retained – at each attempt, it invokes TCO construction algorithm with its current local view, and then builds new overlay connections accordingly.

If $v$ is not the churn coordinator, node $v$ keeps trying to delegate this churn handling to another node – at each attempt, it finds a node $w$ whose id is closest to the churn coordinator in its local view, sends a TCO repairing request message to $w$, and then wait a certain time for $w$’s response.

Thanks to the nodeId circle supported by MEMBERSERV, the churn coordinators (and thus the computation overhead of TCO maintenance and upgrade) are uniformly distributed across all nodes in Elasto. Each node is responsible for a small region on the nodeId circle – it only repairs the TCO when its predecessors join or leave. Another desirable property is that, Elasto only impacts a small subset of nodes when repairing the overlay, in which the crux of \texttt{computeTcoOnChurn()} lies. (See algorithm design in §7.3)
Alg. 33 Local Protocol for Repairing TCO at Node $v \in V$

$\triangleright$ v.repairTcoOnChurn(chnevt)

1: $c \leftarrow \text{getCoordinator}(\text{chnevt})$
2: $\text{done} \leftarrow \text{false}$
3: if $c = v.\text{self}$ then
4: if $\text{readyToHandle}(\text{chnevt})$ is false then
5: prepareLV(\text{chnevt})
6: repeat
7: $E_{\text{new}} \leftarrow \text{computeTcoOnChurn}(\text{chnevt})$
8: for all $e(x, y) \in E_{\text{new}}$ do
9: send($x$, new $\text{AddneiQ}(v.\text{self}, y)$)
10: send($y$, new $\text{AddneiQ}(v.\text{self}, x)$)
11: $X \leftarrow \{x | \exists y \text{ s.t. } (x, y) \in E_{\text{new}}\}$
12: wait until $[(\forall x \in X : \text{received AddneiS from } x)$
13: OR $(X \land v.\mathcal{D} \neq \emptyset)]$
14: if $\forall x \in X : \text{received matched AddneiS}$ then
15: $\text{done} \leftarrow \text{true}$
16: until $\text{done}$ is true
17: else
18: $c \leftarrow \text{getCoordinator}(\text{chnevt})$
19: send($c$, new $\text{RepairS}(v.\text{self}, \text{chnevt})$)
20: wait until [received matched $\text{RepairS}$ from $c$ OR $c \in v.\mathcal{D}$]
21: if received $\text{RepairS}$ from $c$ then
22: $\text{done} \leftarrow \text{true}$
23: until $\text{done}$ is true
Algorithm 34 Local Functions for Repairing TCO at Node $v \in V$

**v.getCoordinator(chnevt)**

1: return node descriptor $w \in (v \setminus v.D)$

\[ (w.id - chnevt.node.id) > 0 \text{ and } (w.id - chnevt.node.id) \text{ is minimum} \]

**v.readyToHandle(chnevt)**

1: $l \leftarrow$ get l s.t. $v$..pred[$l$] = chnevt..node

2: for all $i \in v$..pred[$l$].neighbIds do

3: if $\exists n \in v$..neighb[$l$] s.t. $n.id = i$ then

4: return false

5: $T_B = \bigcup_{u \in v$..backup[$l$]} u$..topics

6: if chnevt..node..topics $\subseteq T_B$ then

7: return true

8: else

9: return false

**v.prepareLV(chnevt)**

1: $w \leftarrow$ chnevt..node

2: $l \leftarrow$ get l s.t. $v$..pred[$l$] = $w$

3: $v$..neighb[$l$] $\leftarrow$ $\emptyset$

4: while $v$..neighb[$l$] = $\emptyset$ do

5: msgReq $\leftarrow$ new NeighbQ($v$..self, $w$..id)

6: send($v$..succ[0], msgReq)

7: wait until [(received matched msgRes) OR ($v$..succ[0] $\in v$..D)]

8: if received matched msgRes then

9: $v$..neighb[$l$] $\leftarrow$ msgRes..neighbs

10: wait until $v$..backup[$l$] is built, i.e.,

\[ w$..topics \subseteq \bigcup_{u \in v$..backup[$l$]} u$..topics \]

7.2.5 The TCOCREATOR Module

The TCOCREATOR module is a conceptually centralized entity to initialize (or to reset) the pub/sub TCO infrastructure at each node. It can apply any existing static algorithms or decentralized protocols (see §2.1).
In the implementation of TcoCreator used in this paper, we employ the LowODA algorithm [69] for initializing the base TCO from scratch. We implement LowODA with an indexing data structure that improves the runtime complexity from $O(|V|^4|T|)$ to $O(|V|^2|T|)$ (see Chapter 3). We also use LowODA as a building block in the TcoUpgrader module (see §7.3).

### 7.2.6 Handling Corner Cases

To handle other corner cases, we adopt a relaxed consistency model that facilitates efficient churn handling:

1. It is possible that the coordinator $v$ is not ready for handling the incoming churn event at node $v'$, i.e., the peer sampling procedure has not built up necessary knowledge for $v'. \mathcal{L}$. However, this situation rarely happens, only when $v$ is a newly joined node, the number of tolerated concurrent churn events $L > 1$, and $v'$ leaves before $v$ knows about the neighbors of $v'$. In that case, $v$ can retrieve the necessary information from its successors.

2. If some other node $w$ (rather than $v'$’s immediate successor) detects the churn event about $v'$, node $w$ would create a ChurnEvt object, wrap the object into a message, and then forward the message to $v'$’s closest successor node chosen from $w. \mathcal{L}$. Because $w$’s knowledge about the successor of $v'$ might be imprecise, it is possible that the node to which $w$ sends a message is not the actual successor, in which case it needs to forward the message further. The constructed ChurnEvt event will be propagated recursively until it reaches node $v$, which invokes the actual operations for churn handling. This forwarding process resembles the routing for locating a key in DHT [80], but we do not use an existing DHT because our local view has different requirements as compared to the DHT routing table (see §7.4).

3. We do not guarantee consistency for the local view among all $v \in V$. Suppose both $v$ and $w$ keep $u$’s node descriptor in its local view, denoted as $u_v$ and $u_w$, respectively, then it is possible that $u_v.\text{neighIds} \neq u_w.\text{neighIds}$. Either $u_v.\text{neighIds}$ or $u_w.\text{neighIds}$ does not have to reflect node $u$’s real TCO neighbor set. However, we do guarantee that $u_v.\text{neighIds}$ (or $u_w.\text{neighIds}$) is always a subset $u$’s TCO neighbors, i.e., $u_v.\text{neighIds}$ might be incomplete, but it never contains a member that is not $u$’s neighbor.

4. We do not impose consensus about the churn coordinator selection among all nodes upon
churn. However, we ensure that at least one node takes charge of each churn. It is possible that more than one node is assigned as the churn coordinator at the same time, but it does not impact the topic-connectivity of the overlay – only introducing some redundant links. We can add some garbage collection mechanism to improve the optimality of the TCO in a lazy manner.

(5) Node \( v \) may receive a number of TCO repair requests from multiple nodes that detect the same churn event simultaneously, but \( v \) only invokes the TCO repairing algorithm once for each churn event.

### 7.3 TCO Repair Algorithms in TCOUpgrader

We concentrate on the strategies in the TCOUpgrader module to dynamically maintain TCO in presence of churn.

When nodes are joining or leaving the overlay, it is possible to re-attain TCO by only adding links among a selected node subset. We try to reduce the size of the node set involved in the overlay repair algorithms. First, this allows us to repair the TCO with a partial knowledge about a small node subset. Second, the number of nodes turns out to be the principal factor for the time complexity of the TCO construction algorithms \[35, 69\], and thus our algorithms can run much faster thanks to the reduced size of the node subset involved.

We define the shadow set as the subset of nodes that are employed for overlay repair upon node churn. For example, given an initial \( TCO(V, T, Int, E) \) and a churn event with only one node \( v' \), the shadow set (or shadows) is a subset of nodes from \( V' \) that are chosen in the overlay repair step, where \( V' = V \cup \{p'\} \) for joining and \( V' = V \setminus \{v'\} \) for leaving. Suppose node \( v \) is the immediate successor of \( v' \) in the nodeId circle. Alg. \( 35 \) specifies the TCO repairing algorithm invoked at the coordinator node \( v \in V \) for a churn event at \( v' \). With the \( v.L' \) maintained by MemberServ, function computeTcoOnChurn() first computes the shadow set \( S \) according to the churn type (Lines 4-7), and then repairs the TCO locally among the shadow set (Lines 8-9). Line 8 estimates the existing overlay edges among shadow set based on its partial knowledge, and function buildEdges() computes the edges that need to be added to restore TCO.
Chapter 7. Elasto: Efficient Maintenance of Pub/Sub TCO Under Churn

Algorithm 35 TCO Repairing invoked at $v$ upon churn at $v'$

1. $v' \leftarrow \text{chnevt.node}$
2. $l \leftarrow \{l \mid \text{v'.pred[l]} = v\}$
3. $S \leftarrow \text{v.backup}\{l\}$
4. if $\text{chnevt.type}$ is JOIN then
   5. $S \leftarrow S \cup \{v'\}$
   else
      7. $S \leftarrow S \cup \text{v'.pneighb[l]}$
5. $E_{\text{cur}} \leftarrow \{(x; y) \mid x, y \in S \land x:id \in y.neighbIds \land y:id \in x.neighbIds\}$
6. $\text{return buildEdges}(S, v', \text{topics}; \int[S])$

Algorithm 36 $\text{buildEdges}(S, T', \int', E_{\text{cur}})$

1. $E_{\text{new}} \leftarrow \emptyset$, $E_{\text{pot}} \leftarrow (S \times S) \setminus E_{\text{cur}}$
2. while $G = (S, E_{\text{new}})$ is not TCO do
   3. for all $(x, y) \in E_{\text{pot}}$ do
      4. $\text{contrib}(e) \leftarrow |\{t \in T'| \int'(x, t) \land \int'(y, t) \land x \& y \text{ are in diff TC-components in } G(t)\}|$
      5. $e \leftarrow \text{findLowEdge}(\rho)$
      6. $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{e\}$
      7. $E_{\text{pot}} \leftarrow E_{\text{pot}} \setminus \{e\}$
   8. $\text{return } E_{\text{new}}$

We implement $\text{buildEdges()}$ by reusing LowODA [69]. Function $\text{buildEdges()}$ operates in a greedy manner: it iteratively adds carefully selected edges one by one until topic-connectivity is attained. At each iteration, $\text{findLowEdge()}$ selects an edge for the overlay (Line 5 of $\text{buildEdges()}$). The edge selection rule is based on a combination of two criteria: node degree and edge contribution, which is defined as reduction in the number of TC-components caused by the addition of the edge to the current overlay. Edge contribution for an edge $e$ is denoted as $\text{contrib}(e)$. Function $\text{findLowEdge()}$ uses a parameter $\rho$ to tread the balance between maximum and average node degree and it makes a weighed selection between two candidate edges: (1) $e_1$ s.t. $\text{contrib}(e_1)$ is the maximum among $E_{\text{pot}}$ and (2) $e_2$ s.t. $\text{contrib}(e_2)$ is the maximum among the subset of $E_{\text{pot}}$ in which all edges increase the maximum degree of $G(V, E_{\text{new}} \cup E_{\text{new}})$ minimally. If $\text{contrib}(e_1) \geq \rho \cdot \text{contrib}(e_2)$, edge $e_1$ is added, otherwise $e_2$ is added.
7.4 Selecting Local View in MEMBERSERV

Alg. 36 Selecting Local View at $v \in V$

1: for $l = 0$ to $L - 1$ do
2:   $v.succ[l] \leftarrow v$’s $l$-th nearest successor in $(buffer \setminus v.D)$
3:   $v.pred[l] \leftarrow v$’s $l$-th nearest predecessor in $(buffer \setminus v.D)$
4:   $v.backup[l] \leftarrow \text{buildBackups}(l, buffer)$
5: for $\forall l$ s.t. $v.pred[l]$ is updated do
6:   $v.pneighb[l] \leftarrow \emptyset, w \leftarrow v.pred[l]$
7:   $msgReq \leftarrow \text{new NeighbQ}(w.id)$
8:   send($w, msgReq$)
9:   wait until [(received expected $msgRes$ from $w$) OR ($w \in v.D$)]
10: if received expected $msgRes$ then
11:   $v.pneighb[l] \leftarrow msgRes.neighbs$

In order to efficiently compute the shadow set upon churn, MEMBERSERV proactively maintains the local view $v.L$ for each node $v \in V$.

We can regard the shadow set $S$ (Line 5 and 7 in Alg. 35) as a sample of nodes that is representative of specific characteristics for the entire node population $V'$. Taking topic-connectivity into account, it is always safe (but not necessarily efficient) to set shadow set as the complete node set $V'$.

However, there may exist many other choices for the shadow set with much fewer nodes. In the case that node $v'$ is leaving, one candidate for the shadow set is the neighbor set around the leaving node $v'$, denoted as $v'.N$. Observe that to re-attain topic-connectivity, it is sufficient to add links among $v'.N$ to the existing overlay. This can be done very efficiently since $v'.N$ is usually much smaller than the complete node set $V'$, but the node degrees of $v'.N$ would usually degrade significantly in the output TCO. The trade-off between the runtime cost and the quality of the output TCO can be balanced by selecting the shadow set in between $v'.N$ and $V'$. In the example of Fig. 7.1, when Node $v_{10}$ is leaving, the neighbor set $v_{10}.N = \{v_{11}, v_3, v_5\}$, in conjunction with a subset of other nodes $B = \{v_1, v_6\}$, forms the shadow set $S$ to repair the “broken” TCO.

Function $\text{buildBackups}()$ in Alg. 36 builds the backup set for each predecessor. For each $v' = v.pred[l]$ where $0 \leq l < L$, we want the backup set $B(v') = v.backup[l]$ to possess two desirable
properties: 1. Each \( u \in B(v') \) shares at least one topic with \( v' \): \((u.\text{topics} \land v'.\text{topics}) \neq \emptyset, \forall u \in B(v')\); 2. All topics subscribed by \( v' \) are covered by \( B(v') \): \( v'.\text{topics} \subseteq (\cup_{u \in B(v')} u.\text{topics}) \).

Note that \( B \) with the above two properties is equivalent to a set cover where \( v' \)'s topics is the universe of ground elements that are supposed to be covered by the topic set of backup nodes. The minimum weighted set cover problem is a well-studied NP-hard problem \([37, 48]\). We can therefore apply classical algorithms to obtain a feasible (not necessarily minimum) backup set.

Still, there is a trade-off concerning the size of the backup set. On the one hand, small cardinality is desired for the backup set with regard to runtime cost, because the number of backups directly impacts the size of the shadow set and thus the time complexity of the overlay repair algorithms. On the other hand, a sufficiently large backup set is preferred for ensuring the quality of the output TCO: a larger backup set means a large shadow set \( S \) and therefore a higher probability for the instance \((S, v'.\text{topics}, \text{Int}|S)\) to approximate \( I'(V', T, \text{Int}') \) with regard to the maximum and average node degrees in the TCO.

We introduce the coverage factor to tune the size of the backup set for a node and seeking balance between the time complexity of the overlay maintenance and the quality of the output TCO. Given \( I(V, T, \text{Int}) \), we build the backup set \( B(v') \) for each node \( v' \in V \). The coverage factor for the backup set \( B(v') \), denoted as \( \lambda(B(v')) \) (or \( \lambda \)), is the minimum number of subscribers to \( t \) within \( B(v') \) taken across all \( t \in v'.\text{topics} \): \( \lambda(B(v')) = \min_{t \in v'.\text{topics}} |\{u | u \in B(v') \land t \in u.\text{topics}\}| \).

The coverage factor is an integer (\( \lambda \geq 0 \)) such that each topic of interest of \( v' \) is covered at least \( \lambda \) times by its backup set \( B(v') \).

The coverage factor selection exhibits the trade-off between the running time and node degrees: On the one hand, \( \lambda = 0 \) minimizes the size of the backup set and running time, but leads to a severe impact on the node degrees. On the other hand, if we choose the coverage factor to be a large value such that all nodes of \( V' \) have to be included in the backup set, then both the maximum and average node degrees are close to those generated by running the static algorithms from scratch, but the runtime cost is not insignificant. According to our experiments in §7.5.3, an increase in \( \lambda \) beyond 5 only marginally improves the node degrees of the TCO under churn. The backup set for \( \lambda = 5 \) is significantly smaller than the complete node set itself so that we choose 5 as the default value for \( \lambda \).

There exist two approximation algorithms for the minimum weight set cover problem: the greedy
algorithm [37] and the primal-dual algorithm [48]. Function buildBackups() has two implementations - each executes the greedy or primal-dual algorithm iteratively until obtaining a λ-backup-set, which we refer to as buildBackupsGreedily() and buildBackupsByPD(), respectively.

The buildBackupsGreedily() procedure applies the greedy set cover heuristic for constructing a backup set [37]. It always chooses the next node $w$ so as to provide the minimum average weight over uncovered topics that would be covered by $w$. This algorithm achieves a log approximation ratio. However, its greediness leads to prioritizing bulk nodes that subscribe to a large number of topics upon backup selection. A small number of bulk nodes would serve as backups for a large number of nodes while the majority of lightweight nodes would not be selected as backups at all. Fairness is lost to a large extent in this case. Furthermore, the impact of accumulated sub-optimality would become progressively severe over time as the number of churn rounds increases.

The buildBackupsByPD() procedure is based on the primal-dual method for computing minimum weight set cover [48]. The algorithm proceeds in an iterative manner: each time randomly picking an uncovered topic $t$ and choosing a subscriber $w$ for $t$ with the minimum weight as a backup. The primal-dual algorithm yields an $f$-approximate solution for minimum weight set cover where $f$ is the maximum frequency of any element. The approximation ratio of primal-dual is higher than that of the greedy set cover algorithm. Yet, it is deemed acceptable in practice for many instances of the problem. Moreover, the primal-dual approach integrates randomness into greediness and effectively mitigates the prioritization of bulk subscribers. Therefore, we decide to leverage the primal-dual algorithm towards building the backup set in Line 4 of Alg. 36. We experimentally compare these two algorithms in §7.5.2.

### 7.5 Evaluation

We implement Elasto and other comparison systems (see Table 7.1) in the PeerSim simulator [67]. We develop Elasto as described in §7.2, §7.3 and §7.4.

We use LowODA [69] as a baseline because it is the only known polynomial time algorithm that achieves sub-linear approximation ratios on both the maximum and average node degrees. We implement two LowODA based algorithms for handling churn: LowODA-Inc and LowODA-Re.
We also compare Elasto with SpiderCast [36], because SpiderCast is highly efficient in constructing TCO in a decentralized peer-to-peer manner and has been adopted in practice [47]. Originally in [36], SpiderCast did not explicitly specify churn handling mechanism; however, we can simply employ SpiderCast’s local neighbor selection heuristics for TCO repairing under churn. More specifically, upon churn event, each node \( v \in V \) would independently invoke the neighbor selection heuristics if \( v \) is rendered as topic-disconnected by the churn event.

<table>
<thead>
<tr>
<th>Elasto</th>
<th>Our proposed pub/sub TCO maintenance system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elasto-L</td>
<td>Local view is maintained at each node for computing shadows and TCOs.</td>
</tr>
<tr>
<td>Elasto-G</td>
<td>Global view is maintained at each node for computing shadows and TCOs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LowODA</th>
<th>Low Max and Avg Degree Overlay Design Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowODA-Inc</td>
<td>Incrementally repair TCO using LowODA-rule regarding existing edges.</td>
</tr>
<tr>
<td>LowODA-Re</td>
<td>Reconstruct TCO from scratch at each churn round, regardless of existing edges.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SpiderCast [36]</th>
<th>A peer-to-peer pub/sub overlay construction protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpiderCast ( k_g, k_r )</td>
<td>Neighbor selection combines two local heuristics: greedy and random coverage. Each node tries to cover its interested topics ( K_g ) or ( K_r ) times. We use SpiderCast((3, 1)) and SpiderCast((4, 0)), as recommended in [36].</td>
</tr>
</tbody>
</table>

We evaluate the above systems with both synthetic pub/sub workloads and real-world traces derived from data sets of Facebook, Twitter and Google. We mainly focus on evaluating the overlay properties (e.g., the node degree and the diameter) and the efficiency of repairing topic-connectivity under churn.

### 7.5.1 Experimental Setup

**Pub/Sub Workloads:** We synthetically generate three types of topic popularity distributions: uniform, Zipfian, and exponential. We also extract the workloads from real-world social networks, namely Twitter and Facebook.

1. **Synthetic workloads:** We initialize the base instance \( I_0(V_0, T, Int) \) with \( |V_0|=2000, |T|=200 \) and \( |v.topics| \in [10, 90] \), where the subscription size of each node follows a power law. Each topic \( t \in T \) is associated with probability \( p(t) \), \( \sum_{t \in T} p(t)=1 \), so that each node subscribes to \( t \) with a probability \( p(t) \). The value of \( p(t) \) is distributed according to either an exponential, a Zipfian (with \( \alpha=2.0 \)), or a uniform distribution, which we call Expo, Zipf, or Unif for short. These distributions are representative of actual workloads used in industrial pub/sub systems today [36]. Expo is
used by stock-market monitoring engines for the study of stock popularity in the New York Stock Exchange [90]. Zipf faithfully describes the feed popularity distribution in RSS feeds [64].

(2) **Facebook dataset**: We use a public Facebook dataset [94], with over 3 million distinct user profiles and 28.3 million social relations as a second workload for our evaluations. In Facebook, when a user has some activity (e.g., updating the status, sharing new photos, or commenting on a blog), all her friends will receive some notifications as subscribed. As such, we model each user, say Alice, as a topic, and all her friends are the respective subscribers. Likewise, the friend set of Alice forms her subscription set. The Facebook relations are bidirectional, so friends in the Facebook social graph subscribe to each other in our model.

(3) **Twitter dataset**: We also use a public Twitter dataset [56], containing 41.7 million distinct user profiles and 1.47 billion social followee/follower relations. Similarly to Facebook, we model users as topics as well as subscribers. However, relations are unidirectional in Twitter, i.e., Alice following Bob does not require that Bob also follows Alice.

We extract the workloads from the original Facebook and Twitter social graphs with a methodology inspired from [76]. More specifically, starting with a random set of a few users as seeds, we traverse the social graph via breadth first search, until it reaches the target number of nodes, and our sample includes all edges among the nodes. The size of our samples is 1K or 10K, i.e., $|V| \approx 1K$ and $|T| \approx 1K$, OR $|V| \approx 10K$ and $|T| \approx 10K$. Fig. 7.3 shows the complementary cumulative distribution function (CCDF) of follower/followee counts for both the original Facebook and Twitter datasets, and for our extracted datasets in the inner plot. The plots indicate that the original dataset properties were retained in the extracted sample. We also observe from Fig. 7.3 that the Twitter data has more correlation than the Facebook data. We denote the instances of our samples by FB 1K, FB 10K, TW 1K, and TW 10K, respectively.

![Figure 7.3: CCDF of followers and followees: Twitter (41.7M users) and Facebook (3M users). Inner plot: 10K-user sample.](image-url)
Churn traces: We first generate sample churn traces that contain over 1000 rounds of churn. Each churn round generates a join or leave with probability 0.5.

We also use the Google cluster data [5] as real-world churn traces to evaluate our system under churn. The Google cluster trace includes data from an 11K-machine cell over about a month-long period. The machine event table contains three types of machine events:

- ADD: a machine became available in the cluster.
- REMOVE: a machine was removed from the cluster.
- UPDATE: a machine available in the cluster had its available resources changed, such as CPU, memory, etc.

We randomly sampled 1K and 10K unique machine ids from the start of the Google cluster trace and then generated join churn rounds via ADD events and leave churn rounds via REMOVE events. We omitted UPDATE events.

7.5.2 Building Backups Greedily vs. by Primal-dual

We first compare two algorithms for building backups as we propose in §7.4. To eliminate other factors in the design domain, we deploy two instances of ElastO-G with buildBackupsGreedi() and buildBackupsByPD(), respectively. For both instances, we set \( \lambda = 3 \), initiate the same TCOs, and feed identical churn traces.

As a complement measure to the backup set, we define the primary set for each node \( v \in V \) as a subset of nodes for which \( v \) serves as backup in the local view, i.e., \( P(v) = \{ u | v \in B(u) \} \). We call \( |P(v)| \) as \( v \)'s primary degree.

Fig. 7.4 and Fig. 7.5 show the output of primary-backup schemes under Unif, respectively. As shown in the figures, both buildBackupsGreedi() and buildBackupsByPD() produce small-sized backup sets compared to the complete node set \( V_0 \): 2.8% of \( |V_0| \) for buildBackupsByPD() and 1.1% of \( |V_0| \) for buildBackupsGreedi() on average across all nodes, respectively. In general, the backup degree is linearly proportional to the subscription size for both algorithms. However, the distribution of the primaries differs considerably between them. The primaries produced by buildBackupsGreedi() are skewed and follow an exponential shape: 42.9% of all nodes have less than 5 primaries, while the highest primary degree of a single bulk subscriber peaks at 581. At
the same time, the distribution of the backups produced by `buildBackupsByPD()` is well-balanced: the lowest primary degree is 21 and the highest is 59. These results confirm our observations in §7.4 about the side effect of greediness on the primary assignment and the fairness introduced by randomness in the primal-dual scheme.

We also compare both backup construction algorithms in terms of the evolution of overlay properties under churn. As shown in Fig. 7.6, both the maximum and average node degrees increase as the TCO instance evolves with node churn. However, overlay quality for ElastO-G utilizing `buildBackupsGreedily()` degrades noticeably, i.e., at churn round 1000, the maximum node degree becomes 84. On the other hand, when the backups are built by primal-dual, both the maximum and average degrees keep a steadily low growth rate. Results presented in Fig. 7.4, 7.5 and 7.6 substantiate our choice of `buildBackupsByPD()` over `buildBackupsGreedily()`. In the rest of the evaluation, we choose `buildBackupsByPD()` in the `selectLV()` procedure (Line 4 of Alg. 36).

### 7.5.3 Selecting Local View and Choosing Coverage Factor

We explore the impact of local view selection with different coverage factors on the output and performance of ElastO-L. We evaluate ElastO-L with different values of the coverage factor ($\lambda = 0, \ldots, 9$).

Fig. 7.7(a) shows that the average local view in ElastO-L is fairly small as compared to the overall size of the network. Under FB 1K, the local view is 21.02 with $\lambda = 1$, 40.49 with $\lambda = 3$, and...
Chapter 7. ElastO: Efficient Maintenance of Pub/Sub TCO Under Churn

Figure 7.7: ElastO-L with λ’s - FB 1K

48.40 with \( \lambda = 5 \), and 57.59 with \( \lambda = 9 \), on average. The average local view also decreases over time, and the majority of nodes keeps a small local view steadily under all instances. This is because, ElastO-L evolves with constant epidemic exchange, and the local view converges promptly with more balanced load distribution. Under all instances, after the first 100 rounds of gossiping, more than 95% of the nodes in ElastO-L|\( \lambda = 5 \) have their local view fewer than 5% of all nodes. This demonstrates that ElastO-L achieves good load balancing in terms of local view maintenance.

Fig. 7.7(b) shows that, as \( \lambda \) increases, \( D_{\text{ElastO-L}} \) decreases, and its growth rate with respect to the churn round decreases. The differences in the maximum node degrees and their growth rates also decrease with successive coverage factors. Under FB 1K, at the end of the Google cluster churn trace, \( D_{\text{ElastO-L}}|\lambda = 0 = 333 \), \( D_{\text{ElastO-L}}|\lambda = 1 = 313 \), \( D_{\text{ElastO-L}}|\lambda = 3 = 302 \), \( D_{\text{ElastO-L}}|\lambda = 5 = 283 \), \( D_{\text{ElastO-L}}|\lambda = 7 = 277 \), and \( D_{\text{ElastO-L}}|\lambda = 9 = 274 \). When \( \lambda \geq 5 \), the difference of the maximum node degrees with different \( \lambda \) values is insignificant. The average node degrees follow the same trends with respect to the coverage factor, and the difference among different values of \( \lambda \) is even more insignificant.

These experiments demonstrate our \text{selectLV}() method achieves good load balancing in terms of local view maintenance across all nodes in ElastO, good balancing of , and confirms the validity of choosing a relatively small coverage factor (see §7.4). We fix \( \lambda = 5 \) for ElastO-L in the rest of §7.5, which demonstrates the scalability, efficiency and robustness of ElastO-L under churn.
7.5.4 Overlay Node Degrees

Fig. 7.8 compares the node degrees produced by different algorithms and protocols as the instances evolve with the Google churn trace, where we initialize all with the same TCO constructed by LowODA from scratch. We do not plot lines for Elasto-G or LowODA-Inc, because Elasto-G lies close to Elasto-L and LowODA-Inc lies close to LowODA-Re, e.g., $D_{\text{Elasto-L}} \leq 1.04 \cdot D_{\text{Elasto-G}}$, and $d_{\text{Elasto-L}} - d_{\text{Elasto-G}} \leq 0.76$, on average under FB 1K.

First, Elasto-L output similar maximum and average node degrees as compared to LowODA-Re. For example, $D_{\text{Elasto-L}} \leq 1.15 \cdot D_{\text{LowODA-Re}}$, $d_{\text{Elasto-L}} - d_{\text{LowODA-Re}} \leq 1.12$, on average over the entire churn sequence under FB.

Second, the node degrees of Elasto-L degrade slowly like a step function along a sequence of churn rounds. From churn round 1 to 2000 under FB 1K, $D_{\text{Elasto-L}}$ increases from 232 to 283, a degradation rate of 0.025 per churn round. This rate becomes even slower as the input instance scales up, which drops to 0.013 under FB 10K.

Third, SpiderCast degrades fast in the first 200 churn rounds, and the output node degrees stay a number of times higher than those of Elasto and LowODA. The gap increases as the instances scale up. For example, $D_{\text{SpiderCast}(3,1)} - D_{\text{Elasto-L}}$ is 355 under FB 1K and 1050 under FB 10K, respectively on average. SpiderCast(3, 1) generates more edges than SpiderCast(4, 0) due to random coverage, which leads to a better chance to attain topic-connectivity (see [36]).
7.5.5 Runtime Cost

Fig. 7.9 and Fig. 7.10 depict cumulative distribution functions (CDF) for the running time of different algorithms over a sequence of churn rounds. As shown in the figures, LowODA-Re runs considerably slower than other dynamic algorithms, because LowODA-Re tears down existing links and reconstructs the TCO from scratch at each churn round. The runtime costs of LowODA-Inc and ElastO-G are of the same order of magnitude. ElastO-L and SpiderCast further improve the time efficiency significantly – ElastO-L is around 4.67% that of ElastO-G on average across all instances, thanks to local operations rather than global computation.

The speedup of ElastO-L as compared to the static LowODA-Re is more profound when the size of instance increases from 1K up to 10K. The running time ratio of ElastO-L against LowODA-Re is around 0.20% under 1K and 0.01% under 10K, on average. This demonstrates the scalability of ElastO-L with respect to the number of nodes, the number of topics, and the subscription sizes.

ElastO, SpiderCast and LowODA-Inc require more time to dynamically repair the TCO under leaves than under joins. This can be explained by the number of nodes involved in the repairing.
phase. Different magnitudes of time costs between handling joins and leaves form clear horizontal lines around 50% in the CDFs of ElastO-G, SpiderCast, and LowODA-Inc. Meanwhile, the shape of running time CDF for ElastO-L exhibits more smoothness and robustness across all churn rounds. This shows that ElastO-L achieves balanced load distribution and fairness among all nodes in the network over the sequence of churns.

7.5.6 Topic Diameters

We also look at topic diameters, which impact many performance factors for efficient routing in pub/sub, e.g., message latency. Given \( TCO(V, T, Int, E) \), the topic diameter for \( t \in T \) is \( diam(t) = diam(G^{(t)}) \), where \( diam(G^{(t)}) \) is the maximum shortest distance between any two nodes in \( G^{(t)} \). We denote the maximum and average topic diameter across all topics as \( Diam \) and \( \overline{diam} \), respectively.

Fig. 7.11 depicts the average topic diameters produced by different algorithms and protocols over the Google churn sequence under TW 1K. All systems start with the same topic diameters, which tend to decrease as the overlays evolve with the churn, because the number of overlay edges increases. Generally, topic diameters are inversely proportional to node degrees: SpiderCast has the lowest topic diameters, and LowODA-Re has the highest topic diameters. Topic diameters of ElastO-L are slightly higher than those of SpiderCast, but the difference is insignificant: \( Diam_{\text{ElastO-L}} - Diam_{\text{SpiderCast}(3,1)} = 3.67 \), and \( \overline{diam}_{\text{ElastO-L}} - \overline{diam}_{\text{SpiderCast}(3,1)} = 1.45 \), respectively on average.

7.5.7 Communication Overhead

We evaluate the communication overhead for overlay maintenance. We measure the number of messages sent and received by all nodes at each churn round. We rule out the communication overhead incurred by the failure detector, which we assume that this component is identical for all systems.

For ElastO, the communication overhead includes: (1) coordination messages for repairing T-\( CO \), including handling concurrent churn events; (2) periodical message exchange for local view
maintenance; and (3) notifications to establish links with new neighbors. Meanwhile, since Spider-Cast is pure peer-to-peer, its communication overhead only contains (2) and (3) in the list.

In Fig. 7.12, Spider-Cast incurs significantly higher communication overhead as compared to ElastO. The main reason is that Spider-Cast has much higher node degrees than ElastO, and therefore churn impacts more nodes in Spider-Cast.

Fig. 7.12 also shows that the systems require more messages when the churn node has a higher subscription size. This comes from the fact that nodes with a large number of subscriptions have more TCO neighbors. Besides, in ElastO, these nodes are more likely to be selected as backups.

7.6 Conclusions

We present a fully dynamic system, ElastO, to construct and maintain low fan-out TCOs for pub/sub systems under churn. We demonstrate that the ElastO system combines the advantages from both the static algorithms and the decentralized protocols under large-scale pub/sub workloads extracted from Twitter and Facebook and real-world cluster churn traces released by Google: (a) both the maximum and average node degrees remain insignificantly higher than those of the static algorithms; (b) the time efficiency is of the same order of magnitude as the decentralized protocols.
Chapter 8

Embarking on Overlay Design for Content-based Pub/Sub

8.1 Introduction

This chapter embarks on the problem of overlay design for content-based pub/sub. We apply a principally different approach as compared to existing approaches (see §1.5.2 and §2.2): construct the overlay and design the pub/sub routing by mining the node-message matching relationships. We attempt to find the “sweet spot” where the combination of overlay construction and pub/sub routing design yields the maximum message dissemination efficiency for a given amount of resources. More specifically, our contributions are as follows:

1. In §8.2, we analyze the challenges for overlay construction and routing protocol design for content-based pub/sub systems. We formalize the problem that captures the core trade-offs between overlay infrastructure and routing protocols. We introduce recall and precision as optimizing metrics for pub/sub messaging. Recall rates the completeness of message delivery to subscribed nodes. Precision represents the accuracy of message delivery, which is often employed to minimize the number of pure forwarders or unwanted messages.

2. In order to judge the quality of the pub/sub overlay, we design a family of pub/sub routing
protocols that embraces many existing approaches (see §8.3). We characterize pub/sub routing by the size of the local view each node needs to maintain in the routing table. Ideally, a “good-quality” overlay should amount to efficient pub/sub routing with a small local view (rather than requiring global knowledge) at each node.

3. We devise overlay design algorithms for content-based pub/sub systems in §8.4. Our algorithms are capable of efficiently producing overlays with bounded node degrees. The output overlays of our algorithms enable less complicated and more scalable pub/sub routing: (1) each node only needs to maintain a partial view around its own neighborhood and (2) message flooding is eliminated. As a result, the pub/sub system achieves higher recall and precision in message dissemination with smaller forwarding tables and a simpler matching engine.

4. In §8.5, we evaluate our algorithms against the state-of-the-art TCO-based approaches under a variety of typical pub/sub workloads. Our algorithms keep the overlay node degree low and support simple pub/sub routing in which each node just maintains the subscriptions of its immediate neighbors. On average, the pub/sub system achieves $\geq 80\%$ recall and $\geq 80\%$ precision for disseminating all messages. By contrast, TCO-based algorithms are computationally expensive and often produce close-to-clique overlays, which leads to unscalable message dissemination.

### 8.2 Problem Formulation

Suppose $V$ is the set of nodes in a pub/sub system (e.g., publishers, subscribers, and brokers), $M$ the set of messages, and $Hit$ a Boolean function over nodes and messages, i.e., $Hit : V \times M \rightarrow \{true, false\}$. Given the hit function $Hit$, we say that message $m \in M$ hits node $v \in V$ if and only if $Hit(v, m) = true$. We then also say that message $m$ matches node $v$’s interest or node $v$ is interested in message $m$. Since the domain of function $Hit$ is a Cartesian product, we also refer to $Hit$ as a hit matrix. Each node $v \in V$ could be a publisher, a subscriber or both. That is, we abstract from the differentiation between clients and brokers in our formalization. Each message $m \in M$ originates from a publishing source $src(m)$ and by default $Hit(src(m), m) = true$.

We denote an instance of a pub/sub system as a triple $I(V, M, Hit)$, which emphasizes the core matching relationship between nodes and messages in the system. This representation embraces
both topic-based and content-based pub/sub.

We denote a pub/sub overlay (PSO) by $PSO(V, M, Hit, E)$, which can be illustrated as an undirected graph $G = (V, E)$ over the node set $V$ with the edge set $E \subseteq V \times V$. Informally speaking, given a set of nodes $V$, a set of messages $M$, and the hit function $Hit$, we want to construct a $PSO(V, M, Hit, E)$ that supports pub/sub communication with maximum efficiency (see precision and recall in §8.2.1) subject to some resource constraints, such as the number of available links at each node.

To formalize the content-based pub/sub overlay design problem, we define two optimization objectives captured by the message dissemination recall and precision in the overlay (cf. §8.2.1.) Then, we present the problem statement in §8.2.2.

### 8.2.1 Recall and Precision for Pub/Sub

The recall and precision metrics are commonly defined in terms of retrieved instances and relevant instances. Both recall and precision are therefore based on an understanding and measure of relevance. These notions are also applicable for measuring the effectiveness of message dissemination in pub/sub systems. In this section, we formally define recall and precision as optimization objectives for overlay construction and routing protocol design in pub/sub systems.

For each message $m \in M$, function $Hit(v, m)$ labels each node $v \in V$ as either true or as false: $v$ is labeled true with respect to $m$, if $Hit(v, m)$ is true, otherwise $v$ is labeled false with respect to $m$, $\forall v \in V$. More specifically, we write:

$$True(m) = \{v \in V| Hit(v, m) = true\} \quad (8.1)$$
$$False(m) = \{v \in V| Hit(v, m) = false\} \quad (8.2)$$

By definition, $True(m) \cap False(m) = \emptyset$, $True(m) \cup False(m) = V$, and $src(m) \in True(m)$.

We denote an instance of a pub/sub system as a triple $I(V, M, Hit)$. Given an instance $I$, an overlay design algorithm $A$, and a pub/sub routing protocol $P$, we classify all nodes into two categories, as either positive or negative, depending on the delivery of message $m$ to the overlay nodes: $v$ is positive with respect to $m$, if $m$ is delivered to $v$, otherwise $v$ is negative with respect to $m,$
∀v ∈ V. More specifically, we denote:

\[ Pos(A,P)(m) = \{ v \in V | v \text{ is positive w.r.t. } m \} \] (8.3)
\[ Neg(A,P)(m) = \{ v \in V | v \text{ is negative w.r.t. } m \} \] (8.4)

We represent these definitions with the confusion matrix known from machine learning and information retrieval [7]. The confusion matrix contains four disjoint categories:

1. True positives (TP) are “hits” – true nodes correctly classified as positives, i.e., nodes that are interested in \( m \) and receive \( m \),

\[ TP(A,P)(m) = True(m) \cap Pos(A,P)(m). \] (8.5)

2. False negatives (FN) refer to “misses” – true nodes incorrectly classified as negatives, i.e., nodes that are interested in \( m \) but do not receive \( m \),

\[ FN(A,P)(m) = True(m) \cap Neg(A,P)(m). \] (8.6)

3. False positives (FP) refer to “false alarms” – false nodes incorrectly classified as positives, i.e., nodes that are not interested in \( m \) but receive \( m \),

\[ FP(A,P)(m) = False(m) \cap Pos(A,P)(m). \] (8.7)

4. True negatives (TN) correspond to “correct rejections.” More specifically, TN consists of false nodes correctly classified as negatives, i.e., nodes that are not interested in \( m \) and do not receive \( m \),

\[ TN(A,P)(m) = False(m) \cap Neg(A,P)(m). \] (8.8)
Table 8.1: Example of confusion matrix for pub/sub communication

<table>
<thead>
<tr>
<th>MESSAGE</th>
<th>DELIVERY</th>
<th>INTEREST HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Pos</td>
<td>( TP_{(A,P)}(m) = {v_2, v_6} )</td>
</tr>
<tr>
<td>False</td>
<td>Neg</td>
<td>( FN_{(A,P)}(m) = {v_1, v_4} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( FP_{(A,P)}(m) = {v_3, v_5, v_8} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( TN_{(A,P)}(m) = {v_7} )</td>
</tr>
</tbody>
</table>

As an example in Fig. 8.1(a), an overlay design algorithm \( A \) constructs an overlay among eight nodes \( V = \{v_1, v_2, \ldots, v_8\} \). We focus on one message \( m \in M \) where \( src(m) = v_2 \) and \( True(m) = \{v_1, v_2, v_4, v_6\} \) (see shaded nodes in the figure.) Some routing protocol \( P \), which we design in §8.3, delivers \( m \) to \( Pos_{(A,P)}(m) = \{v_2, v_3, v_6, v_5, v_8\} \). Table 8.1 shows the confusion matrix for this instance.

Given overlay design algorithm \( A \) and pub/sub routing protocol \( P \), we define recall and precision for \( m \in M \):

\[
Recall_{(A,P)}(m) = \frac{|TP_{(A,P)}(m)|}{|TP_{(A,P)}(m) \cup FN_{(A,P)}(m)|} \tag{8.9}
\]

\[
Precision_{(A,P)}(m) = \frac{|TP_{(A,P)}(m)|}{|TP_{(A,P)}(m) \cup FP_{(A,P)}(m)|} \tag{8.10}
\]

We can interpret recall as a measure of completeness and precision as a measure of exactness. In the context of pub/sub messaging, recall is a measure to Question 1:

*Of all nodes that are actually interested in message \( m \), what fraction of nodes does the system successfully deliver the message to?*

Precision is quantifying Question 2:

*Of all nodes that receive message \( m \), what fraction are actually interested in this particular message?*

High recall means that most of the messages are successfully delivered to matching nodes, while high precision means that nodes receive substantially more matching messages than non-matching
ones. A perfect recall score of 1.0 means that the system delivers message $m$ to all nodes that are interested in $m$ (but says nothing about how many unmatched nodes also have $m$ delivered), whereas a perfect precision score of 1.0 means that all nodes that receive $m$ are interested in $m$ (but says nothing about whether $m$ are delivered to all matched nodes). In the example of Fig. 8.1 and Table 8.1, $Recall(A, P)(m) = \frac{|\{v_2, v_6\}|}{|\{v_2, v_6\} \cup \{v_1, v_4\}|} = 0.5$ and $Precision(A, P)(m) = \frac{|\{v_2, v_6\}|}{|\{v_2, v_6\} \cup \{v_3, v_5, v_8\}|} = 0.4$.

Recall and precision are usually inversely related – it is possible to increase one at the cost of reducing the other. For example, to ensure the completeness of message delivery (i.e., recall), many pub/sub systems [25, 36, 91] employ epidemic flooding mechanism. As a consequence, precision will suffer because a large amount of messages are delivered to irrelevant nodes in order to make sure that nodes do not miss relevant messages.

The two measures are often combined through the $F1$-score [79] to provide a single metric for a system. We also adopt the $F1$-score – the harmonic average of recall and precision – in which recall and precision are evenly weighted:

$$ F1-score(A, P)(m) = \frac{2 \cdot Recall(A, P)(m) \cdot Precision(A, P)(m)}{Recall(A, P)(m) + Precision(A, P)(m)} $$ (8.11)

We define $recall$, $precision$ and $F1$-score for the problem instance $I(V, M, Hit)$ as the average values across $M$:

$$ Recall(A, P) = \frac{\sum_{m \in M} Recall(A, P)(m)}{|M|} $$ (8.12)

$$ Precision(A, P) = \frac{\sum_{m \in M} Precision(A, P)(m)}{|M|} $$ (8.13)

$$ F1-score(A, P) = \frac{\sum_{m \in M} F1-score(A, P)(m)}{|M|} $$ (8.14)

Note that $True(m)$ and $False(m)$ are simply derived from the given instance $I(V, M, Hit)$ and
are independent of the overlay design algorithm $\mathcal{A}$ or the pub/sub routing protocol $\mathcal{P}$. All other definitions (e.g., $\text{Pos}$, $\text{Neg}$, $\text{Recall}$, $\text{Precision}$, etc.) depend on both $\mathcal{A}$ and $\mathcal{P}$.

### 8.2.2 The Overlay Construction Problem

We focus on two optimization objectives in the overlay design for content-based pub/sub: (1) high F1-score (i.e., both high recall and high precision) and (2) low node degrees.

Formally, we define the pub/sub overlay design problem:

**Problem 7.** Given a set of nodes $V$, a set of messages $M$, the interest function $\text{Hit}$, and a routing protocol $\mathcal{P}$, construct a $\text{PSO}(V, M, \text{Hit}, E)$ that has bounded node degree $D$ and achieves a maximum F1-score under $\mathcal{P}$.

To properly evaluate the pub/sub overlay, we need to be more specific about the routing protocol $\mathcal{P}$ in Problem 7. In particular, we propose the family of routing protocols parameterized by the size of the local view in §8.3.

### 8.3 Pub/Sub Routing on $\delta$-Local-View

We judge the quality of an overlay by looking at the efficiency of the pub/sub message exchange that relies on the overlay. However, different pub/sub systems often implement their own routing protocols according to specific application requirements [78, 38, 26, 16, 21]. Instead of looking at each pub/sub routing protocol individually, we design a family of routing protocols that embraces most existing approaches. We then use this family as a standard gauge to examine the qualities of different pub/sub overlays in terms of the message dissemination, routing costs, and so on.

Our pub/sub routing protocol design adopts the concept of the $\delta$-neighborhood of an overlay [52], which we use to tune the size of local knowledge of a node in the system. Formally speaking, the $\delta$-neighborhood of a node $v \in V$ is the set of nodes (and their subscriptions) that are located within distance $\delta$ from $v$, denote by $v.\mathcal{N}(\delta)$. Note the distance between two nodes is the number of edges on the shortest path joining them.

Based on the $\delta$-neighborhood, we define the $\delta$-local-view to perform pub/sub routing on an overlay. Formally speaking, the $\delta$-local-view of a node $v \in V$, denoted as $v.\mathcal{L}(\delta)$, is a pub/sub
forwarding table with the knowledge of \( v \)'s \( \delta \)-neighborhood: each entry represents a map between a node \( n \in v \cdot \mathcal{N}(1) \) and \( P_n \) – the union of all downstream nodes reachable through \( n \) within \( v \)'s \( \delta \)-neighborhood. For example, Table 8.2 shows \( v_2 \cdot \mathcal{L}(\delta) \) parameterized with different values of \( \delta \) for the pub/sub overlay from Fig. 8.1.

<table>
<thead>
<tr>
<th></th>
<th>( \delta = 0 )</th>
<th>( \delta = 1 )</th>
<th>( \delta = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_6 )</td>
<td>( \emptyset )</td>
<td>( {v_6} )</td>
<td>( {v_6, v_5, v_8} )</td>
</tr>
<tr>
<td>( v_3 )</td>
<td>( \emptyset )</td>
<td>( {v_3} )</td>
<td>( {v_3, v_7} )</td>
</tr>
</tbody>
</table>

Figure 8.1: Message routing on an overlay

Table 8.3: F1-score, Recall, and Precision

<table>
<thead>
<tr>
<th></th>
<th>( \delta = 0 )</th>
<th>( \delta = 1 )</th>
<th>( \delta = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F1\text{-score}_{(A,\text{ROL}(\delta))}(m) )</td>
<td>0.444</td>
<td>0.667</td>
<td>0.889</td>
</tr>
<tr>
<td>( \text{Recall}_{(A,\text{ROL}(\delta))}(m) )</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( \text{Precision}_{(A,\text{ROL}(\delta))}(m) )</td>
<td>0.4</td>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

In pub/sub implementations, the \( \delta \)-local-view also needs to include the subscriptions of the nodes in the routing table, which are required for message matching (e.g., \([26, 16]\)). In this work, we focus on the overlay construction and protocol design, which is orthogonal to the development of the message matching scheme. We therefore assume that the hit function, \( \text{Hit} \), available at each node, determines whether node \( v \in V \) is interested in message \( m \in M \).
Based on the definition of the $\delta$-local-view, we propose the pub/sub routing protocol on $\mathcal{L}(\delta)$, ROL($\delta$) for short (see Protocol 37). Each node $v \in V$ runs Protocol 37 independently, and it is possible that different nodes are parameterized with different values of $\delta$. As node $v$ issues or receives message $m$, Protocol 37 first checks whether $v$ has seen $m$ before (Line 2): if $m$ arrives for the first time, then Protocol 37 uses $v \cdot \mathcal{L}(\delta)$ to select a subset of nodes from $v$’s immediate neighbors for forwarding message $m$. Remember that for each $n \in v \cdot \mathcal{N}(1)$,

$$P_n = \{w \in v \cdot \mathcal{L}(\delta) \mid w \text{ is a downstream node reachable through } n\}$$

As $\delta = 0$, $P_n = \emptyset$, and Protocol 37 simply floods $m$ to all of $v$’s neighbors, if $v$ is interested in $m$;
otherwise, node \(v\) drops \(m\) (Lines 5-8). As \(\delta > 0\), Line 9 in Protocol 37 defines \(P_n(m) \subseteq P_n\) as the set of downstream nodes that are interested in \(m\) and reachable through \(n\) within \(v\)'s \(\delta\)-neighborhood. Line 10, let \(P\) represent all downstream nodes that are interested in \(m\) within \(v\)'s \(\delta\)-neighborhood, i.e., all nodes in \(v, \mathcal{L}(\delta)\) that should receive message \(m\). Lines 12-17 compute a forwarding set \(F\) such that \(P \subseteq \bigcup_{n \in F} P_n(m)\). Protocol 37 first includes all neighbor nodes that are interested in message \(m\) (Lines 12-14). Then, to cover the remaining nodes of \(P\), Lines 15-17 greedily select neighbor nodes that are not interested in \(m\) – the classical greedy algorithm for the Minimum Set Cover problem [37].

Protocol 37 guarantees that the forwarding set \(F\) supports message delivery to all \(\delta\)-neighborhood nodes that are interested in \(m\). We formalize this property with Lemma 38.

**Lemma 38.** If message \(m \in M\) reaches node \(v \in V\) for the first time, then Protocol 37 outputs \(F\) that satisfies:

\[\exists n \in F \text{ s.t. } w \in P_n(m), \forall w \in P = \bigcup_{n \in N} P_n(m)\]

Lemma 39 ensures that running Protocol 37 on all overlay nodes delivers \(m\) to each node in \(P\).

**Lemma 39.** Suppose all nodes in \(V\) execute Protocol 37 with the same \(\delta\) value independently and message \(m \in M\) reaches node \(v \in V\), then \(m\) eventually reaches all nodes in \(P = \bigcup_{n \in N} P_n(m)\).

**Proof.** Pick a node \(w \in P\), and \(\text{Hit}(w, m) = \text{true}\). If \(w \in v, \mathcal{N}(1)\), node \(v\) adds \(w\) into its forwarding set \(F\) and then sends \(m\) to \(w\) (Lines 12-14). If \(w \notin v, \mathcal{N}(1)\), then according to Lemma 38, there exists \(n_1 \in F\) s.t. \(w \in v, P_n_1(m)\). Note that node \(n_1\) is closer to node \(w\), i.e., \(w \in n_1, \mathcal{N}(\delta - 1)\). Lemma 38 implies that \(n_1\) forwards \(m\) to a node \(n_2\) s.t. \(w \in n_1, P_n_2(m)\) and \(w \in n_2, \mathcal{N}(\delta - 2)\). As the procedure continues, \(m\) travels through \(v, n_1, n_2, \ldots\) and finally reaches \(w\). \(\square\)

Protocol 37 represents a family of pub/sub routing protocols with the neighborhood parameter \(\delta\). As \(\delta\) increases, each node \(v \in V\) has more local knowledge about its neighborhood, which tends to be beneficial for message dissemination. We again look at the example in Fig. 8.1: some overlay design algorithm \(A\) constructs an overlay for \(V = \{v_1, v_2, \ldots, v_8\}\). Node \(v_2\) issues one message \(m \in M\), i.e., \(\text{src}(m) = v_2\), and \(\text{True}(m) = \{v_1, v_2, v_4, v_6\}\) is the set of nodes that are interested in \(m\) (shaded in the figure). Table 8.3 illustrates F1-score, recall, and precision with regard to different
δ values.

A larger δ value leads to more expensive routing, such as the size of the routing tables, the running time for matching messages against nodes, etc. As δ becomes larger, each node obtains global knowledge about the entire overlay network. In our notation, we write $v, \mathcal{N}(\infty)$ to denote global knowledge of all nodes in the overlay, i.e., $v, \mathcal{N}(\infty) = V$.

We can model many pub/sub routing protocols as a combination of ROL(δ) with different δ values for different nodes. In most classic pub/sub routing protocols [16, 21, 25], a significant portion of the node set has to run ROL(∞), where the nodes are aware of the entire network, including the IDs and the subscriptions of all nodes. Different techniques have been designed to prune irrelevant nodes from the local knowledge, such as advertisement-based forwarding [16, 25] and subscription clustering based on common interests [21]. However, due to the lack of pub/sub overlay design, these systems still require lots of nodes to maintain a large forwarding table for pub/sub routing.

### 8.4 The Overlay Design Algorithm

To construct the overlay network for a content-based pub/sub systems, we design the Confidence-based Overlay Design Algorithm, ConfODA, for short.

**Algorithm 38 The Confidence-based Overlay Design Algorithm**

ConfODA$(V, M, Hit, D, \alpha)$

**Input:** $V, M, Hit, D, \alpha$

**Output:** $E$ – the overlay edge set

1: for all $v, w \in V$ do
2: compute confidence $\text{conf}(\{v\} \Rightarrow \{w\})$
3: for all node $v \in V$ do
4: greedily choose $\alpha$ neighbors with the highest confidence whose node degree are no more than $D$
5: randomly choose $\beta = (D - \alpha)$ neighbors whose node degree are no more than $D$

Alg. 38 takes 5 parameters. The first four $(V, M, Hit, D)$ represent the input problem instance: $V$ is the node set, $M$ is the message set, $Hit$ is the hit function, and $D$ is the bounded node degree for the output overlay. The last parameter $\alpha$ is the greedy neighborhood parameter, which Alg. 38 uses...
to guide the greedy neighbor selection. We also introduce $\beta = (D - \alpha)$ as the random neighborhood parameter for the random neighbor selection.

In Alg. 38, each node $v \in V$ greedily selects $\alpha$ neighbors according to the confidence ranking in Line 4 and then randomly selects $\beta = (D - \alpha)$ neighbors in Line 5. The greedy neighbor selection attempts to cluster nodes with similar interests, and the random neighbor selection is used to ensure the overall connectivity. The algorithm also ensures that the node degrees are bounded by $D = \alpha + \beta$.

We utilize the concepts of support and confidence, known from association rule mining [13], in the design of Alg. 38. In pub/sub messaging, we define support and confidence as measures for the reliability and the frequency of occurrence of node subsets that match different messages.

Table 8.4: Example: 4 nodes and 5 messages

<table>
<thead>
<tr>
<th>$m \in M$</th>
<th>$v \in V$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m_2$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$m_3$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_4$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$m_5$</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The support of a subset of nodes $X \subseteq V$, denoted as $supp(X)$, is the fraction of messages in $M$ that hit $X$:

$$supp(X) = \frac{|\{m \in M | X \subseteq True(m)\}|}{|M|}$$  \hspace{1cm} (8.15)

To illustrate the concepts, we use a small example in Table 8.4: the node set is $V = \{v_1, v_2, v_3, v_4\}$, the message set is $M = \{m_1, m_2, m_3, m_4, m_5\}$, 1 encodes true and 0 encodes false for the value of function $Hit(v, m)$. The node subset $\{v_1, v_2, v_3\}$ has a support of $\frac{1}{5} = 0.2$ since it occurs in 20% of all messages (1 out of 5 messages). This means that 1 message out of 5 messages hits at least the node $v_1$, $v_2$, and $v_3$. Note that this example is extremely small. In practical pub/sub systems with hundreds of nodes, a support needs at least tens of thousands of messages before it can be considered
statistically significant; a reasonable assumption for production pub/sub systems, which disseminate well over millions of messages.

Given $X, Y \subseteq V$ and $X \cap Y = \emptyset$, we define a rule as an implication of the form $X \Rightarrow Y$, where $X$ is the rule body, and $Y$ is the rule head. A message supports a rule if the message hits both the rule body and the rule head. An example rule for the instance of Table 8.4 could be \{v_2, v_3\} \Rightarrow \{v_1\}, meaning that if a message hits both $v_2$ and $v_3$, this message also matches $v_1$’s interest. Among all 5 messages, only $m_4$ supports this rule, i.e., only $m_4$ hits both \{v_2, v_3\} and \{v_1\}

The confidence of a rule is defined as as the percentage of messages supporting the rule out of all messages supporting the rule body. More formally,

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{8.16}$$

The confidence of a rule is its strength or reliability. For example, the rule \{v_2, v_3\} \Rightarrow \{v_1\} has a confidence of $\frac{4}{5} = 0.5$ for the given message set of Table 8.4, which means that for 50% of the time the message hits both $v_2$ and $v_3$, it also hits $v_1$. Note: $supp(X \cup Y)$ means “support for occurrences of messages which hit both X and Y both,” not “support for occurrences of messages which hit either X or Y.”

If we look at a single-node set, $conf(\{v\} \Rightarrow \{w\})$ shows the preference of $v$ to choose $w$ as its neighbor, i.e., the potential for $v$ to connect with $w$. We use $conf(\{v\} \Rightarrow \{w\})$ as the ranking function for node $v$ to choose its neighbors among other nodes in Alg. 38.

### 8.5 Evaluation

In this section, we describe the experimental evaluation of our algorithm. We implemented ConfODA and other auxiliary protocols in Java. We use LowODA [69] as a baseline by letting it interpret each message as a topic, because (1) the output overlay achieves 100% recall and 100% precision under ROL(δ) as $\delta \geq 1$ (see §8.5.1) and (2) LowODA achieves sub-linear approximation ratios for both maximum and average node degrees. However, LowODA does not guarantee the required bound on node degrees.
We synthetically generate pub/sub workloads as follows: Each message \( m \in M \) is associated with probability \( p(m) \) and each node \( v \in V \) hits \( m \) with a probability \( p(m) \). The value of \( p(m) \) is distributed according to either an exponential, a Zipfian, or a uniform distribution, which we refer to as Expo, Zipf, or Unif, respectively. According to [36], these distributions are representative of actual workloads used in industrial pub/sub systems today: Expo is used by stock-market monitoring engines for the study of stock popularity in the New York Stock Exchange [90], and Zipf faithfully captures the feed popularity distribution in RSS feeds [64].

8.5.1 Impact of neighborhood parameter \( \delta \)

We explore the performance of pub/sub overlays under \( \text{ROL}(\delta) \), parameterized by different values of \( \delta \). We set \( |V| = 100 \) and \( |M| = 10^6 \) in this experiment.

![Recall / Precision / F1-score](image)

(a) LowODA under Unif

![Recall / Precision / F1-score](image)

(b) ConfODA under Unif

Figure 8.2: ConfODA vs. LowODA under ROL(\(\delta\)) with different \(\delta\)

Fig. 8.2 depicts how the overlays (constructed by ConfODA and LowODA) perform given \( \text{ROL}(\delta) \) in terms of recall, precision and F1-score. The figure shows that ConfODA yields similar recall, precision, and F1-score as compared to those of LowODA under various \( \text{ROL}(\delta) \). Under Unif, \( (\text{LowODA}, \text{ROL}(\delta)) \) yields 100% recall and 1.22% precision, on average; as \( \delta \geq 1 \), \( \text{Recall}_{(\text{LowODA}, \text{ROL}(\delta))} = 1 \), \( \text{Precision}_{(\text{LowODA}, \text{ROL}(\delta))} = 1 \), and \( \text{F1-score}_{(\text{LowODA}, \text{ROL}(\delta))} = 1 \), which are the best possible values for these metrics. Similarly, ConfODA tends to achieve higher recall, precision and F1-score as \( \text{ROL}(\delta) \) increases \( \delta \). At \( \delta = 1 \), \( \text{Recall}_{(\text{ConfODA}, \text{ROL}(1))} = 0.952 \),
Precision_{\text{ConfODA,ROL(1)}}=1$, and $F1$-score_{\text{ConfODA,ROL(1)}}=0.976$ on average.

![Figure 8.3: (ConfODA, ROL(δ)) vs. (LowODA, ROL(δ)) under different distributions](image)

In Fig. 8.3, we compare between $(\text{ConfODA, ROL}(\delta))$ and $(\text{LowODA, ROL}(\delta))$ with regards to $F1$-score, the average $\delta$-neighborhood degree, and routing cost.

(a) As $\delta$ increases, both $F1$-score_{\text{ConfODA,ROL(δ)}} and $F1$-score_{\text{LowODA,ROL(δ)}} increase, which stay close to 1 when $\delta \geq 1$. The $F1$-score_{\text{LowODA,ROL(δ)}} is slightly higher than $F1$-score_{\text{ConfODA,ROL(δ)}} for most cases, but the difference is insignificant and less than 0.034 under Unif, on average.

(b) The average $\delta$-neighborhood degrees grow as $\delta$ increases. We denote by $|\mathcal{M}(\delta)|_{\text{A}}$ the average $\delta$-neighborhood degree of the overlay produced by overlay design algorithm $\mathcal{A}$. On average, $|\mathcal{M}(\delta)|_{\text{ConfODA}}$ is lower than $|\mathcal{M}(\delta)|_{\text{LowODA}}$, especially when $\delta \leq 2$. At $\delta = 1$, the average $\delta$-neighborhood degree equals the average node degree plus one; we have $|\mathcal{M}(1)|_{\text{ConfODA}} = 14.97$ and $|\mathcal{M}(1)|_{\text{LowODA}} = 84.5$.

(c) We measure the routing cost by the cumulative running time of Protocol 37 invoked at all nodes in $V$ for disseminating all messages in $M$, denoted by $R_{(\mathcal{A}, \mathcal{P})}$, where $\mathcal{A}$ is the overlay construction algorithm and $\mathcal{P}$ is the routing protocol. The figure shows that $R_{(\text{ConfODA,ROL(δ)})}$ is significantly lower than $R_{(\text{LowODA,ROL(δ)})}$ under all conditions: the ratio of $R_{(\text{ConfODA,ROL(δ)})} : R_{(\text{LowODA,ROL(δ)})}$ is 0.19 under Unif, 0.34 under Zipf, and 0.15 under Expo. The routing costs rise as $\delta$ increases,
since the running time of $\mathcal{R}OL(\delta)$ is determined by the $\delta$-neighborhood: as $\mathcal{N}(\delta)$ expands, each node possesses more local knowledge for matching and routing, which incurs more runtime cost for choosing the immediate neighbors for forwarding messages.

In summary, as $\delta$ increases, each node possesses more knowledge of the overlay network, thus both recall and precision tend to improve at the cost of large $\delta$-neighborhood and high routing cost. ConfODA outperforms LowODA in terms of the balance among these multiple dimensions. Although LowODA achieves perfect recall and precision as $\delta \geq 1$, the output overlay requires high node degrees, the local view (i.e., $\delta$-neighborhood) is large, and the routing cost is expensive. Meanwhile, ConfODA is capable of achieving high (but not perfect) recall and precision, while the overlay node degree, the $\delta$-neighborhood in the local view, and routing cost are reasonably small.

For the rest of our evaluations, we set $\delta = 1$ and make the input instance vary in size. We further demonstrate the scalability and efficiency of ConfODA for constructing pub/sub overlays.

### 8.5.2 Node Degrees

![Figure 8.4: Node degrees](image)

We look at both the maximum and average node degrees in the output overlays produced by different algorithms. For a specific algorithm $\mathcal{A}$, we denote the maximum and average node degree as $D_\mathcal{A}$ and $\overline{d}_\mathcal{A}$, respectively.

Fig. 8.4 depicts the comparison between ConfODA and LowODA in terms of node degrees under different distributions, where $|V| = 100$ and $|\mathcal{M}| \in [10^5, 10^6]$. We set $\alpha = 10$ and $\beta = 5$ for ConfODA and $\rho = 3$ for LowODA.
As the figure shows, ConfODA produces bounded node degrees, i.e., $d_{\text{ConfODA}} \leq D_{\text{ConfODA}} \leq \alpha + \beta = 15$. However, the overlays output by LowODA exhibit significantly high node degrees. More specifically, $D_{\text{LowODA}}$ is 98.53 under Unif, 73.90 under Zipf, and 93.97 under Expo, on average; and $d_{\text{LowODA}}$ is 88.25 under Unif, 48.24 under Zipf, and 81.25 under Expo, on average. Besides that, also, the node degrees increase with the number of messages: For example, as $|M| = 10^6$, $D_{\text{LowODA}} = 98$ and $d_{\text{LowODA}} = 84.96$ under Unif, $D_{\text{LowODA}} = 98.7$, and $d_{\text{LowODA}} = 88.34$ under Expo. Since the entire node set is only of size 100, the overlay produced by LowODA, almost forms a clique that contains all possible edges. Also, high node degrees in the overlay directly lead to high degrees of the $\delta$-neighborhood, which determines the size of the routing tables in ROL($\delta$).

These experimental results show that reaching content-connectivity for all messages (via LowODA) is not scalable for general pub/sub systems, especially as the number of messages scales up. This is in line with the observations of the exponential growth in the number of logic topics when transforming content-based pub/sub to topic-based pub/sub [20, 23] (see §8.1). Fortunately, ConfODA does not suffer from this serious drawback, because it relaxes the strict requirement of reaching content-connectivity and it bounds node degrees explicitly by the input parameters. We next show that ConfODA also achieves high performances in other metrics.

### 8.5.3 Recall, Precision and F1-score

![Recall, precision and F1-score](https://via.placeholder.com/150)

Fig. 8.5 depicts recall, precision, and F1-score for (ConfODA, ROL(1)) under different distributions. We do not include the plots for (LowODA, ROL(1)) because they always reach 100%
recall and 100% precision (at the cost of high node degrees). In this experiment, $|V| = 100$ and $|M| \in [10^5, 10^6]$. We set $\alpha = 10$ and $\beta = 5$ for ConfODA, $\rho = 3$ for LowODA, and $\delta = 1$.

As shown in the figure, (ConfODA, ROL(1)) reaches high recall and precision. Also, the metrics tend to further grow as the number of messages in the system increases. This experiment demonstrates that ConfODA scales with the number of messages – the overlay constructed by ConfODA is better capable of message dissemination with more input, i.e., messages and their hit functions.

### 8.5.4 Runtime cost of overlay construction

![Figure 8.6: Runtime cost of overlay construction](image)

We look at the runtime cost of different overlay design algorithms. For a specific algorithm $A$, we denote by $T_A$ the running time of $A$. Fig. 8.6 compares $T_{\text{ConfODA}}$ and $T_{\text{LowODA}}$ under different distributions, where $|V| = 100$ and $|M| \in [10^5, 10^6]$. We set $\alpha = 10$ and $\beta = 5$ for ConfODA and $\rho = 3$ for LowODA.

As Fig. 8.6 shows, ConfODA runs significantly faster than LowODA: the running time ratio $T_{\text{ConfODA}} : T_{\text{LowODA}}$ is 0.18 under Unif, 0.28 under Zipf, and 0.22 under Expo. This speedup stems from two sources: first, ConfODA employs a less complicated and more efficient edge selection heuristic than LowODA; second, ConfODA adds much fewer edges as compared to LowODA and shown in §8.5.2.

In sum, ConfODA outperforms LowODA in terms of running time under all distributions.
8.5.5 Runtime cost of message routing

We measure the efficiency of pub/sub message dissemination on the overlays output by different overlay design algorithms. More specifically, we plot both $R_{\text{ConfODA,ROL}(1)}$ and $R_{\text{LowODA,ROL}(1)}$ where $|V| = 100$ and $|M| \in [10^5, 10^6]$. ConfODA has fixed parameters $\alpha = 10$ and $\beta = 5$, and LowODA has $\rho = 3$.

As Fig. 8.7 shows, ROL(1) is more efficient for message dissemination on the overlays produced by ConfODA as compared to those of LowODA. The time ratio of $R_{\text{ConfODA,ROL}(1)} : R_{\text{LowODA,ROL}(1)}$ is 0.27 under Unif, 0.49 under Zipf, and 0.26 under Expo. The routing cost of ROL(1) is highly dependent on the node degrees of the overlay, and that is why ROL(1) has superior performance with ConfODA. Note that this advantage would be more significant in practical pub/sub systems, because the $Hit$ function does not exist and it requires sophisticated algorithms to match the incoming publication against all subscriptions.

These experiments show that ConfODA provides better overlays to support pub/sub communication as compared to LowODA: faster message delivery, smaller routing tables, and less matching complexity.

8.6 Conclusion

In this work, we formalize the problem for content-based pub/sub overlay design, which attempts to maximize both recall and precision under a specific routing protocol with bounded node degree in the
output overlay. We develop the efficient and scalable ConfODA algorithm that outperforms TCO-based approaches: ConfODA achieves high recall while keeping the node degree low as opposed to TCO-based algorithms, which produce clique-like structures in order to achieve full recall. We experimentally demonstrate the significance of overlay network design for content-based pub/sub. This work embarks on the challenging and promising problem of constructing overlay networks for content-based pub/sub systems. Thus, our algorithms set the stage for future approaches in this line of research.
Chapter 9

Conclusions

9.1 Summary

This thesis contributes to overlay network design for pub/sub systems, with an emphasis on topic-based pub/sub systems. Our work has laid both theoretical groundwork and practical system architecture for the overlays of topic-based pub/sub systems. Our research provides the underlying infrastructure to design, develop, implement, deploy and maintain a large-scale real-world topic-based pub/sub system.

Chapter 3 develops the Gen-ODA framework that covers existing greedy algorithms with different edge selection rules for different optimization criteria for topic-based pub/sub overlays. The Gen-ODA algorithm remains the-state-of-the-art algorithms to construct TCOs from scratch - it has the lowest time complexity and the best approximation ratios with regard to different optimization targets. We use Gen-ODA as the building block for developing more advanced approaches for TCO construction.

Based on the implementation of Gen-ODA, Chapter 4 and Chapter 5 further improve the time and space efficiency of TCO construction by integrating divide and conquer techniques. In particular, we propose a number of divide and conquer algorithms for the MinAvg-TCO problem in Chapter 4 and the MinMax-TCO problem in Chapter 5, respectively.

In Chapter 6, we consider a new family of optimization problems MinAvg-\(k\)-TCO that constructs
reliable overlay networks for topic-based pub/sub. Our GM$_2$ algorithm approximates the MinAvg-2TCO problem within a proven bound, that almost meets the lower bound on the approximation ratio for the problem. We also design a heuristic algorithm for the MinAvg-kTCO problem, namely the HararyPT algorithm, especially for highly correlated pub/sub workloads.

In Chapter 7, we push pub/sub overlay network design from theory to practice. We present a fully dynamic system, ElastO, to construct and maintain low fan-out TCOs for topic-based pub/sub under churn. ElastO combines the advantages from both the static algorithms and the decentralized protocols: (a) ElastO produces low maximum and average node degrees that are insignificantly higher than those of the static algorithms; (b) ElastO repairs the TCO efficiently such that the time efficiency is of the same order of magnitude as the decentralized protocols.

Finally, in Chapter 8, we make the first attempt to move from the topic-based pub/sub overlay to the content-based pub/sub overlay. We analyze the key challenges for content-based pub/sub overlay design, and formalize a family of problems that captures the key trade-offs between overlay design and routing protocols for content-based pub/sub. We devise a novel algorithm that achieves high message hit ratio with bounded node degrees.

This work serves as the foundation and baseline for the future exploration on overlay network design for content-based pub/sub or other distributed systems.

### 9.2 Future Work

We would like to make the GM$_2$ algorithm more practical and scalable by improving the current time efficiency (i.e., $O(|V|^4|T|)$). We need to design efficient data structures for GM$_2$’s implementation. Besides, it is important to formally and experimentally explore more tradeoffs between kTCO and low node degree, such as the MinMax-kTCO problem as defined below:

**Problem 8.** The MinMax-kTCO($V$, $T$, Int) problem parameterized by an integer $k$ is defined as: Given a set of nodes $V$, a set of topics $T$, and the interest function $Int$, construct a $k$TCO with the minimum maximum node degree.

Our first intuition is that the MinMax-kTCO problem is difficult to approximate, which we formally summarized in Conjecture 1:
Conjecture 1. For any given positive integer \( k \), the MinMax-\( k \) TCO problem parameterized by \( k \) is NP-complete and can not be approximated in polynomial time within a logarithmic factor of \( O(\log |T|) \) unless \( P = NP \).

Conjecture 1 is true when \( k = 1 \) [70]. However, the existing proof techniques do not directly apply to general cases. The first obstacle is the formal proof to validate our conjecture. Second, we need to design efficient algorithms to achieve \( k \) TCO with a low maximum node degree, which we can build upon the design and analysis of the \( G\bar{M}^2 \) algorithm and divide and conquer approaches. Third, another important line for future work would be to design distributed algorithms and protocols for constructing and maintaining \( k \) TCO in dynamic environment.

Current research on pub/sub overlay network design (including this thesis) is mostly exploring topic-based pub/sub systems because of its relatively simple and straightforward model. In the topic-based pub/sub, we aim to organize nodes that are interested in the same topic into a connected sub-overlay, which is abstracted in the concept of TCO. TCO is a well-defined property for topic-based pub/sub, which supports a clear-cut decomposition between overlay infrastructure and routing protocols: First, TCO enables simple and efficient routing, and thus we can focus on overlay construction without worrying too much about routing protocols design at the same time. Second, TCO captures the trade-offs between the scalability of the overlay (e.g., node degree) and the routing cost (e.g., forwarding overhead).

Overlay design for the content-based pub/sub is an important and promising direction with new challenges. With regard to the content-based pub/sub, it is intuitive to group together nodes with “similar” content in the overlay. However, unlike the concept of TCO for topic-based pub/sub, it is difficult (at least not straightforward) to define “content-connectivity” for content-based pub/sub without losing its practical characters, especially since there is no explicit boundary to classify pub/sub messages in the continuous content space. Moreover, there is lack of simple routing for content-based pub/sub in general, due to the fuzzy decomposition between the underlaying overlay network and routing protocol design.

One way for content-based pub/sub overlay design is to address several key sub-problems separately: First, cluster nodes into subscriber groups with “similar” content, i.e., transforming the continuous content space into discrete groups. Second, build a graph with “content-connectivity”
based on the subscriber groups produced in the first step. Third, design routing protocols that disseminate messages inside and outside the subscriber groups.
Appendix A

Analysis of MinAvg-\(k\) TCO Problem and \(GM^2\) Algorithm

A.1 Complexity of the Parameterized MinAvg-\(k\) TCO Problem

Theorem A.1.1. MinAvg-2 TCO is NP-complete.

Proof. First, MinAvg-2 TCO is in NP. We consider the decision version of MinAvg-2 TCO: Each instance of the problem has \(I(V,T,Int)\) and a constant \(m_0\). We need to answer the question of whether there exists a 2TCO such that the number of edges is no more than \(m_0\), i.e., \(\leq m_0\). If we are given a candidate overlay TPSO\((V,T,Int,E)\), we can verify in polynomial time (1) whether this candidate is 2TCO by computing the blocks of each sub-graph \(G(t)\) where \(t \in T\) [93, 46], and (2) whether \(|E| \leq m_0\).

Second, we prove MinAvg-2 TCO is NP-hard by a reduction from MinAvg-TCO to MinAvg-2 TCO. We look at the decision versions for both problems. Given an instance \(I(V,T,Int,m)\) for the decision version of MinAvg-TCO, we need to give a yes/no answer to the question of whether there exists TCO\((V,T,Int,E)\) such that \(|E| \leq m\). Without loss of generality, we can denote \(|V| = n\) and number all nodes as \(V = \{v_1, v_2, ..., v_n\}\). We construct an instance \(I'(V',T',Int',m')\) for the decision version of MinAvg-2 TCO, which asks whether there exists 2TCO'\((V',T',Int',E')\)
such that $|E'| \leq m'$. The construction can be achieved in polynomial time as follows (see Fig. A.1):

- $V'$ includes all nodes in $V$ and adds a new node $v'$, i.e., $V' = V \cup \{v'\} = \{v_1, v_2, ..., v_n, v'\}$.
- $T'$ includes all topics in $T$ and adds a topic $t'_i$ for each node $v_i \in V$, i.e., $T' = T \cup \{t'_1, t'_2, ..., t'_n\}$.
- $\text{Int'}(v_i, t) = \text{Int}(v_i, t), \forall t \in T; \text{Int'}(v_i, t'_i) = \text{true}, \text{Int'}(v_i, t'_j) = \text{false}$ if $i \neq j; \text{Int'}(v', t) = \text{true}, \forall t \in T'$.
- $m' = m + n$.

![Figure A.1: The construction of a $2TCO$ instance from a $1TCO$ instance: (a) $1TCO(V, T, \text{Int})$; (b) $2TCO'(V', T', \text{Int'})$.](image)

Suppose there is some $TCO(V, T, \text{Int}, E)$, for instance, $I$ such that $|E| \leq m$, then there is $2TCO'(V', T', \text{Int'}, E')$, for instance, $I'$ such that $|E'| \leq m' = m + n$. Consider the edge set $E' = E \cup \{(v', v_1), (v', v_2), ..., (v', v_n)\}$, then $|E'| \leq m + n$ and $G'(V', E')$ satisfies 2-topic-connectivity for $I'(V', T', \text{Int'})$, because 1. $G'(V', E')$ will remain topic-connected when removing $v'$ from the graph since the remaining graph is identical to $G(V, E)$; 2. $G'(V', E')$ will remain topic-connected when removing any $v_i \in V$ because $v'$ connects to every other node.

On the other hand, if there is some $2TCO'(V', T', \text{Int}, E')$, for instance, $I'$ such that $|E'| \leq m'$, then there is $TCO(V, T, \text{Int}, E)$ such that $|E| \leq m = m' - n$. In $2TCO'$, $v'$ has to connect to every other nodes to attain topic-connectivity for the newly introduced topics $\{t'_1, t'_2, ..., t'_n\}$, i.e., $E'$ contains all edges $\{(v', v_1), (v', v_2), ..., (v', v_n)\}$. We construct $E = E' \setminus \{(v', v_1), (v', v_2), ..., (v', v_n)\}$, then $|E| \leq m$ and $G(V, E)$ satisfy topic-connectivity with regards to $I(V, T, \text{Int})$, because $G'(V', E')$ remains topic-connected when removing $v'$ from the graph, which turns out to be $G(V, E)$.

The lower bound on the approximability of MinAvg-TCO was proven to be $\Omega(\log |V|)$ un-
less \( P = NP \) [15]. Based on this result, we provide Theorem A.1.2 for the inapproximability of \( \text{MinAvg-2TCO} \).

**Theorem A.1.2.** \( \text{MinAvg-2TCO} \) can not be approximated in polynomial time within a factor of \( O(\log |V|) \) unless \( P = NP \).

**Proof.** We follow the same reduction and notation as presented in the proof of Theorem A.1.1. Denote \( m_{\text{opt}} \) as the minimum number of edges for the optimal solution of \( \text{MinAvg-TCO}(V, T, \text{Int}) \) and \( m'_{\text{opt}} \) as the minimum number of edges for the optimal solution of \( \text{MinAvg-2TCO}(V, T, \text{Int}) \), then

\[
m_{\text{opt}} = m'_{\text{opt}} - n \quad \text{(A.1)}
\]

Suppose, by contradiction, that there is a polynomial algorithm which achieves an approximation ratio of \( O(\log n) \) for \( \text{MinAvg-2TCO} \). Let the output be \( 2\text{TCO}'(V', T', \text{Int}', E') \) and \( |E'| = m' \), then there exists a constant \( C \) such that

\[
m' \leq m'_{\text{opt}} \cdot C \log n \quad \text{(A.2)}
\]

With regards to the corresponding \( \text{TCO}(V, T, \text{Int}, E) \) where \( |E| = m \), we have

\[
m = m' - n \leq m'_{\text{opt}} \cdot C \log n - n
\]

\[
= (m_{\text{opt}} + n) \cdot C \log n - n
\]

\[
= m_{\text{opt}} \cdot C \log n - (n - C \log n)
\]

\[
= m_{\text{opt}} \cdot O(\log n) \quad \text{(A.3)}
\]

So, \( \text{TCO}(V, T, \text{Int}, E) \) achieves an approximation ratio of \( O(\log |V|) \), which contradicts Theorem 1 in [15]. \( \square \)

We further generalize the results for \( \text{MinAvg-kTCO} \) by induction on \( k \) with the same proof techniques.
Theorem A.1.3. For any given positive integer $k$, the MinAvg-$k$ TCO problem parameterized by $k$ is NP-complete and cannot be approximated in polynomial time within a factor of $O(\log |V|)$ unless $P = NP$.

### A.2 Correctness and Running Time of the GM$_2$ Algorithm

Lemma A.2.1. Alg. 28 outputs a 2TCO after at most $|V|^2$ iterations of the while loop in Lines 3-8.

Proof. It follows directly from the pseudo code that Alg. 28 always outputs a 2TCO. At each iteration of the while loop in Lines 3-8, one edge is added to the current overlay network. Hence, the algorithm terminates in $|E_{GM_2}|$ iterations, which is bounded by $|V|^2$. □

Lemma A.2.2. The running time of Alg. 28 is $O(|V|^4 |T|)$.

Proof. Consider the runtime cost of each iteration in the while loop in Lines 3-8. For each topic $t \in T$, a Depth-First-Search-based algorithm can find all blocks in the current topic-induced sub-overlay with $O(|V|^2)$ [46]. Thus, it takes $O(|V|^2)$ time to compute the edge estimate on topic $t$ for each potential edge. The time complexity at each iteration is $O(|V|^2 |T|)$ across all topics in $T$. According to Lemma A.2.1, the running time of Alg. 28 is $O(|V|^4 |T|)$. □

### A.3 Building Blocks to Complete the Proof of Lemma 36

Claim A.3.1. The ear $C_j^{(t)}$ reduces the number of TC-blocks on topic $t \in T$ in $(V^{(t)}, S_j^{(t)})$ by $|C_j^{(t)}| - 1$, i.e.,

$$B\left( V^{(t)}, S_{j-1}^{(t)} \right) - B \left( V^{(t)}, S_j^{(t)} \right) = |C_j^{(t)}| - 1 , \ \forall C_j^{(t)} \text{ in } D^{(t)} = \left[ C_1^{(t)}, ..., C_z^{(t)} \right] .$$

Proof. Given an instance $I(V, T, Int)$ and an edge sequence $E$ that produces a 2TCO, we prove this claim by induction on $j$, the index for the ears in the $E$-ear-decomposition $D^{(t)}$.

- **Base case:** $j = 1$. When the edge set is $\emptyset$, there are $|V^{(t)}|$ singleton TC-blocks in $(V^{(t)}, \emptyset)$,

$$B \left( V^{(t)}, \emptyset \right) = |V^{(t)}| . \quad (A.4)$$
Consider the first cycle $C_1^{(t)}$ in the $D^{(t)}$: $C_1^{(t)} = S_1^{(t)}$ has $|C_1^{(t)}|$ edges to connect $|C_1^{(t)}|$ nodes, and all $|C_1^{(t)}|$ nodes belong to one $TC$-block in $\left(V^{(t)}, C_1^{(t)}\right)$. Apart from the $TC$-block that is composed of $|C_1^{(t)}|$ nodes, there are $\left(|V^{(t)}| - |C_1^{(t)}|\right)$ singleton $TC$-blocks in $\left(V^{(t)}, C_0^{(t)}\right)$. Thus the total number of $TC$-blocks in $\left(V^{(t)}, S_0^{(t)}\right)$ is $\left(|V^{(t)}| - |C_0^{(t)}| + 1\right)$, i.e.,

$$B\left(V^{(t)}, S_1^{(t)}\right) = |V^{(t)}| - |C_1^{(t)}| + 1.$$  \hspace{1cm} (A.5)

Therefore, edges in $S_1^{(t)}$ belong to one $TC$-block in $\left(V^{(t)}, S_1^{(t)}\right)$, and

$$B\left(V^{(t)}, \emptyset\right) - B\left(V^{(t)}, S_1^{(t)}\right) = |C_1^{(t)}| - 1.$$  \hspace{1cm} (A.6)

- **Inductive hypothesis**: assume inductively that all edges in $S_r^{(t)}$ belong to one $TC$-block in $\left(V^{(t)}, S_r^{(t)}\right)$, and

$$B\left(V^{(t)}, S_{r-1}^{(t)}\right) - B\left(V^{(t)}, S_r^{(t)}\right) = |C_r^{(t)}| - 1, \quad \forall \ r \leq j - 1, \text{ where } 1 < j \leq z.$$  \hspace{1cm} (A.7)

- **Inductive step**: Based on the inductive hypothesis, all edges in $S_{j-1}^{(t)}$ belong to one $TC$-block in $\left(V^{(t)}, S_{j-1}^{(t)}\right)$ and we denote the node set in this block as

$$W_{j-1}^{(t)} = \{v \in V^{(t)} | \exists e \in S_{j-1}^{(t)} \text{ s.t. } e \text{ is incident to } v\}.$$  

So the number of $TC$-blocks in $\left(V^{(t)}, S_{j-1}^{(t)}\right)$ is

$$B\left(V^{(t)}, S_{j-1}^{(t)}\right) = 1 + \left(|V^{(t)}| - |W_{j-1}^{(t)}|\right).$$  \hspace{1cm} (A.7)

Adding ears preserves 2-connectedness (see the Whitney Theorem in [93]), so all edges in $S_j^{(t)} = S_{j-1}^{(t)} \cup C_j^{(t)}$ belong to one $TC$-block in $\left(V^{(t)}, S_j^{(t)}\right)$, which we denote as

$$W_j^{(t)} = \{v \in V^{(t)} | \exists e \in S_j^{(t)} \text{ s.t. } e \text{ is incident to } v\}.$$  

Similar to Eq. (A.7), the number of TC-blocks in \( \left( V^{(t)}, S_j^{(t)} \right) \) is
\[
B \left( V^{(t)}, S_j^{(t)} \right) = 1 + \left( |V^{(t)}| - |W_j^{(t)}| \right) .
\]  
(A.8)

The ear \( C_j^{(t)} \) has \( |C_j^{(t)}| \) edges to connect \( \left( |C_j^{(t)}| + 1 \right) \) nodes: 2 terminal nodes are in \( W_{j-1}^{(t)} \) and \( \left( |C_j^{(t)}| - 1 \right) \) are in \( V^{(t)} \setminus W_{j-1}^{(t)} \). As compared to \( W_{j-1}^{(t)} \), \( W_j^{(t)} \) contains additional \( \left( |C_j^{(t)}| - 1 \right) \) nodes from the ear \( C_j^{(t)} \), so
\[
|W_j^{(t)}| - |W_{j-1}^{(t)}| = |C_j^{(t)}| - 1
\]  
(A.9)

Combining Eq. (A.7), (A.8) and (A.9), edges in \( S_j^{(t)} \) belong to one TC-block in \( \left( V^{(t)}, S_j^{(t)} \right) \) and
\[
B \left( V^{(t)}, S_{j-1}^{(t)} \right) - B \left( V^{(t)}, S_j^{(t)} \right) = |W_j^{(t)}| - |W_{j-1}^{(t)}| = |C_j^{(t)}| - 1
\]  
(A.10)

\( \square \)

**Claim A.3.2.** Given \( P_i = \langle e_1, ..., e_i \rangle, \forall E, R \in \mathcal{E}(P_i), contrib(e_j, E) \leq contrib(e_j, R) \leq 2 \cdot contrib(e_j, E), 1 \leq j \leq i. \)

**Proof.** Since \( E, R \in \mathcal{E}(P_i) \), by Eq. (6.8), for any \( t \in T \), we have either
\[
contrib^{(t)}(e_j, E) = contrib^{(t)}(e_j, R) = 0, 1 \leq j \leq i
\]
or
\[
contrib^{(t)}(e_j, E), contrib^{(t)}(e_j, R) \in \left[ \frac{1}{2}, 1 \right), 1 \leq j \leq i
\]
As a result,
\[
contrib^{(t)}(e_j, E) \leq contrib^{(t)}(e_j, R) \leq 2 \cdot contrib^{(t)}(e_j, E), 1 \leq j \leq i, \forall t \in T \quad (A.11)
\]
Furthermore, \[
\sum_{t \in T} \text{contrib}^{(t)}(e_j, E) \leq \sum_{t \in T} \text{contrib}^{(t)}(e_j, R) \leq 2 \sum_{t \in T} \text{contrib}^{(t)}(e_j, E), \quad 1 \leq j \leq i \quad \text{(A.12)}
\]

By the definition in Eq. (6.5), we have \[
\text{contrib}(e_j, E) \leq \text{contrib}(e_j, R) \leq 2 \cdot \text{contrib}(e_j, E), \quad 1 \leq j \leq i \quad \text{(A.13)}
\]

Claim A.3.3. Given \(P_i\) and its corresponding \(\bar{P}_i\), \(\forall R \in \mathcal{E}(P_i), \text{contrib}(e, R) \leq \text{estimate}(e, P_i) \leq 2 \cdot \text{contrib}(e, R)\).

Proof. Given \(R \in \mathcal{E}(P_i)\), we first fix some \(t \in T\). Based on the definition in Eq. (6.9), for any \(e \in (V(t) \times V(t))\), we have either \[
\text{contrib}^{(t)}(e, R) = \text{estimate}^{(t)}(e, P_i) = 0
\]
or \[
\text{contrib}^{(t)}(e_j, R) \in \left[\frac{1}{2}, 1\right) \quad \text{and} \quad \text{estimate}^{(t)}(e, P_i) = 1
\]
As a result, \[
\text{contrib}^{(t)}(e, R) \leq \text{estimate}^{(t)}(e, P_i) \leq 2 \cdot \text{contrib}^{(t)}(e, R), \quad \forall t \in T, \text{ where } R \in \mathcal{E}(P_i) \quad \text{(A.14)}
\]

Furthermore, \[
\sum_{t \in T} \text{contrib}^{(t)}(e, R) \leq \sum_{t \in T} \text{estimate}^{(t)}(e, P_i) \leq 2 \sum_{t \in T} \text{contrib}^{(t)}(e, R), \quad \text{where } R \in \mathcal{E}(P_i) \quad \text{(A.15)}
\]
By the definition in equations (6.5) and (6.10), we have

\[ \text{contrib}(e, R) \leq \text{estimate}(e, P_i) \leq 2 \cdot \text{contrib}(e, R), \forall e \in \left(V^{(t)} \times V^{(t)}\right), \text{ where } R \in E(P_i) \]

\[(A.16)\]

\[\square\]

**Derivation A.3.1.** The derivation from Eq. (6.15) to Eq. (6.15) in the proof of Lemma 36.

**Proof.**

\[
\Phi(i, E) - \Phi(i + 1, E) \geq \frac{2\Phi(i, E) - (B_{\text{start}} + B_{\text{end}})}{2m^*}
\]

\[
\Rightarrow \Phi(i, E) - \frac{1}{m^*} \Phi(i, E) \geq \Phi(i + 1, E) - \frac{1}{m^*} \bar{B}, \text{ where } \bar{B} = \frac{B_{\text{start}} + B_{\text{end}}}{2}
\]

\[
\Rightarrow \left(1 - \frac{1}{m^*}\right) \Phi(i, E) - \left(1 - \frac{1}{m^*}\right) \bar{B} \geq \Phi(i + 1, E) - \frac{1}{m^*} \bar{B} - \left(1 - \frac{1}{m^*}\right) \bar{B}
\]

\[
\Rightarrow \left(1 - \frac{1}{m^*}\right) \left(\Phi(i, E) - \bar{B}\right) \geq \Phi(i + 1, E) - \bar{B}
\]

\[(A.17)\]

\[\square\]

**Derivation A.3.2.** The derivation for the bounds of \(\lambda_0\) and \(\lambda_1\) in the proof of Lemma 36, i.e., Eq. (6.17) and (6.18).

**Proof.** Eq. (A.17) shows the progression of the potential function value within successive iterations in \(\mathbb{GM}^2\) as compared to the optimal solution. Based on Eq. (A.17), we derive the bound on the number of iterations of Alg. 28 (i.e., the number of edges in \(E\)) relative to \(m^*\).

Given \(E\), we take \(\Phi(i, E) - \bar{B}\) as a function of \(i\), and it decreases as \(\mathbb{GM}^2\) adds an edge at each iteration. Initially, \(\Phi(0, E) - \bar{B} = \frac{B_{\text{start}} - B_{\text{end}}}{2} > 0\), and finally, \(\Phi(m, E) - \bar{B} = -\frac{B_{\text{start}} - B_{\text{end}}}{2} < 0\). So at some iteration \(\lambda_0\), the function turns from positive to negative. We have the values of the function as a sequence with regard to \(i\):

\[\left\langle \Phi(1, E) - \bar{B}, \ldots, \Phi(\lambda_0, E) - \bar{B}, \Phi(\lambda_0 + 1, E) - \bar{B}, \ldots, \Phi(m, E) - \bar{B} \right\rangle\]

\[(A.18)\]
The difference between successive elements in Eq. (A.18) is

$$
\left( \Phi(i, \mathcal{E}) - \bar{B} \right) - \left( \Phi(i + 1, \mathcal{E}) - \bar{B} \right) = \Phi(i, \mathcal{E}) - \Phi(i + 1, \mathcal{E}) = \text{contrib}(e_{i+1}, \mathcal{E}), \ 0 \leq i < n \tag{A.19}
$$

We look at the edge contribution at each iteration. Based on Eq. (6.4) and (6.8), every edge contribution on $t \in T$ is a value in \{1, 2, ..., $\frac{U-1}{U}$\}, where $U = \max\{|V(t)|, t \in T\}$. More specially,

$$
\text{contrib}^{(t)}(e, \mathcal{E}) \in \left\{ \frac{1}{2}, \frac{2}{3}, \ldots, \frac{U-1}{U} \right\}, \text{ where } U = \max\{|V(t)|, t \in T\}. \tag{A.20}
$$

The overall edge contribution $\text{contrib}(e, \mathcal{E})$ is the sum of at most $|T|$ values chosen from \{1, 2, ..., $\frac{U-1}{U}$\}.

Thus,

$$
\text{contrib}(e, \mathcal{E}) \geq \frac{1}{2}, \forall e \text{ in } \mathcal{E}, \tag{A.21}
$$

and

$$
\left| \Phi(i, \mathcal{E}) - \bar{B} \right| = \left| B_{\text{start}} - \sum_{j=1}^{i} \text{contrib}(e_j, \mathcal{E}) - \frac{B_{\text{start}} + B_{\text{end}}}{2} \right|
$$

$$
= \left| \frac{B_{\text{start}} - B_{\text{end}}}{2} - \sum_{j=1}^{i} \text{contrib}(e_j, \mathcal{E}) \right|
$$

$$
= \begin{cases} 
\text{or } \geq 1/LCM(1, 2, ..., U), & \forall 1 \leq i \leq m \\
\text{either } 0,
\end{cases} \tag{A.22}
$$

where $LCM(1, 2, ..., U)$ is the Least Common Multiple of 1, 2, ..., $U$.

We now return to the sequence in Eq. (A.18). Let $l_{0}^{*}$ be the smallest sequence index in Eq. (A.18) where the function value is smaller than 1/2, i.e.,

$$
l_{0}^{*} = \min \left\{ l_0 \mid \left( \Phi(l_0, \mathcal{E}) - \bar{B} \right) \leq \left( \Phi(0, \mathcal{E}) - \bar{B} \right) \left( 1 - \frac{1}{m^*} \right)^{l_0} \leq \frac{1}{2} \right\}. \tag{A.23}
$$
By Eq. (A.21), at the \((l_0^* + 1)\)-th iteration,

\[
\text{contrib}(e_{l_0^*+1}, E) = \left( \Phi(l_0^*, E) - \bar{B} \right) - \left( \Phi(l_0^* + 1, E) - \bar{B} \right) \geq 1/2 .
\]  
(A.24)

So we have either

\[
0 < \left( \Phi(l_0^*, E) - \bar{B} \right) \leq 1/2 \quad \text{and} \quad \left( \Phi(l_0^* + 1, E) - \bar{B} \right) \leq 0 ,
\]  
(A.25)

or

\[
\left( \Phi(l_0^*, E) - \bar{B} \right) \leq 0
\]  
(A.26)

In any case of Eq. (A.25) and (A.26),

\[
\lambda_0 \leq l_0^*,
\]  
(A.27)

because \( \left( \Phi(\lambda_0, E) - \bar{B} \right) > 0 \) and \( \left( \Phi(\lambda_0 + 1, E) - \bar{B} \right) \leq 0 \) (see the sequence in Eq. (A.18)).

We look back at Eq. (A.23). According to Eq. (A.17), \( \Phi(l_0, E) - \bar{B} \leq \left( \Phi(0, E) - \bar{B} \right) (1 - \frac{1}{m^*})^{l_0} \) always holds, so by definition, \( l_0^* \) is the smallest \( l_0 \) that satisfies

\[
\left( \Phi(0, E) - \bar{B} \right) \left( 1 - \frac{1}{m^*} \right)^{l_0} \leq \frac{1}{2} \\
\Leftrightarrow (B_{\text{start}} - B_{\text{end}}) \left( 1 - \frac{1}{m^*} \right)^{l_0} \leq 1 \\
\Leftrightarrow \ln(B_{\text{start}} - B_{\text{end}}) + l_0 \cdot \ln \left( 1 - \frac{1}{m^*} \right) \leq 0 \\
\Leftrightarrow l_0 \cdot \ln \left( 1 - \frac{1}{m^*} \right) \leq -\ln(B_{\text{start}} - B_{\text{end}}) \\
\Leftrightarrow l_0 \geq \frac{\ln(B_{\text{start}} - B_{\text{end}})}{-\ln \left( 1 - \frac{1}{m^*} \right)} \quad \text{\( \because \) since } \ln \left( 1 - \frac{1}{m^*} \right) < 0
\]  
(A.28)

Further, by the definition of \( l_0^* \) in Eq. (A.23),

\[
l_0^* = \left\lceil \frac{\ln(B_{\text{start}} - B_{\text{end}})}{-\ln \left( 1 - \frac{1}{m^*} \right)} \right\rceil
\]  
(A.29)
By Taylor Expansion,
\[
\ln \left( 1 - \frac{1}{m^*} \right) = -\frac{1}{m^*} - \frac{1}{2(m^*)^2} - \frac{1}{3(m^*)^3} - \cdots \text{, where } m^* > 0
\]
\[\Rightarrow \ln \left( 1 - \frac{1}{m^*} \right) < -\frac{1}{m^*} < 0\]
\[\Leftrightarrow -\ln \left( 1 - \frac{1}{m^*} \right) > \frac{1}{m^*} > 0\]
\[\Leftrightarrow -\ln \left( 1 - \frac{1}{m^*} \right) < m^* \tag{A.30}\]

Putting Eq. (A.30) into Eq. (A.28),

\[l_0^* \leq \left\lceil m^* \cdot \ln(B_{\text{start}} - B_{\text{end}}) \right\rceil. \tag{A.31}\]

Combining, Eq. (A.27) and (A.31),

\[\lambda_0 \leq l_0^* \leq \left\lceil m^* \cdot \ln(B_{\text{start}} - B_{\text{end}}) \right\rceil \leq \left\lceil m^* \cdot \ln B_{\text{start}} \right\rceil = m^* \cdot O(\ln B_{\text{start}}). \tag{A.32}\]

Now we try to derive the bound for \(\lambda_1\). By Eq. (A.18), \(\Phi(\lambda_0 + 1, E) - \bar{B} \leq 0\). We just consider \(\Phi(\lambda_0 + 1, E) - \bar{B} < 0\) is the first negative element in the sequence of Eq. (A.18) – for the case of \(\Phi(\lambda_0 + 1, E) - \bar{B} = 0\), \(\Phi(\lambda_0 + 2, E) - \bar{B} < 0\) would be the first negative element, and we could use the same technique to derive the bound for \((\lambda_1 - 1)\).

Since \(\bar{B} - \Phi(\lambda_0 + 1, E) > 0\), \(\lambda_1\) does not exceed the smallest \(l_1\) that satisfies

\[\left( \bar{B} - \Phi(\lambda_0 + 1, E) \right) \left( 1 - \frac{1}{m^*} \right)^{l_1} \geq \frac{B_{\text{start}} - B_{\text{end}}}{2}. \tag{A.33}\]

Similar to the derivation for Eq. (A.32), we can obtain

\[\lambda_1 \leq \left\lceil m^* \cdot \ln \frac{B_{\text{start}} - B_{\text{end}}}{2(\bar{B} - \Phi(\lambda_0 + 1, E))} \right\rceil. \tag{A.34}\]
By Eq. (A.22),

\[
\tilde{B} - \Phi(\lambda_0 + 1, E) \geq \frac{1}{\text{LCM}(1, 2, \ldots, U)}.
\]  

(A.35)

Recall that \(\text{LCM}(1, 2, \ldots, U)\) is the Least Common Multiple of 1, 2, \ldots, \(U\).

The prime number theorem [6] implies that

\[
\text{LCM}(1, 2, \ldots, U) = e^{U(1+O(1))} \text{ as } U \to \infty.
\]  

(A.36)

Putting Eq. (A.35) and (A.36) into (A.34), we have,

\[
\lambda_1 \leq m^* \cdot O(U + \ln B_{\text{start}}).
\]  

(A.37)
Bibliography


